C++/WinRT gotcha: Not all exceptions derive from hresult_error

devblogs.microsoft.com/oldnewthing/20240115-00

January 15, 2024

Raymond Chen

I often see code that tries to catch all C++/WinRT exceptions by doing a

```
try {
    ... C++/WinRT code ...
} catch (winrt::hresult_error const& ex) {
    ... caught all C++/WinRT exceptions (right?) ...
}
```

Unfortunately, this doesn't catch all C++/WinRT exceptions.

The code in C++/WinRT that converts `HRESULT`s to exceptions can be found in `throw_hresult`. From the code, you can see that every failure `HRESULT` turns into a thrown `winrt::hresult_error`, except for `error_bad_alloc`, which is the C++/WinRT internal name for `E_OUTOFMEMORY`.

Furthermore, your `try` block probably encompasses some C++ library code that could throw things like `std::out_of_range`, `std::invalid_argument`, or a plain old `std::exception`.

And of course if your code interacts with other libraries, you will want to catch the exceptions thrown by those other libraries, like the Windows Implementation Library.

If you want to catch all exceptions, then catch all exceptions. You can ask `winrt::to_hresult()` to convert the current exception to an `HRESULT`.

```
try {
    ... C++/WinRT code ...
} catch (...) {
    LogFailure(winrt::to_hresult());
}
```

In practice, catching `std::bad_alloc` doesn't usually help much. Your exception-handling code is probably going to allocate some memory, so you're back where you started.

Bonus chatter: One of the design principles of the Windows Runtime is that exceptions are intended to be used for unrecoverable errors. If there is a recoverable error, then it should be reported in a non-exceptional way. Some of the older Windows Runtime classes don't follow this principle, but for the important ones, Windows has been slowly adding non-exceptional alternatives. For example, `HttpClient.GetAsync` now has a non-exceptional alternative `HttpClient.TryGetAsync`.