I get a weird error about no matching function when I try to use winrt::capture

devblogs.microsoft.com/oldnewthing/20200918-45

September 18, 2020

Raymond Chen

Last time, we looked at using the C++/WinRT `capture` to call COM ABI methods from C++/WinRT. But sometimes, you follow the cookbook to the letter, and it still fails:

```
winrt::com_ptr<IServiceProvider> provider = ...;

auto top = winrt::capture<IShellBrowser>
    (provider, &IServiceProvider::QueryService, SID_STopLevelBrowser);
```

Here, we are asking `capture` to invoke the `QueryService` method on the provided `IServiceProvider`, passing `SID_STopLevelBrowser` as the first parameter, and passing a `REFIID` + `void**` pair as the second and third parameters.

And yet, when you try it, you get these strange errors. From the Microsoft compiler:

```
error C2672: 'capture': no matching overloaded function found
error C2783: 'void capture(F,Args &&...)': could not deduce template argument for 'F'
```

Somehow, the Microsoft compiler thinks we're trying to use the functor overload.

From clang:

```
error: no matching function for call to 'capture'
    auto top = winrt::capture<IShellBrowser>
               ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~
note: candidate template ignored: couldn't infer template argument 'O'
    auto capture(com_ptr<O> const& object, M method, Args&&...args)
         ^
note: candidate template ignored: substitution failure [with T = IShellBrowser, F =
winrt::com_ptr<IServiceProvider>]: deduced incomplete pack <(no value), const GUID &>
for template parameter 'Args'
        auto capture(F function, Args&&...args)
             ^
```

The gcc compiler says

```
error: no matching function for call to 'capture<IShellBrowser>
(winrt::com_ptr<IServiceProvider>&, <unresolved overloaded function type>, const
GUID&)'
    (provider, &IServiceProvider::QueryService, SID_STopLevelBrowser);
                                                                      ^
note: candidate: 'auto winrt::capture(F, Args&& ...) [with T = IShellBrowser; F =
winrt::com_ptr<IServiceProvider>; Args = {}]'
    auto capture(F function, Args&&...args)
         ^~~~~~~
note:   candidate expects 1 argument, 3 provided

note: candidate: 'template<class T, class O, class M, class ... Args> auto
winrt::capture(const winrt::com_ptr<O>&, M, Args&& ...)'
    auto capture(com_ptr<O> const& object, M method, Args&&...args)
         ^~~~~~~
note:   template argument deduction/substitution failed:
note:   couldn't deduce template parameter 'M'
    (provider, &IServiceProvider::QueryService, SID_STopLevelBrowser);
```

What's going on?

The Microsoft compiler's error message is least helpful. It looks like the Microsoft compiler has selected the functor overload for some reason, but then it rejects the functor because it can't deduce the first parameter type `F`. But our first parameter is `provider`, which should be easily deduced to a `winrt::com_ptr<IServiceProvider>`.

It all makes no sense. And why is it not using the overload we want?

The clang error at least tried both overloads, but for the first overload it deduced `Args = {}`, which is strange because we passed a total of three arguments, so the last two should end up in the `Args`. It somehow just lost those arguments in the cushions of the couch or something.

The failed application of the second overload has an enigmatic phrase "deduced incompete pack ⟨(no value), `const GUID &`⟩ for template parameter '`Args`'". Somehow, it thinks that `&IServiceProvider::QueryService` has no value?

The gcc error steers us closer to the root cause when it mentions in its error message that the parameter list that it saw was

```
capture<IShellBrowser>(winrt::com_ptr<IServiceProvider>&, <unresolved overloaded
function type>, const GUID&)
```

Unresolved overloaded function type. That's strange. Let's take a closer look at `IServiceProvider`. Here it is, after removing a bunch of RPC and other macro noise:

```
MIDL_INTERFACE("6d5140c1-7436-11ce-8034-00aa006009fa")
IServiceProvider : public IUnknown
{
public:
    virtual HRESULT QueryService(
        REFGUID guidService,
        REFIID riid,
        void **ppvObject) = 0;

    template <class Q>
    HRESULT QueryService(REFGUID guidService, Q** pp)
    {
        return QueryService(guidService, __uuidof(Q), (void **)pp);
    }
};
```

Holy cow, somebody added a "helpful" overload of `IServiceProvider::QueryService` that takes only two parameters and manufactures the missing `REFIID` parameter.

That's why we are getting an error: The expression `&IServiceProvider::QueryService` is ambiguous because it could be referring to the three-parameter method, or it could be referring to an entire family of templated methods (which in turn need to be specialized).

Sometimes being helpful backfires.[1]

To be fair, this helpful method was added because the `IID_PPV_ARGS` helper macro hadn't yet been invented. Nowadays, people just write

```
serviceProvider->QueryService(guidService, IID_PPV_ARGS(&q));
```

and don't even realize that there's a helper method available to them.

One way to work around this unwanted helper is to resolve the ambiguity explicitly:

```
auto top = winrt::capture<IShellBrowser>
    (provider,
     static_cast<HRESULT (STDMETHODCALLTYPE IServiceProvider::*)(REFGUID, REFIID,
void**)>
         (&IServiceProvider::QueryService),
     SID_STopLevelBrowser);
```

This is, however, a horrible monstrosity of a cast.

One option is to use a helper lambda.

```
auto top = winrt::capture<IShellBrowser>(
     [](auto&& provider, auto&&... args) { return provider->QueryService(args...); },
     provider,
     SID_STopLevelBrowser);
```

If you're going to be doing this a lot, you can write a helper function:

```
inline HRESULT CapturableQueryService(
    IServiceProvider* provider,
    REFGUID service,
    REFIID riid,
    void**ppv)
{
    return provider->QueryService(service, riid, ppv);
}

auto top = winrt::capture<IShellBrowser>(CapturableQueryService, provider.get(),
SID_STopLevelBrowser);
```

In fact, there's a prewritten function that comes with the system that is basically this: `IUnknown_QueryService`. The `IUnknown_QueryService` function is more general, because it accepts an `IUnknown*` as its first parameter and will perform the `QueryInterface` to `IServiceProvider`.

Finally, in what might be the simplest option, you can give the result of that monster cast a name and let everybody use that name.

```
inline constexpr HRESULT (STDMETHODCALLTYPE
IServiceProvider::*ServiceProviderQueryServiceMethod)(REFGUID, REFIID, void**)
    = &IServiceProvider::QueryService;

auto top = winrt::capture<IShellBrowser>
    (provider, ServiceProviderQueryServiceMethod, SID_STopLevelBrowser);
```

[1] The `IUnknown` and `IAgileReference` interfaces have the same problem:

```
MIDL_INTERFACE("00000000-0000-0000-C000-000000000046")
IUnknown
{
public:
    virtual HRESULT QueryInterface(
                REFIID riid,
                void **ppvObject) = 0;
    virtual ULONG AddRef(void) = 0;
    virtual ULONG Release(void) = 0;

    template<class Q>
    HRESULT
    QueryInterface(Q** pp)
    {
        return QueryInterface(__uuidof(Q), (void **)pp);
    }
};

MIDL_INTERFACE("C03F6A43-65A4-9818-987E-E0B810D2A6F2")
IAgileReference : public IUnknown
{
public:
    virtual HRESULT Resolve(
                REFIID riid,
                void **ppvObjectReference) = 0;

    template<class Q>
    HRESULT Resolve(_COM_Outptr_ Q** pp)
    {
        return Resolve(__uuidof(Q), (void **)pp);
    }
};
```

However, you are unlikely to run into problems with these interfaces in C++/WinRT code, because C++/WinRT already provides native support for `IUnknown` (via `com_ptr`) and `IAgileReference` (via `agile_ref`), so you're not going to have to drop to the ABI to use them.

Raymond Chen

Follow