How to host an IContextMenu, part 6 – Displaying menu help

devblogs.microsoft.com/oldnewthing/20040928-00

September 28, 2004

Raymond Chen

One of the subtleties of context menus is showing help in the status bar. Now, the program we've been developing doesn't have a status bar, so we'll fake it by putting the help text in the title bar.

The key method for this task is IContextMenu::GetCommandString, which allows communication with a context menu handler about the verbs in the menu. We'll have to stash yet another interface in our "instance variables disguised as globals".

```
IContextMenu *g_pcm;
```

(Remember, in a "real program", these would be per-window instance variables, not globals.)

We also need to update that variable during menu tracking.

```
      g_pcm = pcm;
      int iCmd = TrackPopupMenuEx(hmenu, TPM_RETURNCMD, pt.x, pt.y, hwnd, NULL);
      g_pcm = NULL;
```

With that out of the way, we can now provide feedback as the user browses the popup menu.

[Introduction of `g_pcm` variable added 29 September.]

```
// This code is buggy – see below.
void OnMenuSelect(HWND hwnd, HMENU hmenu,
                  int item, HMENU hmenuPopup, UINT flags)
{
  if (g_pcm && item >= SCRATCH_QCM_FIRST &&
      item <= SCRATCH_QCM_LAST) {
    TCHAR szBuf[MAX_PATH];
    if (FAILED(g_pcm->GetCommandString(item – SCRATCH_QCM_FIRST,
                                       GCS_HELPTEXT, NULL,
                                       (LPSTR)szBuf, MAX_PATH))) {
      lstrcpyn(szBuf, TEXT(“No help available.”), MAX_PATH);
    }
    SetWindowText(hwnd, szBuf);
  }
}
```

This function checks whether the menu selection is in the range of items that we allowed the context menu to own. If so, we ask for the help string (or use fallback text if the context menu handler didn’t provide a help string) and display it as our window title.

Finally, we insert this function into our window procedure. We want to update the menu selection status even if the context menu handlers do something with it, so we need to call OnMenuSelect before dispatching to the context menu handlers.

```
LRESULT CALLBACK
WndProc(HWND hwnd, UINT uiMsg, WPARAM wParam, LPARAM lParam)
{
    if (uiMsg == WM_MENUSELECT) {
        HANDLE_WM_MENUSELECT(hwnd, wParam, lParam, OnMenuSelect);
    }
    if (g_pcm3) {
…
```

Wait a second, there was a comment up there that said that the OnMenuSelect function is buggy. Where’s the bug?

Well, technically there is no bug, but if you run this program as-is (and I suggest that you do), you’ll find that what you get is rather erratic.

That’s because there are a lot of buggy context menu handlers out there.

Some context menu handlers don’t support Unicode; others don’t support Ansi. What’s really fun is that instead of returning E_NOTIMPL, they return S_OK but don’t actually do anything. Other context menus have buffer overflow problems and write to the buffer beyond the actual size you specified.

Welcome to the world of application compatibility.

Let’s write a helper function that tries to hide all of these weirdnesses.

```
HRESULT IContextMenu_GetCommandString(
    IContextMenu *pcm, UINT_PTR idCmd, UINT uFlags,
    UINT *pwReserved, LPWSTR pszName, UINT cchMax)
{
  // Callers are expected to be using Unicode.
  if (!(uFlags & GCS_UNICODE)) return E_INVALIDARG;


  // Some context menu handlers have off-by-one bugs and will
  // overflow the output buffer. Let’s artificially reduce the
  // buffer size so a one-character overflow won’t corrupt memory.
  if (cchMax <= 1) return E_FAIL;
  cchMax–;


  // First try the Unicode message.  Preset the output buffer
  // with a known value because some handlers return S_OK without
  // doing anything.
  pszName[0] = L’\0';


  HRESULT hr = pcm->GetCommandString(idCmd, uFlags, pwReserved,
                                     (LPSTR)pszName, cchMax);
  if (SUCCEEDED(hr) && pszName[0] == L’\0') {
    // Rats, a buggy IContextMenu handler that returned success
    // even though it failed.
    hr = E_NOTIMPL;
  }


  if (FAILED(hr)) {
    // try again with ANSI – pad the buffer with one extra character
    // to compensate for context menu handlers that overflow by
    // one character.
    LPSTR pszAnsi = (LPSTR)LocalAlloc(LMEM_FIXED,
                                      (cchMax + 1) * sizeof(CHAR));
    if (pszAnsi) {
      pszAnsi[0] = ‘\0’;
      hr = pcm->GetCommandString(idCmd, uFlags & ~GCS_UNICODE,
                                 pwReserved, pszAnsi, cchMax);
      if (SUCCEEDED(hr) && pszAnsi[0] == ‘\0’) {
        // Rats, a buggy IContextMenu handler that returned success
        // even though it failed.
        hr = E_NOTIMPL;
      }
      if (SUCCEEDED(hr)) {
        if (MultiByteToWideChar(CP_ACP, 0, pszAnsi, -1,
                               pszName, cchMax) == 0) {
          hr = E_FAIL;
        }
      }
      LocalFree(pszAnsi);
    } else {
      hr = E_OUTOFMEMORY;
    }
  }
  return hr;
}
```

The shell has lots of strange functions like this.

[`pszAnsi` comparison fixed, 29 September.]

With this helper function, we can fix our help text function.

```
void OnMenuSelect(HWND hwnd, HMENU hmenu,
                  int item, HMENU hmenuPopup, UINT flags)
{
  if (g_pcm && item >= SCRATCH_QCM_FIRST &&
      item <= SCRATCH_QCM_LAST) {
    WCHAR szBuf[MAX_PATH];
    if (FAILED(IContextMenu_GetCommandString(g_pcm,
                                    item – SCRATCH_QCM_FIRST,
                                    GCS_HELPTEXTW, NULL,
                                    szBuf, MAX_PATH))) {
      lstrcpynW(szBuf, L“No help available.”, MAX_PATH);
    }
    SetWindowTextW(hwnd, szBuf);
  }
}
```

This new version displays help texts for all the context menu handlers that support it, in spite of the attempts of many of those context menu handlers to get it wrong or even create a buffer overflow security vulnerability.

Okay, that was quite a long digression from part 1 of this series. Let’s return to the subject of invoking the default verb next time.

Raymond Chen

Follow