Using Explorer’s fancy drag/drop effects in your own programs

devblogs.microsoft.com/oldnewthing/20210512-00

May 12, 2021

Raymond Chen

The default drag/drop effect is very plain. Let’s try it out. Take the scratch program and make these changes. The COM smart pointer library for today is (rolls dice) C++/WinRT. I will also be using wil for error handling and RAII types.

```
#include <winrt/base.h>
#include <wil/result_macros.h>
#include <wil/resource.h>

struct SimpleDropTarget :
    winrt::implements<SimpleDropTarget, IDropTarget>
{
    SimpleDropTarget(HWND hwnd) : hwndOwner(hwnd)
    {
    }

    STDMETHODIMP DragEnter(IDataObject* pDataObject,
        DWORD grfKeyState, POINTL pt, DWORD* pdwEffect)
    {
        dto.copy_from(pDataObject);
        RETURN_IF_FAILED(CalculateFeedback(grfKeyState, pdwEffect));
        return S_OK;
    }
    STDMETHODIMP DragOver(
        DWORD grfKeyState, POINTL pt, DWORD* pdwEffect)
    {
        RETURN_IF_FAILED(CalculateFeedback(grfKeyState, pdwEffect));
        return S_OK;
    }

    STDMETHODIMP Drop(IDataObject* pDataObject,
        DWORD grfKeyState, POINTL pt, DWORD* pdwEffect)
    {
        dto.copy_from(pDataObject);
        auto cleanup = wil::scope_exit([&] { DragLeave(); });

        RETURN_IF_FAILED(CalculateFeedback(grfKeyState, pdwEffect));
        if (*pdwEffect != DROPEFFECT_NONE) {
            // Do something cool.
        }
        return S_OK;
    }

    STDMETHODIMP DragLeave()
    {
        dto = nullptr;
        return S_OK;
    }

private:
    HWND hwndOwner;
    winrt::com_ptr<IDataObject> dto;

    HRESULT CalculateFeedback(
        DWORD grfKeyState,
        DWORD* pdwEffect)
    {
        if (grfKeyState & MK_CONTROL) {
            *pdwEffect = DROPEFFECT_COPY;
        } else {
            *pdwEffect = DROPEFFECT_MOVE;
        }
        return S_OK;
    }
};
```

This drop target does nothing particularly fancy. The only special thing is that holding the `Ctrl` key provides copy feedback rather than move feedback.

The `Drop` method processes the new data object and either performs the drop or defers to the `DragLeave` feedback if it turns out nothing is being dropped after all. Regardless of how the drop plays out, we call our own `DragLeave()` to clean up. (We use the `wil::scope_exit` function to create a one-off RAII type that performs some action at destruction.)

Let’s hook up this drop target to our window.

```
BOOL
OnCreate(HWND hwnd, LPCREATESTRUCT lpcs)
{
    RegisterDragDrop(hwnd,
        winrt::make<SimpleDropTarget>(hwnd).get());

    return TRUE;
}

int WINAPI WinMain(HINSTANCE hinst, HINSTANCE hinstPrev,
    LPSTR lpCmdLine, int nShowCmd)
{
    ...
    if (SUCCEEDED(OleInitialize(NULL))) {
        ...
    }
    ...
}
```

In the main program, we use `OleInitialize` instead of `CoInitialize` because we are using OLE drag/drop, which is an OLE feature.

And when the window is created, we register our drop target.

When you run this program, the drag/drop feedback is just an empty gray box, possibly with a + sign to indicate that the resulting operation is a copy.

Let’s add fancy Explorer-style drag/drop feedback.

```
struct SimpleDropTarget :
    winrt::implements<SimpleDropTarget, IDropTarget>
{
    SimpleDropTarget(HWND hwnd) : hwndOwner(hwnd),
        helper(winrt::create_instance<IDropTargetHelper>(
            CLSID_DragDropHelper))
    {
    }

    STDMETHODIMP DragEnter(IDataObject* pDataObject,
        DWORD grfKeyState, POINTL pt, DWORD* pdwEffect)
    {
        dto.copy_from(pDataObject);
        RETURN_IF_FAILED(CalculateFeedback(grfKeyState, pdwEffect));
        POINT point{ pt.x, pt.y };
        RETURN_IF_FAILED(helper->DragEnter(hwndOwner, dto.get(),
            &point, *pdwEffect));
        return S_OK;
    }

    STDMETHODIMP DragOver(
        DWORD grfKeyState, POINTL pt, DWORD* pdwEffect)
    {
        RETURN_IF_FAILED(CalculateFeedback(grfKeyState, pdwEffect));
        POINT point{ pt.x, pt.y };
        RETURN_IF_FAILED(helper->DragOver(&point, *pdwEffect));
        return S_OK;
    }

    STDMETHODIMP Drop(IDataObject* pDataObject,
        DWORD grfKeyState, POINTL pt, DWORD* pdwEffect)
    {
        dto.copy_from(pDataObject);
        auto cleanup = wil::scope_exit([&] { DragLeave(); });

        RETURN_IF_FAILED(CalculateFeedback(grfKeyState, pdwEffect));
        POINT point{ pt.x, pt.y };
        RETURN_IF_FAILED(helper->Drop(dto.get(), &point, *pdwEffect));
        if (*pdwEffect != DROPEFFECT_NONE) {
            // Do something cool.
        }
        return S_OK;
    }

    STDMETHODIMP DragLeave()
    {
        dto = nullptr;
        RETURN_IF_FAILED(helper->DragLeave());
        return S_OK;
    }

private:
    HWND hwndOwner;
    winrt::com_ptr<IDataObject> dto;
    winrt::com_ptr<IDropTargetHelper> helper;

    HRESULT CalculateFeedback(
        DWORD grfKeyState,
        DWORD* pdwEffect)
    {
        if (grfKeyState & MK_CONTROL) {
            *pdwEffect = DROPEFFECT_COPY;
        } else {
            *pdwEffect = DROPEFFECT_MOVE;
        }
        return S_OK;
    }
};
```

We create a shell drop target helper and forward all of our drop target methods into it. There is a bit of frustration here due to the impedance mismatch between the way `IDropTarget` and `IDropTargetHelper` represent the point.

With these changes, the drop feedback is much more Explorer-like: The drag image matches the Explorer drag image, showing a thumbnail of the item being dragged, or a collection of items with a numeric badge showing how many items are being dragged. There's also an information box below the image that says what the resulting operation will be.

That's a good start. Next time, we'll add another feature.

Raymond Chen

Follow