If I register the same shell extension as both a context menu extension and a drag/drop extension, how do I know which one the system is using?

devblogs.microsoft.com/oldnewthing/20240701-00

July 1, 2024

Raymond Chen

A customer was developing a shell extension that implemented `IContextMenu`. This extension interface is used for both context menu extensions and drag/drop extensions, and they registered their extension's CLSID in both places. When the user right-clicked on a file, their extension was called to add menu items to the file's context menu. And when the user did a right-mouse drag/drop, their extension was called to add menu items to the drag/drop context menu.

But they wanted their shell extension to know whether it is being used as a context menu extension or a drag/drop extension. How can their extension detect which scenario is active? Other combination shell extensions can detect the scenario by seeing which interface's methods are called. For example, if `IShellPropSheetExt::AddPages` is called, then it knows that it's being used as a property sheet extension. But both context menu extensions and drag/drop extensions use the same interface `IContextMenu`, so they can't use that to detect the scenario.

My kids have a pretend store called "Fred Tire", a pun on Fred Meyer, a local chain of grocery/department stores. When you call Fred Tire, you can order automobile tires, or you can order a pizza.[1]

This customer basically wrote their own Fred Tire.

When a customer calls the store, you can answer the phone “Hello, this is Fred Tire. How can I help you?” If the customer says, “I’d like to buy some tires,” then you take down their tire order. If they say, “I’d like to order a pizza,” then you take down their pizza order. This is the Fred Tire version of using the interface to detect which scenario is being used. Furthermore, after the customer places their tire order, they might say, “And I’d also like to order a pizza,” and you can take their pizza order too. The same order-taker can handle both tires and pizzas. They just follow the customer’s lead as to which order form to fill out.

This customer, however, wanted to answer the phone in a different way depending on what the customer wants. They want to know whether to say “Hello, this is Fred Tire, what kind of tire do you need?” Or whether they should say, “Hello, this is Fred Tire, what kind of pizza would you like?” How can they answer the phone correctly?

If you want to answer the phone differently, what you can do is get two telephone numbers. Put one phone number on your tire flyers and another number on your pizza flyers. Both numbers ring the same phone, but you can see which line is ringing. That way, when somebody calls the tire number, you say, “Hello, this is Fred Tire, what kind of tire do you need?” And if somebody calls the pizzeria number, you say, “Hello, this is Fred Tire, what kind of pizza would you like?”

For shell extensions, this means that you register two different CLSIDs, one for the context menu extension and another for the drag/drop extension. Creating either of the CLSIDs produces the same object, but also passes the CLSID to the constructor so that the object knows which scenario is active.

```
// sketch
class MyShellExtension : IContextMenu, ⟦ other interfaces ⟧
{
    MyShellExtension(REFCLSID clsid) :
        m_clsid(clsid) {}

    CLSID const m_clsid;

    bool IsCreatedForContextMenu()
        { return m_clsid == CLSID_MyContextMenu; }
    bool IsCreatedForDragDrop()
        { return m_clsid == CLSID_MyDragDrop; }

    ⟦ ... other methods ... ⟧
};

class MyShellExtensionFactory : IClassFactory
{
    MyShellExtensionFactory(REFCLSID clsid) :
        m_clsid(clsid) {}

    CLSID const m_clsid;

    STDMETHODIMP CreateInstance(IUnknown* outer, REFIID iid, void** ppv)
    {
        *ppv = nullptr;
        if (outer) return CLASS_E_NOAGGREGATION;
        auto instance = new (std::nothrow) MyShellExtension(clsid);
        if (!instance) return E_OUTOFMEMORY;
        auto hr = instance->QueryInterface(iid, ppv);
        factory->Release();
        return hr;
    }

    ⟦ ... other methods ... ⟧
};

STDAPI DllGetClassObject(REFCLSID clsid, REFIID iid, void** ppv)
{
    *ppv = nullptr;
    if (clsid == CLSID_MyContextMenu || clsid == CLSID_MyDragDrop)
    {
        auto factory = new(std::nothrow) MyShellExtensionFactory(clsid);
        if (!factory) return E_OUTOFMEMORY;
        auto hr = factory->QueryInterface(iid, ppv);
        factory->Release();
        return hr;
    }
    return REGDB_E_CLASSNOTREG;
}
```

You can pass the `clsid` to the factory, who in turn passes it to the `MyShellExtension`, who can then use it to figure out which kind of shell extension it was created for.

If your framework doesn't let you pass parameters to factories, you can just create separate factories.

```
enum class ExtensionKind
{
    ContextMenu,
    DragDrop,
};

class MyShellExtension : IContextMenu, ⟦ other interfaces ⟧
{
    MyShellExtension(ExtensionKind kind) : m_kind(kind) {}

    ExtensionKind const m_kind = kind;

    bool IsCreatedForContextMenu()
        { return m_kind == ExtensionKind::ContextMenu; }
    bool IsCreatedForDragDrop()
        { return m_clsid == ExtensionKind::DragDrop; }

    ⟦ ... other methods ... ⟧
};

class MyContextMenuShellExtension : MyShellExtension
{
    MyContextMenuShellExtension() :
        MyShellExtension(ExtensionKind::ContextMenu) {}
};

class MyDragDropShellExtension : MyShellExtension
{
    MyDragDropShellExtension() :
        MyShellExtension(ExtensionKind::DragDrop) {}
};

CoCreatableClass(MyContextMenuShellExtension)
CoCreatableClass(MyDragDropShellExtension)
```

If your framework doesn't even allow that, then you could templatize the class.

```
template<ExtensionKind kind>
class MyShellExtension : IContextMenu, ⟦ other interfaces ⟧
{
    MyShellExtension() = default;

    ExtensionKind const m_kind = kind;

    bool IsCreatedForContextMenu()
        { return m_kind == ExtensionKind::ContextMenu; }
    bool IsCreatedForDragDrop()
        { return m_clsid == ExtensionKind::DragDrop; }

    ⟦ ... other methods ... ⟧
};


using MyContextMenuShellExtension =
    MyShellExtension<ExtensionKind::ContextMenu>
using MyDragDropShellExtension =
    MyShellExtension<ExtensionKind::DragDrop>
```

[1]Or (and this is where things get fun) you can order a “tire pizza”. I always order the pineapple tire pizza.