The macros for declaring COM interfaces, revisited: C version

devblogs.microsoft.com/oldnewthing/20200909-00

September 9, 2020

Raymond Chen

Quite some time ago, I covered The macros for declaring and implementing COM interfaces. I spelled out the rules, but I didn't go into much detail as to how the rules lead to the desired results.

There have also been some changes to the rules in the intervening years, so this mini-series will be a refresh of the old rules.

```
#undef  INTERFACE
#define INTERFACE   ISample2

DECLARE_INTERFACE_IID_(ISample2, ISample,
                       "5675B786-7BAC-4EA2-A020-F4E7A15E2073")
{
    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
};
```

When this code is compiled by a C compiler, the macros expand as follows:

```
/* DECLARE_INTERFACE_IID_(ISample2, ISample, "...") */
typedef struct ISample2
{
    struct ISample2Vtbl* lpVtbl;
} ISample2;

typedef struct ISample2Vtbl ISample2Vtbl;

struct ISample2Vtbl
{
    /* BEGIN_INTERFACE */
    void* b; /* only on PowerPC */

    /*** IUnknown methods ***/
    /* STDMETHOD(QueryInterface)(THIS_ REFIID riid, void **ppv) PURE; */
    HRESULT (__stdcall* QueryInterface)(ISample2* This, REFIID riid, void** ppv);

    /* STDMETHOD_(ULONG,AddRef)(THIS) PURE; */
    ULONG (__stdcall* AddRef)(ISample2* This);

    /* STDMETHOD_(ULONG,Release)(THIS) PURE; */
    ULONG (__stdcall* Release)(ISample2* This);

    /*** ISample methods ***/
    /* STDMETHOD(Method1)(THIS) PURE; */
    HRESULT (__stdcall* Method1)(ISample2* This);

    /* STDMETHOD_(int, Method2)(THIS) PURE; */
    int (__stdcall* Method2(ISample2* This);

    /*** ISample2 methods ***/
    /* STDMETHOD(Method3)(THIS_ int iParameter) PURE; */
    HRESULT (__stdcall* Method3)(int iParameter);

    /* STDMETHOD_(int, Method4)(THIS_ int iParameter) PURE; */
    int (__stdcall* Method4)(int iParameter);

    /* END_INTERFACE */
};
```

When compiled as C, the interface is formally defined as a structure consisting only of a vtable. The vtable is a sequence of function pointers, one for each virtual method.

The parameters to the `DECLARE_ INTERFACE_ IID_` macro are the interface being declared, its base interface, and the UUID for the interface. There's also a version without the trailing underscore: `DECLARE_ INTERFACE_ IID`. That version is for the case where there is no base interface. In practice, you will never use that version because every interface derives from `IUnknown`. Basically, the only interface that would use the no-base-interface version is `IUnknown` itself.

There's also a no-IID version of the macro: `DECLARE_ INTERFACE_` (and `DECLARE_ INTERFACE` for the one interface with no base interface). If you use this version, then you won't be able to say `__uuidof(ISample2)` in the C++ expansion to obtain the IID of the interface. We'll see more about this when we dig into the C++ expansion.

The `BEGIN_ INTERFACE` macro does nothing on most systems, but on PowerPC, it generates a mysterious `void*` at the start of the vtable. I don't know why it's there, but apparently that was part of the PowerPC ABI for vtables. A common mistake is forgetting the `BEGIN_ INTERFACE` and `END_ INTERFACE` macros. You don't notice until somebody tries to use your header file on a PowerPC.

Notice that all the methods of the base classes need to be redeclared in the derived class, so that the vtable is laid out properly. A common mistake is to omit the base interface methods, and you get away with it when compiling as C++ because the base interface methods are inherited. But C doesn't have inheritance. You have to write it out.

The `STDMETHOD` and `STDMETHOD_` macros generate a function pointer structure member, corresponding to a C++ virtual method, using the calling convention for COM methods, which happens to be `__stdcall`. The `STDMETHOD` macro is for methods returning `HRESULT`, and the `STDMETHOD_` macro is for methods that return something else.

The `PURE` macro expands to nothing. It's used by the C++ expansion, which we'll cover later.

The `THIS` and `THIS_` macros expand to the `This` parameter declaration, corresponding to the hidden `this` parameter in C++. For methods with no parameters, use `THIS` as the single parameter. For methods that have parameters, use `THIS_` before the first parameter.

The `END_ INTERFACE` macro expands to nothing. There hasn't yet been an architecture that required special treatment of the end of the vtable. But the macro is there in case some future architecture ends up needing something to go there.

Next time, we'll look at how these macros expand in C++.

Raymond Chen

Follow