How do I use C++/WinRT to implement a classic COM interface that derives from another classic COM interface?

devblogs.microsoft.com/oldnewthing/20200424-00

April 24, 2020

Raymond Chen

The C++/WinRT library can be used to implement both Windows Runtime interfaces as well as classic COM interfaces. One feature of classic COM that is absent (intentionally) from the Windows Runtime is interface derivation. If you're writing a class that needs to implement a derived COM interface, how do you express it? (The WRL library calls this a "chained interface".)

For concreteness, let's suppose that you are implementing `IFileSystemBindData` and `IFileSystemBindData2`.

The naïve way is to say that you implement both interfaces:

```
struct MyFileSystemBindData :
    implements<MyFileSystemBindData,
        IFileSystemBindData,
        IFileSystemBindData2>
{
    // IFileSystemBindData
    HRESULT SetFindData(const WIN32_FIND_DATAW* pfd) override;
    HRESULT GetFindData(WIN32_FIND_DATAW* pfd) override;

    // IFileSystemBindData2
    HRESULT SetFileID(LARGE_INTEGER liFileID) override;
    HRESULT GetFileID(LARGE_INTEGER *pliFileID) override;
    HRESULT SetJunctionCLSID(REFCLSID clsid) override;
    HRESULT GetJunctionCLSID(CLSID *pclsid) override;
};
```

If you do this, you get ambiguous cast errors because the `QueryInterface` provided by the `implements` template ends up doing something like this:

```
if (is_guid_of<IFileSystemBindData>(iid)) {
  *result = static_cast<IFileSystemBindData*>(this);
} else if (is_guid_of<IFileSystemBindData2>(iid)) {
  *result = static_cast<IFileSystemBindData2*>(this);
}
```

The cast to `IFileSystemBindData*` is ambiguous because the compiler can't tell whether you want the `IFileSystemBindData` that is the immediate base class, or whether you want the `IFileSystemBindData` that is the base class of the `IFileSystemBindData2` interface.

But you didn't need to do that anyway. The COM interfaces derive from each other, so you probably want them to share a vtable. Declaring that you implement both interfaces means that you get two vtables (one for each interface) rather than a shared vtable.

The way to define your object is to say that you implement only the derived interface:

```
struct MyFileSystemBindData :
    implements<MyFileSystemBindData,
        IFileSystemBindData2>
{
    ...
};
```

This gets rid of the ambiguous cast, because there is now only one way to get a `IFileSystemBindData`.

However, you also need to get the `QueryInterface` to respond to `IID_IFileSystemBindData`.

To do that, you can overload the `winrt::is_guid_of` function so that a check for `IFileSystemBindData2` includes a test for `IFileSystemBindData`.

```
namespace winrt
{
  template<>
  bool is_guid_of<IFileSystemBindData2>(guid const& id) noexcept
  {
    return is_guid_of<IFileSystemBindData2, IFileSystemBindData>(id);
  }
}
```

This takes advantage of the variadic template overload of `is_guid_of` introduced in PR 107.

Raymond Chen

Follow