Using Windows Runtime interop methods from C++/WinRT: Some helper functions

devblogs.microsoft.com/oldnewthing/20210806-00

Raymond Chen

Last time, we saw how to combine various pieces of C++/WinRT in order to call a Windows Runtime interop interface and capture the result as a C++/WinRT object.

There is some redundancy in the pattern, since you have to name the interface twice: Once to obtain it from the factory, and again to name the member function.

A little helper function will save some typing:

```
template<
    typename Result,
    typename WinRTType,
    typename InteropInterface,
    typename... InteropArgs,
    typename... Args>
auto capture_interop(
    HRESULT (STDMETHODCALLTYPE InteropInterface::*method)(InteropArgs...),
    Args&&... args)
{
    return winrt::capture<Result>(
        winrt::get_activation_factory<WinRTType, InteropInterface>(),
        method, std::forward<Args>(args)...);
}
```

Our example last time then simplifies to

```
capture_interop<
    winrt::IAsyncOperation<winrt::WebTokenRequestResult>,
    winrt::WebAuthenticationCoreManager>(
    &::IWebAuthenticationCoreManagerInterop::RequestTokenForWindowAsync,
    window,
    static_cast<::IInspectable*>(winrt::get_abi(request)));
```

There is a corresponding pattern for interop interfaces on instance objects.

```
template<
    typename Result,
    typename InteropInterface,
    typename... InteropArgs,
    typename... Args>
auto capture_interop(
    winrt::IUnknown const& o,
    HRESULT (STDMETHODCALLTYPE InteropInterface::*method)(InteropArgs...),
    Args&&... args)
{
    return winrt::capture<Result>(
        o.as<InteropInterface>(), method, std::forward<Args>(args)...);
}
```

Usage for this would be something like

```
winrt::UserActivity activity = ...;

auto session = capture_interop<winrt::UserActivitySession>(
    activity, &::IUserActivityInterop::CreateSessionForWindow, window);
```

These helper functions are part of the Windows Implementation Library, documented here:
capture_interop.

Raymond Chen

Follow