What are the duck-typing requirements of MFC IPTR?

devblogs.microsoft.com/oldnewthing/20230509-00

May 9, 2023

Raymond Chen

We continue our survey of duck-typing requirements of various C++ COM smart pointer libraries by looking at MFC's `IPTR`, running it through our standard tests.

```
// Dummy implementations of AddRef and Release for
// testing purposes only. In real code, they would
// manage the object reference count.
struct Test
{
    void AddRef() {}
    void Release() {}
    Test* AddressOf() { return this; }
};

struct Other
{
    void AddRef() {}
    void Release() {}
};

// Pull in the smart pointer library
// (this changes based on library)
#include <afxole.h>

extern const GUID IID_Test;
extern const GUID IID_Other;
using TestPtr = IPTR(Test);
using OtherPtr = IPTR(Other)

void test()
{
    Test test;

    // Default construction
    TestPtr ptr;

    // Construction from raw pointer
    TestPtr ptr2(&test);

    // Copy construction
    TestPtr ptr3(ptr2);

    // Attaching and detaching
    auto p = ptr3.GetInterfacePtr();
    ptr3.Detach();
    ptr.Attach(p);

    // Assignment from same-type raw pointer
    ptr3 = &test;

    // Assignment from same-type smart pointer
    ptr3 = ptr;

    // Accessing the wrapped object
    // (this changes based on library)
    if (ptr.GetInterfacePtr() != &test) {
        std::terminate(); // oops
    }
    if (ptr->AddressOf() != &test) {
        std::terminate(); // oops
    }

    // Returning to empty state
    ptr3.Release(); // requires ptr3 be non-empty
    ptr3 = static_cast<Test*>(nullptr);
    ptr3.Attach(nullptr);

    // Receiving a new pointer
    // (this changes based on library)
    Test** out = &ptr3;

    // Bonus: Comparison.
    if (ptr == ptr2) {}
    if (ptr != ptr2) {}
    if (ptr < ptr2) {}

    // Litmus test: Accidentally bypassing the wrapper
    ptr->AddRef();
    ptr->Release();

    // Litmus test: Construction from other-type raw pointer
    Other other;
    TestPtr ptr4(&other);

    // Litmus test: Construction from other-type smart pointer
    OtherPtr optr;
    TestPtr ptr5(optr);

    // Litmus test: Assignment from other-type raw pointer
    ptr = &other;

    // Litmus test: Assignment from other-type smart pointer
    ptr = optr;

    // Destruction
}
```

The `IPTR` macro assumes that the `IID` that corresponds to the alleged interface is named `IID_Blah`. We give it a dummy ID interface in the form of a reference to a variable that is never defined. This ensures a linker error if the code ever tries to use that fake interface ID.

The `IPTR.Detach` method does not return the detached pointer. You have to fetch it yourself via `GetInterfacePtr()` if you want to save it.

Returning the smart pointer to an empty state can be done in multiple ways.

- The `Release()` method releases the wrapped pointer, but there must be a non-null wrapped pointer in the first place. It is an error to call `ptr3.Release()` on an empty `ptr3`.
- You can use the assignment operator to assign a `nullptr`. However, there are multiple assignment operators,[1] so you have to cast the `nullptr` to `Test*` to say, “I am assigning a new pointer to be wrapped.”
- A sneaky way to avoid the extra typing is to use the `Attach()` method to attach a null pointer. The first parameter to `Attach()` is always a `Test*`, so there is no ambiguity in passing just `nullptr`.

The only way to receive a new pointer is to use the `&` operator. This operator releases any previous wrapped pointer before receiving the new one.

The `IPTR` fails the bonus comparison test, but not through any fault of the `Test` class. The `IPTR` simply doesn’t support comparison at all, not even for COM interfaces.

The `IPTR` macro fails the “accidental bypass” litmus test, in the same way that `_com_ptr_t` did. As with `_com_ptr_t`, there are two ways of releasing the object that are dangerously similar:

```
ptr2.Release(); // good
ptr2->Release(); // bad
```

The same cautions apply.

The `IPTR` passes the other litmus tests: All of the attempts to convert or assign from another type of smart pointer or raw pointer generate errors.

Okay, so here’s the scorecard for `IPTR`.

MFC `IPTR` scorecard	
Default construction	Pass
Construct from raw pointer	Pass
Copy construction	Pass
Destruction	Pass
Attach and detach	Pass
Assign to same-type raw pointer	Pass
Assign to same-type smart pointer	Pass

Fetch the wrapped pointer	`GetInterfacePtr()`
Access the wrapped object	`->`
Receive pointer via `&`	release old
Release and receive pointer	`&`
Preserve and receive pointer	N/A
Return to empty state	Pass
Comparison	Not supported
Accidental bypass	Fail
Construct from other-type raw pointer	Pass
Construct from other-type smart pointer	Pass
Assign from other-type raw pointer	Pass
Assign from other-type smart pointer	Pass

Next time, we’ll look at ATL’s `CComPtr`.

[1] The other assignment operators accept `CLSID` and `PCWSTR` and create a new object identifed by the `CLSID` or ProgId. Fortunately, it’s unlikely that you will assign one of these things to a a COM smart pointer by mistake.