INTRODUCTION TO SIMPLE MESSAGING EXAMPLES

So far, we've introduced a lot of patterns. We've seen the basic Messaging Components, such as *Message Channel*, *Message*, and *Message Endpoint*. We've also seen detailed patterns for Messaging Channels and for Message Construction.

So how do all of these patterns fit together? How does a developer integrate applications using these patterns? What does the code look like, and how does it work?

This is the chapter where we really get to see the code. We have two examples:

- **Request/Reply** — Demonstrates (in Java and .NET/C#) how to use messaging to send a request message and respond with a reply message.
- **Publish/Subscribe** — Explores how to use a JMS Topic to implement the Observer pattern.

These two simple examples should get you started on how add messaging to your own applications.

Request/Reply Example

This is a simple but powerful example, transmitting a request and transmitting back a reply. It consists of two main classes:

- **Requestor** — The object that sends the request message and expects to receive the reply message.
- **Replier** — The object that receives the request message and sends a reply message in response.

These two simple classes sending simple messages illustrate a number of the patterns:

- ***Message Channel*** and ***Point-to-Point Channel*** — One channel for transmitting the requests, another for transmitting the replies.
- ***Document Message*** — The default type of message, used as both the request and the reply.
- ***Request-Reply*** — A pair of messages sent over a pair of channels, allowing the two applications to have a two-way conversation.
- ***Return Address*** — The channel to send the response on.
- ***Correlation Identifier*** — The ID of the request that caused this response.

- ***Datatype Channel*** — All of the messages on each channel should be of the same type.
- ***Invalid Message Channel*** — What happens to messages that aren't of the right type.

The example code also demostrates a couple of patterns from the Messaging Endpoints chapter later in the book:

- ***Polling Consumer*** — How the requestor consumes reply messages.
- ***Event-Driven Consumer*** — How the replier consumes request messages.

While this book is technology-, product-, and language-neutral, code cannot be. So we've choosen two messaging programming platforms to implement this example:

- The JMS API in Java J2EE
- The MSMQ API in Microsoft .NET using C#

The same request/reply example is implemented in both platforms. So choose your favorite platform as an example of how messaging works. If you'd like to see how messaging works on the other platform, even if you don't know how to write code for that platform, you should be able to figure out how that code works by comparing it to the code in the language you already know.

Publish/Subscribe Example

This example explores how to implement the Observer pattern using a *Publish-Subscribe Channel*. It considers distribution and threading issues and discusses how messaging greatly simplifies these issues. The example shows how to implement both the push and pull models of notification and compares the consequences of each. It also explores how to design an adaquate set of channels needed for a complex enterprise with numerous subjects notifing numerous observers.

The discussion and sample code will illustrate several patterns:

- ***Publish-Subscribe Channel*** — The channel that provides publish/subscribe notification.
- ***Event Message*** — The message type used to send notifications.
- ***Request-Reply*** — The technique used as part of the pull model for an observer to request state from the subject.
- ***Command Message*** — The message type used by an observer to request state from the subject.
- ***Document Message*** — The message type used by a subject to send its state to an observer.
- ***Return Address*** — Tells the subject how to send the state to the observer.

- ***Datatype Channel*** — The main guideline for whether two unrelated subjects can use the same channel to update the same group of observers.

The example code also demostrates a couple of patterns from the Messaging Endpoints chapter later in the book:

- ***Messaging Gateway*** — How the subject and observer encapsulate the messaging code so that they are not messaging-specific.
- ***Event-Driven Consumer*** — How the observers consume notification messages.
- ***Durable Subscriber*** — An observer that does not want to miss notifications, even if the observer is temporarily disconnected when the notification is sent.

This example is implemented in Java using JMS because JMS supports *Publish-Subscribe Channel* as an explicit feature of the API through its Topic interface. .NET does not provide a similar level of support for using the publish/subscribe semantics in MSMQ; when it does, the techniques in the JMS example should be readily applicable to .NET programs as well.

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