The evolution of menu templates: 16-bit classic menus

devblogs.microsoft.com/oldnewthing/20080709-00

July 9, 2008

Raymond Chen

Menus aren't as complicated as dialogs. There are no fonts, no positioning, it's just a list of menu items and flags. Well, okay, there's the recursive part, when a menu has a submenu. But that's really the only wrinkle. Most of it is pretty boring.

The 16-bit classic menu template begins with the following header:

```
struct MENUHEADER16 {
 WORD wVersion;
 WORD cbHeaderSize;
 BYTE rgbExtra[cbHeaderSize];
};
```

The version is zero for 16-bit classic menu templates, and the `cbHeaderSize` is the number of extra bytes in the menu header that have to be skipped over to find the first actual menu item. In practice, `cbHeaderSize` is always zero. This header exists only on the top level menu; recursive submenus do not have a `MENUHEADER16`.

After the header (and any extra bytes specified by `cbHeaderSize`) comes a packed array of menu item templates. There are two types of menu item templates, normal items and pop-up submenus. First, let's look at the normal items:

```
struct NORMALMENUITEM16 {
 WORD wFlags;       // menu item flags (MFT_*, MFS_*)
 WORD wID;          // menu item ID
 CHAR szText[];     // null terminated ANSI string
};
```

Normal items represent menu items that are not pop-up submenus, and they take a pretty straightforward form. All you get are flags, the item ID, and the menu item text. The flags are values such as `MFT_STRING`, `MFT_MENUBARBREAK`, and `MFS_DISABLED`. Of course, the `MF_POPUP` flag is not allowed, since this is a normal item template. The flag `MFS_HILITE` is also not allowed, for reasons we will see later.

The other type of menu item template is the pop-up submenu.

```
struct POPUPMENUITEM16 {
 WORD wFlags;       // menu item flags (MFT_*, MFS_*)
 CHAR szText[];     // null terminated ANSI string
};
```

The pop-up item template doesn't have an ID, the `MF_POPUP` flag must be set in the flags (naturally), the `MFS_HILITE` flag must not be set, and it is immediately followed by… another menu resource, minus the resource header, which describes the pop-up submenu itself. (This is the recursive part.)

The end of the list of menu item templates is reached when an item with the `MF_END` flag is set in its flags. And now you see why `MFS_HILITE` is disallowed:

```
#define MF_END              0x00000080L
#define MF_HILITE           0x00000080L
#define MFS_HILITE          MF_HILITE
```

If you set the `MF_HILITE` flag, it would be mistaken for the end of the menu template. Fortunately, there's no need to set the `MFS_HILITE` flag in the menu item template since highlighting happens at runtime based on the user's mouse and keyboard activity, not at menu creation time.

To make all this discussion concrete, let's convert this rather uninteresting menu resource into a menu template:

```
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
```

The menu template for this classic 16-bit menu would go something like this: We start with the header, which always looks the same.

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

Next comes the list of menu items. Our first is a pop-up submenu, so the `MF_POPUP` flag is set, indicating that we have a `POPUPMENUITEM16`:

```
0004  10 00     // wFlags = MF_POPUP
                // no wID
0006  26 46 69 6C 65 00 // "&File" + null terminator
```

Since this is a pop-up menu, the contents of the pop-up menu follow. This is the recursive part of the menu template format: we have a menu template inside the outer one. The first item of the pop-up menu is a string and therefore takes the form of a `NORMALMENUITEM16`:

```
000C  00 00     // wFlags = MFT_STRING
000E  64 00     // wID = 100
0010  26 4F 70 65 6E 09 43 74 72 6C 2B 4F 00
                // "&Open\tCtrl+O" + null terminator
```

The next item of the pop-up menu is a separator. If you have been following the rules strictly, you would generate the separator like this:

```
001D  00 08     // wFlags = MFT_SEPARATOR
001F  00 00     // wID = 0
0021  00        // ""
```

However, it turns out that there is an alternate form for separators, namely to pass all zeroes:

```
001D  00 00     // wFlags = 0
001F  00 00     // wID = 0
0021  00        // ""
```

The existence of this alternate form is actually an artifact of history, which we'll look at next time. But for now, just realize that you can express a separator in two different ways, either the official way with `MFT_SEPARATOR` or the alternate way with `wFlags = 0`. Either works just fine.

Anyway, let's finish up that submenu with the final item, which is a string. We set the `MF_END` flag to indicate that this is the end of the (nested) menu.

```
0022  80 00    // wFlags = MFT_STRING | MF_END
0024  65 00    // wID = 101
0026  26 45 78 69 74 09 41 6C 74 2B 58 00
               // "&Exit\tAlt+X" + null terminator
```

With the completion of the nested menu, we pop back to the top-level menu. Next comes the "View" submenu.

```
0032  90 00     // wFlags = MF_POPUP | MF_END
                // no wID
0034  26 56 69 65 77 00 // "&View" + null terminator
```

The `MF_POPUP` flag marks this as a `POPUPMENUITEM16`, which means that there is no `wID`. And look, the `MF_END` flag is set, which means that this is the last item on the top-level menu. But we're not finished yet, since we still have to read the nested submenu. (Notice that

the “end of menu” marker is far away from the actual end of the menu!)

```
003A  88 00    // wFlags = MFT_STRING | MFS_CHECKED | MF_END
003C  66 00    // wID = 102
003E  26 53 74 61 74 75 73 20 42 61 72 00
                // “&Status Bar” + null terminator
```

The submenu consists of a single item, so its first item is also its last (`MF_END`). Now that the submenu is complete, we pop back to the main menu again, but as we saw, the main menu is also complete, so that concludes the entire menu template.

Next time, we’ll look at that strange alternate form for separator items before returning to the history of menu templates.

Raymond Chen

Follow