# Design Rules, Naming Convention and A Minimum Symbol Set For MQSeries System Design

**Benjamin F. Zhou**

IBM Certified Solution Expert

**Financial Sciences Corp., USA**

Graphical design is the most effective approach for describing a clean messaging system. Although a messaging system, including application programs can be very complex; it is of key importance that developers and administrators can easily understand its design.

With simplicity in mind, this author proposes a small set of symbols for use in this graphical design approach. This set has been proven effective during the design, implementation and training of staff both in-house and at major bank clients in the Wall Street Pre-Issuance Messaging (PIM) project. Also presented in this paper is a set of design rules and naming conventions based on design practice.

## 1. Graphical design – a minimum symbol set

It is easy to do system design on a graphical basis. To do this, an easy-to-understand set of symbol definitions is needed. The following set of ten symbols (Fig.1), which are included in a Visio template stencil file, can be used to describe any MQSeries design.

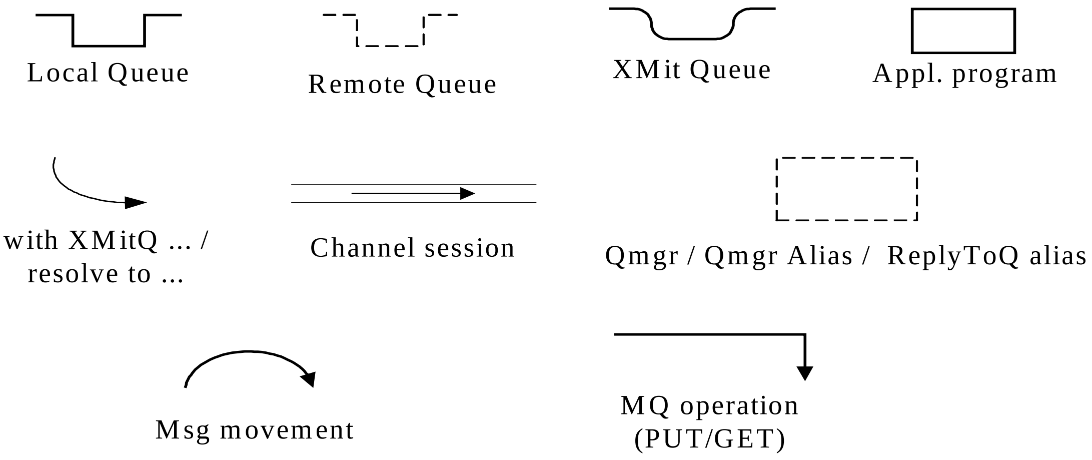

**Figure 1. Minimum symbol set for MQ system design**

(Note: depends on your reader tool version, the symbols may look differently. For better readability, all diagrams are provided in Microsoft Visio format.)

Figure 2 is a sample multi-hop message flow diagram:

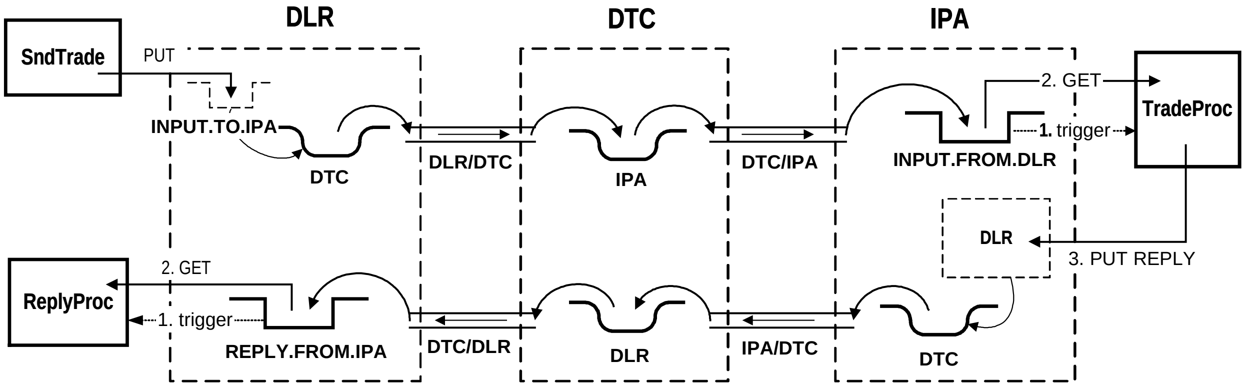

**Figure 2. sample design diagram**

With the same symbol set, Figure 3 illustrates the use of queue manager alias to remap the queue manager name specified in an MQOPEN call. This diagram refers to a section on queue manager alias in chapter 3 of MQSeries Intercommunication Guide.

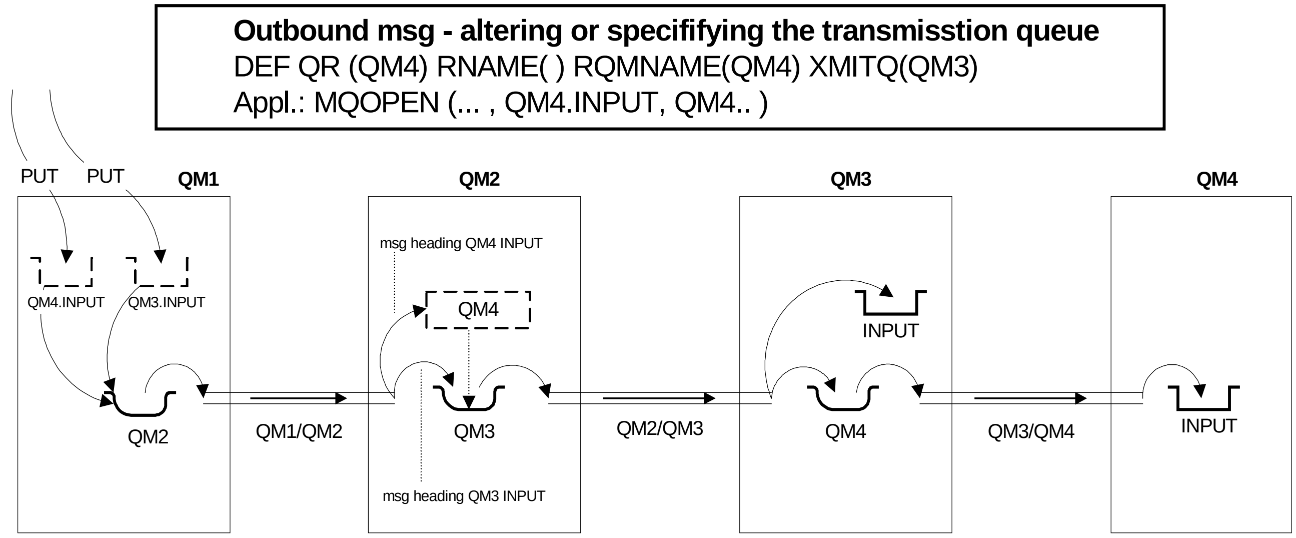
**Outbound msg - altering or specififying the transmisstion queue**
DEF QR (QM4) RNAME( ) RQMNAME(QM4) XMITQ(QM3)
Appl.: MQOPEN (... , QM4.INPUT, QM4.. )

**Figure 3. Use of Queue Manager Alias to remap queue manager name**

## 2. Naming convention

With such a diagram, the most suitable naming convention becomes apparent. In our definition, I setup the following naming convention:

- Local queue: take name from its role inside the Qmgr, including an implied message flow direction, like INPUT.FROM.CSFB, REPLY.TO.GS, REPLY.FROM.CHASE. Do not include Qmgr name in queue name.
- Transmission queues: use name of the immediate next destination Qmgr.
- Remote queue: specify(or just take name from) destination Qmgr, including an implied message flow direction, like OUTPUT.TO.MSDW.
- ReplyToQ-alias: take name of the destination Qmgr, here, DLR at IPA
- Trigger process: use same name as the application name it is supposed to trigger. For example, the process used to trigger TradeProc should be named as TradeProc
- Assign an alias queue for every sender party. Never give them your local input queue name. This will shield your system from that of others. For controlled test, you may want to point some of the alias to your DLQ.
- Do not put alias objects on your design diagram.

## 3. Use triggering if possible

- You cannot guarantee that your message processing application will be 100% reliable. Using triggering can reduce the impact of failure to a minimum.
- A trigger monitor will not trigger another application until the current one it just triggered ends. So if there is only one trigger monitor running, performance on triggered queues will suffer if messages arrive at different triggered queues in quick succession. Start a trigger monitor for every such queue to solve this problem.

## 4. You don’t need persistence as often as you may believe

After careful analysis, you may find that guaranteeing your messages never get lost is a much more expensive endeavor than making your application handle the case when a message does get lost. Actually, under most circumstances, this can fit easily into your application logic. Overuse of persistent messages can easily bring your system to its knees. Non-persistent messages typically have 20 times the throughput and respond much quicker.

## 5. Absolute separation of message-flow from message processing applications

Let MQSeries take care of message flow, and let your applications take care of message processing. This is the rule and ideal case. Unfortunately, you may encounter situations when other parties do not follow these rules.

- You specify a ReplyToQ in the message header of your request message. But the other party may not use it; instead, it puts all the messages into the queue you have specified for request only.
- Other party's application may not even mark its reply message as of type REPLY in the message header, thus forcing you to look at the message body to decide what it is.

If this happens to your system (we have encountered both), don't compromise the rule by letting your message processing application get involved with message flow. Just define a few internal queues and write a separate application to redistribute such messages. This way, you not only preserve the integrity of your application, but also make it possible to switch back to the ideal setting if the rule-offenders conform later.

Figure 4 illustrates the use of a simple application **redistributor** in the case mentioned above.

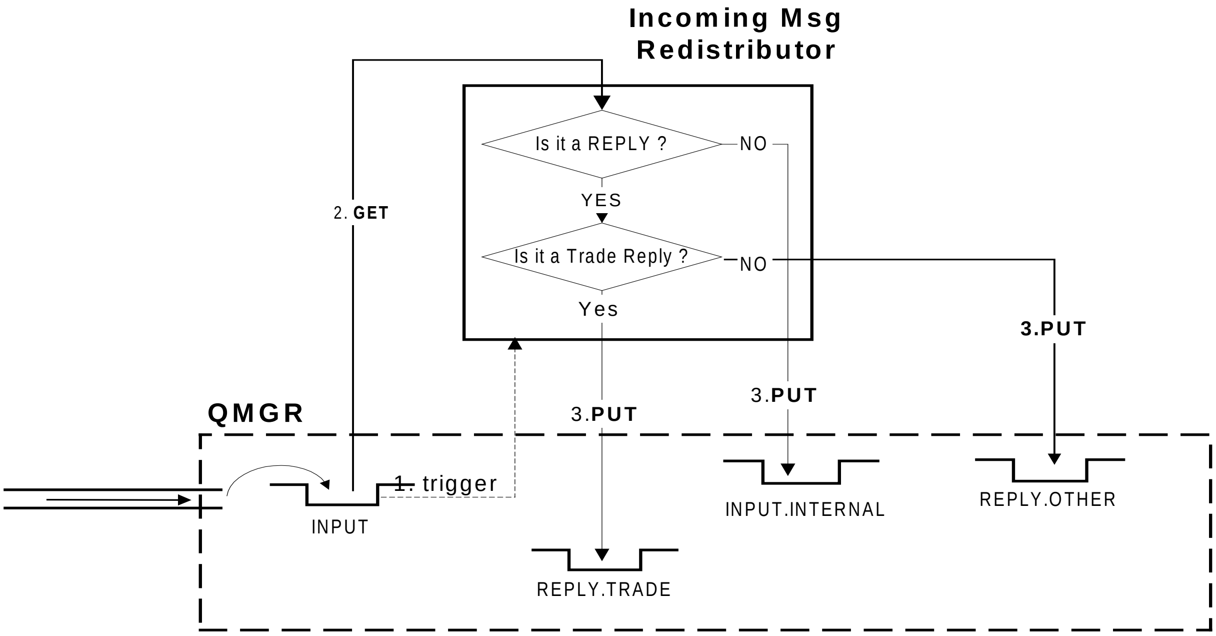

**Figure 4. the use of redistributor**

Please send your feedback to: bfzhou@netscape.net