There’s nothing wrong with making bold treeview items

devblogs.microsoft.com/oldnewthing/20090406-00

April 6, 2009

Raymond Chen

Commenter Frans Bouma asks,

Actually, bold treeview items work just fine. Watch:

Start with our scratch program and make these changes:

```
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
  g_hwndChild = CreateWindow(
      WC_TREEVIEW, NULL, WS_CHILD | WS_VISIBLE | WS_TABSTOP |
      TVS_HASBUTTONS | TVS_HASLINES | TVS_LINESATROOT,
      0, 0, 0, 0, hwnd, (HMENU)1, g_hinst, 0);


  TVINSERTSTRUCT tvis;
  tvis.hParent = TVI_ROOT;
  tvis.hInsertAfter = TVI_LAST;
  tvis.item.mask = TVIF_TEXT | TVIF_STATE;
  tvis.item.stateMask = TVIS_BOLD;
  tvis.item.state = 0;
  tvis.item.pszText = TEXT(“First”);


  tvis.hParent = TreeView_InsertItem(g_hwndChild, &tvis);


  tvis.item.pszText = TEXT(“Second”);
  tvis.item.state = TVIS_BOLD;
  TreeView_InsertItem(g_hwndChild, &tvis);


  tvis.item.pszText = TEXT(“Third”);
  tvis.item.state = 0;
  TreeView_InsertItem(g_hwndChild, &tvis);


  return TRUE;
}
```

I elided error checking for expository purposes.

This code creates a tree view and populates it as follows:

First
- **Second**
- Third

When you run this program, you can see that the text for all the items appears as expected; nothing is truncated.

As for the backward compability remark, it’s quite simple: *Every change, no matter how seemingly insignificant, has compatibility consequences*. The common controls are heavily used in third party programs, many of which make strange assumptions about how the controls work, relying on quirks of implementations in strange ways. For example, those who have read the first bonus chapter of my book will know that even something as seemingly harmless as fixing a flicker bug in the status bar resulted in a broken status bar in a program from a major software publisher. Every change is taken with great trepidation, and the bias is to preserve bug-for-bug compatibility.

In this case, the issue was with the way the width of the treeview item is calculated. You can easily imagine programs which worked around the existing behavior by artificially padding the item with spaces to compensate for the miscalculation. If the treeview suddenly fixed the bug, these treeview items would now be undesirably large, possibly creating a horizontal scroll bar where there previously had been none, resulting in bugs like "After upgrading, the last item in my treeview is being covered by a scroll bar." We saw something like this before when we looked at the effects of the `DS_SHELLFONT` dialog style: Fixing the bug described in that article would result in property sheet pages coming out undesirably large (because their sizes were artificially inflated to compensate for the erroneous calculation).

That doesn't mean the bug can't get fixed, however. Just as the `DS_SHELLFONT` flag is a signal to say that your property sheet page wants to use the new calculations, you can tell the tree view "Please give me the version of the treeview that fixes the font bug" by sending it the `CCM_SETVERSION` message and specifying that you want version 5. Similarly, you can opt into version 6 of the common controls by providing a manifest.

Update: I slipped a subtlety into this article which it appears people didn't pick up on, so I'll make it explicit.

The original question was about "switching the font from normal to bold", but there are multiple ways of doing this. My sample code used the recommended (declarative) method of setting the `TVIS_BOLD` flag. But if you click through the link, you'll see that the original commenter was using the procedural method of handling the `NM_CUSTOMDRAW` notification, selecting a new font (a boldface variation of the normal font), and returning `CRF_NEWFONT`. This is a technique I had illustrated previously with list views and tool tips.

The compatibility behavior is for fonts customized via `NM_CUSTOMDRAW`. The declarative method was added specifically to address the bug in item size calculations when people change the font procedurally: Older versions of the treeview control asked for the custom font only when painting; it didn't ask for the custom font when measuring. Adding the font query to version 6 was actually quite risky, since the only way to ask the application for the procedurally-applied font is to *actually go through the motions of drawing it*, generating a dummy `NM_CUSTOMDRAW` notification with an empty paint rectangle. If an application painted outside the rectangle, you would have seen seen random painting on the screen.

Raymond Chen

Follow