An initial look at the mechanics of how COM marshaling is performed

devblogs.microsoft.com/oldnewthing/20220615-00

June 15, 2022

Raymond Chen

Last time, we looked at the usage patterns for manually-marshaled interfaces. Today, we'll look at the contract from the other side.

A marshaler has to implement a number of methods, which break down nicely into three groups:

Method	Operation	Group
`GetUnmarshalClass`	Report the object to create when unmarshaling.	Marshaling
`GetMarshalSizeMax`	Calculate how much memory is needed.	
`MarshalInterface`	Generate the marshaling data.	
`UnmarshalInterface`	Load object from marshaling data.	Unmarshaling
`ReleaseMarshalData`	Clean up marshaling data.	Cleaning up
`DisconnectObject`	Shut down the unmarshaled object.	

The general sequence of operations goes like this:

- Marshaling an object (happens in the source context):
  - COM asks `GetUnmarshalClass` for the class ID to create when unmarshaling.
  - COM asks `GetMarshalSizeMax` how much memory will be needed for the marshaling data.
  - COM allocates the required amount of data.
  - COM asks `MarshalInterface` to produce that data into a provided stream.

- Unmarshaling an object (happens in the destination context):
  - COM creates the unmarshaler object previously specified by `GetUnmarshalClass`.
  - COM calls `UnmarshalInterface` so the unmarshaler object can produce a new object from the marshal data that it had previously generated by its `MarshalInterface` method. (In most cases, the unmarshaler uses itself as the produced object.)
- Cleaning up:
  - COM calls `ReleaseMarshalData` on any thread to tell the unmarshaler to clean up any resources associated with the marshaled data. (The documentation says that under rare conditions, COM might ask the marshaler to do the cleanup.)
  - COM calls `DisconnectObject` in the destination context to tell the unmarshaled object that it should break any connection with the original object. In practice, this is called only for proxies created by the standard marshaler, because COM doesn't know how to hunt down custom proxies.

An important note is that, as we noted last time, the `UnmarshalInterface` automatically performs an internal equivalent of `ReleaseMarshalData` if the marshaling was performed as `MSHLFLAGS_NORMAL`.

Let walk through these steps with an analogy: Suppose the object is an oracle, say `COracle`. People can talk to the oracle, but the nature of the sense of hearing is that in order for people to do that, they need to be standing nearby, or at least a sound source carrying their voice needs to be nearby.

Now, suppose somebody who lives far away would like to talk to the oracle.

The `COracle`'s `GetUnmarshalClass` method would return the class ID for a custom `CLSID_OracleProxy` class.

The `COracle`'s `GetMarshalSizeMax` method would return 10, the maximum number of digits in a Greek telephone number. (The oracle lives in Delphi.)

The `COracle`'s `MarshalInterface` method would buy an eSIM card, write the number of the eSIM card on a papyrus scroll, and add the eSIM to the oracle's mobile phone plan. (The is a modern oracle who knows how to use a mobile phone.) For simplicity, let's assume that the marshaling was performed as `TABLESTRONG`. Associated with each eSIM is a reference count, that is the number of active remote clients plus the number of papyrus scrolls that have the number written on it. At the start, the reference count of the eSIM is 1 because the number is written on a papyrus scroll.

The courier delivers the papyrus scroll to whoever it was that wanted to talk to the oracle. The recipient unrolls the papyrus scroll, transcribes the bytes, and gives them to COM.

COM looks at the bytes of the stream and says "Okay, it says here that I need to create a `CLSID_OracleProxy` object." COM therefore creates a `COracleProxy` object, in an uninitialized state.

COM then calls the `COracleProxy`'s `UnmarshalInterface` with the remaining bytes of the stream. That method takes the ten digits of the oracle's eSIM card and commits them to memory. It calls the number (including the +30 dialing prefix if the proxy is outside Greece), and says, "Hi, this is `COracleProxy`. Just telling you that there's somebody over here who wants to talk to you." The oracle increments the reference count on the eSIM number.

Since the marshaling was done as `TABLESTRONG`, the courier can deliver the papyrus scroll to another client, and the unmarshal ceremony is repeated.

At some point, the papyrus scroll will be disposed of. But before they perform a ceremonial burning, they must call `CoReleaseUnmarshalData`. That function creates a new instance of the unmarshaler object `CLSID_OracleProxy` and this time calls the `ReleaseMarshal-Data` method. That method retrieves the phone number, calls it, and tells the oracle, "I'm destroying the papyrus scroll now." The oracle decrements the reference count on the eSIM and since it is not yet zero, she doesn't cancel the eSIM card yet.

Now, it's also possible that the courier was unable to deliver the papyrus scroll to the remote client (maybe they moved and left no forwarding address). In that case, the courier is the one who calls `CoReleaseMarshalData` before destroying the papyrus scroll. The courier might do this when they reach the remote client's home and finds that the house is empty. The courier might do this even before getting to the remote client's house, because the entire city has been burnt to the ground by an invading army. Or the courier might do this even before leaving the oracle because they realize that the remote client's address is outside his delivery area. Whatever the reason, the `ReleaseMarshalData` function extracts the phone number and calls the oracle to say, "I'm destroying the papyrus scroll now". In this case, the oracle decrements the reference count on the eSIM to zero, so she cancels the eSIM account and removes it from her mobile phone plan.

Anyway, assuming the courier reaches the destination and set up an oracle proxy, we are now in the state where the remote client can now ask questions to the `COracleProxy`, and the oracle proxy will pick up the phone, call the oracle at Delphi, and relay the question and its answer.

Eventually, the remote client decides that they are finished asking questions and `Release`s the oracle proxy. The oracle proxy makes one last call to the oracle, saying, "The client is finished asking questions. You can cancel the account as soon as I hang up." Once the call ends, the oracle decrements the reference count on the eSIM, and if it goes to zero, then she cancels the eSIM account and deletes it from her mobile phone plan.

The oracle herself might decide that she wants to retire. In that case, the oracle asks COM to disconnect all remote clients, and then cancels all of her eSIM accounts and dramatically throws her mobile phone into a fire. (There's a lot of burning in a fire in my imaginary version of ancient Greece.)

COM finds all of the oracle proxies associated with the remote client and calls the `DisconnectObject` method. Each oracle proxy makes a mental note that the oracle has retired, and the next time their remote client says, "Hey, please ask the oracle a question for me," the oracle proxy can say "Sorry, the oracle has retired. She is no longer accepting questions."

Next time, we'll use this understanding to start writing our own marshaler.

Raymond Chen

Follow