An example of using Windows Runtime interop methods from C++/WinRT: RequestTokenForWindowAsync

devblogs.microsoft.com/oldnewthing/20210805-00

August 5, 2021

Raymond Chen

A customer was trying to use the `IWebAuthenticationCoreManagerInterop::RequestTokenForWindowAsync` method from C++/WinRT. The `IWebAuthenticationCoreManagerInterop` interface follows the interop pattern and lets a Win32 program use the Windows Runtime `WebAuthenticationCoreManager` by associating it with a `HWND` instead of a `CoreWindow`.

A customer was having trouble getting this to work, though:

```
#include <WebAuthenticationCoreManagerInterop.h>
#include <winrt/Windows.Foundation.h>
#include <winrt/Windows.Security.Authentication.Web.Core.h>

namespace winrt
{
    using namespace Windows::Foundation;
    using namespace Windows::Security::Authentication::Web::Core;
}

// Note: Code in italics is wrong; see discussion
winrt::IAsyncOperation<winrt::WebTokenRequestResult>
RequestTokenForWindowAsync(HWND window, winrt::WebTokenRequest const& request)
{
    auto interop = winrt::get_activation_factory<winrt::WebAuthenticationCoreManager,

::IWebAuthenticationCoreManagerInterop>();

    winrt::com_ptr<winrt::IAsyncOperation<winrt::WebTokenRequestResult>> operation;
    auto requestInspectable = static_cast<::IInspectable*>(winrt::get_abi(request));

    winrt::check_hresult(
        interop->RequestTokenForWindowAsync(
        window,
        requestInspectable,
        __uuidof(operation),
        operation.put_void()));

    co_return co_await operation;
}
```

The `__uuid(operation)` fails to compile, producing the error

```
error C2787: 'winrt::com_ptr<winrt::Windows::Foundation::IAsyncOperation<winrt::
Windows::Security::Authentication::Web::Core::WebTokenRequestResult>>': no GUID has
been associated with this object
```

What's going on?

The first order of business is understanding the error message.

The `__uuidof` nonstandard extension keyword can be applied to a type or a variable. If you apply it to a variable, then it uses the type of that variable. If the type is a pointer or reference, the pointed-to or referenced type is used. And then the compiler checks if the resulting type has a `__declspec(uuid(...))` attribute.

What happened here is that we passed a variable whose type is `winrt::com_ptr<something>`, and `winrt::com_ptr` doesn't have a `__declspec(uuid(...))` attribute because the UUID associated with a `winrt::com_ptr` depends on what the `something` is.

Okay, so let's fix that by using the `something`.

```
winrt::check_hresult(
    interop->RequestTokenForWindowAsync(
    window,
    requestInspectable,
    __uuidof(winrt::IAsyncOperation<winrt::WebTokenRequestResult>),
    operation.put_void()));
```

That still doesn't work. We just get the same error again:

```
error C2787: 'winrt::Windows::Foundation::IAsyncOperation<winrt::Windows::Security::
Authentication::Web::Core::WebTokenRequestResult>': no GUID has been associated with
this object
```

It's the same problem again. `winrt::IAsyncOperation<winrt::WebTokenRequest-Result>` doesn't have a `__declspec(uuid(...))` because the UUID depends on the thing inside.

But this time, we can't unwrap it because we really do want the UUID of the `IAsyncOperation<something>`, not the UUID of the `something`.

It turns out that we ran astray much earlier:

```
winrt::com_ptr<winrt::IAsyncOperation<winrt::WebTokenRequestResult>> operation;
```

We created a `winrt::com_ptr` to a `winrt::IAsyncOperation<T>`. But the catch is that `winrt::IAsyncOperation<T>` is itself already a smart pointer. You created a smart pointer to a smart pointer, and that's the source of confusion.

The type wrapped by a `winrt::com_ptr` is expected to be a type that has methods `Query-Interface`, `AddRef`, and `Release`, according to the conventions of the ABI `::IUnknown`. The thing to pass here is not another smart pointer, but rather the ABI COM interface type.

```
// Oh my goodness, please don't make me type this.
winrt::com_ptr<ABI::Windows::Foundation::IAsyncOperation<
    ABI::Windows::Security::Authentication::Web::Core::WebTokenRequestResult*>>
    operation;
```

That is a horrible mouthful, and requires you to include the Windows Runtime ABI header files. Mixing the Windows Runtime ABI header files with C++/WinRT does work, but it usually creates mass confusion for the developer, so let's try not to do that.

Even if you wade into this world, you have to take the COM ABI version of `IAsyncOperation<WebTokenResult>` and convert it back to its C++/WinRT equivalent before you can `co_await` it, which is another level of annoying typing.

Fortunately, there's a way out of this mess. One is to realize that `winrt::IAsyncOperation<winrt::WebTokenRequestResult>` is the thing we want: A smart pointer around a raw ABI pointer. So use that.

```
winrt::IAsyncOperation<winrt::WebTokenRequestResult>
RequestTokenForWindowAsync(HWND window, winrt::WebTokenRequest const& request)
{
    auto interop = winrt::get_activation_factory<winrt::WebAuthenticationCoreManager,

::IWebAuthenticationCoreManagerInterop>();

    winrt::IAsyncOperation<winrt::WebTokenRequestResult> operation;
    auto requestInspectable = static_cast<::IInspectable*>(winrt::get_abi(request));

    winrt::check_hresult(
        interop->RequestTokenForWindowAsync(
        window,
        requestInspectable,
        winrt::guid_of<decltype(operation)>(),
        operation.put_void()));

    co_return co_await operation;
}
```

Once we have it in this form, we can call upon our old friend `winrt::capture`, whose job is to help obtain COM ABI objects and convert them to C++/WinRT objects.

```
winrt::IAsyncOperation<winrt::WebTokenRequestResult>
RequestTokenForWindowAsync(HWND window, winrt::WebTokenRequest const& request)
{
    auto interop = winrt::get_activation_factory<winrt::WebAuthenticationCoreManager,

::IWebAuthenticationCoreManagerInterop>();

    auto requestInspectable = static_cast<::IInspectable*>(winrt::get_abi(request));

    co_return co_await
        winrt::capture<winrt::IAsyncOperation<winrt::WebTokenRequestResult>>(
            interop,
            &::IWebAuthenticationCoreManagerInterop::RequestTokenForWindowAsync,
            window,
            requestInspectable);
}
```

For style points, you could collapse the entire function into a one-liner.

```
winrt::IAsyncOperation<winrt::WebTokenRequestResult>
RequestTokenForWindowAsync(HWND window, winrt::WebTokenRequest const& request)
{
    return
        winrt::capture<winrt::IAsyncOperation<winrt::WebTokenRequestResult>>(
            winrt::get_activation_factory<winrt::WebAuthenticationCoreManager,
                                          ::IWebAuthenticationCoreManagerInterop>(),
            &::IWebAuthenticationCoreManagerInterop::RequestTokenForWindowAsync,
            window,
            static_cast<::IInspectable*>(winrt::get_abi(request)));
}
```

Replacing `co_return co_await` with a simple `return` works here because we are just returning the operation, so we can pass it through instead of wrapping it inside a coroutine. (This sort of trick is not available in general, but we can use it here.)

One final tweak is using the WIL helper method `com_raw_ptr` to extract the ABI `IInspectable*` from a C++/WinRT object.

```
winrt::IAsyncOperation<winrt::WebTokenRequestResult>
RequestTokenForWindowAsync(HWND window, winrt::WebTokenRequest const& request)
{
    return
        winrt::capture<winrt::IAsyncOperation<winrt::WebTokenRequestResult>>(
            winrt::get_activation_factory<winrt::WebAuthenticationCoreManager,
                                          ::IWebAuthenticationCoreManagerInterop>(),
            &::IWebAuthenticationCoreManagerInterop::RequestTokenForWindowAsync,
            window,
            wil::com_raw_ptr(request));
}
```

Raymond Chen

Follow