IBM Software Group

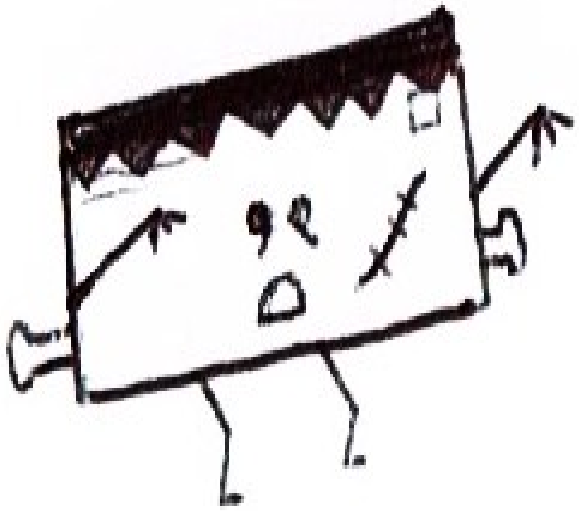
Handling undelivered messages in WebSphere MQ: DLQ, Poison Messages

Angel Rivera (rivera@us.ibm.com)
WebSphere MQ Unix® and Windows Level 2 Support
Date: 31-Oct-2013

WebSphere® Support Technical Exchange

Perfect topics for Halloween!

Agenda



- Undeliverable messages
  - Types
  - How to handle them
- Dead Letter Queue
- Poison Messages
  - J2EE application server (WAS) using JMS client

Undeliverable messages

- The Undeliverable Messages discussed in this presentation are those that are not delivered successfully by the queue manager.
- These messages are successfully placed in a queue inside a queue manager ...
  - ... but they cannot be delivered to the intended destination
- In contrast, if the putting application cannot put messages into the queue (such as when a queue is put-inhibited or is full) then:
  - This type is not considered here

Notes: Scenarios that are not covered here

notes

- An example of a scenario that does not fit the criteria from the previous slide is:
- An MQ client application (such as amqsput) tries to put a message into a queue that does not exist, or a queue that is PUT_INHIBITED, or that is FULL. For example:

- $ amqsput Q6 QM_VER75
- Sample AMQSPUT0 start
- target queue is Q6
- test
- MQPUT ended with reason code 2053
- Sample AMQSPUT0 end

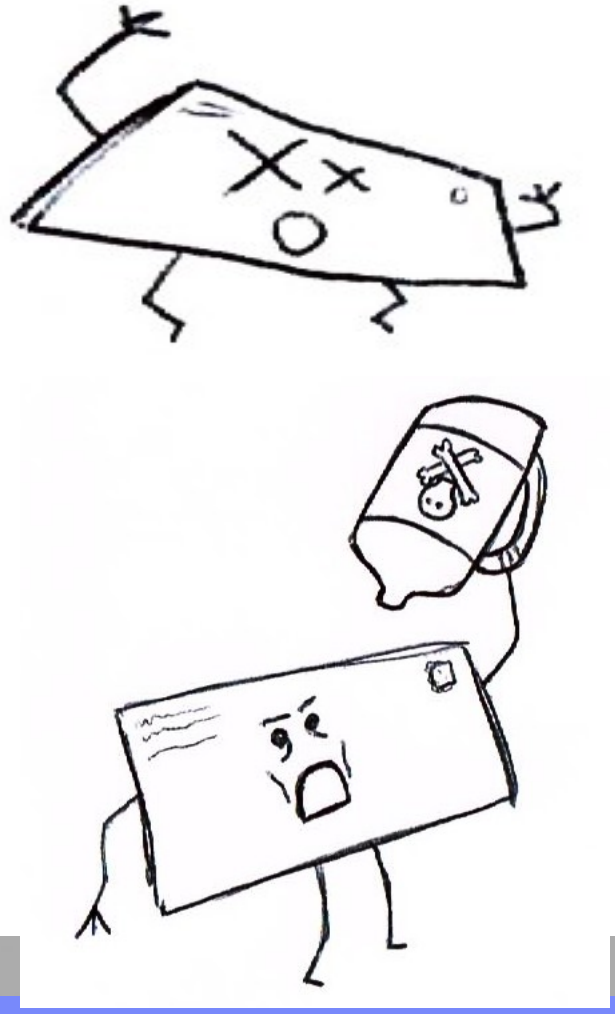
- $ mqrc 2053
- 2053 0x00000805 MQRC_Q_FULL

- Notice that the message was not accepted by the queue manager.

- You may argue that the application could not delivered the message, and thus, technically it is an "undelivered message", but this presentation concerns those messages that were accepted by the queue manager, but that the queue manager could not deliver in a subsequent action.

Undeliverable messages - types

- There are 2 types of undeliverable messages:
- 1) Messages that are placed in the Dead Letter Queue.
- 2) Messages that cannot be delivered to the client application during a get (Poison Messages)

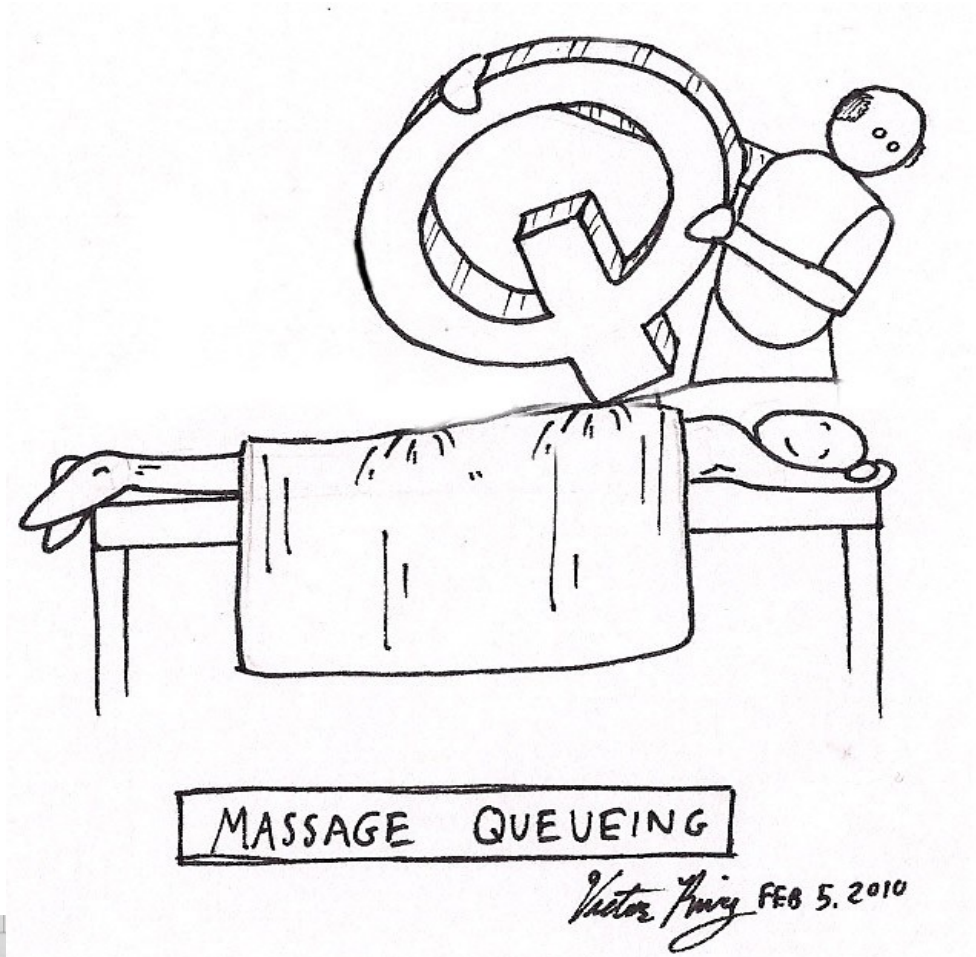


About the cartoons

- The cartoons in this presentation were created by my son Victor Rivera, who is an artist.
- I commissioned him this artwork that combine the theme of Halloween and MQ messages.

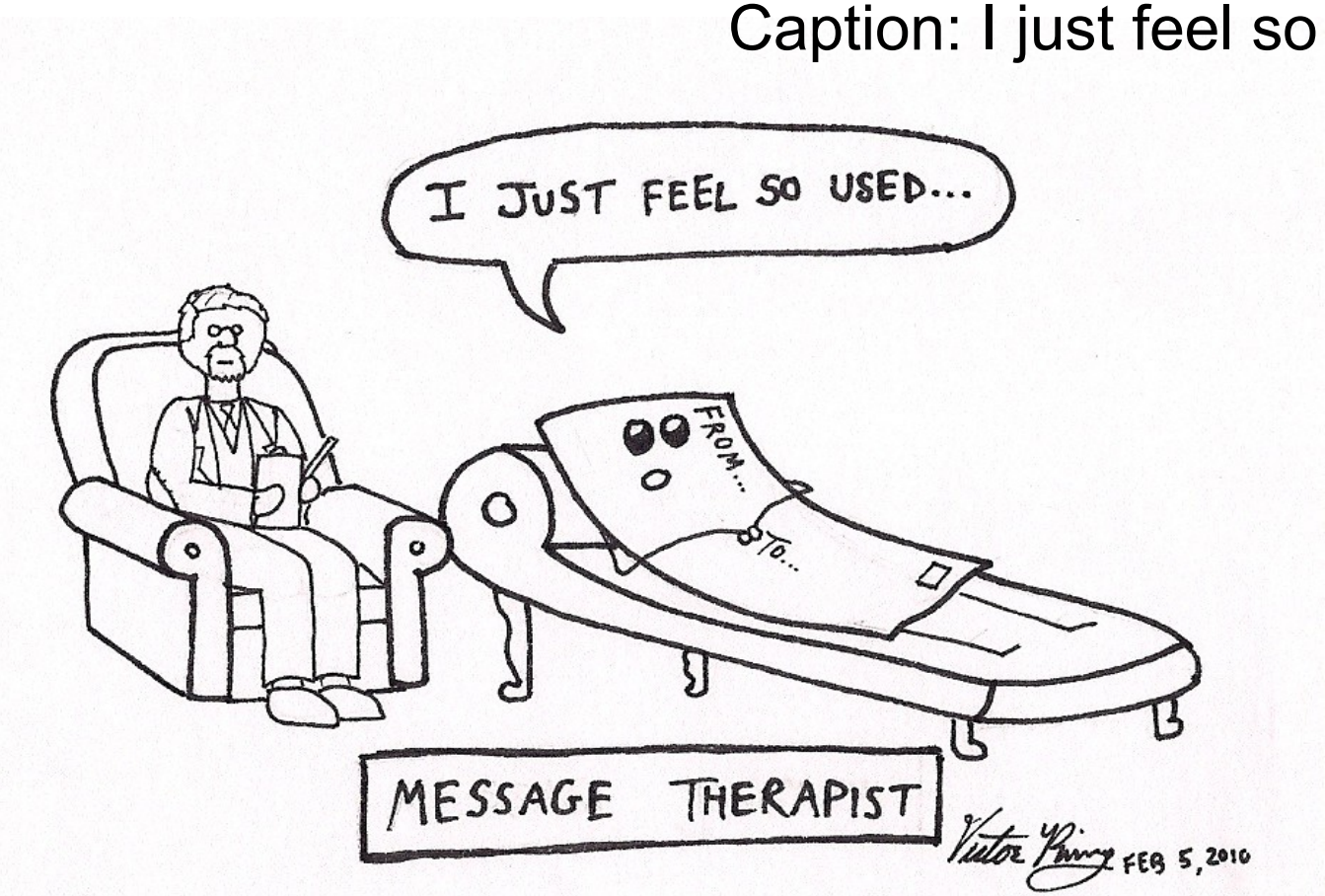
- The next 2 slides show artwork that he created for another presentation introducing MQ.
- He combined the theme of MQ and massage therapy.
- Some PMRs have a typo referring to “massage” instead of “message”.

Cartoon: MQ is not “massage queueing”



Cartoon: Message Therapist - 1

Caption: I just feel so used ...

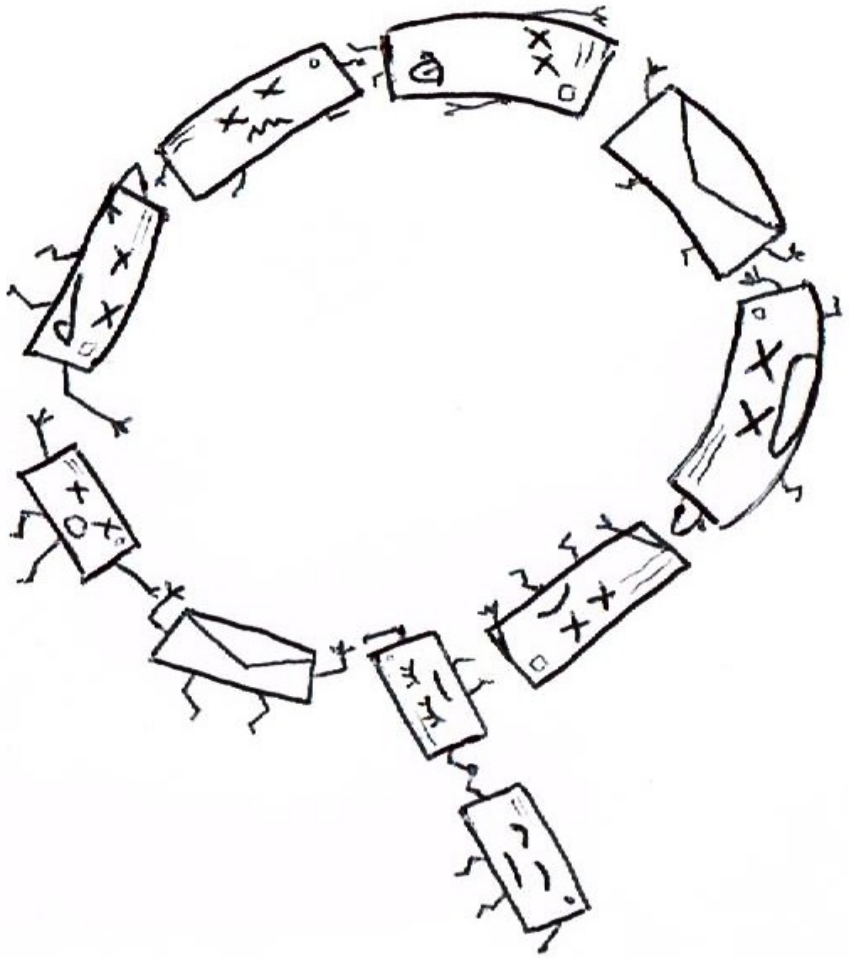


Undeliverable messages - DLQ

- Messages that fail to arrive to the final destination queue are placed in:
  - The Dead Letter Queue (DLQ), if defined.
- There is no default DLQ.
  - The MQ Administrator needs to define a DLQ
- If a message is destined to the DLQ but if there is no DLQ:
  - If message is not persistent, it is discarded
  - If message is persistent, channel is stopped. Message is NOT discarded.

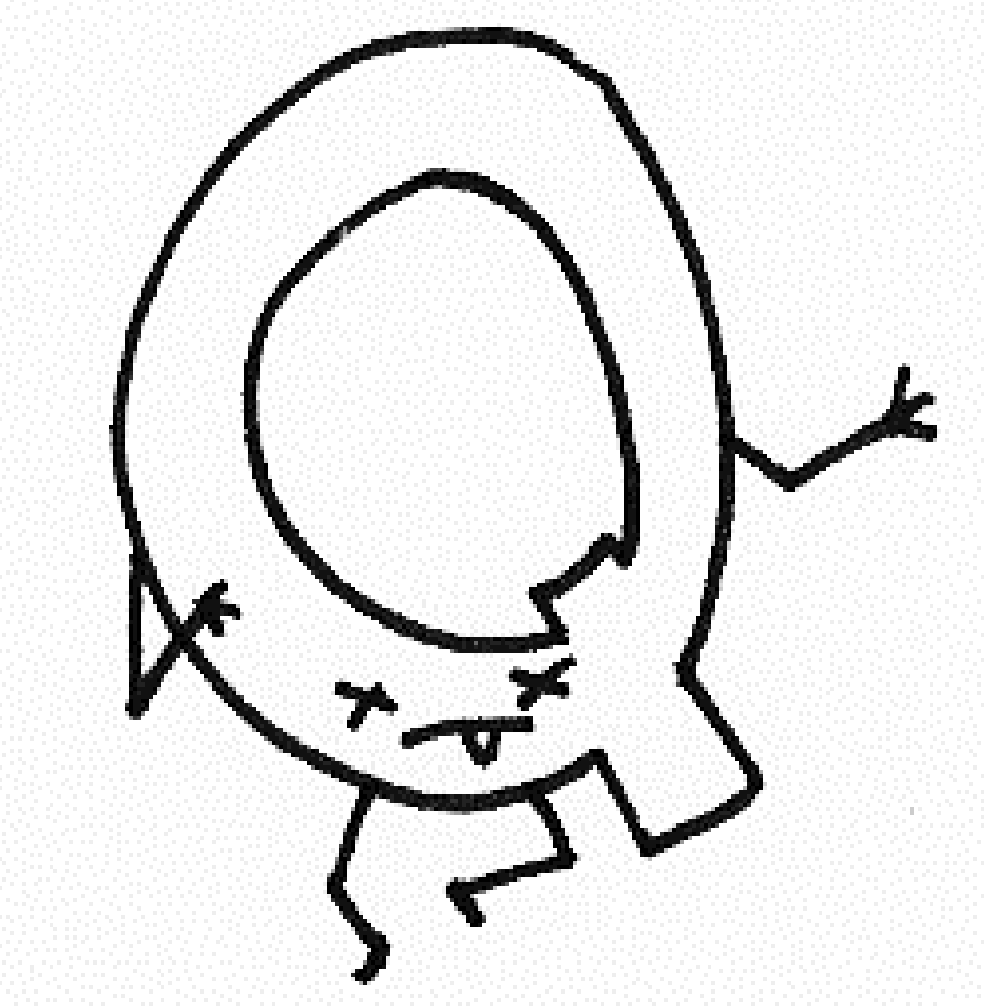
Undeliverable messages - DLQ

- It is important to understand that the Dead Letter Queue is a queue of dead letters, or dead messages ...



Undeliverable messages - DLQ

- ... and not a Letter Q that is dead.
- “dead” Letter Q
- Smiley: ;-)



Typical scenario - message arrives fine

- Let's study a detailed example in which the message arrives fine.

- Application connects to QJ in QMGR1 and puts a message.
- QJ in QMGR1 is a remote queue definition that references QJ in the remote QMGR2
- The message is transmitted from QJ in QMGR1 via the transmission queue in QMGR1 that goes to QMGR2.
- When the message arrives in QMGR2, the message is successfully delivered to queue QJ.

Notes: Overview of the components of distributed queuing

notes

The following was copied from the WSTE:

A Day in the Life of a WebSphere MQ Transmission Queue

http://www-01.ibm.com/support/docview.wss?uid=swg27021403

It provides detailed examples, using the theme of Romeo and Juliet

In this example, the putting application and the getting application are connected to different queue managers, two MCAs and their associated network connection are involved in the transfer, as shown in the figure in the previous page. In this case, the target queue is considered to be a remote queue to the putting application.

The sequence of events is as follows:

1. The putting application issues MQOPEN and MQPUT calls to put messages to the target queue.

2. During the MQOPEN call, the name resolution function detects that the target queue is not local, and decides which transmission queue is appropriate. Thereafter, on the MQPUT calls associated with the MQOPEN call, all messages are placed on this transmission queue.

Notes: Overview of the components of distributed queuing - 3

notes

3. The sending MCA gets the messages from the transmission queue and passes them to the receiving MCA at the remote computer.

4. The receiving MCA puts the messages on the target queue, or queues.

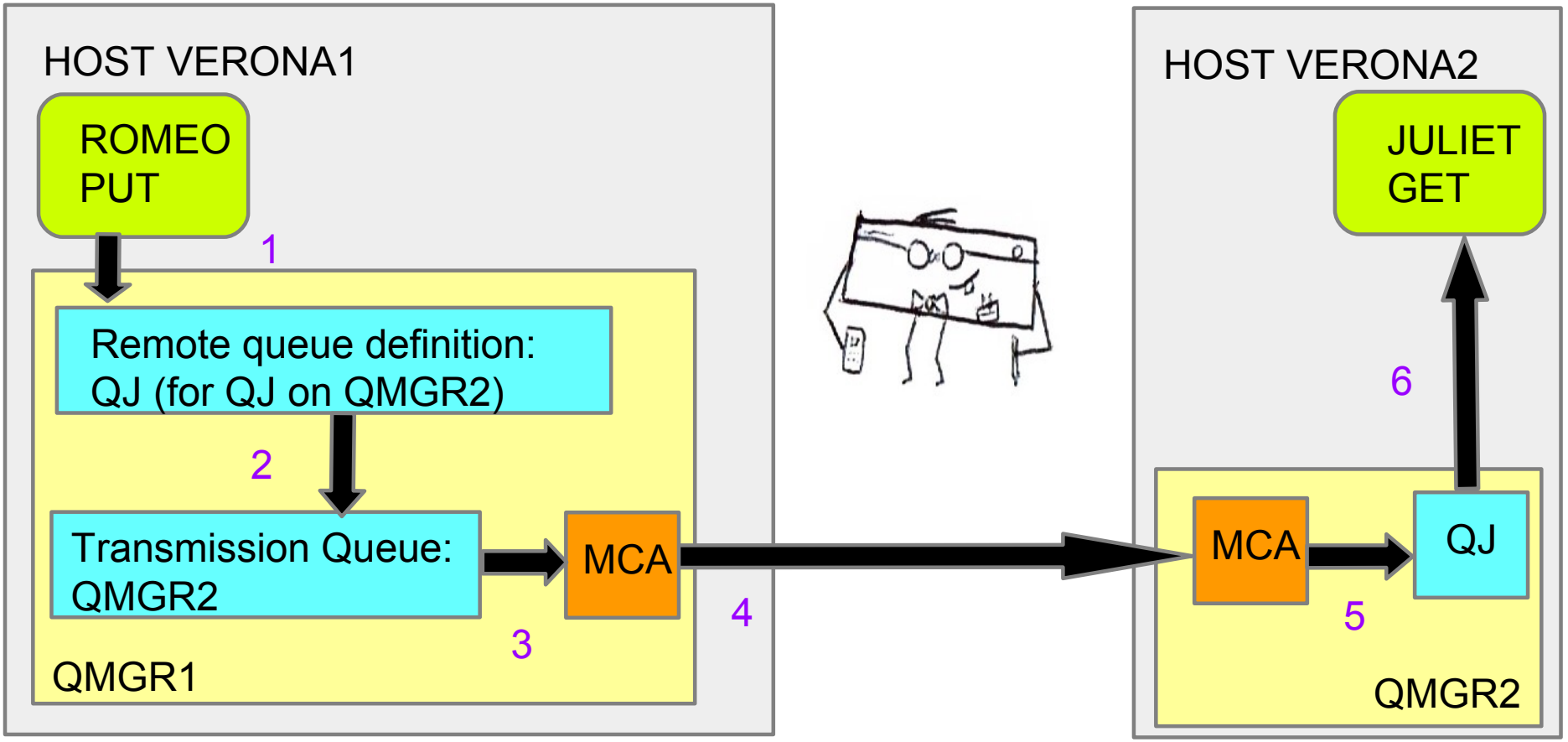
5. The getting application issues MQOPEN and MQGET calls to get the messages.

Note: Only step 1 and step 5 involve application code; steps 2 through 4 are performed by the local queue managers and the MCA programs. The putting application is unaware of the location of the target queue, which could be in the same processor, or in another processor on another continent.

The combination of sending MCA, the network connection, and the receiving MCA, is called a message channel, and is inherently a unidirectional device.

Normally, it is necessary to move messages in both directions, and two channels are set up for this, one in each direction. The following technote has concrete examples:
http://www-01.ibm.com/support/docview.wss?uid=swg21470997
Commands to setup both ways communication between 2 queue managers via Sender and Receiver channels

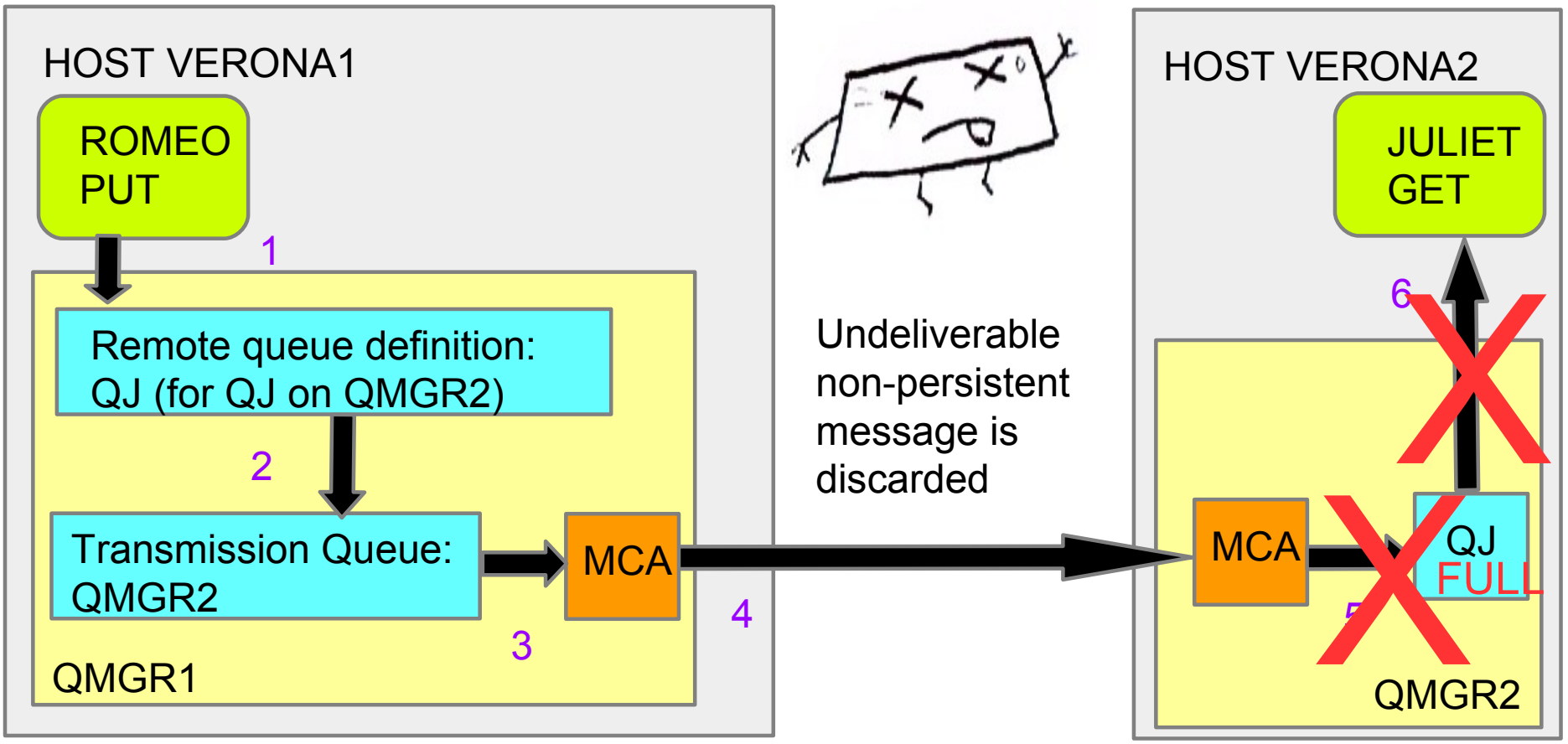
Sending a message via XMITQ-MCA



Typical scenario - destination queue is full

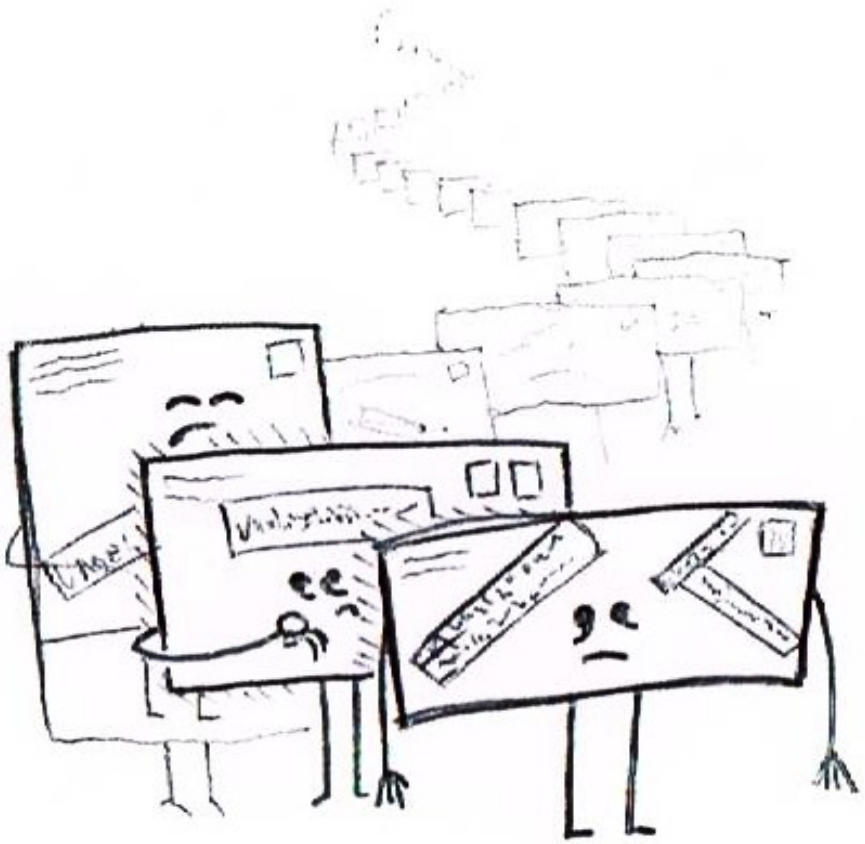
- Now, let's explore what happens when the destination queue is full and **there is no DLQ**
- Application connects to Q1 in QMGR1 and puts a message.
- Q1 is a remote queue definition that references Q2 in the remote QMGR2
- The message is transmitted from Q1 in QMGR1 via the transmission queue in QMGR1 that goes to QMGR2.
- But when the message arrives in QMGR2, the queue Q2 is full and the message cannot be delivered.
- Notice that there were NO errors reported when the message was received by either queue manager.

Destination queue is full - no DLQ

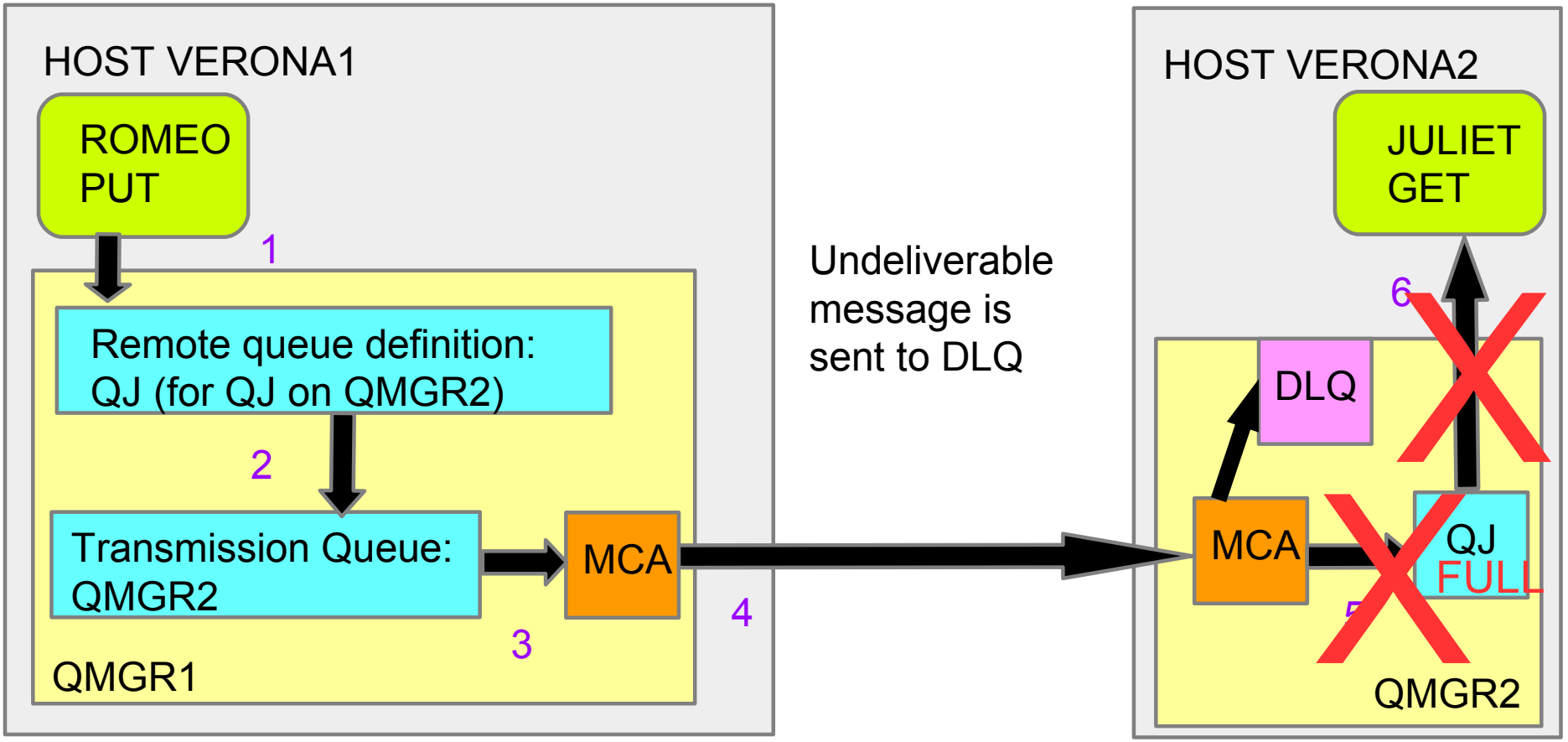


Typical scenario - with DLQ

- In this new scenario, there is a DLQ defined.
- Thus, when the destination queue is full, the dead message is sent to the DLQ



Destination queue is full - with DLQ



How to find the reason code in a DLQ

- This topic is explained in the following document:
- **Browsing Message Fields, Properties and Contents in WebSphere MQ**
- http://www-01.ibm.com/support/docview.wss?uid=swg27018059

- The following slides were copied from the above presentation.

Dead Letter Queue – amqsbcg 1

- When messages are placed onto the Dead Letter Queue (such as SYSTEM.DEAD.LETTER.QUEUE), a special header is attached to the message giving information on why the message has been put onto the queue (reason code).
- You can use the sample amqsbcg to find out the reason code:
- amqsbcg SYSTEM.DEAD.LETTER.QUEUE QMgrName

Dead Letter Queue – amqsbcg 2

- In the message header, notice: **Format : ‘MQDEAD ‘**

- StrucId : 'MD ' Version : 2
- Report : 0 MsgType : 8
- Expiry : -1 Feedback : 0
- Encoding : 273 CodedCharSetId : 819

- **Format : 'MQDEAD '**
- Priority : 0 Persistence : 0
- MsgId : X'414D5120415744545732303220202020945178452 01F4483'

Dead Letter Queue – amqsbcg 3

- In the message data, see the first line in the output.
- It has an eye catcher: DLH

```
****   Message       ****

 length - 697 bytes

00000000:  444C 4820 0000 0001 0000 07F3 4157 442E 'DLH ....... AWD.'
                               *********      ***
```

- Notice the value for the 9-13 bytes (5th and 6th pairs from the left).
  - In this case it is hexadecimal 0000 007F3 or 000007F3

Dead Letter Queue – amqsbcg 4

- **For Unix (except for Linux on x86 – Intel):**
- You can use the "mqrc" command to find out the meaning of 000007F3
- You have to compose the code to be as follows, in hexadecimal:
- $ mqrc 0x000007f3
- 2035 0x000007f3 MQRC_NOT_AUTHORIZED

Dead Letter Queue – amqsbcg 5

- Keep in mind that the following shortcuts will NOT work with mqrc:

- $ mqrc x'7F3'
- No matching return codes

- $ mqrc x7F3
- No matching return codes

- $ mqrc x07F3
- No matching return codes

Dead Letter Queue – amqsbcg 6

- **For Windows and Linux for Intel:**
- You need to take into account the swapping of the bytes.
- The following is obtained from a DLQ in Windows.
- `00000000:  444C 4820 0100 0000 2508 0000 5133 2020 'DLH ....%...Q3  '`
  `byte2 byte1`

- You cannot use 0x25080000 because it is not a valid reason code:

- C:\> mqrc 0x25080000
  - No matching return codes

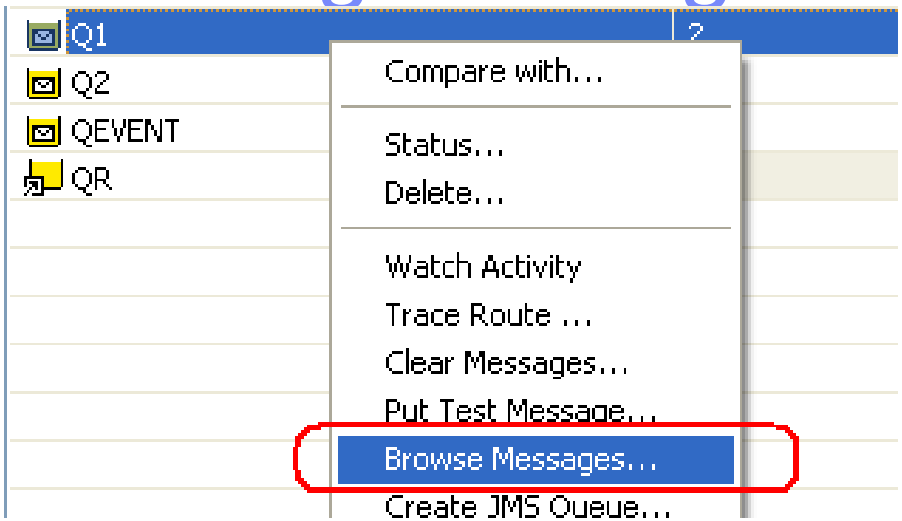
Dead Letter Queue – amqsbcg 7

- In Intel, you need to swap the bytes and reverse the order:

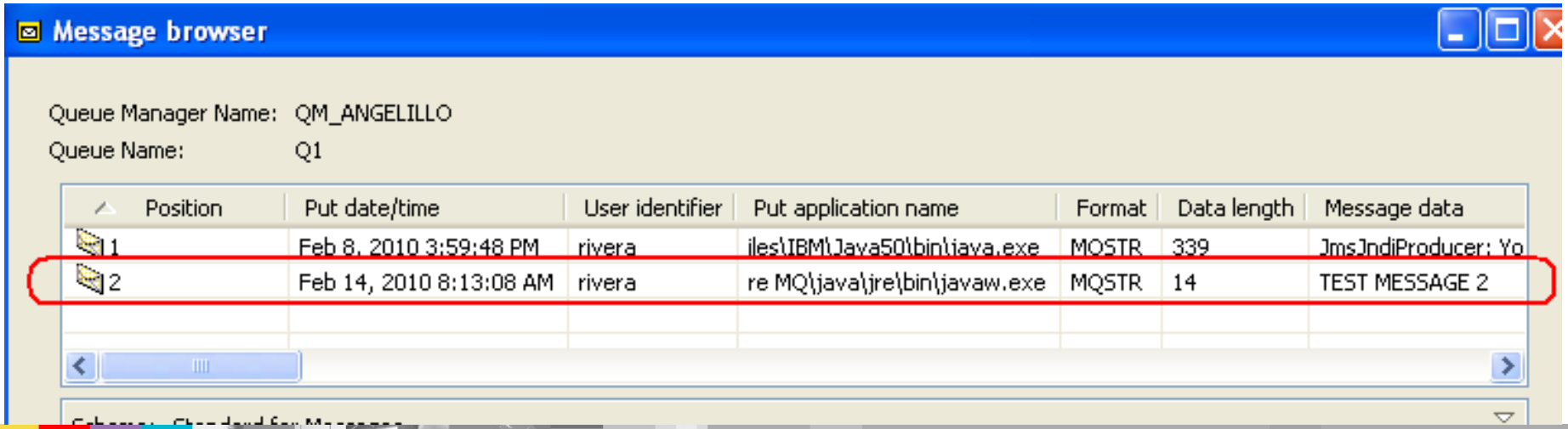
```
2508 0000  => 00 00 08 25
        ++----->>
      ++ --------->>
   ++ --------------->>
++ -------------------->>
```

- Thus, the reason code is:
  - 0x00000825
- C:\> mqrc 0x00000825
- 2085 0x00000825 MQRC_UNKNOWN_OBJECT_NAME

Browsing messages – Explorer

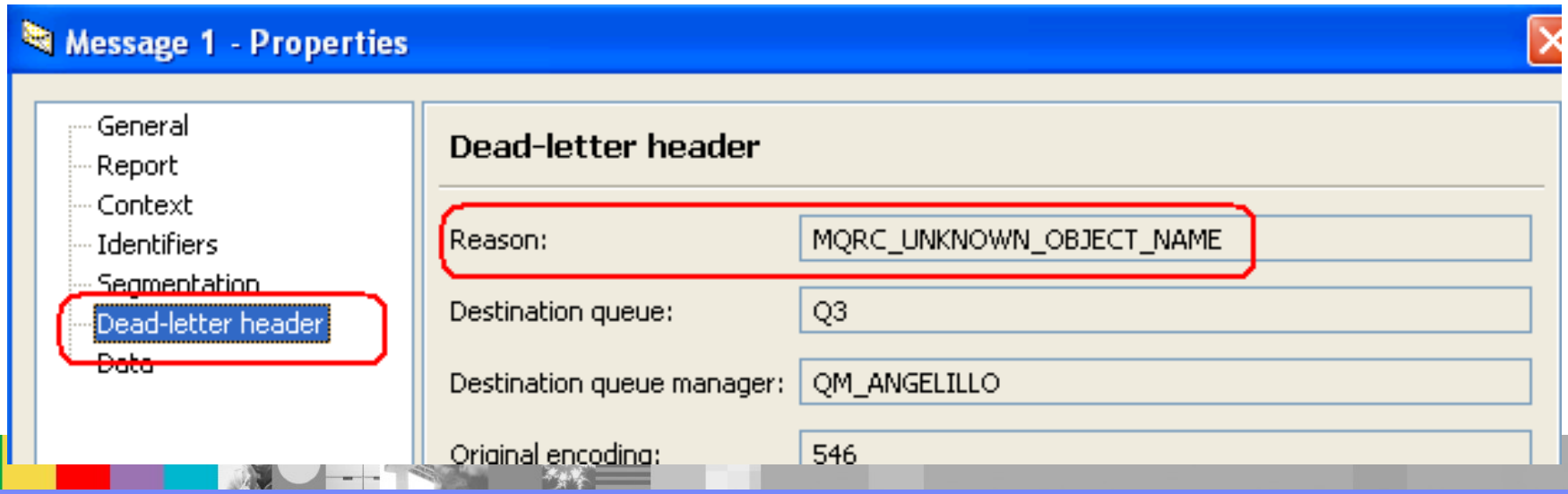


Select the desired queue, such as a DLQ, then right-click and select Browse Messages ...



Dead Letter Queue – Explorer

- It is far easier to find out the reason code by browsing a message from the DLQ via MQ Explorer.
- It handles the swapping of bytes according to platform.
- You can do reverse lookup via mqrc:
  - $ mqrc MQRC_UNKNOWN_OBJECT_NAME
- 2085 0x00000825



Basic troubleshooting

- Situation: You put successfully a message to a remote queue definition in QMGR1 and you expect the message to be received by QMGR2, but the message is not found in the expected queue
- The following slides provide some basic troubleshooting recommendations.

Notes: Basic troubleshooting

notes

Configuration for these concrete scenarios:
Windows 7: queue manager QM_ANG75, MQ 7.5.0.2
Linux SuSE 11: queue manager QM_VER75, QM 7.5.0.2

To expedite the full connectivity between these queue managers, I copied the commands from the following technote and I customized those commands and execute them in each queue manager.

http://www-01.ibm.com/support/docview.wss?uid=swg21470997
Commands to setup both ways communication between 2 queue managers via Sender and Receiver channels

C\:> crtmqm QM_ANG75
Remote Q definition: Q6_VER75 which points to local Q6 in QM_VER75
Transmission Queue: QM_VER75
At this time there is NO DLQ

$ crtmqm QM_VER75
Local Queue: Q6
  Attributes: PUT(DISABLED)

Notes: Basic troubleshooting

notes

At this time there is NO DLQ in the receiving queue manager
ALTER QMGR DEADQ('')
display qmgr DEADQ
AMQ8408: Display Queue Manager details.
QMNAME(QM_VER75) **DEADQ( )**

When dealing with a remote queue definition, the application was ABLE to deliver the message to the local queue manager
C:\>amqsput Q6_VER75 QM_ANG75
Sample AMQSPUT0 start
target queue is Q6_VER75
test-put-inhibited
Sample AMQSPUT0 end

Notice that there were NO errors reported in QM ANG75.
But the message did not arrive to its destination (Q6 at QM_VER75)

Let's do some troubleshooting along the whole line of communication.

Notes: Basic troubleshooting

notes

Start with the remote queue definition (RQD) in local queue manager QM_ANG75. Notice that this queue does not have a CURDEPTH attribute, because such definition does NOT have any physical storage directly associated to it.

Step 1: Use runmqsc to check for the RQD to confirm if it is properly defined. The fields to check are:
RQMNAME(QM_VER75)
RNAME(Q6)
XMITQ(QM_VER75)

display qremote(Q6_VER75)
AMQ8409: Display Queue details.
QUEUE(Q6_VER75) TYPE(QREMOTE)
PUT(ENABLED) **RQMNAME(QM_VER75)**
RNAME(Q6) SCOPE(QMGR)
XMITQ(QM_VER75)

Notes: Basic troubleshooting

notes

Step 2: Check if the message is located in the transmission queue
From Step 1, we see that the transmission queue is: XMITQ(QM_VER75)
Let's check the current depth:

display qlocal(QM_VER75) curdepth
AMQ8409: Display Queue details.
QUEUE(QM_VER75) TYPE(QLOCAL)
CURDEPTH(0)

A value of 0 for CURDEPTH indicates that there are no messages to be transmitted.

A value of 1 or greater indicates that the sender channel that gets the messages from the transmission queue is not working or that the receiver channel at the remote queue manager did not start.

Step 3: Let's find the sender channel associated with the transmission queue.
Notice that the command to display the details for the transmission queue will NOT show the sender channel that uses this transmission queue
display qlocal(QM_VER75)

Notes: Basic troubleshooting

notes

Instead, we need to query which is the channel that uses the transmission queue. We can take advantage of the WHERE clause (similar to an SQL query when using databases):

show me the channels that have a value in XMITQ that is equal to QM_VER75

```
display channel(*) where(XMITQ EQ QM_VER75)
AMQ8414: Display Channel details.
   CHANNEL(QM_ANG75.QM_VER75)          CHLTYPE(SDR)
   XMITQ(QM_VER75)
```

Now let's display the details on the sender channel:

```
display CHANNEL(QM_ANG75.QM_VER75)
AMQ8414: Display Channel details.
   CHANNEL(QM_ANG75.QM_VER75)          CHLTYPE(SDR)
    COMPHDR(NONE)                      COMPMSG(NONE)
   CONNAME(veracruz.raleigh.ibm.com(1455))
   USEDLQ(YES)                        USERID( )
   XMITQ(QM_VER75)
```

Notes: Basic troubleshooting

notes

The attributes that are relevant are:
CONNAME which indicate the hostname and port number of the remote queue manager.
USEDLQ (default is YES) ==> new attribute in MQ 7.5.
See: http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.ref.dev.doc/q109050_.htm
And http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.pro.doc/q002680_.htm

Step 4: Check the status of the channel
Notice that the previous command shows only the definition for the channel, but it does not tell us that status: is it running?
We need to use the command:
display chstatus(QM_ANG75.QM_VER75)
AMQ8417: Display Channel Status details.
CHANNEL(QM_ANG75.QM_VER75) CHLTYPE(SDR)
CONNAME(9.27.46.234(1455)) CURRENT
RQMNAME(QM_VER75) **STATUS(RUNNING)**
SUBSTATE(MQGET) XMITQ(QM_VER75)

Notice that the channel status is RUNNING

Notes: Basic troubleshooting

notes

Thus, at this point we need to reach the conclusion that the message that was placed into the RQD in the local queue manager QM_ANG75 was internally forwarded to the transmission queue QM_VER75 and that the sender channel QM_ANG75.QM_VER75 sent the message to the receiving queue manager CONNAME(veracruz.raleigh.ibm.com(1455))

Step 5: Let's check the destination queue in the receiving queue manager

Runmqsc QM_VER75
display qlocal(Q6) curdepth
AMQ8409: Display Queue details.
QUEUE(Q6) TYPE(QLOCAL)
CURDEPTH(0)

If the message was successfully sent from QM_ANG75 via the transmission queue and the sender channel, then it was received by QM_VER75, but the desired queue Q6 does not have it.
Hum, where is the message?

Notes: Basic troubleshooting

notes

Step 6: Let's see the Dead Letter Queue

Find out the name of the DLQ in the receiving queue manager:

```
display qmgr deadq
     4 : display qmgr deadq
AMQ8408: Display Queue Manager details.
   QMNAME(QM_VER75)                        DEADQ( )
```

The DEADQ field is empty, which means that there is no DLQ for this queue manager.

When dealing with non-persistent messages, if the message arrived to a queue manager, but if the message cannot be placed in the destination queue, and if there is no DLQ defined, then the message is DISCARDED!

Notes: Basic troubleshooting

notes

SCENARIO WITH DLQ - similar to the previous SCENARIO but with DLQ defined in destination queue manager

We are going to define and use a local queue in the destination queue manager to serve as the DLQ.

Hint: Many MQ Explorer users hide the SYSTEM* queues and thus, if you use as the dlq the SYSTEM.DEAD.LETTER.QUEUE, then it will be hidden and you may not notice if there are messages in the dlq.

Note: You can specify the DLQ during crtmqm:
$ crtmqm -u DLQ QM_VER75
Caveat: You must define the DLQ. The above parameter does NOT actually create the queue, it just tells the qmgr which is the name of the dlq.

```
runmqsc QM_VER75
  ## Define the DLQ
  define qlocal(DLQ) like(SYSTEM.DEAD.LETTER.QUEUE)
  ## Specify the DLQ for the queue manager
  alter qmgr DEADQ(DLQ)
```

Notes: Basic troubleshooting

notes

Repeat Steps 0 thru 5 from previous SCENARIO.

Step 6: Let's find out which is the Dead Letter Queue.

```
display qmgr deadq
AMQ8408: Display Queue Manager details.
   QMNAME(QM_VER75)                        DEADQ(DLQ)
```

Step 7: Display the current depth of DLQ

```
display qlocal(DLQ) curdepth
AMQ8409: Display Queue details.
   QUEUE(DLQ)                              TYPE(QLOCAL)
   CURDEPTH(1)
```

Notice that the value for CURDEPTH is 1, which indicates that a message arrived to the DLQ

Notes: Basic troubleshooting

notes

Step 8: How to find out the reason for the message to be placed in the DLQ.

```
$ amqsbcg DLQ QM_VER75
****Message descriptor****
  StrucId  : 'MD  '  Version : 2
  Report   : 0  MsgType : 8
  Expiry   : -1  Feedback : 0
  Encoding : 546  CodedCharSetId : 1208
=>  Format : 'MQDEAD  '
  Priority : 0  Persistence : 0
...
****   Message     ****
00000000:  444C 4820 0100 0000 0308 0000 5136 2020          'DLH ........Q6  '
                                *********
```

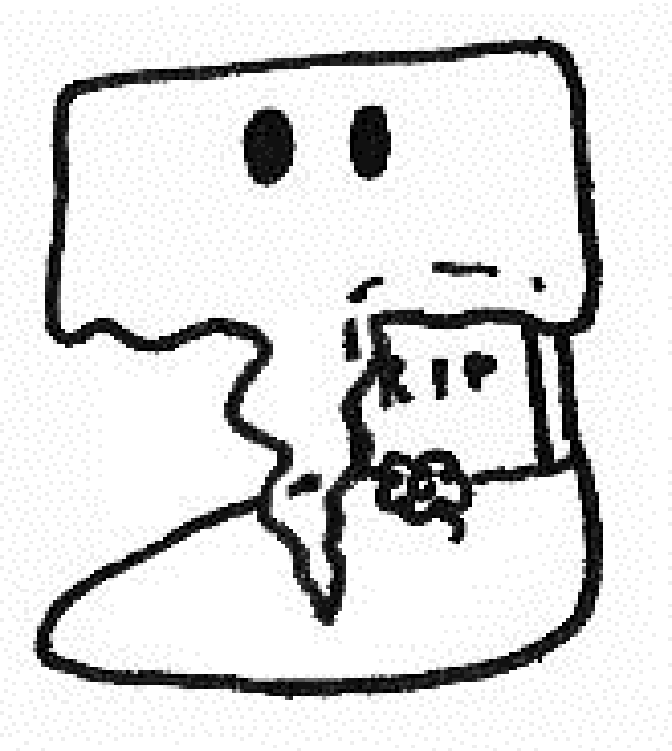
The host is Linux Intel: we need to swap the bytes: 03 08 00 00 => 00 00 08 03

```
$ mqrc 0x00000803
     2051  0x00000803  MQRC_PUT_INHIBITED
```

Dead Letter Queue handler

- Question: Now that the reason for sending the message to the DLQ has been identified ...
  what should we do to re-process the message from DLQ?

- Answer:
- Use the Dead Letter Queue Handler



Rising from the tomb

Notes: DLQ handler

notes

For more information see the following material:

http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg27008375
How to Automate Handling of WebSphere MQ Dead Letter Messages

http://www.ibm.com/developerworks/websphere/library/techarticles/1204_gupta/1204_gupta.html
Using the Dead Letter Queue Handler in WebSphere MQWebsphere MQ

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/index.jsp?topic=%2Fcom.ibm.mq.pla.doc%2Fq005720_.htm
WebSphere MQ > Planning > Designing a WebSphere MQ architecture > Handling undelivered messages with the dead-letter queue handler
IBM WebSphere MQ information center, Version 7.5: UNIX, Linux, Windows
An example DLQ handler rules table

Notes: DLQ handler

notes

Let's continue with the detailed example described earlier, in which the local queue Q6 in QM_VER75 had the attribute PUT(DISABLED) and the messages destined to Q6 were sent to the DLQ.

Now we want to have another local queue Q7 and we will run the DLQ Handler to move the messages from the dead letter queue to Q7.

Let's change to a directory where we can create the DLQ rules table file that will be used by the DLQ Handler. The host for this example is Linux:

Create a DLQ rules table file. There are no restrictions regarding the file name or suffix.

```
   cd /var/mqm
   vi dlq.rules
INPUTQM(' ')  INPUTQ(' ')
DESTQ(Q6) ACTION(FWD) FWDQ(Q7) FWDQM('') HEADER(NO)
```

There are only 2 lines: the first one is for control purposes and we need to explicitly indicate the names of the queue manager and of the dead letter queue.
The 2nd line will only handle those messages that were destined to Q6, forward them to the queue Q7, in the same queue manager and removing the MQDEAD header.

Notes: DLQ handler

notes

The DLQ handler is called runmqdlq and it will not return the prompt while it is executing.
You need to redirect as input, the rules table file.
This example invokes the handler with queue DLQ in queue manager QM_VER75.

```
$ runmqdlq DLQ QM_VER75 < /var/mqm/dlq.rules
10/02/2013  02:55:02 PM  AMQ8708: Dead-letter queue handler started to process
INPUTQ(DLQ).
```

You will need to terminate it with Ctrl-Z

```
^Z
[2]+  Stopped                 runmqdlq DLQ QM_VER75 < /var/mqm/dlq.rules
```

After you stop it, ensure that it is really dead. You may need to issue “kill -9” too.

```
$ ps -ef | grep runmqdlq
mqm      30311 25457  0 15:07 pts/0    00:00:00 runmqdlq DLQ QM_VER75
rivera   30777 25457  0 15:07 pts/0    00:00:00 grep runmqd

$ kill -9 30311
[1]+  Killed                  runmqdlq DLQ QM_VER75 < /var/mqm/dlq.rules
```

Notes: DLQ handler

notes

Now let's take a look at both queues in this scenario and check for the current depth.
You can use the runmqsc command such as:
display ql(Q7) curdepth
AMQ8409: Display Queue details.
QUEUE(Q7) TYPE(QLOCAL)
CURDEPTH(1)

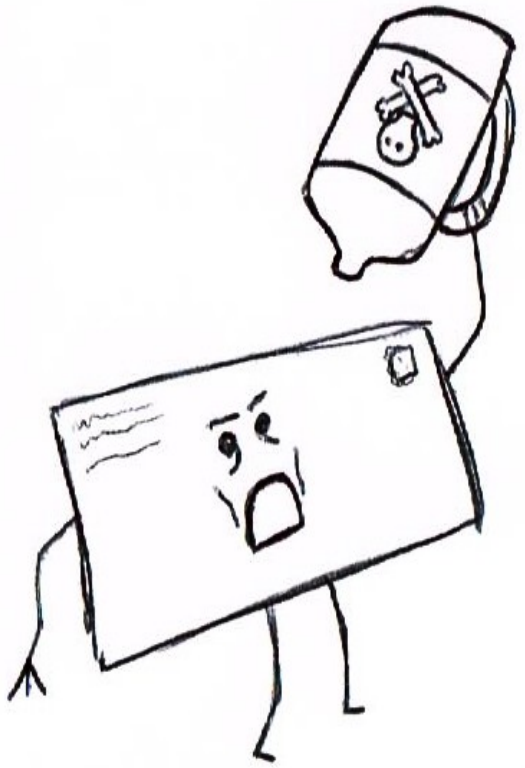
Before running runmqdlq the current depth was:
Q6 -> 0
Q7 -> 0
DLQ -> 1

After runmqdlq:
Q6 -> 0
Q7 -> 1
DLQ -> 0

The message is now back “alive” in Q7 !

Poison Message

- A 'poison message' is one which cannot be processed by the receiving application and the message is returned back to the queue:
- Example: wrong format of the message, or a player in the transaction (a database) is not ready
- The application will get the message again and will try to process it.
- If it cannot process it, then the message is returned back, potentially causing an infinite loop.



Poison Message - C versus JMS

- When using the C-based MQ API, the application is responsible for handling poison messages.

- In contrast, when using JMS, the MQ client code will handle poison messages:
- It reads the BackoutCount field of the message.
- If this field is greater than the attribute BOTHRESH (backout threshold) of the reading queue,
- then the MQ JMS client will move the message to the queue indicated by the queue attribute BOQNAME (backout queue)

Poison Message in WAS

- The most common scenario is when using a J2EE application server such as WAS, which uses the MQ JMS client code.

- For more information see:
- **How WebSphere Application Server V6 handles poison messages**
- http://www.ibm.com/developerworks/websphere/library/techarticles/0803_titheridge/0803_titheridge.html

- **Using WebSphere MQ V7 as JMS Provider for WebSphere Application Server V7**
- http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg27016505

Poison Message in WAS

- The JMS client opens the queue and checks these attributes:
- BOTHRESH (backout threshold) - default is 0
- BOQNAME (backout queue) - default is null

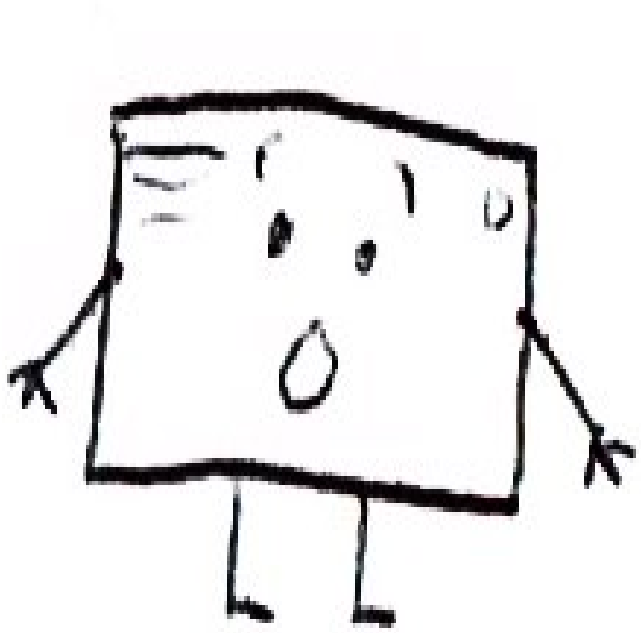
- You can use the following runmqsc command:
- display ql(Q1) BOQNAME BOTHRESH
- AMQ8409: Display Queue details.
- QUEUE(Q1) TYPE(QLOCAL)
- **BOQNAME( )** **BOTHRESH(0)**

Poison Message in WAS - default

- The MQ JMS client gets a message from the queue. Let's assume that it is the first time.
- The BackoutCount field is 0.
- The MQ JMS client checks if this field is greater than the BackoutThreshold, which is 0.
- The answer is NO.
- Then the message is sent to the receiving application, a Message Driven Bean (MDB).
- The MDB rejects the message.

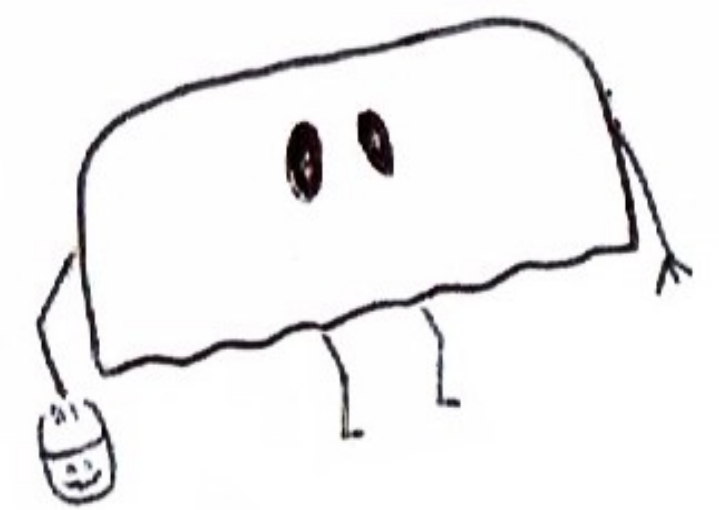
Poison Message in WAS - default

- Because the MDB rejected the message, the MQ JMS code puts back the message into the queue.
- The BackoutCount of the message is incremented from 0 to 1.
- Assuming default priorities, this message is placed back at the the first position of the queue.
- That is, at the next GET, this message will be read.



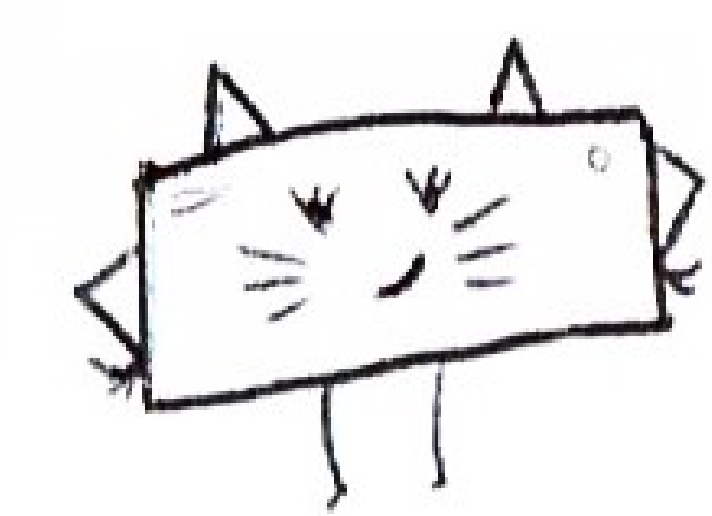
Poison Message in WAS - default

- The MQ JMS client reads the message again.
- It checks that the BackoutCount is 1.
- This field is greater than the Backout Threshold (0).
- At this point the message is considered to be undeliverable and it is now a poison message.
- If it is returned back to the queue. This could cause an infinite loop!



Poison Message in WAS - default

- The MQ JMS client will try to move the poison message from the originating queue into the queue specified by the Backout Queue field of the queue.
- Because it is null, then the MQ JMS client will stop the WAS Listener Port or the Activation Specification, and this will require an administrator to manually handle the poison message (delete it or move it)



Poison Message in WAS - configured

- The MQ and WAS administrators can customize the environment to automatically handle poison messages.
- In WAS, the number of retries for the Listener Port (LP) or Activation Specification (AS) can be customized. Let's assume that is 5.
- In MQ, the queue can be altered to indicate the desired values:
- Backout Threshold (1)
  Backout Queue (Q1_BO)

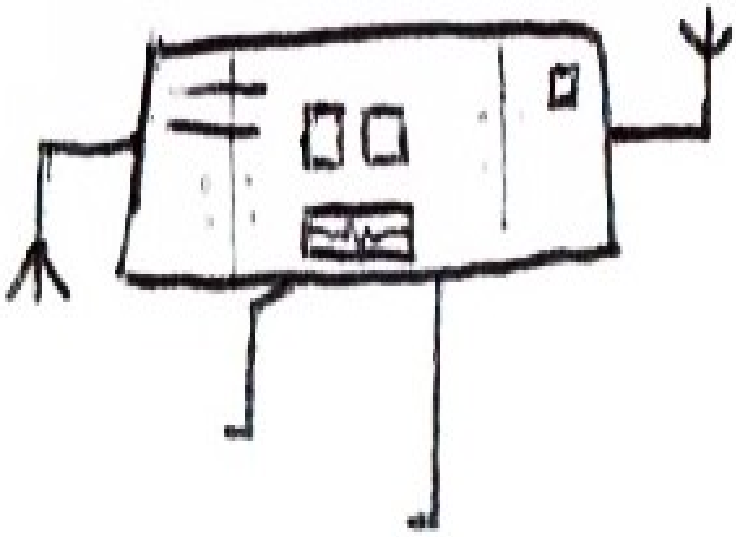
Poison Message in WAS - configured

- Let's repeat the scenario.
- The MQ JMS client reads the message (BackoutCount 0).
- The field is less than the BackoutThreshold (1).
- The message is sent to the MDB.
- The MDB rejects it.
- The MQ JMS client puts back the message and the BackoutCount field is set to 1.
- The retry counter for the LP or AS is incremented from 0 to 1.

Poison Message in WAS - configured

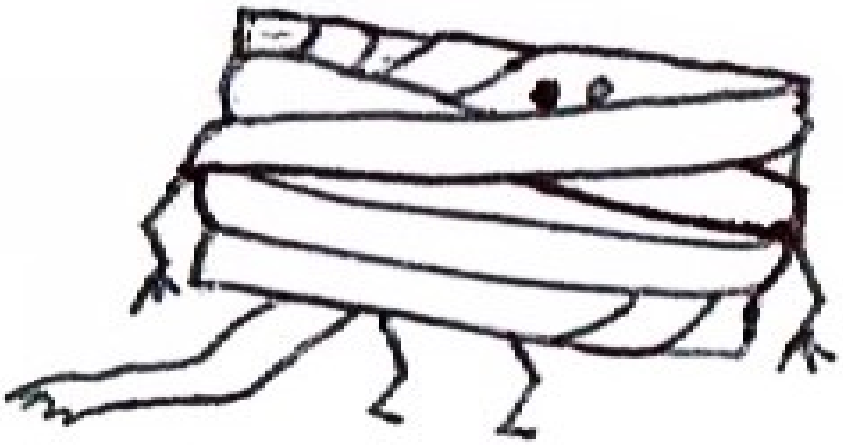
- The LP or AS check if the retry counter (1) is greater than the maximum retries (5).
- The answer is NO and proceeds to handle messages.

- The MQ JMS client reads the message again
- BackoutCount = 1
- The field is equal than the BackoutThreshold (1).



Poison Message in WAS - configured

- The message is sent to the MDB.
- The MDB rejects it.
- The MQ JMS client puts back the message and the BackoutCount field is set to 2.
- The retry counter for the LP o AS is incremented from 1 to 2.

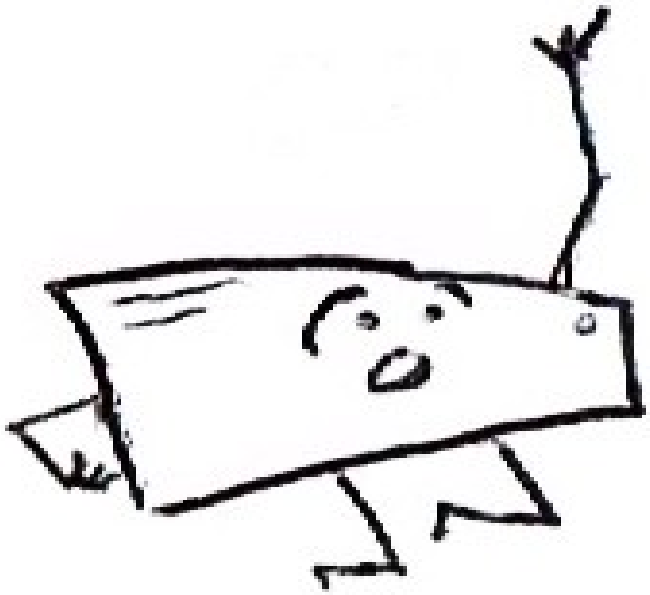


Poison Message in WAS - configured

- The LP or AS check if the retry counter (2) is greater than the maximum retries (5).
- The answer is NO and proceeds to handle messages.

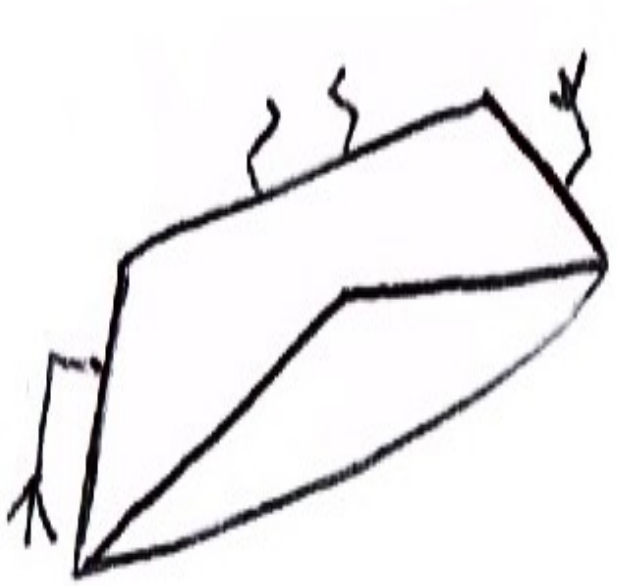
Poison Message in WAS - configured

- The MQ JMS client reads the message again.
- BackoutCount = 2.
- The field is greater than the BackoutThreshold (1).
- The message is NOT sent to the MDB.



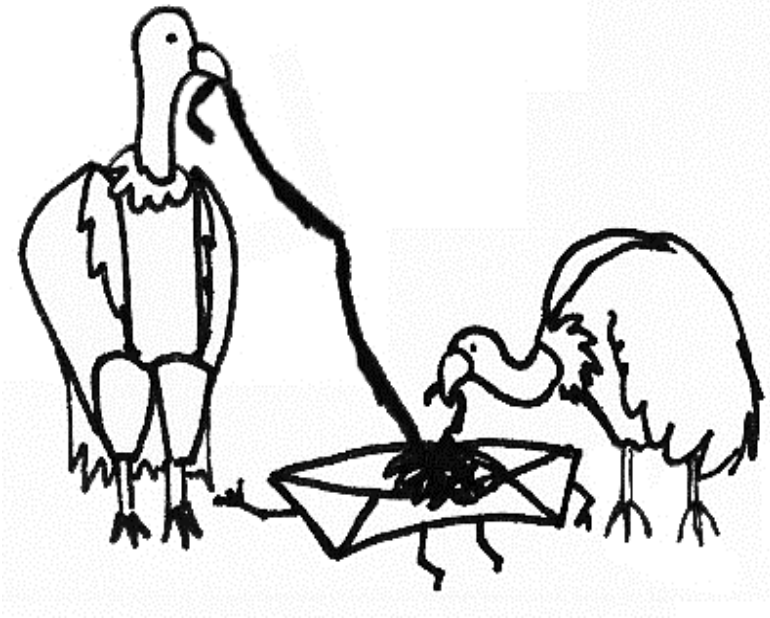
Poison Message in WAS - configured

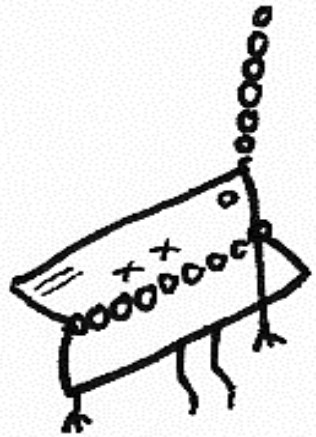
- The MQ JMS client:
- Deletes the message from the reading queue (Q1)
- Moves the message into the queue defined in the BOQNAME: Q1_BO
- Resets the BackoutCount field to 0.
- Resets the retry counter to 0 for the LP or AS in WAS



MQ Halloween

- There are MQ scavenger threads ...

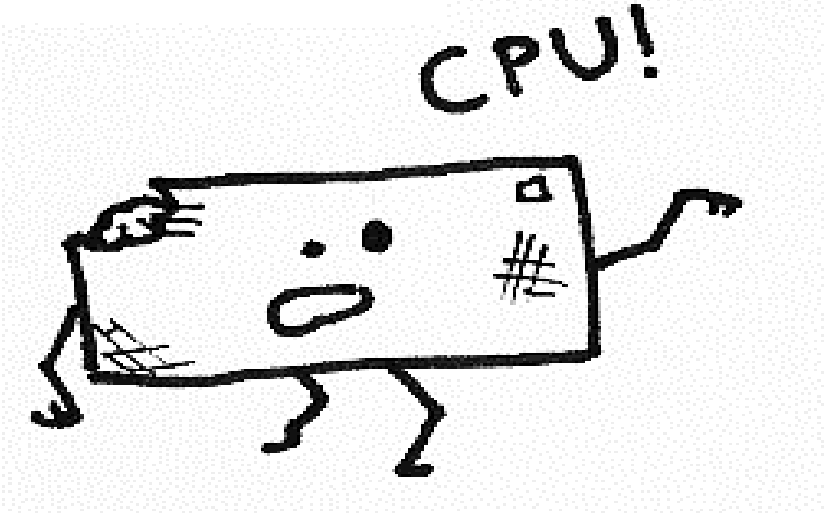




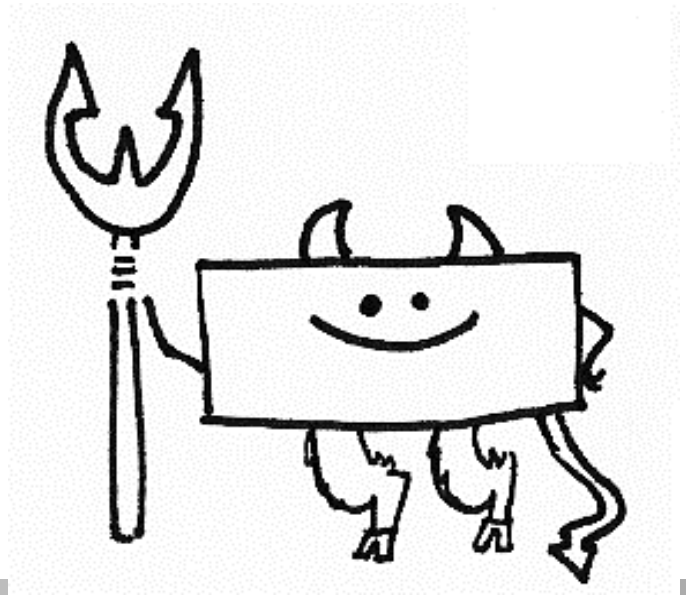
- ... and hanging processes!

MQ Halloween

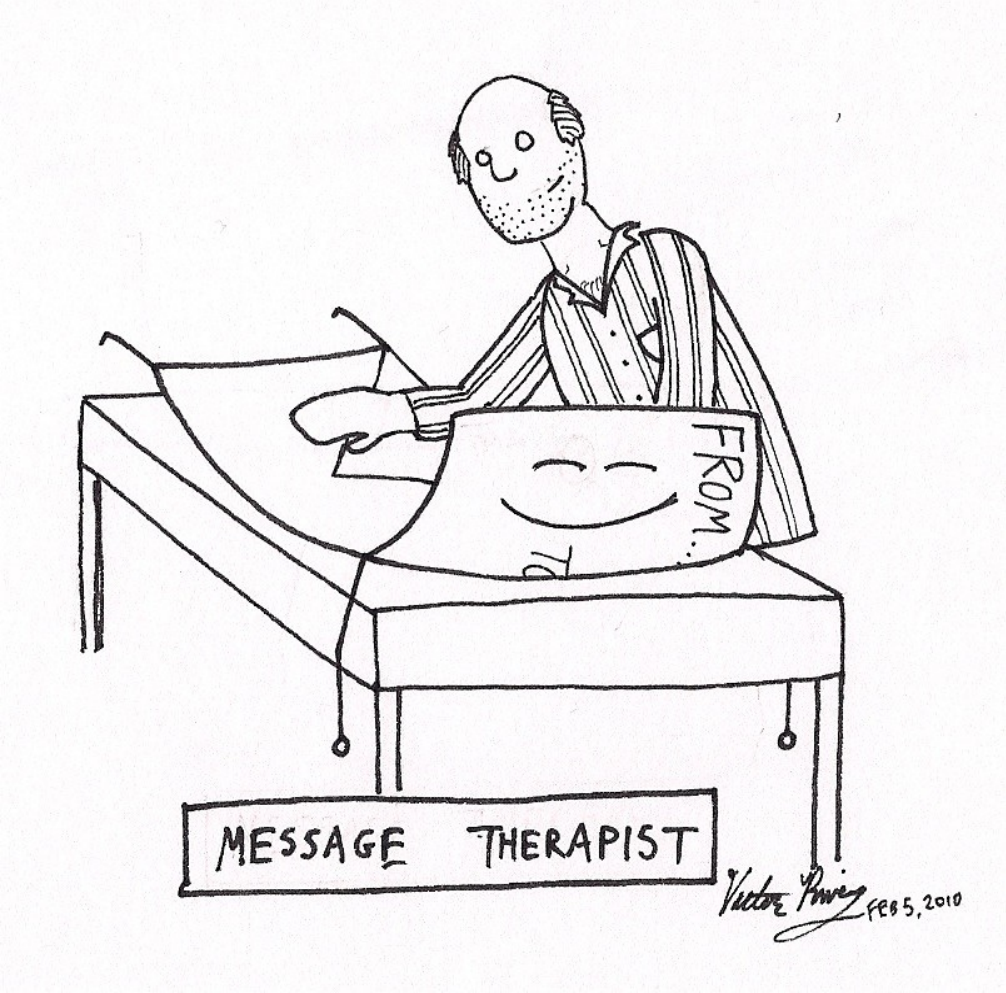
- Zombie processes ...



- ... and daemon processes!



Cartoon: Message Therapist - 2



Additional WebSphere Product Resources

- Learn about upcoming WebSphere Support Technical Exchange webcasts, and access previously recorded presentations at:
  http://www.ibm.com/software/websphere/support/supp_tech.html
- Discover the latest trends in WebSphere Technology and implementation, participate in technically-focused briefings, webcasts and podcasts at:
  http://www.ibm.com/developerworks/websphere/community/
- Join the Global WebSphere Community:
  http://www.websphereusergroup.org
- Access key product show-me demos and tutorials by visiting IBM® Education Assistant:
  http://www.ibm.com/software/info/education/assistant
- View a webcast replay with step-by-step instructions for using the Service Request (SR) tool for submitting problems electronically:
  http://www.ibm.com/software/websphere/support/d2w.html
- Sign up to receive weekly technical My Notifications emails:
  http://www.ibm.com/software/support/einfo.html

Connect with us!

1. **Get notified on upcoming webcasts**
   Send an e-mail to wsehelp@us.ibm.com with subject line “wste subscribe” to get a list of mailing lists and to subscribe
2. **Tell us what you want to learn**
   Send us suggestions for future topics or improvements about our webcasts to wsehelp@us.ibm.com
3. **Be connected!**
   Connect with us on Facebook
   Connect with us on Twitter

Questions and Answers