A survey of how implementations of Windows Runtime events deal with errors

devblogs.microsoft.com/oldnewthing/20220413-00

April 13, 2022

Raymond Chen

There are three commonly-seen implementations of Windows Runtime events (one of which with four variants), and they deal with errors differently.

Class	Disconnect errors			
	`RPC_E_DISCONNECTED`	`RPC_S_SERVER_UNAVAILABLE`	`RPC_E_SERVER_DIED`	`RPC_E`
`WRL::EventSource<FireAll>`	•	•		
`WRL::EventSource<StopOnFirstError>`	•	•		
`WRL::EventSource<ReportUnhandled-OnFirstError-WithWin8Quirk>`	•	•	•	

`WRL::EventSource< StopOnFirstError- WithWin8Quirk>`	•	•	•	
C++/CX `event`	•	•		
`winrt::event`	•	•		

The code for `WRL::EventSource` is in `wrl/event.h`. The code for C++/CX `event` is in `vccorlib.h`, where it is called `Platform::EventSource`. And the code for C++/WinRT `winrt::event` is in `winrt/base.h`. I built the above table by reading the code. (You can do it too.)

For WRL, the default for `EventSource` is `FireAll` for third-party code, and `Report-UnhandledOnFirstErrorWithWin8Quirk` for Windows operating system code.

In all cases, a handler that returns one of the recognized Disconnect error codes is removed as an event handler and therefore will not receive any future events. (However, it will not be considered an error to decide whether or not to stop calling further handlers.) To avoid accidentally propagating one of these secret error codes out of your event handler, wrap your event handler in a `try` ... `catch (...)` or mark your event handlers as `noexcept`.

In fact, given that different event sources deal with errors differently, you probably should simply avoid returning errors from your event handler. Wrap the event handler in a `try` ... `catch (...)` so you can decide what to do in case of an error, or mark the event handler as `noexcept` to say that all errors are fatal to the application.

Raymond Chen

Follow