Building a dialog template at run-time

devblogs.microsoft.com/oldnewthing/20050429-00

April 29, 2005

Raymond Chen

We've spent quite a bit of time over the past year learning about dialog templates and the dialog manager. Now we're going to put the pieces together to do something interesting: Building a dialog template on the fly.

What we're going to write is an extremely lame version of the `MessageBox` function. Why bother writing a bad version of something that Windows already does? Because you can use it as a starting point for further enhancements. For example, once you learn how to generate a template dynamically, you can dynamically add buttons beyond the boring "OK" button, or you can add additional controls like a "Repeat this answer for all future occurrences of this dialog" checkbox or maybe insert an animation control.

I'm going to start with a highly inefficient dialog template class. This is not production-quality, but it's good enough for didactic purposes.

```
#include <vector>
class DialogTemplate {
public:
 LPCDLGTEMPLATE Template() { return (LPCDLGTEMPLATE)&v[0]; }
 void AlignToDword()
  { if (v.size() % 4) Write(NULL, 4 - (v.size() % 4)); }
 void Write(LPCVOID pvWrite, DWORD cbWrite) {
  v.insert(v.end(), cbWrite, 0);
  if (pvWrite) CopyMemory(&v[v.size() - cbWrite], pvWrite, cbWrite);
 }
 template<typename T> void Write(T t) { Write(&t, sizeof(T)); }
 void WriteString(LPCWSTR psz)
  { Write(psz, (lstrlenW(psz) + 1) * sizeof(WCHAR)); }
private:
 vector<BYTE> v;
};
```

I didn't spend much time making this class look pretty because it's not the focus of this article. The `DialogTemplate` class babysits a `vector` of bytes to which you can `Write` data. There is also a little `AlignToDword` method that pads the buffer to the next `DWORD` boundary. This'll come in handy, too.

Our message box will need a dialog procedure which ends the dialog when the `IDCANCEL` button is pressed. If we had made any enhancements to the dialog template, we would handle them here as well.

```
INT_PTR CALLBACK DlgProc(HWND hwnd, UINT wm, WPARAM wParam, LPARAM lParam)
{
 switch (wm) {
 case WM_INITDIALOG: return TRUE;
 case WM_COMMAND:
  if (GET_WM_COMMAND_ID(wParam, lParam) == IDCANCEL) EndDialog(hwnd, 0);
  break;
 }
 return FALSE;
}
```

Finally, we build the template. This is not hard, just tedious. Out of sheer laziness, we make the message box a fixed size. If this were for a real program, we would have measured the text (using `ncm.lfCaptionFont` and `ncm.lfMessageFont`) to determine the best size for the message box.

```
BOOL FakeMessageBox(HWND hwnd, LPCWSTR pszMessage, LPCWSTR pszTitle)
{
 BOOL fSuccess = FALSE;
 HDC hdc = GetDC(NULL);
 if (hdc) {
  NONCLIENTMETRICSW ncm = { sizeof(ncm) };
  if (SystemParametersInfoW(SPI_GETNONCLIENTMETRICS, 0, &ncm, 0)) {
   DialogTemplate tmp;
   // Write out the extended dialog template header
   tmp.Write<WORD>(1); // dialog version
   tmp.Write<WORD>(0xFFFF); // extended dialog template
   tmp.Write<DWORD>(0); // help ID
   tmp.Write<DWORD>(0); // extended style
   tmp.Write<DWORD>(WS_CAPTION | WS_SYSMENU | DS_SETFONT | DS_MODALFRAME);
   tmp.Write<WORD>(2); // number of controls
   tmp.Write<WORD>(32); // X
   tmp.Write<WORD>(32); // Y
   tmp.Write<WORD>(200); // width
   tmp.Write<WORD>(80); // height
   tmp.WriteString(L""); // no menu
   tmp.WriteString(L""); // default dialog class
   tmp.WriteString(pszTitle); // title
   // Next comes the font description.
   // See text for discussion of fancy formula.
   if (ncm.lfMessageFont.lfHeight < 0) {
     ncm.lfMessageFont.lfHeight = -MulDiv(ncm.lfMessageFont.lfHeight,
              72, GetDeviceCaps(hdc, LOGPIXELSY));
   }
   tmp.Write<WORD>((WORD)ncm.lfMessageFont.lfHeight); // point
   tmp.Write<WORD>((WORD)ncm.lfMessageFont.lfWeight); // weight
   tmp.Write<BYTE>(ncm.lfMessageFont.lfItalic); // Italic
   tmp.Write<BYTE>(ncm.lfMessageFont.lfCharSet); // CharSet
   tmp.WriteString(ncm.lfMessageFont.lfFaceName);
   // Then come the two controls.  First is the static text.
   tmp.AlignToDword();
   tmp.Write<DWORD>(0); // help id
   tmp.Write<DWORD>(0); // window extended style
   tmp.Write<DWORD>(WS_CHILD | WS_VISIBLE); // style
   tmp.Write<WORD>(7); // x
   tmp.Write<WORD>(7); // y
   tmp.Write<WORD>(200-14); // width
   tmp.Write<WORD>(80-7-14-7); // height
   tmp.Write<DWORD>(-1); // control ID
   tmp.Write<DWORD>(0x0082FFFF); // static
   tmp.WriteString(pszMessage); // text
   tmp.Write<WORD>(0); // no extra data
   // Second control is the OK button.
   tmp.AlignToDword();
   tmp.Write<DWORD>(0); // help id
   tmp.Write<DWORD>(0); // window extended style
   tmp.Write<DWORD>(WS_CHILD | WS_VISIBLE |
                    WS_GROUP | WS_TABSTOP | BS_DEFPUSHBUTTON); // style
```

```
   tmp.Write<WORD>(75); // x
   tmp.Write<WORD>(80-7-14); // y
   tmp.Write<WORD>(50); // width
   tmp.Write<WORD>(14); // height
   tmp.Write<DWORD>(IDCANCEL); // control ID
   tmp.Write<DWORD>(0x0080FFFF); // static
   tmp.WriteString(L"OK"); // text
   tmp.Write<WORD>(0); // no extra data
   // Template is ready - go display it.
   fSuccess = DialogBoxIndirect(g_hinst, tmp.Template(),
                                hwnd, DlgProc) >= 0;
  }
  ReleaseDC(NULL, hdc); // fixed 11 May
 }
 return fSuccess;
}
```

The fancy formula for determining the font point size is not that fancy after all. The dialog manager converts the font height from point to pixels via the standard formula:

> `fontHeight = -MulDiv(pointSize, GetDeviceCaps(hdc, LOGPIXELSY), 72);`

Therefore, to get the original pixel value back, we need to solve this formula for `pointSize` so that when it is sent through the formula again, we get the original value back.

The template itself follows the format we discussed earlier, no surprises.

One subtlety is that the control identifier for our OK button is `IDCANCEL` instead of the `IDOK` you might have expected. That's because this message box has only one button, so we want to let the user hit the ESC key to dismiss it.

Now all that's left to do is take this function for a little spin.

```
void OnChar(HWND hwnd, TCHAR ch, int cRepeat)
{
 if (ch == TEXT(' ')) {
  FakeMessageBox(hwnd,
   L"This is the text of a dynamically-generated dialog template. "
   L"If Raymond had more time, this dialog would have looked prettier.",
   L"Title of message box");
 }
}
    // add to window procedure
    HANDLE_MSG(hwnd, WM_CHAR, OnChar);
```

Fire it up, hit the space bar, and observe the faux message box.

Okay, so it's not very exciting visually, but that wasn't the point. The point is that you now know how to build a dialog template at run-time.

Raymond Chen

Follow