Why does my C++/WinRT project get errors of the form “abi<…>::… is abstract see reference to producer<…>“?

devblogs.microsoft.com/oldnewthing/20200206-00

February 6, 2020

Raymond Chen

Let’s say you want to implement an interface in your C++/WinRT class.

```
namespace winrt
{
  using namespace ::winrt::Windows::Foundation;
  using namespace ::winrt::Windows::Web::Http;
  using namespace ::winrt::Windows::Web::Http::Filters;
}

struct MyFilter :
  public winrt::implements<MyFilter,
                           winrt::IHttpFilter>
{
  MyFilter();
  winrt::IAsyncOperationWithProgress<
    winrt::HttpResponseMessage, winrt::HttpProgress>
  SendRequestAsync(winrt::HttpRequestMessage request);
};
```

Looks great.

And you get this horrible compiler error, which I’ve reformatted to make it slightly less horrible:

```
error C2259: 'winrt::impl::produce<D,I>': cannot instantiate abstract class
    with
    [
        D=MyFilter,
        I=winrt::Windows::Web::Http::Filters::IHttpFilter
    ]

due to following members:

  'int32_t winrt::impl::abi<
    winrt::Windows::Web::Http::Filters::IHttpFilter,
    void>::type::SendRequestAsync(void *,void **)
    noexcept': is abstract

see declaration of

  'winrt::impl::abi<
    winrt::Windows::Web::Http::Filters::IHttpFilter,
    void>::type::SendRequestAsync'

see reference to class template instantiation

  'winrt::impl::producer<
    D,winrt::Windows::Web::Http::Filters::IHttpFilter,
    void>' being compiled

    with
    [
        D=MyFilter
    ]

see reference to class template instantiation

  'winrt::impl::producer_convert<
    D,winrt::Windows::Web::Http::Filters::IHttpFilter,
    void>' being compiled

    with
    [
        D=MyFilter
    ]

see reference to class template instantiation

  'winrt::impl::producers_base<D,
    std::tuple<winrt::Windows::Web::Http::Filters::IHttpFilter>
    >' being compiled

    with
    [
        D=MyFilter
    ]
```

```
see reference to class template instantiation

  'winrt::implements<MyFilter,
    winrt::Windows::Web::Http::Filters::IHttpFilter>' being compiled
```

This seems to be saying that a method was not implemented, but it's some weird `int32_t abi<...>` thing that doesn't resemble anything you've seen in any documentation anywhere.

What's even more confusing is that one of your colleagues copied this exact same code into their project, and it compiled just fine.

The problem is that you forgot to include the header file for the namespace that contains the interface you are trying to implement. In this case, it means that you forgot to include the `winrt/Windows.Web.Http.Filters.h` header file.

That also explains why your colleague doesn't have the same problem: Your colleague included the header file.

What makes this error message particularly insidious is that you might say, "Well, I must have forgotten to implement a method," and try in vain to implement the missing methods and end up digging yourself into deeper and deeper holes.

Starting in C++/WinRT build 20190923, the error message has been made much less confusing:

```
error C2079: 'winrt::impl::producer<D
    winrt::Windows::Web::Http::Filters::IHttpFilter,
    void>::vtable'
    uses undefined struct 'winrt::impl::produce<D,I>'

    with
    [
        D=MyFilter
    ]
```

The new error message says that something is undefined, which is hopefully a better clue that you need to include the corresponding header file in order to get the definition.

This is one of the things about writing libraries that is often overlooked: Arranging things so that when the developer makes a mistake, the compiler's error message leads the developer in the right direction, or at least not in the *wrong* direction. This is hard, because you the library writer don't actually control the compiler's error messages. You just have to try to have things set up so that the error message the compiler generates is somehow indicative of how the library was incorrectly used.

It's indirect programming.

Bonus chatter: It is the job of the `produce` class to implement the ABI methods. In the original version, the `produce` class was defined as a template with a default definition:

```
template<typename D, typename I>
struct produce
{
};
```

This means that if you forget to include the header file that contains the required specialization, the compiler happily uses the default. But the default definition doesn't actually implement any of the ABI methods, resulting in a bunch of "cannot instantiate abstract class" errors.

The fix is to leave the template class declared but not defined:

```
template<typename D, typename I>
struct produce;
```

With this version, if somebody tries to produce an interface without having included the header file, there is no specialization, and there is no default definition either, so you get a compiler error that tells you that the structure isn't defined yet.

Raymond Chen

Follow