The macros for declaring COM interfaces, revisited: C++ version

devblogs.microsoft.com/oldnewthing/20200910-00

September 10, 2020

Raymond Chen

Last time, we looked at the macros for declaring COM interfaces and how they expand when compiled for C.

When compiled as C++, the macros do something entirely different.

```
/* DECLARE_INTERFACE_IID_(ISample2, ISample, "...") */
struct __declspec(uuid("5675B786-7BAC-4EA2-A020-F4E7A15E2073"))
       __declspec(novtable)
       ISample2 : public ISample
{
    /* BEGIN_INTERFACE */
    virtual void a() {} // only on PowerPC

    // *** IUnknown methods ***
    /* STDMETHOD(QueryInterface)(THIS_ REFIID riid, void **ppv) PURE; */
    virtual __declspec(nothrow) HRESULT __stdcall
        QueryInterface(REFIID riid, void** ppv) = 0;

    /* STDMETHOD_(ULONG,AddRef)(THIS) PURE; */
    virtual __declspec(nothrow) ULONG __stdcall AddRef(void) = 0;

    /* STDMETHOD_(ULONG,Release)(THIS) PURE; */
    virtual __declspec(nothrow) ULONG __stdcall Release(void) = 0;

    // ** ISample methods ***
    /* STDMETHOD(Method1)(THIS) PURE; */
    virtual __declspec(nothrow) HRESULT __stdcall Method1(void) = 0;

    /* STDMETHOD_(int, Method2)(THIS) PURE; */
    virtual __declspec(nothrow) HRESULT __stdcall Method2(void) = 0;

    // *** ISample2 methods ***
    /* STDMETHOD(Method3)(THIS_ int iParameter) PURE; */
    virtual __declspec(nothrow) HRESULT __stdcall Method3(int iParameter) = 0;

    /* STDMETHOD_(int, Method4)(THIS_ int iParameter) PURE; */
    virtual __declspec(nothrow) int __stdcall Method4(int iParameter) = 0;

    /* END_INTERFACE */
};
```

The `DECLARE_ INTERFACE` macros declare a structure that consists solely of pure virtual methods. If you use the `DECLARE_ INTERFACE_` version, you can specify a base interface. You will pretty much always use this two-parameter version, since you need to derive from `IUnknown` if nothing else.

The `__declspec(uuid(...))` specifier enables the use of `__uuidof` to auto-generate a GUID when you write `__uuidof(ISample2)`. This is very handy for macros like `IID_PPV_ARGS` which automatically pass the interface GUID that corresponds to the macro parameter, thereby avoiding errors due to mismatches.

Normally, C++ objects change identity during construction and destruction, which means that constructing the interface object involves setting up a vtable filled with `__purecall` entries, only to have that vtable be immediately overwritten when the derived class is

constructed. Similarly, at destruction, the vtable regresses from the derived class's vtable to the `__purecall` vtable when destruction reaches the interface object.

The `__declspec(novtable)` specifier tells the compiler not to bother setting up the vtable for this class during construction and destruction, because the class promises not to call any of its own virtual methods during constructor or destruction. (Vacuously true for interfaces because they have trivial constructors and destructors.) The `novtable` specifier avoids the code needed to set up the vtables as well as not needing to produce a vtable in the first place.

Related: The sad history of Visual Studio's custom `__if_exists` keyword.

As we learned last time, the `BEGIN_INTERFACE` macro usually does nothing, but on PowerPC, it generates an extra dummy entry in the vtable for reasons lost to history.

The `STDMETHOD` macro generates the method declaration. The method is `virtual`, as you would expect. It also is marked `__declspec(nothrow)`, which is a promise that calling the method will not throw an exception. There is no enforcement of this promise; if you break the rules and allow an exception to escape, then the behavior is undefined. COM methods are not allowed to throw exceptions, so this annotation is accurate, assuming everybody plays by the rules.

Related: The sad history of the C++ `throw(...)` exception specifier.

The `PURE` expands to `= 0` for C++, which makes it a pure virtual method.

Related: COM interfaces do not implement their own pure virtual methods, even though the language permits it.

The rest is fairly straightforward. The `THIS` and `THIS_` macros expand to nothing; they exist to keep C happy.

Every macro in this entire sequence does something, either in C or C++. Well, with the exception of `END_INTERFACE`, which nobody has yet to find a use for. But it's there just in case.[1]

Next time, we'll look at the implementation macros.

[1] For example, it might be used to declare an explicitly nonvirtual destructor, should the C++ language someday decide to make destructors virtual by default in polymorphic classes.

Raymond Chen

Follow