Where can I find the standard asynchronous stream?

devblogs.microsoft.com/oldnewthing/20150114-00

January 14, 2015

Raymond Chen

In the documentation for XmlLite, one of the features called out is that XmlLite is a non-blocking parser. If the input stream returns `E_PENDING`, then XmlLite propagates that status to its caller, and a subsequent request to XmlLite to parse will resume where it left off.

That documentation calls out two scenarios in which this can happen, the second of which is

> 2. The input Stream is a standard asynchronous stream. The E_PENDING HRESULT may be raised when the data is temporarily unavailable on the network. In this case, you need to try again later in a callback or after some interval of time.

A customer was kind of confused by this explanation. "Where do I get a standard asynchronous stream so I can use it in scenario 2?"

The documentation here is trying to be helpful by expanding on the original statement that XmlLite is a non-blocking parser and providing examples of how you can take advantage of this non-blocking behavior. The normative statement is the one that says, "XmlLite propagates the E_PENDING from the input stream to its caller, and a subsequent request to read data from the XmlLite parser will resume where it left off." The rest is informational, but it seems that the informational text was more confusing than helpful.

The informational text is trying to say, "Here are some examples where you can take advantage of this behavior." The first scenario is an example where you provided an IStream that returns `E_PENDING` when it wants to force the XmlLite parser to stop parsing. You might do this, for example, if you have out-of-band data in your XML stream. The stream would return `E_PENDING` when it encounters the out-of-band data, and this causes the XmlLite parser to stop parsing and return `E_PENDING`. You can then process the out-of-band data, and then when you are ready to resume parsing, you reissue the call that returned `E_PENDING` so the parser can resume where it left off.

The second scenario is an example where you provided an IStream that returns `E_PENDING` to indicate that there is more data in the stream, but it is not available right now. For example, the stream may be the result of a streaming download, and the next chunk of the download hasn't arrived yet. Instead of blocking the read, the stream returns `E_PENDING` to

say, “There is more data, but I can’t provide it right now. Go do something else for a while.” The download stream presumably has some way of notifying when the next download chunk is ready. Your program can subscribe to that notification, and when it is received, you can resume parsing with XmlLite.

The adjective “standard” here in the phrase “a standard asynchronous stream” does not refer to a specific reference implementation. It’s using the word “standard” in the sense of “regularly and widely used, seen, or accepted; not unusual or special.” (This was subtly implied by the use of the indefinite rather than the definite article, but that use of the indefinite could be interpreted to mean “an instance of the standard asynchronous stream”.) In other words, the opening sentence is saying, “The input Stream is any asynchronous stream that behaves in the usual manner.”

By analogy, consider the sentence “This service is available from a standard touch-tone phone.” This doesn’t mean “There is a specific model of touch-tone phone that is the standard touch-tone phone, and you must use that one.” It’s just saying “Any touch-tone phone (that conforms to the standard) will work.”

Raymond Chen

Follow