What are the rules for CoMarshalInterface and CoUnmarshalInterface?

devblogs.microsoft.com/oldnewthing/20151022-00

October 22, 2015

Raymond Chen

Last time, we looked at the rules for `CoMarshalInterThreadInterfaceInStream` and `CoGetInterfaceAndReleaseStream`, the functions you use for sharing an object with another thread in the sample case where you there is only one other thread you want to share with, and you need to share it only once. Let's continue with the Q&A.

What if I want to unmarshal more than once?

In this case, you use the more general `CoMarshalInterface`. You can pass the `MSHLFLAGS_TABLESTRONG` flag to indicate that you want to be able to unmarshal many times. In that case, you need to tell COM when you are finished unmarshaling so it knows when to clean up, because it cannot assume that you are finished after the first unmarshal. The pattern goes like this:

- On the originating apartment, create an empty stream.
- On the originating apartment, call `CoMarshalInterface` with the empty stream and the `MSHLFLAGS_TABLESTRONG` flag.
- Transmit a *copy* of the stream to each of the threads you want to share the object with. (You need to use a copy so that the multiple threads don't all try to use the same stream and step on each other's stream position. Alternatively, you could be clever and use the same stream, but use a mutex or other synchronization object to make sure only one thread uses the stream at a time.)
- The receiving threads rewind the stream copy to the beginning.
- The receiving threads call `CoGetInterfaceAndReleaseStream` to reconstitute the object from the stream and release the stream.1
- The receiving threads happily accesses the object.
- When the originating apartment decides that it doesn't want to share the object any more, it calls `CoReleaseMarshalData` to tell COM to clean up all the bookkeeping.
- The originating apartment destroys the master stream.

What is the relationship between `CoMarshalInterThreadInterfaceInStream` and `CoMarshalInterface`?

The `CoMarshalInterThreadInterfaceInStream` function is a helper function that does the following:

- `CreateStreamOnHGlobal`.
- `CoMarshalInterface` with `MSHCTX_INPROC` and `MSHLFLAGS_NORMAL`.
- Rewinds the stream to the beginning.
- Returns the stream.

Similarly, `CoGetInterfaceAndReleaseStream` is a helper function that does

- `CoUnmarshalInterface`
- `IStream::Release`

Since a one-shot marshal to another thread within the same process is by far the most common case, the helper functions exist to let you get the job done with just one function call on each side.

What if I want to marshal only once, but to another process?

Again, you need to use the more general `CoMarshalInterface` function. This time, you pass the `MSHCTX_LOCAL` flag if you intend to marshal to another process on the same computer, or the `MSHCTX_DIFFERENTMACHINE` flag if you intend to marshal to another computer. For the marshal flags, use `MSHLFLAGS_NORMAL` to indicate that you want a one-shot marshal. The recipient can unmarshal with `CoGetInterfaceAndReleaseStream` as before.

What if I want to marshal to another process and unmarshal more than once?

This is just combining the two axes. On the marshaling side, you do the same as a one-shot cross-process marshal, except you pass the `MSHLFLAGS_TABLESTRONG` flag to indicate that you want to be able to unmarshal many times. You then send copies of that stream to all your intended recipients, and each of them calls `CoGetInterfaceAndReleaseStream`, just like before.

Can you marshal a proxy? Does it get all Inception-like?

Go ahead and marshal a proxy. COM detects that you're marshaling a proxy and does the Right Thing. For example, if you marshal a proxy back to the originating thread, then when you unmarshal, you get a direct pointer again!

[1] If the thread wants to unmarshal from the stream than once, it could call `CoUnmarshal-Interface` and not release the stream immediately. Then each time it wants to unmarshal from the stream, it calls `CoUnmarshalInterface` again, releasing the stream only when it has decided that it will not do any more unmarshaling. This seems silly because once you

unmarshal the first time, you can just `AddRef` the pointer if you want to make another copy. I guess this is for the case where the thread wants to pass the stream off to yet another thread? Definitely a fringe case.

Raymond Chen

Follow