How to host an IContextMenu, part 1 – Initial foray

devblogs.microsoft.com/oldnewthing/20040920-00

September 20, 2004

Raymond Chen

Most documentation describes how to plug into the shell context menu structure and be a context menu provider. If you read the documentation from the other side, then you also see how to **host** the context menu. (This is the first of an eleven-part series with three digressions. Yes, eleven parts—sorry for all you folks who are in it just for the history articles. I'll try to toss in an occasional amusing diversion.)

The usage pattern for an IContextMenu is as follows:

- Creation.
- IContextMenu::QueryContextMenu. This initializes the context menu. During this call, the context menu decides which items appear in it, based on the flags you pass.
- Display the menu or otherwise select a command to execute, using IContextMenu::GetCommandString, IContextMenu2::HandleMenuMsg and IContextMenu3::HandleMenuMsg2 to faciliate the user interaction.
- IContextMenu::InvokeCommand. This executes the command.

The details of this are explained in Creating Context MenuHandlers from the point of view of the IContextMenu implementor.

> The Shell first calls IContextMenu::QueryContextMenu. It passes in an HMENU handle that the method can use to add items to the context menu. If the user selects one of the commands, IContextMenu::GetCommandString is called to retrieve the Help string that will be displayed on the Microsoft Windows Explorer status bar. If the user clicks one of the handler's items, the Shell calls IContextMenu::InvokeCommand. The handler can then execute the appropriate command.

Read it from the other side to see what it says you need to do as the IContextMenu host:

> The *IContextMenu host* first calls IContextMenu::QueryContextMenu. It passes in an HMENU handle that the method can use to add items to the context menu. If the user selects one of the commands, IContextMenu::GetCommandString is called to retrieve the Help string that will be displayed on *the host's* status bar. If the user clicks one of the handler's items, the *IContextMenu host* calls IContextMenu::InvokeCommand. The handler can then execute the appropriate command.

Exploring the consequences of this new interpretation of the context menu documentation will be our focus for the next few weeks.

Okay, let's get started. We begin, as always, with our scratch program. I'm going to assume you're already familiar with the shell namespace and pidls so I can focus on the context menu part of the issue.

```
#include <shlobj.h>


HRESULT GetUIObjectOfFile(HWND hwnd, LPCWSTR pszPath, REFIID riid, void **ppv)
{
  *ppv = NULL;
  HRESULT hr;
  LPITEMIDLIST pidl;
  SFGAOF sfgao;
  if (SUCCEEDED(hr = SHParseDisplayName(pszPath, NULL, &pidl, 0, &sfgao))) {
    IShellFolder *psf;
    LPCITEMIDLIST pidlChild;
    if (SUCCEEDED(hr = SHBindToParent(pidl, IID_IShellFolder,
                                      (void**)&psf, &pidlChild))) {
      hr = psf->GetUIObjectOf(hwnd, 1, &pidlChild, riid, NULL, ppv);
      psf->Release();
    }
    CoTaskMemFree(pidl);
  }
  return hr;
}
```

This simple function takes a path and gets a shell UI object from it. We convert the path to a pidl with SHParseDisplayName, then bind to the pidl's parent with SHBindToParent, then ask the parent for the UI object of the child with IShellFolder::GetUIObjectOf. I'm assuming you've had enough experience with the namespace that this is ho-hum.

(The helper functions SHParseDisplayName and SHBindToParent don't do anything you couldn't have done yourself. They just save you some typing. Once you start using the shell namespace for any nontrivial amount of time, you build up a library of little functions like these.)

For our first pass, all we're going to do is invoke the "Play" verb on the file when the user right-clicks. (Why right-click? Because a future version of this program will display a context menu.)

```
#define SCRATCH_QCM_FIRST 1
#define SCRATCH_QCM_LAST  0x7FFF


void OnContextMenu(HWND hwnd, HWND hwndContext, UINT xPos, UINT yPos)
{
  IContextMenu *pcm;
  if (SUCCEEDED(GetUIObjectOfFile(hwnd, L"C:\\Windows\\clock.avi",
                  IID_IContextMenu, (void**)&pcm))) {
    HMENU hmenu = CreatePopupMenu();
    if (hmenu) {
      if (SUCCEEDED(pcm->QueryContextMenu(hmenu, 0,
                           SCRATCH_QCM_FIRST, SCRATCH_QCM_LAST,
                           CMF_NORMAL))) {
        CMINVOKECOMMANDINFO info = { 0 };
        info.cbSize = sizeof(info);
        info.hwnd = hwnd;
        info.lpVerb = "play";
        pcm->InvokeCommand(&info);
      }
      DestroyMenu(hmenu);
    }
    pcm->Release();
  }
}


    HANDLE_MSG(hwnd, WM_CONTEXTMENU, OnContextMenu);
```

As noted in the checklist above, first we create the IContextMenu, then initialize it by calling IContextMenu::QueryContextMenu. Notice that even though we don't intend to display the menu, we still have to create a popup menu because IContextMenu::QueryContextMenu requires on. We don't actually display the resulting menu, however; instead of asking the user to pick an item from the menu, we make the choice for the user and select "Play", filling in the CMINVOKECOMMANDINFO structure and invoking it.

But how did we know that the correct verb was "Play"? In this case, we knew because we hard-coded the file to "clock.avi" and we knew that AVI files have a "Play" verb. But of course that doesn't work in general. Before getting to invoking the default verb, let's first take the easier step of asking the user what verb to invoke. That exercise will actually distract us for a while, but we'll come back to the issue of the default verb afterwards.

If the code above is all you really wanted (invoking a fixed verb on a file), then you didn't need to go through all the context menu stuff. The code above is equivalent to calling the ShellExecuteEx function, passing the SEE_MASK_INVOKEIDLIST flag to indicate that you want the invoke to go through the IContextMenu.

[Typo fixed 25 September.]

Raymond Chen

Follow