The evolution of dialog templates – 16-bit Classic Templates

devblogs.microsoft.com/oldnewthing/20040618-00

June 18, 2004

Raymond Chen

In the history of Windows, there have been four versions of dialog templates. And despite the changes, you'll see that they're basically all the same.

First, there was the classic Windows 1.0 dialog template. It starts like this:

```
DWORD dwStyle; // dialog style
BYTE  cItems;  // number of controls in this dialog
WORD  x;       // x-coordinate
WORD  y;       // y-coordinate
WORD  cx;      // width
WORD  cy;      // height
```

Notice that this is where the 255-controls-per-dialog limit comes from on 16-bit Windows, since the field that records the number of controls on the dialog is only a byte.

After this header come a series of strings. All strings in the 16-bit dialog template permit a null-terminated ANSI string. For example, if you wanted to store the string "Hello", you would write out the six bytes

```
48 65 6C 6C 6F 00  ; "Hello"
```

(As a special case of this: If you write out a single 00 byte, then that represents a null string. Handy when you don't actually want to store a string but the dialog format requires you to store one.)

Sometimes you are allowed to specify a 16-bit ordinal value instead of a string. In that case, you write out the byte 0xFF followed by the ordinal. For example, if you wanted to specify the ordinal 42, you would write out the three bytes

```
FF 2A 00           ; FF followed by WORD (little-endian)
```

Okay, back to the dialog template. After the header, there are three strings:

- The menu name, which can be a string or an ordinal. This is typically null, indicating that you don't want a menu. If non-null, then the menu will be loaded via LoadMenu using the specified string or resource from the instance handle passed to the dialog creation function via the HINSTANCE parameter.
- The class, which must be a string (no ordinals allowed). This is typically also null, indicating that you want the default dialog class. We have seen earlier how you can override the default dialog class to get special behavior. If non-null, the class will be also be looked up relative to the instance handle passed to the dialog creation function via the HINSTANCE parameter.
- The dialog title, which must be a string (no ordinals allowed).

If the DS_SETFONT style is set, then what follows next is a WORD indicating the point size and a string specifying the font name. Otherwise, there is no font information.

That's the end of the header section. Next come a series of dialog item templates, one for each control.

Each item template begins the same way:

```
WORD  x;       // x-coordinate (DLUs)
WORD  y;       // y-coordinate (DLUs)
WORD  cx;      // width (DLUs)
WORD  cy;      // height (DLUs)
WORD  wID;     // control ID
DWORD dwStyle; // window style
```

Recall that the dialog coordinates are recorded in dialog units (DLUs). Four x-DLUs and eight y-DLUs equals one "average" character.

After the fixed start of the item template comes the class name, either as a null-terminated ANSI string or (and this is particularly weird) as single byte in the range 0x80 through 0xFF which encodes one of the "standard" window classes:

- 0x80 = "button"
- 0x81 = "edit"
- 0x82 = "static"
- 0x83 = "listbox"
- 0x84 = "scrollbar"
- 0x85 = "combobox"

(Note that this encoding means that the first character of a window class name cannot be an extended character if you want to use it in a dialog template!)

After the class name comes the control text, either as a null-terminated string or as an ordinal. If you use an ordinal, then the lpszName member of the CREATESTRUCT is a pointer to the three-byte ordinal sequence (0xFF followed by the ordinal); otherwise it's a pointer to the string. The only control I know of that knows what to do with the ordinal is the static control if you put it into one of the image modes (SS_ICON or SS_BITMAP), in which case the ordinal is a resource identifier for the image that the static displays.

After the control text comes up to 256 bytes of "extra data" in the form of a byte count, followed by the actual data. If there is no "extra data", then use a byte count of zero.

When the dialog manager creates a control, it passes a pointer to the "extra" data as the final LPVOID parameter to the CreateWindowEx function. (As far as I can tell, there is no way to tell the resource compiler to insert this extra data. It's one of those lurking features that nobody has taken advantage of yet.)

Okay, that's all great and theoretical. But sometimes you just need to see it in front of you to understand it. So let's take apart an actual 16-bit dialog resource. I took this one from COMMCTRL.DLL; it's the search/replace dialog.

```
0000  C0 00 C8 80 0B 24 00 2C-00 E6 00 5E 00 00 00 52  .....$.,...^...R
0010  65 70 6C 61 63 65 00 08-00 48 65 6C 76 00 04 00  eplace...Helv...
0020  09 00 30 00 08 00 FF FF-00 00 00 50 82 46 69 26  ..0........P.Fi&
0030  6E 64 20 57 68 61 74 3A-00 00 36 00 07 00 72 00  nd What:..6...r.
0040  0C 00 80 04 80 00 83 50-81 00 00 04 00 1A 00 30  .......P.......0
0050  00 08 00 FF FF 00 00 00-50 82 52 65 26 70 6C 61  ........P.Re&pla
0060  63 65 20 57 69 74 68 3A-00 00 36 00 18 00 72 00  ce With:..6...r.
0070  0C 00 81 04 80 00 83 50-81 00 00 05 00 2E 00 68  .......P.......h
0080  00 0C 00 10 04 03 00 03-50 80 4D 61 74 63 68 20  ........P.Match 
0090  26 57 68 6F 6C 65 20 57-6F 72 64 20 4F 6E 6C 79  &Whole Word Only
00A0  00 00 05 00 3E 00 3B 00-0C 00 11 04 03 00 01 50  ....>.;........P
00B0  80 4D 61 74 63 68 20 26-43 61 73 65 00 00 AE 00  .Match &Case....
00C0  04 00 32 00 0E 00 01 00-01 00 03 50 80 26 46 69  ..2........P.&Fi
00D0  6E 64 20 4E 65 78 74 00-00 AE 00 15 00 32 00 0E  nd Next......2..
00E0  00 00 04 00 00 03 50 80-26 52 65 70 6C 61 63 65  ......P.&Replace
00F0  00 00 AE 00 26 00 32 00-0E 00 01 04 00 00 03 50  ....&.2........P
0100  80 52 65 70 6C 61 63 65-20 26 41 6C 6C 00 00 AE  .Replace &All...
0110  00 37 00 32 00 0E 00 02-00 00 00 03 50 80 43 61  .7.2........P.Ca
0120  6E 63 65 6C 00 00 AE 00-4B 00 32 00 0E 00 0E 04  ncel....K.2.....
0130  00 00 03 50 80 26 48 65-6C 70 00 00              ...P.&Help..
```

Let's start with the header.

```
0000  C0 00 C8 80  // dwStyle
0004  0B           // cItems
0005  24 00 2C 00  // x, y
0009  E6 00 5E 00  // cx, cy
```

In other words, the header says

dwStyle	= 0x80C800C0	= WS_POPUP | WS_CAPTION | WS_SYSMENU | DS_SETFONT | DS_MODALFRAME
cItems	= 0x0B	= 11
x	= 0x0024	= 36
y	= 0x002C	= 44
cx	= 0x00E6	= 230
cy	= 0x005E	= 94

After the header come the menu name, class name, and dialog title:

```
000D  00            // no menu
000E  00            // default dialog class
000F  52 65 70 6C 61 63 65 00 // "Replace"
```

Now, since the DS_SETFONT bit is set in the style, the next section describes the font to be used by the dialog:

```
0017  08 00         // wSize = 8
0019  48 65 6C 76 00 // "Helv"
```

Aha, this dialog box uses 8pt Helv.

Next come the eleven dialog item templates.

```
001E  04 00 09 00   // x, y
0022  30 00 08 00   // cx, cy
0026  FF FF         // wID
0028  00 00 00 50   // dwStyle
```

So this dialog item template says

x	= 0x0004	= 4
y	= 0x0009	= 9
cx	= 0x0030	= 48
cy	= 0x0008	= 8
wID	= 0xFFFF	= -1
dwStyle	= 0x50000000	= WS_CHILD | WS_VISIBLE | SS_LEFT

How did I know that the style value 0x0000 should be interpreted as SS_LEFT and not, say, BS_PUSHBUTTON? Because the window class tells me that what I have is a static control.

```
002C  82            // "static"
```

After the class name comes the control text.

```
002D  46 69 26 6E 64 20 57 68 61 74 3A 00 // "Fi&nd What:"
```

And finally (for this dialog item template), we specify that we have no extra data:

```
0039  00            // no extra data
```

Now we repeat the above exercise for the other ten controls. I'll just summarize here:

```
// Second control
003A  36 00 07 00   // x, y
003E  72 00 0C 00   // cx, cy
0042  80 04         // wID
0044  80 00 83 50   // dwStyle
0048  81            // "edit"
0049  00            // ""
004A  00            // no extra data
// Third control
004B  04 00 1A 00   // x, y
004F  30 00 08 00   // cx, cy
0053  FF FF         // wID
0055  00 00 00 50   // dwStyle
0059  82            // "static"
005A  52 65 26 70 6C 61 63 65 20 57 69 74 68 3A 00
                    // "Re&place With:"
0069  00            // no extra data
// Fourth control
006A  36 00 18 00   // x, y
006E  72 00 0C 00   // cx, cy
0072  81 04         // wID
0074  80 00 83 50   // dwStyle
0078  81            // "edit"
0079  00            // ""
007A  00            // no extra data
// Fifth control
007B  05 00 2E 00   // x, y
007F  68 00 0C 00   // cx, cy
0083  10 04         // wID
0085  03 00 03 50   // dwStyle
0089  80            // "button"
008A  4D 61 74 63 68 20 26 57 68 6F 6C 65 20 57
      6F 72 64 20 4F 6E 6C 79 00
                    // "Match &Whole Word Only"
00A1  00            // no extra data
// Sixth control
00A2  05 00 3E 00   // x, y
00A6  3B 00 0C 00   // cx, cy
00AA  11 04         // wID
00AC  03 00 01 50   // dwStyle
00B0  80            // "button"
00B1  4D 61 74 63 68 20 26 43 61 73 65 00
                    // "Match &Case"
00BD  00            // no extra data
// Seventh control
00BE  AE 00 04 00   // x, y
00C2  32 00 0E 00   // cx, cy
00C6  01 00         // wID
00C8  01 00 03 50   // dwStyle
00CC  80            // "button"
00CD  26 46 69 6E 64 20 4E 65 78 74 00
                    // "&Find Next"
```

```
00D8  00            // no extra data
// Eighth control
00D9  AE 00 15 00   // x, y
00DD  32 00 0E 00   // cx, cy
00E1  00 04         // wID
00E3  00 00 03 50   // dwStyle
00E7  80            // "button"
00E8  26 52 65 70 6C 61 63 65 00
                    // "&Replace"
00F1  00            // no extra data
// Ninth control
00F2  AE 00 26 00   // x, y
00F6  32 00 0E 00   // cx, cy
00FA  01 04         // wID
00FC  00 00 03 50   // dwStyle
0100  80            // "button"
0101  52 65 70 6C 61 63 65 20 26 41 6C 6C 00
                    // "Replace &All"
010E  00            // no extra data
// Tenth control
010F  AE 00 37 00   // x, y
0113  32 00 0E 00   // cx, cy
0117  02 00         // wID
0119  00 00 03 50   // dwStyle
011D  80            // "button"
011E  43 61 6E 63 65 6C 00
                    // "Cancel"
0125  00            // no extra data
// Eleventh control
0126  AE 00 4B 00   // x, y
012A  32 00 0E 00   // cx, cy
012E  0E 04         // wID
0130  00 00 03 50   // dwStyle
0134  80            // "button"
0135  26 48 65 6C 70 00
                    // "&Help"
013B  00            // no extra data
```

And that’s the dialog template. We can now reconstruct the resource compiler source code from this template:

```
DIALOG 36, 44, 230, 94
STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU | DS_MODALFRAME | NOT WS_VISIBLE
CAPTION "Replace"
FONT 8, "Helv"
BEGIN
    CONTROL "Fi&nd What:", -1, "static", SS_LEFT,
            4, 9, 48, 8
    CONTROL "", 0x0480, "edit",
            WS_BORDER | WS_GROUP | WS_TABSTOP | ES_AUTOHSCROLL,
            54, 7, 114, 12
    CONTROL "Re&place With:", -1, "static", SS_LEFT,
            4, 26, 48, 8
    CONTROL "", 0x0481, "edit",
            WS_BORDER | WS_GROUP | WS_TABSTOP | ES_AUTOHSCROLL,
            54, 24, 114, 12
    CONTROL "Match &Whole Word Only", 0x0410, "button",
            WS_GROUP | WS_TABSTOP | BS_AUTOCHECKBOX,
            5, 46, 104, 12
    CONTROL "Match &Case", 0x0411, "button",
            WS_TABSTOP | BS_AUTOCHECKBOX,
            5, 62, 59, 12
    CONTROL "&Find Next", IDOK, "button",
            WS_GROUP | WS_TABSTOP | BS_DEFPUSHBUTTON,
            174, 4, 50, 14
    CONTROL "&Replace", 0x0400, "button",
            WS_GROUP | WS_TABSTOP | BS_PUSHBUTTON,
            174, 21, 50, 14
    CONTROL "Replace &All", 0x0401, "button",
            WS_GROUP | WS_TABSTOP | BS_PUSHBUTTON,
            174, 38, 50, 14
    CONTROL "Cancel", IDCANCEL, "button",
            WS_GROUP | WS_TABSTOP | BS_PUSHBUTTON,
            174, 55, 50, 14
    CONTROL "Cancel", 0x040E, "button",
            WS_GROUP | WS_TABSTOP | BS_PUSHBUTTON,
            174, 75, 50, 14
END
```

Notice that we didn’t explicitly say “DS_SETFONT” in the dialog’s STYLE directive since that is implied by the “FONT” directive. And since WS_VISIBLE is on by default, we didn’t have to say it; rather we had to explicitly refute it in the places it wasn’t wanted.

Now if you take a look in your SDK header files, you’ll find dlgs.h and findtext.dlg which pretty much match up with the template above, giving names to the magic values like 0x0400 and positioning the controls in the same place as above. You’ll find some minor differences, though, since the header files in the SDK are for the 32-bit Find/Replace dialog and the one above is the 16-bit Find/Replace dialog, but you’ll see that it still matches up pretty well.

Next time: The 32-bit DIALOG template.

Raymond Chen

Follow