The useless IMemoryBufferReference.Closed event

devblogs.microsoft.com/oldnewthing/20240123-00

January 23, 2024

Raymond Chen

The `IMemoryBufferReference` interface has a `Closed` event which is signaled when the underlying object is closed or destructed. Closing the `IMemoryBufferReference` invalidates any pointers that had been obtained from `GetBuffer`.

Unfortunately, this event is basically useless.

The idea was that a consumer of the buffer could be notified that the underlying data has been made unavailable, but multithreading means that this notification doesn't really help: Suppose that you have one thread that is doing some computation with the buffer, and another thread that notifies you that the buffer is invalid. The notification thread can't stop the computation thread in its tracks. At best, it would have to signal the computation thread to wrap up and block the notification thread until the computation thread reports that it has stopped accessing the buffer. But the notification might be delivered on a single-threaded apartment, in which case blocking is ill-advised.

What's worse, some implementations of `IMemoryBufferReference` raise the event *after* the buffer becomes invalid, so this event doesn't even give you a chance to stop your computation. It's just telling you, "Oh, hey, so, like, you probably just corrupted memory a few millseconds ago."

Fortunately, you don't really need the notification because you're generally just notifying yourself. Each `IMemoryBufferReference` is a separate reference to the buffer, and if you have two components that want to access the buffer, you can just give each one a different `IMemoryBufferReference`. That way, one component closing the `IMemoryBufferReference` has no effect on the other. The only time your `IMemoryBufferReference` should be closed is when *you* close it.

And hopefully you can arrange so that you never surprise yourself.

Next time, we'll look at how the Closed event is not merely useless but also dangerous.