INTRODUCTION TO MESSAGE CONSTRUCTION

In *Introduction to Messaging Systems*, we discussed *Message*. When two applications wish to exchange a piece of data, they do so by wrapping it in a message. Whereas a *Message Channel* cannot transmit raw data per se, it can transmit the data wrapped in a message.

Deciding to create a *Message* and send it raises several other issues:

Message intent — Messages are ultimately just bundles of data, but the sender can have different intentions for what it expects the receiver to do with the message. It can send a *Command Message*, specifying a function or method on the receiver that the sender wishes to invoke. The sender is telling the receiver what code to run. It can send a *Document Message*, enabling the sender to transmit one of its data structures to the receiver. The sender is passing the data to the receiver, but not specifying what the receiver should necessarily do with it. Or it can send an *Event Message*, notifying the receiver of a change in the sender. The sender is not telling the receiver how to react, just providing notification.

Returning a response — When an application sends a message, it often expects a response confirming that the message has been processed and providing the result. This is a *Request-Reply* scenario. The request is usually a *Command Message*, and the reply is a *Document Message* containing a result value or an exception. The requestor should specify a *Return Address* in the request to tell the replier what channel to use to transmit the reply. The requestor may have multiple requests in process, so the reply should contain a *Correlation Identifier* that specifies which request this reply corresponds to. [JMS11, pp.27-28]

There are two common *Request-Reply* scenarios worth noting that both involve a *Command Message* request and a corresponding *Document Message* reply. In the first scenario, *Messaging RPC*, the requestor not only wants to invoke a function on the replier, but also wants the return value from the function. This is how applications perform an RPC (Remote Procedure Call) using *Messaging*. In the other scenario, *Messaging Query*, the requestor is performing a query that the replier will execute and return the results in the reply. This is how applications use messaging to perform a query remotely.

Huge amounts of data — Sometimes applications want to transfer a really large data structure, one that may not fit comfortably in a single message. In this case, break the data into more managable chunks and send them as a *Message Sequence*.

The chunks have to be sent as a sequence, and not just a bunch of messages, so that the receiver can reconstruct the original data structure.

Slow messages — A concern with messaging is that the sender often does not know how long it will take for the receiver to receive the message. Yet the message contents may be time-sensitive, such that if the message isn't received by a deadline, it should just be ignored and discarded. In this situation, the sender can use *Message Expiration* to specify an expiration date. If the messaging system cannot deliver a message by its expiration, it should discard the message. If a receiver gets a message after its expiration, it should discard the message.

Source:
http://www.enterpriseintegrationpatterns.com/patterns/messaging/MessageConstructionIntro.html