Why does my C++/WinRT project get errors of the form “unresolved external symbol … consume_Something”?

devblogs.microsoft.com/oldnewthing/20190529-00

May 29, 2019

Raymond Chen

You set up a new C++/WinRT project and build it, and everything looks great.

```
#include <winrt/Windows.Gaming.Input.h>

void CheckGamepads()
{
    auto gamepads =
        winrt::Windows::Gaming::Input::Gamepad::Gamepads();
    for (auto&& gamepad : gamepads)
    {
        check(gamepad);
    }
}
```

The code builds just fine except that you get a linker error that makes no sense. (Let’s face it, most linker errors make no sense until you put on your linker-colored glasses.)

```
error LNK2019: unresolved external symbol "public: struct winrt::Windows::
Foundation::Collections::IIterator<struct winrt::Windows::Gaming::Input::Gamepad>
__thiscall winrt::impl::consume_Windows_Foundation_Collections_IIterable<struct
winrt::Windows::Foundation::Collections::IIterable<struct winrt::Windows::Gaming::
Input::Gamepad>,struct winrt::Windows::Gaming::Input::Gamepad>::First(void)const " (?
First@?$consume_Windows_Foundation_Collections_IIterable@U?$IIterable@W4Gamepad@
Gaming@Input@Windows@winrt@@@Collections@Foundation@Windows@winrt@@W4Gamepad@Gaming@
Input@45@@impl@winrt@@QBE?AU?$IIterator@W4Gamepad@Gaming@Input@Windows@winrt@@@
Collections@Foundation@Windows@3@XZ) referenced in function "struct winrt::Windows::
Foundation::Collections::IIterator<struct winrt::Windows::Gaming::Input::Gamepad>
__stdcall winrt::impl::begin<struct winrt::Windows::Foundation::Collections::
IIterable<struct winrt::Windows::Gaming::Input::Gamepad>,0>(struct winrt::Windows::
Foundation::Collections::IIterable<struct winrt::Windows::Gaming::Input::Gamepad>
const &)" (??$begin@U?$IIterable@W4Gamepad@Gaming@Input@Windows@winrt@@@Collections@
Foundation@Windows@winrt@@$0A@@impl@winrt@@YG?AU?$IIterator@W4Gamepad@Gaming@Input@
Windows@winrt@@@Collections@Foundation@Windows@1@ABU?$IIterable@W4Gamepad@Gaming@
Input@Windows@winrt@@@3451@@Z)
```

What the heck is going on here?

Take away all the decorations and it boils down to this:

```
unresolved external symbol "winrt::impl::consume_...IIterable<...>::First()"
referenced in function "begin(winrt::IIterable<...> const&)."
```

The linker couldn't find a definition for the `First` method.

The answer from the linker's point of view is obvious: You called this `consume_BlahBlah` method but never defined it.

Yeah, so tell me something I don't know.

Each C++/WinRT header file contains the information needed to call methods on the classes in that namespace. In our case, we included `Windows.Gaming.Input.h`, which tells us how to call methods on `winrt:: Windows:: Gaming:: Input::Gamepad` objects. That made it possible to call `Gamepad:: Gamepads()`. The resulting `gamepads` variable is a `winrt:: Windows:: Foundation:: Collections:: IVectorView<Gamepad>`. We then use a ranged `for` statement to enumerate them, and that means that we're calling methods on the `gamepads` object, which means that we're calling methods on a `winrt:: Windows:: Foundation:: Collections:: IVectorView` object.

Ah, but we never told the compiler how to call the methods of `IVectorView`. The `Windows.Gaming.Input.h` header file included only the information to allow the methods of `Gamepad` to be called. "Okay, I got you all set up for `Gamepad`." Any types required from other interfaces were left as forward declarations. "If you need them, you can get the definitions yourself."[1]

We used those forward declarations without ever defining them, hence the linker error.

The solution is to include the required header file for the namespace.

```
#include <winrt/Windows.Foundation.Collections.h>
```

This is one of those rookie mistakes that make you scratch your head the first time you encounter it. The need to include the header file is mentioned in a big green box in the documentation, but that's not much consolation after you lost a few hours trying to figure it out.

There's some good news and bad news about this error message.

The good news is that this error message is going away. The bad news is that it's being replaced with a different error message. (But hopefully the new one is easier to understand.) More details next time.

¹ The idea is that you pay only for the namespaces you use. If every header file included its transitive closure of dependencies, (1) you would create circular dependencies, and (2) including a single header file would end up including all the other header files when you chased through all the dependencies.

The idea of "pay for play" is not unique to C++/WinRT. The C++ standard library follows the same principle. If you want `std::string`, you need to `#include <string>`. If you include a header file that has a method that takes a string, you will end up with only enough information to call that method. It doesn't mean that you get all of `<string>` automatically.

Raymond Chen

Follow