How to host an IContextMenu, part 9 – Adding custom commands

October 4, 2004

Raymond Chen

The indexMenu, idCmdFirst and idCmdLast parameters to the IContextMenu::QueryContextMenu method allow you, the host, to control where in the context menu the IContextMenu will insert its commands. To illustrate this, let’s put two bonus commands on our context menu, with the boring names “Top” and “Bottom”.

We need to reserve some space in our menu identifiers, so let’s carve some space out for our private commands:

```
#define SCRATCH_QCM_FIRST 1
#define SCRATCH_QCM_LAST  0x6FFF
#define IDM_TOP           0x7000
#define IDM_BOTTOM        0x7001
```

We reserved 0x1000 commands for ourselves, allowing the IContextMenu to play with commands 1 through 0x6FFF. (We could have carved our space out of the low end, too, by increasing SCRATCH_QCM_FIRST instead of decreasing SCRATCH_QCM_LAST.)

Go back to the program we had in part 6 and make these changes:

```
void OnContextMenu(HWND hwnd, HWND hwndContext, int xPos, int yPos)
{
  POINT pt = { xPos, yPos };
  if (pt.x == -1 && pt.y == -1) {
    pt.x = pt.y = 0;
    ClientToScreen(hwnd, &pt);
  }


  IContextMenu *pcm;
  if (SUCCEEDED(GetUIObjectOfFile(hwnd, L”C:\\Windows\\clock.avi”,
                 IID_IContextMenu, (void**)&pcm))) {
    HMENU hmenu = CreatePopupMenu();
    if (hmenu) {
      if (InsertMenu(hmenu, 0, MF_BYPOSITION,
                     IDM_TOP, TEXT(“Top”)) &&
          InsertMenu(hmenu, 1, MF_BYPOSITION,
                     IDM_BOTTOM, TEXT(“Bottom”)) &&
          SUCCEEDED(pcm->QueryContextMenu(hmenu, 1,
                             SCRATCH_QCM_FIRST, SCRATCH_QCM_LAST,
                             CMF_NORMAL))) {
        pcm->QueryInterface(IID_IContextMenu2, (void**)&g_pcm2);
        pcm->QueryInterface(IID_IContextMenu3, (void**)&g_pcm3);
        int iCmd = TrackPopupMenuEx(hmenu, TPM_RETURNCMD,
                                    pt.x, pt.y, hwnd, NULL);
        if (g_pcm2) {
          g_pcm2->Release();
          g_pcm2 = NULL;
        }
        if (g_pcm3) {
          g_pcm3->Release();
          g_pcm3 = NULL;
        }
        if (iCmd == IDM_TOP) {
          MessageBox(hwnd, TEXT(“Top”), TEXT(“Custom”), MB_OK);
        } else if (iCmd == IDM_BOTTOM) {
          MessageBox(hwnd, TEXT(“Bottom”), TEXT(“Custom”), MB_OK);
        } else if (iCmd > 0) {
          CMINVOKECOMMANDINFOEX info = { 0 };
          info.cbSize = sizeof(info);
          info.fMask = CMIC_MASK_UNICODE | CMIC_MASK_PTINVOKE;
          if (GetKeyState(VK_CONTROL) < 0) {
            info.fMask |= CMIC_MASK_CONTROL_DOWN;
          }
          if (GetKeyState(VK_SHIFT) < 0) {
            info.fMask |= CMIC_MASK_SHIFT_DOWN;
          }
          info.hwnd = hwnd;
          info.lpVerb  = MAKEINTRESOURCEA(iCmd – SCRATCH_QCM_FIRST);
          info.lpVerbW = MAKEINTRESOURCEW(iCmd – SCRATCH_QCM_FIRST);
          info.nShow = SW_SHOWNORMAL;
          info.ptInvoke = pt;
          pcm->InvokeCommand((LPCMINVOKECOMMANDINFO)&info);
        }
      }
      DestroyMenu(hmenu);
    }
    pcm->Release();
  }
}
```

[Corrected insertion location for “Bottom” 9:42am.]

Before calling IContextMenu::QueryContextMenu, we added our own custom commands (with menu identifiers outside the range we offer to IContextMenu::QueryContextMenu so they won’t conflict), and then call IContextMenu::QueryContextMenu passing the new reduced range as well as specifying that the insertion position is 1 instead of 0.

When we pass the context menu to to IContextMenu::QueryContextMenu, the menu looks like this:

Top

Bottom

By passing 1 as the insertion point, we are telling the context menu handler that it should insert its commands at position 1 (pushing out what is currently at positions 1 and onwards).

Top

… new stuff …

Bottom

After displaying this enhanced context menu, we check which command the user picked, whether it’s one of ours (which we handle directly) or one from the inserted portion of the context menu (which we dispatch to the handler).

Raymond Chen

Follow