What kind of caller diagnostic information can I get from exceptions thrown by C++/WinRT and wil?

devblogs.microsoft.com/oldnewthing/20221121-00

November 21, 2022

Raymond Chen

C++/WinRT throws `winrt::hresult_error` to represent COM exceptions. The wil framework throws `ResultException` for this purpose. How do they interact, and what diagnostic information do they provide?

C++/WinRT `hresult_error` uses `RoOriginateError` to generate a *stowed exception* which records, among other things, a stack trace for the current thread. This stack trace is stored as part of the `hresult_error` in the form of a `IRestrictedErrorInfo`.

The wil framework by default does not use `RoOriginateError`, so there is no captured stack trace. However, it does capture the file name and line number in the `FailureInfo` that is stored in the `ResultException`. The `FailureInfo` also contains information to let you correlate multiple failures and see which ones are manifestations of the same underlying failure.

Here's a little table of what we have so far:

	C++/WinRT	**wil**
Thrown type	`hresult_error`	`ResultException`
Stack trace in thrown object	Yes	No
File/line number in thrown object	No	Yes
Recorded in wil error log	No	Yes

Things get more complicated if you include `wil/result_originate.h`: This tells wil to call `RoOriginateError` before throwing the exception, thereby capturing a stack trace. The stack trace is not explicitly saved in the exception object, however. It is stored in a thread-local object that can be retrieved via `GetErrorInfo()`, and many parts of the system (including C++/WinRT) understand how to retrieve and preserve this extended information, though determining whether any specific scenario preserves the extended information requires investigation.

So now we have this:

	C++/WinRT	wil
Thrown type	`hresult_error`	`ResultException`
Stack trace in thrown object	Yes	No
Stack trace in thread data	Yes	Requires `result_originate.h`
File/line number in thrown object	No	Yes
Recorded in wil error log	No	Yes

But wait, we're not done yet. There's another header file that affects how wil throws exceptions, and that's `wil/cppwinrt.h`. This header file enables various C++/WinRT+wil interop features, including exception handling. Exceptions propagated by the C++/WinRT library (for example, by `check_hresult()`) are filtered through wil, which logs them through its own error logging channel. However, since the file and line number were generated from the `__FILE__` and `__LINE__` preprocessor symbols captured by the `THROW_IF_FAILED` macro, C++/WinRT cannot capture file and line number information about the origination point, so you don't get line number information in your wil trace log. But you still get a stack trace in the `hresult_error` object.

Exceptions that are thrown explicitly via `throw hresult_error()` do not go through wil filtering.

	C++/WinRT			wil
		with `wil/cppwinrt.h`		
	no `wil/cppwinrt.h`	`throw hresult_error`	`check_hresult`	`THROW_IF_FAILED`
Thrown type	`hresult_error`	`hresult_error`	`hresult_error`	`ResultException`
Stack trace in thrown object	Yes	Yes	Yes	No

Stack trace in thread data	Yes	Yes	Yes	Requires `result_originate.h`
File/line number in thrown object	No	No	No	Yes
Recorded in wil error log	No	No	Yes	Yes

But wait, we're not finished yet. The wil framework alters its behavior if C++/CX is enabled. If so, then it throws a `Platform::Exception^` instead of a `wil::ResultException`. The `Platform::Exception^` captures a stack trace but not file/line number information.

	C++/WinRT			**wil**	
		with `wil/cppwinrt.h`		**no C++/CX**	**with C++/CX**
	no `wil/cppwinrt.h`	**`throw hresult_error`**	**`check_hresult`**	**`THROW_IF_FAILED`**	**`THROW_IF_FAILED`**
Thrown type	`hresult_error`	`hresult_error`	`hresult_error`	`Result-Exception`	`Exception^`
Stack trace in thrown object	Yes	Yes	Yes	No	Yes
Stack trace in thread data	Yes	Yes	Yes	Requires `result_originate.h`	Yes
File/line number in thrown object	No	No	No	Yes	No
Recorded in wil error log	No	No	Yes	Yes	Yes

Raymond Chen

Follow