The evolution of menu templates: 32-bit classic menus

devblogs.microsoft.com/oldnewthing/20080711-00

July 11, 2008

Raymond Chen

Now that we've got a handle on 16-bit classic menu templates, we can move on to the next evolutionary step, namely 32-bit classic menu templates.

The 32-bit classic menu template is in fact nearly identical to the 16-bit classic menu template. The only change is that the menu text is now a Unicode string instead of an ANSI string. Consequently, the discussion below will be rather brief when there is nothing new being introduced.

The 32-bit classic menu template begins with the same header, and the fields have the same meaning.

```
struct MENUHEADER32 {
 WORD wVersion;
 WORD cbHeaderSize;
 BYTE rgbExtra[cbHeaderSize];
};
```

Actually, there's a bonus wrinkle: Whereas the `cbHeaderSize` is always zero in practice for 16-bit menu templates, the 32-bit `cbHeaderSize` *must be* zero if you intend your menu template to be used on the Windows 95 series of Windows operating systems.

Why must the value be zero on Windows 95? I just discovered this now doing the research for this series of articles. It's a bug in the code that converts between 32-bit and 16-bit resources! When converting from a 32-bit menu template to a 16-bit menu template, the conversion code dutifully copies the `cbHeaderSize` and uses it to skip ahead in the 32-bit menu template, but it neglects to skip ahead the same amount in the 16-bit menu template. Fortunately, nobody ever sets this value to anything other than zero, so the bug never manifests itself in practice. (I suspect I'm the first person ever to notice this bug. First, because nobody generates menu templates at runtime; everybody uses the resource compiler or some other tool, and those tools all set the field to zero. And second, because those extra bytes aren't used for anything, so there's no reason for the count to be nonzero.)

Even if you don't care about Windows 95, the `cbHeaderSize` must still be an even number so that the menu item templates are suitably aligned.

The rest of the 32-bit classic menu template is just stuff you've seen before. We have a packed array of menu item templates, either a `POPUPMENUITEM32` if the menu item is a pop-up submenu, or a `NORMALMENUITEM32` if not:

```
struct NORMALMENUITEM32 {
 WORD wFlags;       // menu item flags (MFT_*, MFS_*)
 WORD wID;          // menu item ID
 WCHAR szText[];    // null terminated Unicode string
};


struct POPUPMENUITEM32 {
 WORD wFlags;       // menu item flags (MFT_*, MFS_*)
 WCHAR szText[];    // null terminated UNICODE string
};
```

Aside from changing `CHAR` to `WCHAR`, everything is exactly the same. Let's use that same resource script we used to illustrate the 16-bit classic menu template and convert it to a 32-bit classic menu template instead.

```
1 MENU
BEGIN
  POPUP “&File”
  BEGIN
    MENUITEM “&Open\tCtrl+O”, 100
    MENUITEM SEPARATOR
    MENUITEM “&Exit\tAlt+X”,  101
  END
  POPUP “&View”
  BEGIN
    MENUITEM “&Status Bar”, 102, CHECKED
  END
END
```

Compiling this as a 32-bit classic menu template would result in something like this:

```
// MENUHEADER32
0000  00 00      // wVersion = 0
0002  00 00      // cbHeaderSize = 0
```

After the header come the top-level menu items:

```
// POPUPMENUITEM32 for top-level menu
0004  10 00     // wFlags = MF_POPUP
                // no wID
0006  26 00 46 00 69 00 6C 00 65 00 00 00
                // “&File” + null terminator
```

Since we have a pop-up submenu, the contents of the submenu come next, and we put the top-level menu items on hold.

```
// NORMALMENUITEM32 for nested menu
0012  00 00     // wFlags = MFT_STRING
0014  64 00     // wID = 100
0016  26 00 4F 00 70 00 65 00 6E 00 09 00
      43 00 74 00 72 00 6C 00 2B 00 4F 00 00 00
                // “&Open\tCtrl+O” + null terminator
```

For the separator, we can either do it the formally correct way:

```
// NORMALMENUITEM32 for nested menu – separator
0030  00 08     // wFlags = MFT_SEPARATOR
0032  00 00     // wID = 0
0034  00 00     // “”
```

or we can use the alternate compatibility form:

```
0030  00 00     // wFlags = 0
0032  00 00     // wID = 0
0034  00 00     // “”
```

The last item on the submenu has the `MF_END` flag.

```
// NORMALMENUITEM32 for nested menu – last item
0036  80 00    // wFlags = MFT_STRING | MF_END
0038  65 00    // wID = 101
003A  26 00 45 00 78 00 69 00 74 00 09 00
      41 00 6C 00 74 00 2B 00 58 00 00 00
               // “&Exit\tAlt+X” + null terminator
```

We now pop back to the top-level menu, whos second item and final item is another pop-up submenu.

```
// POPUPMENUITEM for top-level menu
0052  90 00     // wFlags = MF_POPUP | MF_END
                // no wID
0054  26 00 56 00 69 00 65 00 77 00 00 00
                // “&View” + null terminator
```

And finally, the contents of that last pop-up submenu.

```
0060  88 00    // wFlags = MFT_STRING | MFS_CHECKED | MF_END
0062  65 00    // wID = 102
0064  26 00 53 00 74 00 61 00 74 00 75 00
      73 00 20 00 42 00 61 00 72 00 00 00
                // “&Status Bar” + null terminator
```

And that’s all there is to it. A pretty straightforward extension of the 16-bit classic menu template to a 32-bit Unicode version.

After a short break, we’ll look at the extended menu templates.

Raymond Chen

Follow