The COM interface contract rules exist for a reason

devblogs.microsoft.com/oldnewthing/20051101-54

November 1, 2005

Raymond Chen

Some people believe that the COM rules on interfaces are needlessly strict. But the rules are there for a reason.

Suppose you ship some interface in version N of your product. It's an internal interface, not documented to outsiders. Therefore, you are free to change it any time you want without having to worry about breaking compatibility with any third-party plug-ins.

But remember that if you change an interface, you need to generate a new Interface Identifier (IID). Because an interface identifier uniquely identifies the interface. (That's sort of implied by its name, after all.)

And this rule applies even to internal interfaces.

Suppose you decide to violate this rule and use the same IID to represent a slightly different interface in version N+1 of your program. Since this is an internal interface, you have no qualms about doing this.

Until you have to write a patch that services both versions.

Now your patch is in trouble. It can call `IUnknown::QueryInterface` and ask for that IID, and it will get something back. But you don't know whether this is the version N interface or the version N+1 interface. If you're not even aware that this has happened, your patch will probably just assume it has the version N+1 interface, and strange things happen when it is run on version N.

Debugging this problem is not fun. Neither is fixing it. Your patch has to use some other cues to decide which interface it actually got back. If your program has been patched previously, you need to have the version numbers of every single patch so that you can determine which version of the interface you have.

Note that this dependency can be hidden behind other interfaces. Consider:

```
[
    uuid(“ABC”)
]
interface IColorInfo
{
    HRESULT GetBackgroundColor([out] COLORREF *pcr);
    …
};


[
    uuid(“XYZ”)
]
interface IGraphicImage
{
    …
    HRESULT GetColorInfo([out] IColorInfo **ppci);
};
```

Suppose you want to add a new method to the `IColorInfo` interface:

```
[
    uuid(“DEF“)
]
interface IColorInfo
{
    HRESULT GetBackgroundColor([out] COLORREF *pcr);
    …
    HRESULT AdjustColor(COLORREF clrOld,
                        COLORREF clrNew);
};


[
    uuid(“XYZ”)
]
interface IGraphicImage
{
    …
    HRESULT GetColorInfo([out] IColorInfo **ppci);
};
```

You changed the interface, but you also changed the IID, so everything is just fine, right?

No, it isn’t.

The `IGraphicImage` interface is dependent upon the `IColorInfo` interface. When you changed the `IColorInfo` interface, you implicitly changed the `IGraphicImage::GetColorInfo` method, since the returned interface is now the version N+1 `IColorInfo` interface.

Consider a patch written with the version N+1 header files.

```
void AdjustGraphicColorInfo(IGraphicImage* pgi,
                            COLORREF clrOld, COLORREF clrNew)
{
 IColorInfo *pci;
 if (SUCCEEDED(pgi->GetColorCount(&pci)) {
  pci->AdjustColor(clrOld, clrNew);
  pci->Release();
 }
}
```

If run against version N, the call to `IGraphicImage::GetColorCount` will return a version N `IColorInfo`, and that version doesn't support the `IColorInfo::AdjustColor` method. But you're going to call it anyway. Result: Walking off the end of the version N vtable and calling into space.

The quick solution is to change the IID for the `IGraphicImage` function to reflect the change on the `IColorInfo` interface on which it depends.

```
[
    uuid("UVW")
]
interface IGraphicImage
{
    …
    HRESULT GetColorInfo([out] IColorInfo **ppci);
};
```

A more robust fix would be to change the `IGraphicImage::GetColorInfo` method so that you pass the interface you want to receive.

```
[
    uuid("RST")
]
interface IGraphicImage
{
    …
    HRESULT GetColorInfo([in] REFIID riid,
                         [iid_is(riid), out] void** ppv);
};
```

This allows interfaces on which `IGraphicImage` depends to change without requiring a change to the `IGraphicImage` interface itself. Of course, the implementation needs to change to respond to the new value of `IID_IColorInfo`. But now the caller can feel safe in the knowledge that when it asks for an interface, it's actually getting it and not something else that coincidentally has the same name.

Raymond Chen

Follow