What's the difference between throwing a winrt::hresult_error and using winrt::throw_hresult?

devblogs.microsoft.com/oldnewthing/20210716-00

July 16, 2021

Raymond Chen

There are two ways to throw an exception in C++/WinRT. You can throw the exception object directly:

```
throw winrt::hresult_invalid_argument();
throw winrt::hresult_error(D3DERR_DEVICELOST);
```

Or you can use the `throw_hresult` function.

```
winrt::throw_hresult(E_INVALIDARG);
winrt::throw_hresult(D3DERR_DEVICELOST);
```

What's the difference?

If you look at the code for the `throw_hresult` function, you'll see that it eventually throws the underlying exception object, but it constructs the exception object with the `take_ownership_from_abi` parameter. So the real question is "What does the `take_ownership_from_abi` parameter do?"

The `take_ownership_from_abi` parameter means that this exception is taking over the error context from the existing ABI error context. The error context is what is used by error reporting tools and debuggers to show the root cause of the error.

So it boils down to this:

- If this error was detected by your code, then you want the debugger and other error reporting tools to point to your code as the source of the error, and you should use `throw winrt::hresult_error` or a specific derived exception type if applicable, such as `hresult_invalid_argument`.
- If you are propagating an error received by another component, then you want the debugger and other error reporting tools to direct the developer to the component from which you received the error. In that case, you should use `winrt::throw_hresult`.

Note that in the second case (propagation), the component you received the error from could itself be propagating an error from yet another component. As long as everybody propagates the error context along with the error, the debugging tools will point at the code that originated the error.[1]

[1] The intermediate components are also reported, so you can also follow how the error traveled from the originator to the final destination, but the origination usually gives you the best information about what went wrong.[2]

[2] Propagating and transforming error context brings us full circle to the very early days of COM when HRESULT was a handle to an error object. The old new thing has become the new old thing.

Raymond Chen

Follow