Why am I getting an unresolved external from C++/WinRT if it is a header-only C++ library?

devblogs.microsoft.com/oldnewthing/20210927-00

September 27, 2021

Raymond Chen

A customer was getting unresolved external errors from their C++/WinRT code. How is that possible? C++/WinRT is a header-only library; there is no static library that needs to be linked in. I mean, the way to resolve an unresolved external from a header is to link in the associated static or dynamic library, but for header-only libraries, there is no associated *anything* to link in. So how can something be unresolved?

C++/WinRT is not just one header, but a series of headers. To reduce compile times, the C++/WinRT headers for a particular component are split into multiple layers of headers. The lowest layer provides forward declarations for types, but almost no definitions. As you go higher up the layers, you get a little more: Maybe a class gets a definition, but the methods don't get implemented until a still higher layer.

When you include a C++/WinRT header file, it includes declarations, definitions, and implementations for everything for which that header file is responsible, as well as its parent namespaces, but no more.

Consider:

```
namespace Contoso.Widgets
{
    runtimeclass Widget
    {
        Widget();
        Contoso.Gadgets.Gadget GetGadget();
        void Reset();
    }
}
```

If you include `winrt/Contoso.Widgets.h`, you get everything you need to create a `Widget` and call its methods. However, you *don't* get everything you need to operate on `Gadget` objects. You do have enough to call the `GetGadget()` method, but you can't do anything with the gadget until you include `winrt/Contoso.Gadgets.h`.

Normally, when you make this mistake, you get a compile-time error thanks to a trick: Declare the methods as `auto`. An `auto` method must be defined before it can be called, and that's the language feature that the trick relies on.[1]

However, this trick doesn't work for constructors and destructors, since they don't have return values that can be `auto`-ized. If you use a constructor for a type without having included the corresponding namespace header file, the code will compile, and you'll get a linker error later.[2]

Sorry.

[1] The methods are, however, defined with a return value in the normal way if `__INTELLISENSE__` is defined, so that you get more useful feedback from IntelliSense.

[2] Well, if you're lucky and some other translation unit included the namespace header file and used the constructor, then you will get a definition, but that's only because you were relying on the kindness of strangers.

Raymond Chen

Follow