The secret signal that tells Windows Runtime events that the event recipient no longer exists

devblogs.microsoft.com/oldnewthing/20190521-00

May 21, 2019

Raymond Chen

There is a convention in the Windows Runtime that if an event handler returns the error code `RPC_E_DISCONNECTED`, then that means that the event recipient no longer exists and can be removed from the list of handlers. The C++/CX name for this error code is `Platform::DisconnectedException`. C# doesn't have a specific name for it; it's just a special case of `COMException`.

By convention, the `RPC_E_DISCONNECTED` is returned in the case where you subscribe to an event with a weak reference to the event recipient, but when the event is raised, the weak reference fails to resolve to an object, meaning that the object no longer exists. It is a convention in the Windows Runtime that the event source auto-unregisters the handler when it learns that the receipient no longer exists. In other words, once you report that the event recipient is gone, it cannot magically resurrect itself.

In practice, you get this error in two cases. The first is the case where the event recipient forgot to unregister itself from the event before it destructed. In that case, the handler is never going to be deregistered (the object forgot to clean up prior to destruction), and automatically unregistering the handler avoids a leak.

The second is the case where the event recipient unregistered the handler, but an event was in flight at the time the handler was unregistered, so you hit a race condition where the event source was all primed to call the handler just as you unregistered it. In that case, the event handler was already removed, so the auto-unregistration is redundant and harmless.

The WRL event source goes one step further and gives this auto-unregister treatment to the `RPC_S_SERVER_UNAVAILABLE` error code as well. And the `winrt::event` goes two steps further and also auto-unregisters upon receipt of `JSCRIPT_E_CANTEXECUTE`.

What this means for you is that you should avoid making this secret signal accidentally. If you return one of these secret error codes (or throw the corresponding exceptions), you are going to tell the event source that your handler is unrecoverably dead. The most common case where you might do it by mistake is if you in turn call out to another object, and that other object returns one of these error codes or throws one of these exception, and you propagate the error to the event or allow the exception to escape your event handler.

Next time, we'll look at how event handlers interact with garbage collection.

Bonus chatter: Many years ago, one of my friends accidentally invoked a secret code. When she and her friends ordered drinks, she asked for a sidecar, and was perplexed when she received something that didn't resemble the cocktail at all. It turns out that in that area, the term *sidecar* was a secret code meaning that you'd like an additional shot of alcohol served with your existing drink. (I assume it was a secret code because strict liquor laws prevented you from asking for the extra shot explicitly.)

Raymond Chen

Follow