Dragging a shell object, part 2: Enabling the Move operation

devblogs.microsoft.com/oldnewthing/20041207-00

December 7, 2004

Raymond Chen

Let's say that we did want to support Move in our drag/drop program, for whatever reason. Let's do it with some scratch file instead of `clock.avi`, though. Create a file somewhere that you don't mind losing; let's say it's `C:\throwaway.txt`. Change the function `OnLButtonDown` as follows:

```
void OnLButtonDown(HWND hwnd, BOOL fDoubleClick,
                   int x, int y, UINT keyFlags)
{
  IDataObject *pdto;
  if (SUCCEEDED(GetUIObjectOfFile(hwnd,
                   L"C:\\throwaway.txt",
                   IID_IDataObject, (void**)&pdto))) {
    IDropSource *pds = new CDropSource();
    if (pds) {
      DWORD dwEffect;
      if (DoDragDrop(pdto, pds,
                 DROPEFFECT_COPY | DROPEFFECT_LINK | DROPEFFECT_MOVE,
                 &dwEffect) == DRAGDROP_S_DROP) {
        if (dwEffect & DROPEFFECT_MOVE) {
          DeleteFile(TEXT("C:\\throwaway.txt"));
        }
      }
      pds->Release();
    }
    pdto->Release();
  }
}
```

Oh wait, there are people out there who think I'm advocating hard-coded paths, so let me change the program to operate on a path passed on the command line. This is code that is purely a distraction from the point of this article, which is why I avoided it originally. Personally I dislike it when somebody hands me a sample program that is 90% unrelated to the technology the program is trying to demonstrate. I have to go digging through the code hunting for the 10% of stuff that matters.

```
#include <shellapi.h>


LPWSTR *g_argv;
LPCWSTR g_pszTarget;


void OnLButtonDown(HWND hwnd, BOOL fDoubleClick,
                   int x, int y, UINT keyFlags)
{
  IDataObject *pdto;
  if (SUCCEEDED(GetUIObjectOfFile(hwnd,
                    g_pszTarget,
                    IID_IDataObject, (void**)&pdto))) {
  …
          DeleteFileW(g_pszTarget);
  …
}


BOOL
InitApp(void)
{
  int argc;
  g_argv = CommandLineToArgvW(GetCommandLineW(), &argc);
  if (!g_argv || argc != 2) return FALSE;
  g_pszTarget = g_argv[1];
  if (PathIsRelative(g_pszTarget)) return FALSE;
  …
}
```

Woo-hoo, eight distracting lines of code that have nothing to do with the subject of dragging shell objects around. I hope you're happy.

Where was I? Oh right, explaining the first batch of blue code that by now has scrolled off your screen thanks to the intervening meaningless drivel.

Now that we allow move, we need to check whether the resulting effect was `DROPEFFECT_MOVE`, which tells us, "The drop target wanted to perform a move operation, but it only got as far as copying the object; please finish the move operation by deleting the original."

Notice that `DROPEFFECT_MOVE` does **not** mean, "The drop target performed a move." Rather, it tells you that the drop target wants you to delete the original. If the drop target was able to delete the original (or move it directly), then you will not get `DROPEFFECT_MOVE` back.

(One case where `DROPEFFECT_MOVE` doesn't even mean that a Move operation occurred at all is if the user dragged the object to an "Incinerator" icon, the purpose of which is to destroy whatever is dropped onto it. In this case the Incinerator would return `DROPEFFECT_MOVE` without even making a copy. Result: The object is deleted. A better name for `DROPEFFECT_MOVE` would have been `DROPEFFECT_DELETEORIGINAL`.)

If the data object represents a file, then the shell is pretty good at figuring out how to move the file to the destination instead of copying it and asking you to delete the original. You will typically get `DROPEFFECT_MOVE` back only if the data object represents a non-file, since in that case the shell doesn't know how to delete the original.

But what if you want to know whether the operation was a move, regardless of whether the operation was optimized by the drop target? We'll look at that next time.

(By the way, if you execute a Move of the throwaway file, don't forget to move it back so you can run the scratch program again!)

Raymond Chen

Follow