The evolution of menu templates: 16-bit extended menus

devblogs.microsoft.com/oldnewthing/20080715-00

July 15, 2008

Raymond Chen

Windows 95 introduced a new menu format, known as “extended menus”. You declare these in a resource file with the `MENUEX` keyword. The 16-bit extended menu is really just a temporary stopping point on the way to the 32-bit extended menu, since the 16-bit form is supported only by the Windows 95 family of operating systems. It’s sort of the missing link of menu templates.

Things start off the same as the 16-bit classic menu, with a structure I’ve been calling `MENUHEADER16`:

```
struct MENUHEADER16 {
 WORD wVersion;
 WORD cbHeaderSize;
 BYTE rgbExtra[cbHeaderSize-4];
};
```

The version number for extended menus is one instead of zero, and the `cbHeaderSize` now includes the size of the `wVersion` and `cbHeaderSize` fields in the header size count; therefore, the number of interstitial bytes four less than the value specified by the `cbHeaderSize` member.

Due to a bug in Windows 95 (and its descendants), the `cbHeaderSize` is ignored, and its value is assumed to be four. Fortunately, every version of the 16-bit resource compiler that supports 16-bit extended menu templates sets the `cbHeaderSize` to four. Consequently, nothing goes wrong in practice. And I suspect nobody has noticed this bug in the over fifteen years (not twenty-five as I had originally written) the code has been in existence.

Unlike the classic menu, there is a prefix structure that comes before the list of menu items.

```
struct MENUPREFIX16 {
 DWORD dwContextHelpID;
};
```

New to extended menus is the addition of context help IDs. These values can be set and retrieved programmatically with the `GetMenuContextHelpId` and `SetMenuContextHelpId` functions.

The template then continues with a packed array of structures I will call `MENUITEMEX16`:

```
struct MENUITEMEX16 {
 DWORD dwType;
 DWORD dwState;
 WORD  wID;
 BYTE  bFlags;
 CHAR  szText[]; // null terminated ANSI string
};
```

Whereas the members of the classic `MENUITEM16` were designed to be passed to the function `InsertMenu`, the members of the extended `MENUITEMEX16` were designed to be passed to the function `InsertMenuItem`. The `dwType`, `dwState`, and `wID` members correspond to the `fType`, `fState`, and `wID` members of the 16-bit `MENUITEMINFO` structure. Similarly, the `szText` goes into the `dwItemData` if the item requires a string. (If the item doesn't require a string, then the `szText` should be an empty string; i.e., should consist solely of the null terminator.)

Notice that a new feature of extended menus is that pop-up menus can have IDs as well as normal menu items.

The `bFlags` describes other information about the menu item, information that in the classic menu was hidden in spare bits in the `wFlags`. But here, the `bFlags` is where this information is kept. The following flags are currently defined:

`0x01`	This item is a pop-up submenu
`0x80`	This item is the last item in the menu

If indeed the bottom bit is set, then after the `MENUITEMEX16` comes a description of the submenu, recursively. (Note that the submenu does not have a `MENUHEADER16`.)

As before, we'll illustrate this format with an example.

```
1 MENUEX 1000
BEGIN
  POPUP "&File", 200,,, 1001
  BEGIN
    MENUITEM "&Open\tCtrl+O", 100
    MENUITEM "", -1, MFT_SEPARATOR
    MENUITEM "&Exit\tAlt+X",  101
  END
  POPUP "&View", 201,,, 1002
  BEGIN
    MENUITEM "&Status Bar", 102,, MFS_CHECKED
  END
END
```

The resulting 16-bit extended menu template begins with the header:

```
0000  01 00          // wVersion = 1
0002  04 00          // cbHeaderSize = 4
```

Since this is the start of a menu, we get a context help ID:

```
0004  E8 03 00 00    // dwContextHelpID = 1000
```

After the context help ID come the menu items. Our first is a pop-up submenu, so the `bFlags` indicates that a submenu is coming:

```
0008  00 00 00 00    // dwType = MFT_STRING
000C  00 00 00 00    // dwState = 0
0010  C8 00          // wID = 200
0012  01             // bFlags = “pop-up submenu”
0013  26 46 69 6C 65 00 // “&File” + null terminator
```

Since we have a pop-up submenu, we recursively include a template for that submenu directly after the menu item template. Consequently, we begin with the context help ID:

```
0019  E9 03 00 00    // dwContextHelpID = 1001
```

And then the contents of the submenu:

```
001D  00 00 00 00    // dwType = MFT_STRING
0021  00 00 00 00    // dwState = 0
0025  64 00          // wID = 100
0027  00             // bFlags = 0
0028  26 4F 70 65 6E 09 43 74 72 6C 2B 4F 00
                     // “&Open\tCtrl+O” + null terminator


0035  00 08 00 00     // dwType = MFT_SEPARATOR
0039  00 00 00 00     // dwState = 0
003D  FF FF           // wID = -1
003F  00              // bFlags = 0
0040  00              // “”


0041  00 00 00 00     // dwType = MFT_STRING
0045  00 00 00 00     // dwState = 0
0049  65 00           // wID = 101
004B  80              // bFlags = “this is the last menu item”
004C  26 45 78 69 74 09 41 6C 74 2B 58 00
                     // “&Exit\tAlt+X” + null terminator
```

When we reach the end of the pop-up submenu, we pop up a level. Therefore, the next entries describe more top-level menu items.

```
0058  00 00 00 00     // dwType = MFT_STRING
005C  00 00 00 00     // dwState = 0
0060  C9 00           // wID = 201
0062  81              // bFlags = “pop-up submenu” |
                      //          “this is the last menu item”
0063  26 56 69 65 77 00 // “&View” + null terminator
```

Ah, no sooner do we pop up than we push back down with another submenu. And the “last menu item” flag is set, which means that once the submenu is finished, we are done with the extended menu template.

```
0069  EA 03 00 00    // dwContextHelpID = 1002


006D  00 00 00 00    // dwType = MFT_STRING
0071  08 00 00 00    // dwState = MFS_CHECKED
0075  66 00          // wID = 102
0077  80              // bFlags = “this is the last menu item”
0078  26 53 74 61 74 75 73 20 42 61 72 00
                      // “&Status Bar” + null terminator
```

After the context help ID, we have the sole menu item for this pop-up submenu, so the first item is also the last item.

Next time, we’ll wrap up by looking at the final menu template format, the 32-bit extended menu. I bet you all can’t wait.

Raymond Chen

Follow