The macros for declaring and implementing COM interfaces

devblogs.microsoft.com/oldnewthing/20041005-00

October 5, 2004

Raymond Chen

There are two ways of declaring COM interfaces, the hard way and the easy way.

The easy way is to use an IDL file and let the MIDL compiler generate your COM interface for you. If you let MIDL do the work, then you also get __uuidof support at no extra charge, which is a very nice bonus.

The hard way is to do it all by hand. If you choose this route, your interface will look something like this:

```
#undef  INTERFACE
#define INTERFACE   ISample2


DECLARE_INTERFACE_(ISample2, ISample)
{
    BEGIN_INTERFACE


    // *** IUnknown methods ***
    STDMETHOD(QueryInterface)(THIS_ REFIID riid, void **ppv) PURE;
    STDMETHOD_(ULONG,AddRef)(THIS) PURE;
    STDMETHOD_(ULONG,Release)(THIS) PURE;


    // ** ISample methods ***
    STDMETHOD(Method1)(THIS) PURE;
    STDMETHOD_(int, Method2)(THIS) PURE;


    // *** ISample2 methods ***
    STDMETHOD(Method3)(THIS_ int iParameter) PURE;
    STDMETHOD_(int, Method4)(THIS_ int iParameter) PURE;


    END_INTERFACE
};
```

What are the rules?

- You must set the `INTERFACE` macro to the name of the interface being declared. Note that you need to `#undef` any previous value before you `#define` the new one.
- You must use the `DECLARE_INTERFACE` and `DECLARE_INTERFACE_` macros to generate the preliminary bookkeeping for an interface. Use `DECLARE_INTERFACE` for interfaces that have no base class and `DECLARE_INTERFACE_` for interfaces that derive from some other interface. In our example, we derive the `ISample2` interface from `ISample`. **Note**: In practice, you will never find the plain `DECLARE_INTERFACE` macro because all interfaces derive from IUnknown if nothing else.
- You must list all the methods of the base interfaces in exactly the same order that they are listed by that base interface; the methods that you are adding in the new interface must go last.
- You must use the `STDMETHOD` or `STDMETHOD_` macros to declare the methods. Use `STDMETHOD` if the return value is `HRESULT` and `STDMETHOD_` if the return value is some other type.
- If your method has no parameters, then the argument list must be `(THIS)`. Otherwise, you must insert `THIS_` immediately after the open-parenthesis of the parameter list.
- After the parameter list and before the semicolon, you must say `PURE`.
- Inside the curly braces, you must say `BEGIN_INTERFACE` and `END_INTERFACE`.

There is a reason for each of these rules. They have to do with being able to use the same header for both C and C++ declarations and with interoperability with different compilers and platforms.

- You must set the `INTERFACE` macro because its value is used by the `THIS` and `THIS_` macros later.
- You must use one of the `DECLARE_INTERFACE*` macros to ensure that the correct prologue is emitted for both C and C++. For C, a vtable structure is declared, whereas for C++ the compiler handles the vtable automatically; on the other hand, since C++ has inheritance, the macros need to specify the base class so that upcasting will work.
- You must list the base class methods in exactly the same order as in the original declarations so that the C vtable structure for your derived class matches the structure for the base class for the extent that they overlap. This is required to preserve the COM rule that a derived interface can be used as a base interface.
- You must use the `STDMETHOD` and `STDMETHOD_` macros to ensure that the correct calling conventions are declared for the function prototypes. For C, the macro creates a function pointer in the vtable; for C++, the macro creates a virtual function.
- The `THIS` and `THIS_` macros are used so that the C declaration explicitly declares the “this” parameter which in C++ is implied. Different versions are needed depending on the number of parameters so that a spurious trailing comma is not generated in the zero-parameter case.

- The word `PURE` ensures that the C++ virtual function is pure, because one of the defining characteristics of COM interfaces is that all methods are pure virtual.
- The `BEGIN_INTERFACE` and `END_INTERFACE` macros emit compiler-specific goo which the compiler vendor provides in order to ensure that the generated interface matches the COM vtable layout rules. Different compilers have historically required different goo, though the need for goo is gradually disappearing over time.

And you wonder why I called it “the hard way”.

Similar rules apply when you are implementing an interface. Use the `STDMETHODIMP` and `STDMETHODIMP_` macros to declare your implementations so that they get the proper calling convention attached to them. We’ll see examples of this next time.

Raymond Chen

Follow