Mind your C++/WinRT namespaces

devblogs.microsoft.com/oldnewthing/20230316-00

March 16, 2023

Raymond Chen

When you implement a Windows Runtime class in C++/WinRT, each class name appears in three different namespace, so you need to mind your namespaces.

Projection	`winrt::Namespace::ClassName`
Factory	`winrt::Namespace::factory_implementation::ClassName`
Implementation	`winrt::Namespace::implementation::ClassName`

If you write an unqualified `ClassName`, the compiler searches for the name in the current namespace, then the parent namespace, then the grandparent namespace, and so on until it reaches the global namespace.

Now, when you say `ClassName`, there's a decent chance that you intend to refer to the name in the projection namespace, particularly if you copy/pasta'd some code from a tutorial or from another project. What you didn't realize is that the code you copied was intended to be compiled outside the implementation namespaces. But if you happen to be in one of the implementation namespaces, you will pick up the name in that other namespace by mistake.

```
namespace winrt::MyNamespace::implementation
{
    struct ClassName : ClassNameT<ClassName>
    {
        ClassName CreateChild()
        { return make<ClassName>(...); }

        Windows::Foundation::IAsyncOperation<ClassName>
            CreateChildAsync()
        {
            /* do stuff */
            co_return make<ClassName>(...);
        }
    };
}

namespace winrt::MyNamespace::factory_implementation
{
    struct ClassName : ClassNameT<ClassName, implementation::ClassName>
    {
        static ClassName Create()
        { return make<ClassName>(); }
    };
}
```

The above code wants the `CreateChild()` method to return a Windows Runtime `ClassName` object, but since the name `ClassName` is being used inside the `winrt::Namespace::implementation` namespace, it actually refers to the `winrt::Namespace::implementation::ClassName` implementation type, not the projection type.

Similarly, the `Create()` method on the factory class intends to return a Windows Runtime `ClassName` object, but instead it returns a `winrt::Namespace::factory_implementation::ClassName` object.

The result of this incorrect name lookup is usually a pair of really confusing error messages.

You get one error message when the compiler realizes that the `return make<ClassName>(...)` is trying to return a projected type, but the declared return type is one of the implementation types, so you are scolded that there is no conversion from the projection type to the implementation type.

You get a second error message when the compiler instantiates the `ClassNameT` template, which uses the Curiously Recurring Template Pattern (commonly known as CRTP). The `ClassNameT` template expects the `CreateChild()` and `Create()` methods to return the projected type, but their declared return type is an implementation type, and you get scolded a second time because there is no conversion from the implementation type to the projection type.

The `CreateChildAsync()` method is even worse. In this case, we accidentally said that it returns an `IAsyncOperation<implementation::ClassName>`. This mistake is also rewarded with not just two but *three* confusing error messages, which could be reported in any order.

As before, there is a problem from the CRTP code that the declared return type doesn't match what the CRTP code expects.

And analogously, you get an error at the `co_return` because the coroutine expects you to `co_return` the implementation type (since that's the accidental template type parameter to `IAsyncOperation`), but you `co_return`ed the projection type. This error message is a little confusing because it is typically reported as a problem with the promise's `return_value` method, since the argument to `co_return` gets passed to the promise's `return_value` method.

The third mysterious error message comes from `IAsyncOperation` because one of the requirements is that the template type parameter (the thing produced by the `IAsyncOperation`) must be a Windows Runtime type, and the Windows Runtime type is your projection type, not the implementation types.

Okay, so we learned that using an unqualified type name from inside the implementation or factory implementation namespace gives you the corresponding implementation type, not the projection type. But what if you want the implementation type?

In theory, you could type the full name `winrt::Namespace::ClassName`, but really, all you have to say is `Namespace::ClassName`. The lookup proceeds through the parent namespaces, and it finds a match when it gets to `winrt`.

This shortcut is particularly handy when you have a deep namespace. Instead of the full name `winrt::Grandparent::Parent::Namespace::ClassName`, you can write just `Namespace::ClassName`.

Bonus chatter: All this confusion stems from the fact that we used the same name in three namespaces. We could have avoided this by using different names for our two implementation classes, so that they don't collide with the projected class name, or with each other.

```
namespace winrt::MyNamespace::implementation
{
    struct ClassNameImpl : ClassNameT<ClassNameImpl>
    {
        ClassName CreateChild()
        { return make<ClassNameImpl>(...); }

        Windows::Foundation::IAsyncOperation<ClassName>
            CreateChildAsync()
        {
            /* do stuff */
            co_return make<ClassNameImpl>(...);
        }
    };
}

namespace winrt::MyNamespace::factory_implementation
{
    struct ClassNameFact : ClassNameT<ClassNameFact, implementation::ClassNameImpl>
    {
        static ClassName Create()
        { return make<ClassNameFact>(); }
    };
}
```

This way, `ClassName` always refers to the projection class.