Rendering menu glyphs is slightly trickier

devblogs.microsoft.com/oldnewthing/20050802-13

August 2, 2005

Raymond Chen

Last time, we saw how to draw themed and unthemed radio buttons, and I mentioned that menu glyphs are trickier. They're trickier because they are provided as raw monochrome bitmaps instead of fully-formed color-coordinated bitmaps. First, let's do it wrong in order to see what we get. Then we'll try to fix it. Start with a clean new scratch program

```
class RootWindow : public Window
{
 …
protected:
 void PaintContent(PAINTSTRUCT *pps);
 BOOL WinRegisterClass(WNDCLASS *pwc)
 {
  pwc->hbrBackground = (HBRUSH)(COLOR_MENU + 1);
  return __super::WinRegisterClass(pwc);
 }
 …
};


void RootWindow::PaintContent(PAINTSTRUCT *pps)
{
 int cxCheck = GetSystemMetrics(SM_CXMENUCHECK);
 int cyCheck = GetSystemMetrics(SM_CYMENUCHECK);
 RECT rc = { 0, 0, cxCheck, cyCheck };
 DrawFrameControl(pps->hdc, &rc, DFC_MENU, DFCS_MENUCHECK);
}
```

This naïvely uses the `DrawFrameControl` function to draw the menu check mark directly into the paint DC. If you are running with the default Windows XP theme you probably won't notice anything amiss, but switch to the Windows Classic theme and you'll see that the check mark is drawn in black and white even though the Classic menu background color is gray.

The reason for this is called out in the documentation for `DrawFrameControl`:

> If uType is either DFC_MENU or DFC_BUTTON and uState is not DFCS_BUTTONPUSH, the frame control is a black-on-white mask (that is, a black frame control on a white background).

All we get from `DrawFrameControl` is a monochrome mask. It is our responsibility to colorize it as necessary. To do this, we draw the mask into a monochrome bitmap, and then use the `BitBlt` function to colorize it. Recall that when blitting from a monochrome bitmap to a color bitmap, the color black in the source bitmap becomes the destination DC's text color, and the color white in the source bitmap becomes the destination DC's background color.

```
void RootWindow::PaintContent(PAINTSTRUCT *pps)
{
 HDC hdcMem = CreateCompatibleDC(pps->hdc);
 if (hdcMem) {
  int cxCheck = GetSystemMetrics(SM_CXMENUCHECK);
  int cyCheck = GetSystemMetrics(SM_CYMENUCHECK);
  HBITMAP hbmMono = CreateBitmap(cxCheck, cyCheck, 1, 1, NULL);
  if (hbmMono) {
   HBITMAP hbmPrev = SelectBitmap(hdcMem, hbmMono);
   if (hbmPrev) {
    RECT rc = { 0, 0, cxCheck, cyCheck };
    DrawFrameControl(hdcMem, &rc, DFC_MENU, DFCS_MENUCHECK);
    COLORREF clrTextPrev = SetTextColor(pps->hdc,
                                        GetSysColor(COLOR_MENUTEXT));
    COLORREF clrBkPrev = SetBkColor(pps->hdc,
                                    GetSysColor(COLOR_MENU));
    BitBlt(pps->hdc, 0, 0, cxCheck, cyCheck,
           hdcMem, 0, 0, SRCCOPY);
    SetBkColor(pps->hdc, clrBkPrev);
    SetTextColor(pps->hdc, clrTextPrev);
    SelectBitmap(hdcMem, hbmPrev);
   }
   DeleteObject(hbmMono);
  }
  DeleteDC(hdcMem);
 }
}
```

The key steps here are (1) drawing into a temporary monochrome bitmap to generate the mask, (2) setting the text and background colors of the destination DC, (3) using `BitBlt` to do the color mapping. The rest of the function is just boring bookkeeping.

Observe that the checkmark's colors now match the system menu colors because we set them as the text and background colors for the mono-to-color blit.

Armed with this knowledge, perhaps you can help this person, who is trying to draw the menu check marks transparently. I can think of two different solutions off the top of my head.

Raymond Chen

Follow