How should I report errors from my Windows Runtime API?

devblogs.microsoft.com/oldnewthing/20190228-00

February 28, 2019

Raymond Chen

A customer was implementing a custom Windows Runtime component and wanted to know what the best practices are for reporting errors.

Well, it depends on what kind of error it is.

One class of errors are unrecoverable errors. For example, if there is an error that represents an internal logic error or other bug in the program, then there is not going to be any meaningful recovery. The program thought it was doing the right thing, and it is unlikely to know what to do when told that it's entire world view is wrong.

For example, suppose that you must stop the widget before you change its polarity. If somebody tries to change the polarity of a widget that is not stopped, well, there's a bug in that program. Passing an invalid parameter is another example of something that indicates an internal logic error in the program.

Another type of unrecoverable errors is the error that indicates that things are so far gone that any attempt at recovery is unlikely to succeed. An "out of memory" error when trying to allocate memory for a 50-byte object is an example of this type of unrecoverable error. Sure, you could report, "Sorry, I couldn't complete the operation because I was out of memory," but any action the program takes is probably going to try to allocate at least 50 bytes of memory at some point, so there's really not much point in continuing. All you're doing is delaying the crash by a few milliseconds. (And making the crash dump harder to debug because the crash will be at a point far away from the root cause.)

Another class of errors are the recoverable or anticipated errors. These are errors which applications could meaningfully anticipate recovering from, typically because they are caused by external factors not within the programmer's control.

For example, attempting to download a file from a URL can legitimately fail for reasons complete out of the control of the program. The system could be offline, or the server could be unavailable. A programmer would anticipate these possible errors and have an alternative

plan of action if they occur.

Another case where an error would be anticipated is if there is a natural race condition with an external source. A media playback control might reach the end of the video just at the moment the program decides to skip backward 10 seconds. This is an unavoidable race condition and is therefore an anticipated error.

The guidance for the Windows Runtime is that unrecoverable errors are reported by COM error codes at the ABI level, which are converted to a language-specific exception reporting mechanism by the corresponding language runtime. On the other hand, recoverable or anticipated errors are reported by some in-API mechanism like a status code, rather than by using a COM error code.

The error handling guidance for C++/WinRT captures this guidance, but wearing C++-colored glasses.

The idea is that the Windows Runtime considers exceptions to be fatal errors. Application code is not expected to catch exceptions thrown by Windows Runtime objects because they represent unrecoverable errors. The application should just let the exception trigger an application crash so it can be debugged.

If you are a Windows Runtime class and wish to report an unrecoverable error, then before returning the COM failure code, call the `RoOriginateError` function to provide a helpful message to the developer explaining what they did wrong.

Note that many Windows Runtime classes provided by Windows do not adhere to this guidance because they predate the guidance. But even those classes are gradually being improved to align with the guidance. For example, `StorageFolder.TryGetItem` lets you access a file or folder without raising an exception if the item doesn't exist or is inaccessible. Similarly, `HttpClient.TryGetAsync` attempts a `GET` operation but does not raise an exception if the operation fails.

Raymond Chen

Follow