How do I convert from the C++/WinRT implementation type to the C++/WinRT projection type?

devblogs.microsoft.com/oldnewthing/20200827-00

August 27, 2020

Raymond Chen

Working with C++/WinRT projections is relatively straightforward. Everything is a smart pointer type, and you invoke methods on them in the usual way, and the only methods you can invoke are the ones in the interface definition.

For concreteness, let's suppose you have this class interface definition:

```
namespace Sample
{
    runtimeclass Class
    {
        Class();
        void DoSomething();
    };
};
```

And you implemented it like this:

```
namespace winrt::Sample::implementation
{
    struct Class : ClassT<Class>
    {
        Class();
        void DoSomething();

        void ImplementationMethod();
        int implementation_value = 42;
    };
}
```

Working from the projection side, here's what you can and cannot do:

```
winrt::Sample::Class c;   // can construct it
c.DoSomething();          // can invoke interface method
c.ImplementationMethod(); // not allowed
c.implementation_value++; // not allowed
```

On the other hand, working with C++/WinRT implementations is a bit gnarlier, because you are now straddling two worlds: You have the simple projection world, and you have the ugly implementation world.

For convenience, I'm going to omit the `winrt::` namespace qualifier from everything.

First, we have the object creation patterns:

```
// Create an object from the implementation,
// returning the projection.
Sample::Class c =
    winrt::make<implementation::Class>();

// Create an object from the implementation,
// returning a com_ptr to the implementation.
com_ptr<implementation::Class> c =
    winrt::make_self<implementation::Class>();
```

And then we have the object conversion patterns:

```
// Go from the implementation to the projection.
implementation::Class* p;
Sample::Class c = *p;

// Go from the implementation's com_ptr to the projection.
com_ptr<implementation::Class> ptr;
Sample::Class c = *ptr;
```

The implementation class contains conversions to its projected runtime class as well as all of its interfaces. So you just need to dereference the raw pointer or `com_ptr`, and then let the conversion kick in.

There's a gotcha here: You have to make sure not to copy the object by mistake!

```
implementation::Class* p;
auto o = *p; // bad idea
if (case1) {
  Sample::Class c = o;
} else {
  IInspectable i = o;
}
```

This code wants to factor out the dereference operator and then perform the conversion later, depending on what case it is in. The problem is that `o` is going to be a *copy* of the object, which is going to cause all sorts of problems, because the object is managed by its internal reference count, and copying it into a local is going to cause all sorts of problems: The copy will get a copy of the reference count, which is now incorrect. And the copy is going to destruct when it goes out of scope, regardless of its actual reference count. Fortunately, the debug build will catch this problem with the error

```
error C2259: 'winrt::Sample::implementation::Class': cannot instantiate abstract
class due to following members:
'void winrt::impl::root_implements<D,winrt::Sample::Class>::use_make_function_to_
create_this_object(void)': is abstract
```

Next time, we'll look at the harder job of getting from the projection back to the implementation.

Raymond Chen

Follow