The evolution of menu templates: 32-bit extended menus

devblogs.microsoft.com/oldnewthing/20080716-00

July 16, 2008

Raymond Chen

At last we reach the 32-bit extended menu template. Introduced in Windows 95, this remains the most advanced menu template format through Windows Vista. As you might expect, the 32-bit extended menu template is just a 32-bit version of the 16-bit extended menu template, so if you've been following along, you should find no real surprises here; all the pieces have been telegraphed far in advance.

The header remains the same:

```
struct MENUHEADER32 {
 WORD wVersion;
 WORD cbHeaderSize;
 BYTE rgbExtra[cbHeaderSize-4];
};
```

The differences here from the 32-bit classic menu template header are analogous to the changes between the 16-bit classic menu template and the 16-bit extended menu template. The `wVersion` is set to one for extended templates, and the `cbHeaderSize` includes the `wVersion` and `cbHeaderSize` fields themselves, so the number of extra bytes is four less than the value specified in `cbHeaderSize`. There is one additional constraint: The `cbHeaderSize` must be a multiple of four because extended menu item templates must be aligned on `DWORD` boundaries. But as with 32-bit classic templates, the `cbHeaderSize` must be four in order to avoid a bug in the Windows 95 family of operating systems.

After the header comes the menu itself, and like the 16-bit extended menu template, there is a prefix structure that comes before the items and which serves the same purpose as in the 16-bit extended menu template:

```
struct MENUPREFIX32 {
 DWORD dwContextHelpID;
};
```

The list of menu items is basically the same as the 16-bit version, just with some expanded fields.

```
struct MENUITEMEX32 {
 DWORD dwType;
 DWORD dwState;
 DWORD dwID;
 WORD  wFlags;
 WCHAR szText[]; // null terminated UNICODE string
};
```

As we saw before when we studied the 16-bit extended menu template, the big difference between classic and extended menu items is that classic menu items were designed for the `InsertMenu` function, whereas extended menu items were designed for the `InsertMenuItem` function. The `dwType`, `dwState`, and `dwID` members correspond to the `fType`, `fState`, and `wID` members of the `MENUITEMINFO` structure, and the the `szText` goes into the `dwItemData` if the item requires a string.

One additional quirk of 32-bit extended menu item templates which the 16-bit version does not have is that 32-bit extended menu item templates must begin on a 32-bit boundary; therefore, you must insert a `WORD` of padding after the menu text if the text is an odd number of characters long. (Fourteen bytes of the fixed-length part of the `MENUITEMEX32` plus an odd number of `WCHAR`s plus the null terminator `WCHAR` leaves a value that is 2 mod 4; therefore, you need an additional `WORD` to return to a `DWORD` boundary.)

The `wFlags` field has the same values as in the 16-bit extended menu item templates; the high byte is always zero. And, as before, if the bottom bit is set, then the menu item describes a pop-up submenu, which is inserted directly after the extended menu item template.

That's all there is to it. Let's see how our example menu resource looks when converted to a 32-bit extended menu template:

```
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
```

First comes the header, whose contents are fixed:

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

Before the list of extended menu item templates, we have the context help ID:

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

Since our first menu item is a pop-up submenu, the `wFlags` will have the bottom bit set:

```
0008  00 00 00 00    // dwType = MFT_STRING
000C  00 00 00 00    // dwState = 0
0010  C8 00 00 00    // wID = 200
0014  01 00          // wFlags = “pop-up submenu”
0016  26 00 46 00 69 00 6C 00 65 00 00 00
                     // “&File” + null terminator
0022  00 00          // Padding to restore alignment
```

Notice the two bytes of padding so that we return to `DWORD` alignment.

The `wFlags` promised a pop-up submenu, so here it is.

```
0024  E9 03 00 00    // dwContextHelpID = 1001


// First item
0028  00 00 00 00    // dwType = MFT_STRING
002C  00 00 00 00    // dwState = 0
0030  64 00 00 00    // dwID = 100
0034  00 00          // wFlags = 0
0036  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


// Second item
0050  00 08 00 00     // dwType = MFT_SEPARATOR
0054  00 00 00 00     // dwState = 0
0058  FF FF FF FF     // dwID = -1
005C  00 00           // wFlags = 0
005E  00 00           // “”


// Third (final) item
0060  00 00 00 00     // dwType = MFT_STRING
0064  00 00 00 00     // dwState = 0
0068  65 00 00 00     // dwID = 101
006C  80 00           // wFlags = “this is the last menu item”
0070  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
0086  00 00          // Padding to restore alignment
```

When we see the “end” marker, we pop one level back to the main menu.

```
0088  00 00 00 00     // dwType = MFT_STRING
008C  00 00 00 00     // dwState = 0
0090  C9 00 00 00     // dwID = 201
0094  81 00           // wFlags = “pop-up submenu” |
                      //          “this is the last menu item”
0096  26 00 56 00 69 00 65 00 77 00 00 00
                      // “&View” + null terminator
00A2  00 00          // Padding to restore alignment
```

The set bottom bit in the `wFlags` indicates that another pop-up submenu is coming, and the “end” marker means that once the submenu is finished, we are done.

```
00A4  EA 03 00 00    // dwContextHelpID = 1002


00A8  00 00 00 00    // dwType = MFT_STRING
00AC  08 00 00 00    // dwState = MFS_CHECKED
00B0  66 00 00 00    // dwID = 102
00B4  80 00          // wFlags = “this is the last menu item”
00B6  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
00CE  00 00          // Padding to restore alignment
```

Since the pop-up submenu has only one item, the first item is also the last.

That’s it for the evolution of menu templates, starting from a series of calls to the ANSI version of `InsertMenu` to a series of calls to the Unicode version of `InsertMenuItem`. Menu templates get much less attention than dialog templates, but if you wanted to know how they work, well, there you have it.

Raymond Chen

Follow