How to display a string without those ugly boxes

devblogs.microsoft.com/oldnewthing/20040716-00

July 16, 2004

Raymond Chen

You've all seen those ugly boxes. When you try to display a string and the font you have doesn't support all of the characters in it, you get an ugly box for the characters that aren't available in the font.

Start with our scratch program and add this to the `PaintContent` function:

```
void
PaintContent(HWND hwnd, PAINTSTRUCT *pps)
{
    TextOutW(pps->hdc, 0, 0,
            L"ABC\x0410\x0411\x0412\x0E01\x0E02\x0E03", 9);
}
```

That string contains the first three letters from three different alphabets: "ABC" from the Roman alphabet; "АБВ" from the Cyrillic alphabet; and "กขฃ" from the Thai alphabet.

If you run this program, you get a bunch of ugly boxes for the non-Roman characters because the SYSTEM font is very limited in its character set support.

But how to pick the right font? What if the string contained Korean or Japanese characters? There is no single font that contains every character defined by Unicode. (Or at least, none that is commonly available.) What do you do?

This is where font linking comes in.

Font linking allows you to take a string and break it into pieces, where each piece can be displayed in an appropriate font.

The IMLangFontLink2 interface provides the methods necessary to do this breaking. GetStrCodePages takes the string apart into chunks, such that all the characters in a chunk can be displayed by the same font, and MapFont creates the font.

Okay, so let's write our font-link-enabled version of the TextOut function. We'll do this in stages, starting with the idea kernel.

```
#include <mlang.h>
HRESULT TextOutFL(HDC hdc, int x, int y, LPCWSTR psz, int cch)
{
  ...
  while (cch > 0) {
    DWORD dwActualCodePages;
    long cchActual;
    pfl->GetStrCodePages(psz, cch, 0, &dwActualCodePages, &cchActual);
    HFONT hfLinked;
    pfl->MapFont(hdc, dwActualCodePages, 0, &hfLinked);
    HFONT hfOrig = SelectFont(hdc, hfLinked);
    TextOut(hdc, ?, ?, psz, cchActual);
    SelectFont(hdc, hfOrig);
    pfl->ReleaseFont(hfLinked);
    psz += cchActual;
    cch -= cchActual;
  }
  ...
}
```

After figuring out which code pages the default font supports, we walk through the string asking GetStrCodePages to give us the next chunk of characters. From that chunk, we create a matching font and draw the characters in that font at "the right place". Repeat until all the characters are done.

The rest is refinement and paperwork.

First of all, what is "the right place"? We want the next chunk to resume where the previous chunk left off. For that, we take advantage of the TA_UPDATECP text alignment style, which says that GDI should draw the text at the current position, and update the current position to the end of the drawn text (therefore, in position for the next chunk).

Therefore, part of the paperwork is to set the DC's current position and set the text mode to TA_UPDATECP:

```
  SetTextAlign(hdc, GetTextAlign(hdc) | TA_UPDATECP);
  MoveToEx(hdc, x, y, NULL);
```

Then we can just pass "0,0" as the coordinates to TextOut, because the coordinates passed to TextOut are ignored if the text alignment mode is TA_UPDATECP; it always draws at the current position.

Of course, we can't just mess with the DC's settings like this. If the caller did not set TA_UPDATECP, then the caller is not expecting us to be meddling with the current position. Therefore, we have to save the original position and restore it (and the original text alignment mode) afterwards.

```
POINT ptOrig;
DWORD dwAlignOrig = GetTextAlign(hdc);
SetTextAlign(hdc, dwAlignOrig | TA_UPDATECP);
MoveToEx(hdc, x, y, &ptOrig);
while (cch > 0) {
  ...
  TextOut(hdc, 0, 0, psz, cchActual);
  ...
}
// if caller did not want CP updated, then restore it
// and restore the text alignment mode too
if (!(dwAlignOrig & TA_UPDATECP)) {
  SetTextAlign(hdc, dwAlignOrig);
  MoveToEx(hdc, ptOrig.x, ptOrig.y, NULL);
}
```

Next is a refinement: We should take advantage of the second parameter to GetStrCodePages, which specifies the code pages we would prefer to use if a choice is avialable. Clearly we should prefer to use the code pages supported by the font we want to use, so that if the character can be displayed in that font directly, then we shouldn't map an alternate font.

```
HFONT hfOrig = (HFONT)GetCurrentObject(hdc, OBJ_FONT);
DWORD dwFontCodePages = 0;
pfl->GetFontCodePages(hdc, hfOrig, &dwFontCodePages);
...
while (cch > 0) {
  pfl->GetStrCodePages(psz, cch, dwFontCodePages, &dwActualCodePages, &cchActual);
  if (dwActualCodePages & dwFontCodePages) {
    // our font can handle it - draw directly using our font
    TextOut(hdc, 0, 0, psz, cchActual);
  } else {
    ... MapFont etc ...
  }
}
...
```

Of course, you probably wonder this magical `pfl` comes from. It comes from the Multilanguage Object in mlang.

```
IMLangFontLink2 *pfl;
CoCreateInstance(CLSID_CMultiLanguage, NULL,
                 CLSCTX_ALL, IID_IMLangFontLink2, (void**)&pfl);
...
pfl->Release();
```

And of course, all the errors we've been ignoring need to be taken care of. This does create a big of a problem if we run into an error after we have already made it through a few chunks. What should we do?

I’m going to handle the error by drawing the string in the original font, ugly boxes and all. We can’t erase the characters we already drew, and we can’t just draw half of the string (for our caller won’t know where to resume). So we just draw with the original font and hope for the best. At least it’s no worse than it was before font linking.

Put all of these refinements together and you get this final function:

```
HRESULT TextOutFL(HDC hdc, int x, int y, LPCWSTR psz, int cch)
{
  HRESULT hr;
  IMLangFontLink2 *pfl;
  if (SUCCEEDED(hr = CoCreateInstance(CLSID_CMultiLanguage, NULL,
                      CLSCTX_ALL, IID_IMLangFontLink2, (void**)&pfl))) {
    HFONT hfOrig = (HFONT)GetCurrentObject(hdc, OBJ_FONT);
    POINT ptOrig;
    DWORD dwAlignOrig = GetTextAlign(hdc);
    if (!(dwAlignOrig & TA_UPDATECP)) {
      SetTextAlign(hdc, dwAlignOrig | TA_UPDATECP);
    }
    MoveToEx(hdc, x, y, &ptOrig);
    DWORD dwFontCodePages = 0;
    hr = pfl->GetFontCodePages(hdc, hfOrig, &dwFontCodePages);
    if (SUCCEEDED(hr)) {
      while (cch > 0) {
        DWORD dwActualCodePages;
        long cchActual;
        hr = pfl->GetStrCodePages(psz, cch, dwFontCodePages, &dwActualCodePages,
&cchActual);
        if (FAILED(hr)) {
          break;
        }
        if (dwActualCodePages & dwFontCodePages) {
          TextOut(hdc, 0, 0, psz, cchActual);
        } else {
          HFONT hfLinked;
          if (FAILED(hr = pfl->MapFont(hdc, dwActualCodePages, 0, &hfLinked))) {
            break;
          }
          SelectFont(hdc, hfLinked);
          TextOut(hdc, 0, 0, psz, cchActual);
          SelectFont(hdc, hfOrig);
          pfl->ReleaseFont(hfLinked);
        }
        psz += cchActual;
        cch -= cchActual;
      }
      if (FAILED(hr)) {
        //  We started outputting characters so we have to finish.
        //  Do the rest without font linking since we have no choice.
        TextOut(hdc, 0, 0, psz, cch);
        hr = S_FALSE;
      }
    }
    pfl->Release();
    if (!(dwAlignOrig & TA_UPDATECP)) {
      SetTextAlign(hdc, dwAlignOrig);
      MoveToEx(hdc, ptOrig.x, ptOrig.y, NULL);
    }
  }
```

```
  return hr;
}
```

Finally, we can wrap the entire operation inside a helper function that first tries with font linking and if that fails, then just draws the text the old-fashioned way.

```
void TextOutTryFL(HDC hdc, int x, int y, LPCWSTR psz, int cch)
{
  if (FAILED(TextOutFL(hdc, x, y, psz, cch)) {
    TextOut(hdc, x, y, psz, cch);
  }
}
```

Okay, now that we have our font-linked TextOut with fallback, we can go ahead and adjust our `PaintContent` function to use it.

```
void
PaintContent(HWND hwnd, PAINTSTRUCT *pps)
{
  TextOutTryFL(pps->hdc, 0, 0,
               TEXT("ABC\x0410\x0411\x0412\x0E01\x0E02\x0E03"), 9);
}
```

Observe that the string is now displayed with no black boxes.

One refinement I did not do was to avoid creating the IMlangFontLink2 pointer each time we want to draw text. In a “real program” you would probably create the multilanguage object once per drawing context (per window, perhaps) and re-use it to avoid going through the whole object creation codepath each time you want to draw a string.

[Raymond is currently on vacation; this message was pre-recorded.]

Raymond Chen

Follow