IBM WebSphere MQ V7 System Administration

(Course code WM201 / VM201)

Student Exercises Guide

ERC 1.0

WebSphere Education

Trademarks

IBM® is a registered trademark of International Business Machines Corporation.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX®	CICS®	FFST™
First Failure Support Technology™	HACMP™	i5/OS®
iSeries®	MQSeries®	MVS™
Notes®	OMEGAMON®	OS/400®
QMF™	RACF®	SP™
SupportPac™	System z™	Tivoli®
TXSeries®	VSE/ESA™	VTAM®
WebSphere®	z9™	z10™
z/OS®	zSeries®	

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux® is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows and Windows Vista are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX® is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

October 2008 edition

Contents

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM® is a registered trademark of International Business Machines Corporation.

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX®	CICS®	FFST™
First Failure Support Technology™	HACMP™	i5/OS®
iSeries®	MQSeries®	MVS™
Notes®	OMEGAMON®	OS/400®
QMF™	RACF®	SP™
SupportPac™	System z™	Tivoli®
TXSeries®	VSE/ESA™	VTAM®
WebSphere®	z9™	z10™
z/OS®	zSeries®	

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux® is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows and Windows Vista are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX® is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Exercise 1. Working with queues

Estimated time

01:00

What this exercise is about

In this exercise, you start working with control and WebSphere MQ script commands through the command line interface and script command files.

What you should be able to do

At the end of this exercise, you should be able to:

- Use crtmqm to create a queue manager
- Use strmqm to start a queue manager
- Use runmqsc to create queue manager objects
- Use amqsput to put test messages onto a queue
- Use amgsgbr and amqsbcg to browse messages on a queue
- Use amqsget to get messages from a queue

Introduction

First, you create and start a queue manager. Then you enter WebSphere MQ commands on the command line and using a script file.

To access put, browse, and get messages, use the WebSphere MQ sample programs. The following is a brief description of the programs you may need to use. These programs are invoked from a command prompt and accept two parameters. The first parameter is required and is the name of a queue. The second parameter is optional and is the name of a queue manager. If the second parameter is omitted, the default queue manager is assumed. Both parameters are case-sensitive.

- amqsput - This program connects to the queue manager and opens the queue for output, reads lines of text from the standard input device, generates a message from each line of text, and puts the messages on the named queue. After reading a null line or the

EOF character from the standard input device, it closes the queue, and disconnects from the queue manager. The command to start the program is `amqsput queuename queuemanagername,` for example, `amqsput Q1 QM1`.

- amqsget - This program is invoked the same way as amqsput and has the same parameter structure. The program connects to the queue manager, opens the queue for input, gets all the messages from the queue, writes the text within the message to the standard output device, waits 15 seconds (60 seconds if there is no message at the start) in case any more messages are put on the queue and, if none arrive, closes the queue and disconnects from the queue manager. The command to start the program is `amqsget queuename queuemanagername`, for example `amqsget Q2 QM1`
- amqsbcg - This program is invoked the same way as amqsput or amqsget and has the same parameter structure. The program connects to the queue manager, opens the queue for browsing, browses all the messages on the queue, and writes their contents, in both hexadecimal and character format, to the standard output device. It also displays, in a readable format, the fields in the message descriptor for each message. It then closes the queue and disconnects from the queue manager. The command to start the program is `amqsbcg queuename queuemanagername`, for example `amqsbcg Q3 QM1`.

Requirements

- The appropriate WebSphere MQ product installed on the system.
- The sample programs amqsput, amqsbcg, and amqsget are invocable from any position within the file system hierarchy.
- A text editor

Exercise instructions

Step 1: Create and start a queue manager

__ 1. Create a queue manager. Call it QMC##, where ## is your team or machine number. Use this queue manager for all steps in this exercise.

__ 2. Start the queue manager.

__ 3. Create the system sample queues.

Step 2: Use the WebSphere MQ commands interactively

__ 1. Use the control command runmqsc to perform the following tasks. Enter all the WebSphere MQ commands interactively and display the results at the terminal.

- __ a. Display all the attributes of the queue manager.
- __ b. List the names of all the queues whose names begin with the characters 'SYSTEM'.
- __ c. Create a local queue QL.A. Its definitions should include a text description.
- __ d. Display all the attributes of the queue.
- __ e. Change the maximum number of messages allowed on the queue to 1000.
- __ f. Again, display the queue attributes. Was the modification successful and is the queue description unchanged? Type DISPLAY QL(QL.A)
- __ g. Define a second local queue QL.B with a text description.
- __ h. Display all the attributes of the queue. Type DISPLAY QL(QL.B)
- __ i. Change the maximum number of messages allowed on the queue to 2000 this time using DEFINE with REPLACE instead of the ALTER command.
- __ j. Display again the queue attributes of QL.B. Was the creation of the queue successful and is there a queue description?

__ 2. Exit the runmqsc mode.

Step 3: Use the WebSphere MQ commands in a command file

__ 1. Prepare a WebSphere MQ command file with the same commands you entered interactively in Step 2. Call the file 'exer1.txt'.

__ 2. Process the command file and direct the results to the file report. Have a look at the file containing the results.

Step 4: Use the sample programs

__ 1. Put messages on the local queue QL.A.

__ 2. Use one of the utilities mentioned in the introduction to this exercise to browse the messages of the queue. Direct the output to a file (or pipe to a paging program like more) and look at the results.

__ 3. Use one of the utilities mentioned in the introduction to this exercise to get the messages from the queue to make the queue empty.

Step 5: Work with alias queues

__ 1. Create an alias queue QA.A which resolves to your local queue QL.A.

__ 2. Inhibit put requests on the alias queue.

__ 3. Change QL.B to inhibit put requests on your local queue QL.B.

__ 4. Create an alias queue QA.B which resolves to your local queue QL.B.

__ 5. Exit the runmqsc mode.

__ 6. Try to put messages on both alias and local queues using the sample program.

__ 7. Reenable QL.B for put requests

End of exercise

Optional exercise instructions

Step 1: Working with queues

__ 1. Put some messages on your queues QL.A and QL.B.

__ 2. Show the CURDEPTH of all the queues you have created.

__ 3. Define a new local queue QL.X with the attributes of QA.A.

__ 4. Error? Now use the attributes of QL.A.

__ 5. Put some messages on QL.X.

__ 6. Change the description of QL.A.

__ 7. Display the DESCR attribute of all your queues.

__ 8. Is there any change?

__ 9. Clear queue QL.A.

__ 10. Clear queue QA.A.

__ 11. Were you successful? Give the reason why.

__ 12. Delete QL.X.

__ 13. Was it successfully deleted? If not, modify the command.

End of exercise

Exercise instructions with hints

Step 1: Create and start a queue manager

__ 1. Create a queue manager and call it QMC##, where ## is your team or machine number. Use this queue manager for all steps in this exercise. Type **crtmqm QMC##**

Note: The queue manager name is case-sensitive.

__ 2. Start the queue manager. Type **strmqm QMC##**

__ 3. Create the system sample queues. These objects are defined to be used by the WebSphere MQ sample programs. Use the following depending on your system:

AIX

```
runmqsc QMC## < /usr/mqm/samp/amqscos0.tst
```

Windows

```
runmqsc QMC## < "C:\Program Files\IBM\Websphere
MQ\tools\mqsc\samples\amqscos0.tst
```

SUN

```
runmqsc QMC## < /opt/mqm/samp/amqscos0.tst
```

Step 2: Use the WebSphere MQ control commands interactively

Enter all the WebSphere MQ commands interactively and display the results at the terminal.

__ 1. Use the control command runmqsc to perform the following tasks. Type **runmqsc QMC##**

__ a. Display all the attributes of the queue manager. Type **DISPLAY QMGR**

QMGR is a literal and is not replaced with your queue manager name.

__ b. List the names of all the queues whose names begin with the characters 'SYSTEM'. Type **DISPLAY Q(SYSTEM*)**

__ c. Create a local queue QL.A. Its definitions should include a text description. Type **DEFINE QL(QL.A) DESCR('QL.A Text')**

__ d. Display all the attributes of the queue. Type **DISPLAY QL(QL.A)**

__ e. Change the maximum number of messages allowed on the queue to 1000. Type **ALTER QL(QL.A) MAXDEPTH(1000)**

__ f. Display the queue attributes. Type **DISPLAY QL(QL.A)**. Was the modification successful and is the queue description unchanged?

__ g. Define a second local queue QL.B with a text description. Type **DEFINE QL(QL.B) DESCR('QL.B Text')**

__ h. Display all the attributes of the queue. Type **DISPLAY QL(QL.B)**.

__ i. Change the maximum number of messages allowed on the queue to 2000. This time using the commands DEFINE with REPLACE instead of the ALTER command. Type **DEFINE QL(QL.B) REPLACE +**.

__ j. Display again the queue attributes of QL.B. Type **DISPLAY QL(QL.B)**. Was the creation of the queue successful and is there a queue description?

__ 2. Exit the runmqsc mode. Type **END.**

Step 3: Use the WebSphere MQ control commands in a command file

__ 1. Prepare a WebSphere MQ command file with the same commands you entered interactively in Step 2. Call the file 'exer1.txt'.

```
DIS QMGR
DIS Q(SYSTEM*)
DEF QL(QL.A) DESCR('QL.A Text')
DIS QL(QL.A)
ALTER QL(QL.A) MAXDEPTH(1000)
DIS QL(QL.A)
DEF QL(QL.B) DESCR('QL.B Text')
DEF QL(QL.B) REPLACE +
```

```
MAXDEPTH(2000)
DIS QL(QL.B)
```

__ 2. Process the command file and direct the results to the file 'report'. Type **runmqsc QMC## < exer1.txt > report**

Have a look at the file containing the results.

Step 4: Use the sample programs

UNIX

The supplied utilities (amqsput, amqsget, amqsbcg) are in the samp/bin sub-directory. You may change directory; or you may modify your search path. Both solutions are provided below.

- To change the directory:
  - AIX Systems
    `cd /usr/mqm/samp/bin`
  - SUN Systems
    `cd /opt/mqm/samp/bin`
- To modify your search path:
  - AIX Systems
    `export PATH=$PATH:/usr/mqm/samp/bin/`
  - SUN Systems
    `export PATH=$PATH:/opt/mqm/samp/bin/`

 Windows

The PATH statement is already set. You can start the sample programs from any directory.

__ 1. Put messages on the local queue QL.A. Type **amqsput QL.A QMC##**

What reason code did you receive? _____ What does it mean?

To interpret MQ reason codes, refer to Appendix B in the Student Notebook; or at a c: or $ prompt, use the control command mqrc. Type **mqrc nnnn** where nnnn is the four-digit reason code, for example, `mqrc 2085`.

__ 2. Browse the messages of the queue. Direct the output to a file and look at the results. Type **amqsbcg QL.A QMC## > out**.

What reason code did you receive? _____ What does it mean?

__ 3. Get the messages from the queue to make the queue empty. Type **amqsget QL.A QMC##**.

What reason code did you receive? _____ What does it mean?

Step 5: Work with alias queues

__ 1. Create an alias queue QA.A which resolves to your local queue QL.A. Type **runmqsc QMC##**. Type **DEF QA(QA.A) TARGQ(QL.A)**.

__ 2. Inhibit put requests on the alias queue. Type **ALTER QA(QA.A) PUT(DISABLED)**.

__ 3. Change QL.B to inhibit put requests on your local queue QL.B. Type **ALTER QL(QL.B) PUT(DISABLED)**.

__ 4. Create an alias queue QA.B which resolves to your local queue QL.B. Type **DEF QA(QA.B) TARGQ(QL.B)**.

__ 5. Try to put messages on both alias and local queues using the sample program.

Type **amqsput QL.A QMC##** for QL.A.

Type **amqsput QA.A QMC##** for QA.A.

Type **amqsput QL.B QMC##** for QL.B.

Type **amqsput QA.B QMC##** for QA.B.

__ 6. Now reenable QL.B for put requests. Type **alter ql(QL.B) put(enabled)**.

End of exercise

Optional exercise instructions with hints

Step 1: Working with queues

__ 1. Put some messages on your queues QL.A and QL.B.

__ 2. Show the CURDEPTH of all the queues you have created. Type **DISPLAY Q(Q*) CURDEPTH**.

__ 3. Define an new local queue QL.X with the attributes of QA.A. Type **DEFINE QLOCAL(QL.X) LIKE(QA.A)**.

__ 4. Error? Now use the attributes of QL.A. Type **DEFINE QLOCAL(QL.X) LIKE(QL.A)**.

__ 5. Put some messages on QL.X. Type **amqsput QL.X QMC##**.

__ 6. Change the description of QL.A. Type **ALTER QL(QL.A) DESCR('QL.A Text new')**.

__ 7. Display the DESCR attribute of all your queues. Type **DISPLAY Q(Q*) DESCR**.

__ 8. Is there any change?

__ 9. Clear queue QL.A. Type **CLEAR QL(QL.A)**.

__ 10. Clear queue QA.A. Type **CLEAR QA(QA.A)**.

__ 11. Were you successful? Give the reason why.

__ 12. Delete QL.X. Type **DELETE QL(QL.X)**

__ 13. Was it successfully deleted? If not, modify the command **DELETE QL(QL.X) purge**

End of exercise

Exercise review and wrap-up

You should now be able to:

- Create and start a queue manager
- Use control commands and command files to create local queues and display attributes of WebSphere MQ objects
- Change queue attributes and create alias queues
- Use sample programs to put messages on queues, browse messages and get messages from queues
- Clear and delete queues

Exercise 2. Implementing triggering

Estimated time

01:00

What this exercise is about

In this exercise, you extend your experience by using an WebSphere MQ application involving triggering to work. In addition, you can use a request-reply scenario to see how a reply-to queue can be used.

What you should be able to do

At the end of this exercise, you should be able to:

- Apply triggering parameters to queues
- Test triggering using WebSphere MQ utilities

Introduction

You use a sample program to put request messages on a queue which is enabled for triggering. The other sample programs are started by the trigger monitor and build the reply messages.

- amqsreq - This program is invoked from a command prompt in exactly the same way as amqsput, but accepts three input parameters. The first parameter is the name of the request queue, the second the queue manager name and the third is the reply-to queue name. The command to start the program is `amqsreq q_name qmgr_name replytoqueuename`, for example `amqsreq Q1 QM1 RTQ1`.

  The program reads lines of text from the standard input device, converts them to request messages, and MQPUTs the messages on the named queue.

  When a blank line is entered, the program begins to MQGET the messages from the reply-to queue and display those messages on the standard output device.

- amqsech - This program is designed to be started by a trigger monitor and not from a command prompt. The program connects to the queue manager named in the structure passed to it by the

trigger monitor and opens the queue also named in the structure. This is called the *request queue* in the description that follows.

The program gets a message from the request queue, creates a new message containing the same application data as the original message, and puts the new message on the reply-to queue named in the message descriptor of the original message. The program then gets each of the remaining messages on the request queue in turn and generates a reply in the same way. When the request queue is empty, the program closes the queue and disconnects from the queue manager.

- amqsinq - This program is designed to be started by a trigger monitor in the same way as amqsech. The program connects to the queue manager, opens the request queue, gets a message from the queue and interprets the application data as the name of a queue. The program opens this queue, calls MQINQ to inquire on the values of three of its attributes, constructs a reply message containing these values, and puts the message on the reply-to queue. The program then gets each of the remaining messages on the request queue in turn and generates a reply in the same way. When the request queue is empty, the program closes the queue and disconnects from the queue manager.

Requirements

- The sample programs amqsreq, amqsech, and amqsinq are invocable from any position within the file system hierarchy.
- A text editor.

Exercise instructions

Step 1: Configure the queue manager for triggering

__ 1. Prepare a WebSphere MQ command file 'exer2.txt.' to create all the objects required.

Use the REPLACE parameter on all relevant WebSphere MQ commands so that the command file can be run again if necessary.

__ a. An initiation queue named QL.INITQ.

__ b. A local queue which is enabled for triggering. Call it QL.A. The arrival of the first message on QL.A should trigger the appropriate application. (Make sure puts and gets are enabled and clear any messages from the prior lab)

__ c. A process to identify the application to be started. Define the process in such a way that amqsech is started synchronously with respect to the execution of the trigger monitor. Call the process PR.ECHO.

__ d. A model queue called QM.REPLY, that amqsreq opens to create a temporary dynamic reply to queue.

__ 2. Process the command file using runmqsc. Check the output to confirm that all the objects were successfully created. If unsuccessful, re-edit and rerun the command file, until all the objects are created.

__ 3. Start the trigger monitor using the control command runmqtrm specifying your queue-manager and the name of the initiation queue.

Step 2: Test triggering

__ 1. Run amqsreq to put some request messages on the local queue that is enabled for triggering.

__ 2. Type a message, press **Enter**.

__ 3. Press **Enter** again to end the sample program.

__ 4. Observe the trigger monitor starting amqsech and the replies received by amqsreq.

__ 5. If it did not work, investigate the possible causes.

End of exercise

Optional exercises

Step 1: Test triggering with amqsinq

__ 1. Create a second process object so that amqsinq is started by the trigger monitor instead of amqsech.

__ 2. Alter the queue QL.A appropriately.

__ 3. Run amqsreq again. Type

amqsreq QL.A QMC## QM.REPLY

Remember that amqsinq now expects the application data in each message to contain the name of a queue. For example, you can use the names of the following queues:

SYSTEM.ADMIN.COMMAND.QUEUE

QM.REPLY

SYSTEM.SAMPLE.REPLY

QL.A

Try an invalid queue name and see what happens.

Step 2: Alter the trigger conditions

__ 1. Alter the values of the attributes of your local queue which control triggering. For example, try using depth triggering.

__ 2. Check that the application is triggered according to the TRIGDPTH value.

__ 3. Alter the process object so the trigger monitor starts amqsech or amqsinq as a separate asynchronous process, use a trigger type of first again.

__ 4. Check that for the duration of the triggered application a separate window is opened.

End of exercise

Exercise instructions with hints

Step 1: Configure the queue manager for triggering

__ 1. Prepare a WebSphere MQ command file 'exer2.txt.' to create all the objects required.

Use the REPLACE parameter on all relevant WebSphere MQ commands so that the command file can be run again if necessary.

__ a. An initiation queue named QL.INITQ. Type **DEFINE QLOCAL(QL.INITQ) REPLACE**

__ b. A local queue which is enabled for triggering. Call it **QL.A**. The arrival of the first message on **QL.A** should trigger the appropriate application. (Make sure puts and gets are enabled and clear any messages from the prior lab). Type

```
DEFINE QL(QL.A) REPLACE +
TRIGGER +
TRIGTYPE(FIRST) +
PROCESS(PR.ECHO)+
INITQ(QL.INITQ)
```

__ c. A process to identify the application to be started. Define the process in such a way that amqsech is started synchronously with respect to the execution of the trigger monitor. Call the process **PR.ECHO.** Use the following commands depending on your system type:

Windows

```
DEFINE PROCESS(PR.ECHO) REPLACE + APPLICID('amqsech')
```

UNIX

AIX Systems

```
DEFINE PROCESS(PR.ECHO) REPLACE +
APPLICID('/usr/mqm/samp/bin/amqsech')
```

SUN Systems

```
DEFINE PROCESS(PR.ECHO) REPLACE +
APPLICID('/opt/mqm/samp/bin/amqsech')
```

__ d. Use a model queue called QM.REPLY, that amqsreq opens to create a temporary dynamic reply to queue. Type **DEFINE QMODEL(QM.REPLY) REPLACE**.

__ 2. Process the command file using `runmqsc`. Check the output to confirm that all the objects are successfully created. If unsuccessful re-edit the command file, until all the objects are created. Type **runmqsc QMC## < exer2.txt > report2**.

__ 3. Start the trigger monitor using the control command runmqtrm specifying your queue-manager and the name of the initiation queue. Type **runmqtrm -q QL.INITQ -m qmgrname**

Step 2: Test triggering

__ 1. Open a new command window and run amqsreq in order to put some request messages on the local queue that is enabled for triggering. Type **amqsreq QL.A QMC## QM.REPLY**

__ 2. Type a message, press **Enter**. For example, type **Hello this is a message from TeamXX**.

__ 3. Press **Enter** again to end the sample program.

__ 4. Observe the trigger monitor starting amqsech and the replies received by `amqsreq`.

__ 5. If it did not work, investigate the possible causes.

Was one of the conditions for a trigger event not satisfied?

End of exercise

Optional exercise instructions with hints

Step 1: Test triggering with amqsinq

__ 1. Create a second process object so that amqsinq is started by the trigger monitor instead of amqsech.

Windows

```
DEFINE PROCESS(PR.INQ) REPLACE +
APPLICID('amqsinq')
```

UNIX

AIX systems

```
DEFINE PROCESS(PR.INQ) REPLACE +
APPLICID('/usr/mqm/samp/bin/amqsinq')
```

SUN systems

```
DEFINE PROCESS(PR.INQ) REPLACE +
APPLICID('/opt/mqm/samp/bin/amqsinq')
```

__ 2. Alter the queue QL.A appropriately. Type **ALTER QL(QL.A) PROCESS(PR.INQ)**

__ 3. Run `amqsreq` again. Remember that amqsinq now expects the application data in each message to contain the name of a queue. For example, you can use the names of the following queues. Type **amqsreq QL.A QMC## QM.REPLY**

```
SYSTEM.ADMIN.COMMAND.QUEUE
QM.A_REPLY
SYSTEM.SAMPLE.REPLY
QL.A
```

Try an invalid queue name and see what happens.

__ 4. Observe the trigger monitor starting amqsinq and the replies received by amqsreq. The replies consist of an inquire performed on attributes of the named queues.

Step 2: Alter the trigger conditions

__ 1. Alter the values of the attributes of your local queue which control triggering. For example, try using depth triggering. Type

```
ALTER QL(QL.A) +
TRIGTYPE(DEPTH) TRIGDPTH(5)
```

__ 2. Check that the application is triggered according to the TRIGDPTH value. Type **amqsreq QL.A QMC## QM.REPLY**

__ 3. Alter the process object so the trigger monitor starts amqsech or amqsinq as a separate asynchronous process, use a trigger type of first again. Type the following commands according to your system type:

UNIX

```
ALTER PROCESS(PR.name) ENVRDATA('&')
```

Windows

```
ALTER PROCESS(PR.name) APPLICID('start amqsech')
ALTER PROCESS(PR.name) APPLICID('start amqsinq')
```

__ 4. Check that for the duration of the triggered application a separate window is opened.

End of exercise

Exercise review and wrap-up

Having completed this exercise, you should be able to:

- Apply triggering parameters to queues
- Test triggering using MQ utilities

Exercise 3. Queue recovery

Estimated time

01:00

What this exercise is about

This exercise demonstrates the survival of persistent messages across a queue manager restart. It also demonstrates how a queue that has become damaged and can be recovered from its media image.

What you should be able to do

At the end of this exercise, you should be able to:

- Use rcdmqimg to capture an object media image
- Use rcrmqobj to re-create a WebSphere MQ object

Introduction

In this exercise, you create a queue manager with a linear log. In order to demonstrate media recovery, you then damage a queue. Of course, the act of deliberately damaging a queue should not be done in practice. It is done in this exercise in order to simulate a disk failure.

Requirements

WebSphere MQ V7 installed on one of the following systems:

- Windows
- AIX
- Solaris

Exercise instructions

Step 1: Queue manager restart

__ 1. Create a new queue manager named QML## with linear logging, where # is your team or machine number. Indicate that it is not to be the default queue manager.

__ 2. Start the queue manager.

__ 3. Prepare a WebSphere MQ command file "exer3.txt" to create the following objects:

  __ a. Two local queues named QL.A and QL.B. The default persistence of each queue should be set to YES.

  __ b. Two alias queues named QA.A and QA.B. QA.A resolves to the local queue QL.A and QA.B to the queue QL.B. The default persistence of each alias queue should be set to NO.

  __ c. Clear the local queues.

__ 4. Process the command file using runmqsc. Make sure you specify your queue manager name.

__ 5. Put a mixture of persistent and nonpersistent messages on the local queues QL.A and QL.B and alias queues QA.A and QA.B using the sample program amqsput. When amqsput is invoked, the value of the DefPersistence attribute of the queue which is first (explicitly) opened determines whether the message it puts on the queue is persistent or nonpersistent. In respect to the definitions made before amqsput create:

  __ a. Persistent messages naming the local queue

  __ b. Non-persistent messages naming the alias queue

__ 6. Use the WebSphere MQ Explorer or the sample program amqsbcg to browse the messages on the local queue QL.A and check that there is a mixture of persistent and nonpersistent messages on it.

__ 7. Stop the queue manager QML## and then start it again.

__ 8. Browse the messages on queue QL.A again. Check that only the persistent messages are on the local queue QL.A have survived the restart.

Step 2: Media recovery

__ 1. Execute rcdmqimg against the queue QL.B. This allows you to see how the record image can force a checkpoint which is useful for queues that never (or infrequently) reach a depth of zero where a checkpoint is done automatically.

__ 2. Locate the file implementing local queue QL.B within the file system and damage the queue by deleting the file. This file is called 'q'.

__ 3. Display the attributes of local queue QL.B using the DISPLAY QUEUE command. This should still work as the queue manager does not need to access the queue file in order to provide the requested information.

__ 4. Put some messages to local queue QL.B using amqsput. It might take 10 or more messages (due to buffering) but the queue manager will eventually detect that the queue has been damaged and report the fact by returning MQRC 2101 (MQRC_OBJECT_DAMAGED).

__ 5. Try to display the attributes of local queue QL.B again using the DISPLAY QUEUE command. The queue manager reports an error now as it knows the queue has been damaged.

__ 6. Recover local queue QL.B from its media image using the re-create object (rcrmqimg) command.

__ 7. Check that you can now display the attributes of the queue.

__ 8. Browse the local queue QL.B again to check if the messages are recovered successfully. What about the nonpersistent messages?

__ 9. Clear the messages from the queue.

__ 10. Stop and delete the queue manager QML##.

End of exercise

Exercise instructions with hints

Step 1: Queue manager restart

__ 1. Create a new queue manager named QML## with linear logging, where # is your team or machine number. Indicate that it is not to be the default queue manager. Type **crtmqm -ll QML##**

__ 2. Start the queue manager. Type **strmqm QML##**

__ 3. Prepare an WebSphere MQ command file "exer3.txt" to create the following objects:

__ a. Two local queues named QL.A and QL.B. The default persistence of each queue should be set to YES. Type

DEF QL(QL.A) DEFPSIST(YES) REPLACE

DEF QL(QL.B) DEFPSIST(YES) REPLACE

__ b. Two alias queues named QA.A and QA.B. QA.A resolves to the local queue QL.A and QA.B to the queue QL.B. The default persistence of each alias queue should be set to NO. Type

DEF QA(QA.A) TARGQ(QL.A) DEFPSIST(NO) REPLACE

DEF QA(QA.B) TARGQ(QL.B) DEFPSIST(NO) REPLACE

__ c. Clear the local queues. Type

CLEAR QL(QL.A)

CLEAR QL(QL.B)

__ 4. Process the command file using **runmqsc**. Make sure you specify your queue manager name.

__ 5. Put a mixture of persistent and nonpersistent messages on the local queues QL.A and QL.B and alias queues QA.A and QA.B using the sample program **amqsput**. When **amqsput** is invoked, the value of the *DefPersistence* attribute of the queue which is first (explicitly) opened determines whether the message it puts on the queue is persistent or nonpersistent. In respect to the definitions made before **amqsput** create:

- Persistent messages naming the local queue
- Non-persistent messages naming the alias queue
- **amqsput** QL.A [*qmgrname*]
- **amqsput** QA.A [qmgrname]
- **amqsput** QL.B [qmgrname]
- **amqsput** QA.B [qmgrname]

__ 6. Use the WebSphere MQ Explorer or the sample program amqsbcg to browse the messages on the local queue QL.A and check that there is a mixture of persistent and nonpersistent messages on it. Type **amqsbcg QL.A QML##**

__ 7. Stop the queue manager QML## and then start it again. Type

endmqm -i QML##

strmqm QML##

__ 8. Browse the messages on queue QL.A again. Check that only the persistent messages are on the local queue QL.A have survived the restart. Type **amqsbcg QL.A QMC##**

Step 2: Media recovery

__ 1. Execute **rcdmqimg** against the queue QL.B. This allows you to see how the record image can force a checkpoint that is useful for queues that never (or infrequently) reach a depth of zero where a checkpoint is done automatically. Type **rcdmqimg -m QML## -t ql QL.B**

__ 2. Locate the file implementing local queue QL.B within the file system and damage the queue by deleting the file. Type the following according to your system type:

UNIX

```
/var/mqm/qmgrs/QML##/queues/QL!B/q (Unix)
```

Windows

```
C:\Program Files\IBM\WebSphere MQ\qmgrs\QML##\queues\QL!B\Q
```

__ 3. Display the attributes of local queue QL.B using the DISPLAY QUEUE command. This should still work as the queue manager does not need to access the queue file in order to provide the requested information. Type

runmqsc QML##

DIS Q(QL.B)

__ 4. Put some messages to local queue QL.B using amqsput. It might 10 or more messages (due to buffering) but the queue manager will eventually detect that the queue has been damaged and report the fact by returning MQRC 2101 (MQRC_OBJECT_DAMAGED). Type **amqsput QL.B QML##**

__ 5. Try to display the attributes of local queue QL.B again using the DISPLAY QUEUE command. The queue manager reports an error now as it knows the queue has been damaged. Type

runmqsc QML##

DIS Q(QL.B)

__ 6. Recover local queue QL.B from its media image using the re-create object (rcrmqimg) command. Type **rcrmqobj -m QML## -t ql QL.B**

__ 7. Check that you can display the attributes of the queue.

runmqsc QML##

DIS Q(QL.B)

__ 8. Browse the local queue QL.B to check if the messages are recovered successfully. Type **amqsgbr QL.B QML##**

What about the nonpersistent messages?

__ 9. Clear the messages from the queue. Type **amqsget QL.B QML##**

__ 10. Stop and delete the queue manager. Type

endmqm -i QML##

dltmqm QML##

End of exercise

Exercise review and wrap-up

Having completed this exercise, you should be able to:

- Create a queue manager with linear logging
- Put persistent and nonpersistent messages on local queues
- Stop and restart a queue manager
- Recover a damaged local queue from its media image
- Record manually media images to optimize media recovery

Exercise 4. Queue manager interconnection

Estimated time

01:30

What this exercise is about

In this exercise, you connect two queue managers, and exchange messages between applications connected to those queue managers.

What you should be able to do

At the end of this exercise, you should be able to:

- Design a networked WebSphere MQ architecture consisting of two or more interconnected queue managers
- Use runmqsc to create the necessary channels and supporting objects to implement distributed queuing
- Use utilities to test a remote queuing environment

Introduction

You need to configure two queue managers, each on a different system, with two message channels between them for message flow in each direction. Use your original queue managers with circular logging for this and the remaining exercises.

The exercise uses TCP/IP as the communications protocol. The instructor tells you the host names or IP addresses of your system.

Requirements

- The sample programs amqsput, amqsget, amqsreq and amqsech are invocable from any position within the file system hierarchy.
- A text editor.

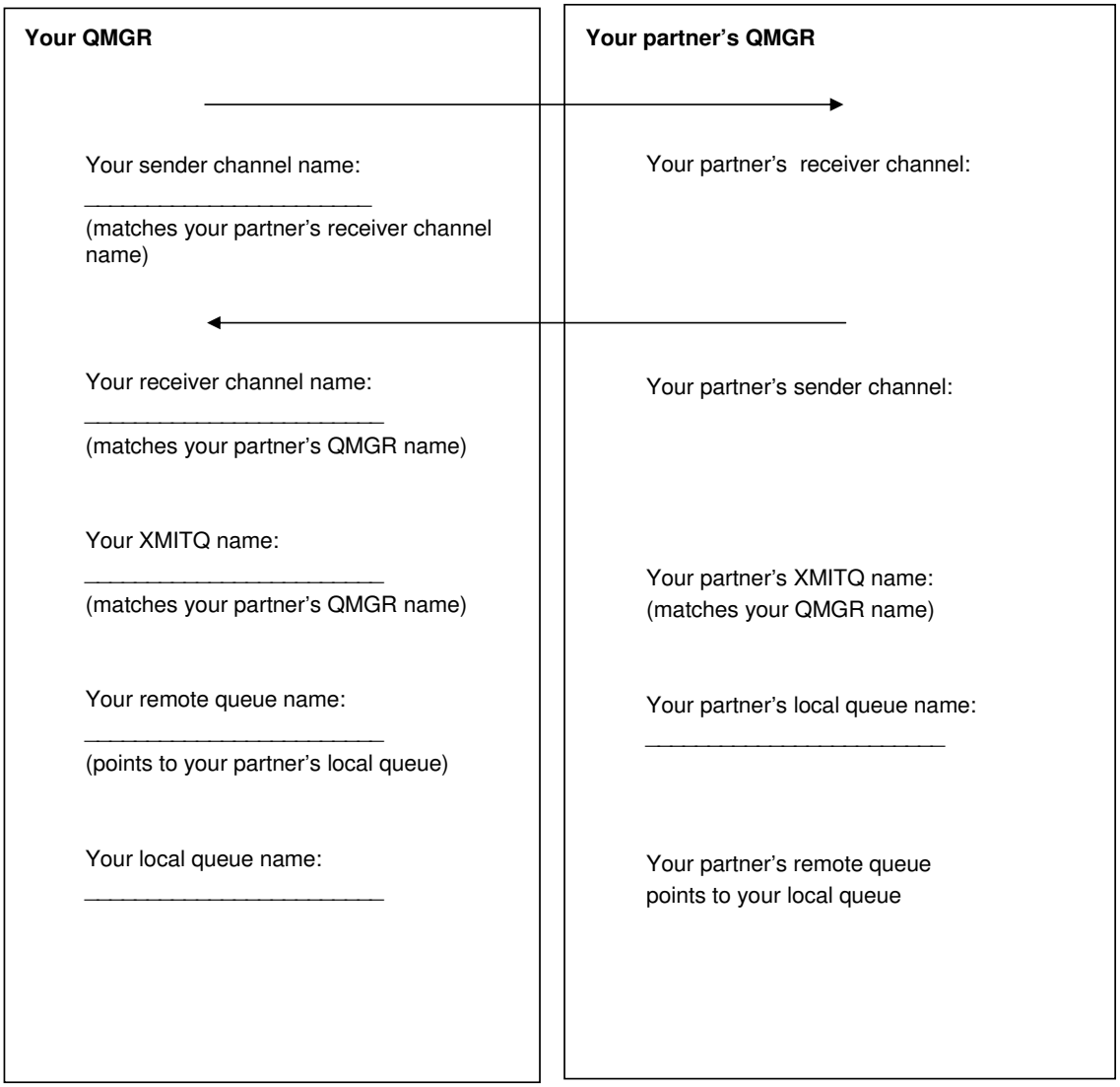
Exercise 4 worksheet

Originating QMGR (you)
Your QMGR name: _____
(agrees with partner's XMITQ name)

CONNAME:
Your IP address or system name:
_____
Your listener port number: _____

Destination QMGR (your partner)
Partner's QMGR name: _____
(agrees with your XMITQ name)

CONNAME:
Partner's IP address or system name:
_____
Partner's listener port number: _____



Your remote queue name = _____ (points to your partner's local queue)

RNAME =_____ (matches your partner's local queue name)

RQMNAME =_____ (matches your partner's QMGR name)

XMITQ (optional) =_____ (agrees with partners QMGR name)

Your local queue name = _____ (pointed at by your partner's remote queue)

Your Dead Letter Queue name = _____

Note

When using **amqsreq** to test, the default Model Queue name = SYSTEM.DEFAULT.MODEL.QUEUE or your chosen Model Queue name = _____.

Exercise instructions

Step 1: Create and configure the required connection objects

__ 1. For this exercise, use the queue manager with circular logging from Exercise 1.

__ 2. Prepare a WebSphere MQ command file 'exer4a.txt' to define the WebSphere MQ objects required for a connection between your local queue manager and the queue manager of your partner team.

The necessary objects to be defined are:

__ a. A definition of a message channel (type = sender).

- Channel name = QMC##.QMC$$ (where QMC## is your local queue manager name and QMC$$ is your partner's remote queue manager name)
- Network protocol = TCP/IP
- Network address = NNN(90$$) (where NNN is your partner's IP address or system name and 90$$ is your partner's port address)
- Transmission queue name = the same as the name of your partner's remote queue manager

__ b. A definition of a message channel (type = receiver). The attributes should match to sender channel of partner team.

__ c. A transmission queue with the same name as the remote queue manager.

__ d. A dead letter queue named DLQ.

__ e. Change the queue manager to use the just created queue as its Dead Letter Queue.

__ 3. Use runmqsc to create the WebSphere MQ objects.

Step 2: Configure and activate the required TCP listener function

Start the WebSphere MQ listener listening on port 90##.

Step 3: Test and start the connection

__ 1. Ping the message channel from the sender end in order to test the channel definitions. Check for successful completion.

__ 2. Start the channel using the runmqsc command START CHANNEL and verify that it is working.

Step 4: Create the required application objects

__ 1. Prepare a second WebSphere MQ command file 'exer4b.txt' to define the application queues on your queue manager.

__ a. Delete and redefine the local queue QL.A.

__ b. Create a local definition of a remote queue pointing to the local queue QL.A on your partner's queue manager using the name QRMT$$, where $$ is your partner's team or machine number.

__ 2. Use runmqsc to create the WebSphere MQ objects.

Step 5: Test distributed queueing

__ 1. Use the sample program amqsput to send messages to the queue QL.A on the partner queue manager.

__ 2. Use amqsget or amqsbcg to check for successful arrival of messages from your partner.

__ 3. If the messages do not arrive, investigate the possible causes and solve the problem together with your partner.

End of exercise

Optional exercise instructions

Step 1: Set up remote triggering

Use the sample program amqsreq to send request messages to the queue QL.A on your partner's queue manager QMC##. The target queue should be enabled for triggering so that the sample program amqsech is started automatically and generates reply messages which are subsequently received by amqsreq.

__ 1. Reactivate the trigger function for QL.A to handle request messages send to your QL.A now by the partner team.

__ 2. Restart the trigger monitor.

__ 3. Put request messages on QL.A in your partner team queue manager.

__ 4. Check for a reply of each request message.

Step 2: Use the channel initiator

__ 1. Create a new WebSphere MQ command file 'exer4c.txt"to define and modify WebSphere MQ objects to setup automatic channel operation using a channel initiator.

__ 2. Enable the transmission queue for triggering.

__ 3. Set the disconnect interval of the channel to 30 seconds.

__ 4. Stop and restart the sender channel using runmqsc.

__ 5. Wait until the channel is terminated by the DISCINT.

__ 6. Put some messages on the queue QRMT$$ using amqsput.

__ 7. The channel should restart automatically.

__ 8. If the messages do not arrive, investigate the possible causes and solve the problem together with your partner team.

__ 9. Put some messages on the queue QRMT$$ using amqsreq.

__ 10. Both channels should restart automatically.

__ 11. Prove that by checking the channel status on both sides. (Be sure that the DISCINT has not elapsed.)

__ 12. If the messages do not arrive, investigate the possible causes and solve the problem together with your partner team.

End of exercise

Exercise instructions with hints

Step 1: Create and configure the required connection objects

__ 1. For this exercise, use the queue manager with circular logging from Exercise 1. Use QMC##

__ 2. Prepare an WebSphere MQ command file 'exer4a.txt' to define the WebSphere MQ objects required for a connection between your local queue manager and the queue manager of your partner team. The necessary objects follows:

__ a. A definition of a message channel (type = sender).

- Channel name = **QMC##.QMC$$** (where **QMC##** is your local queue manager name and **QMC$$** is your partner's remote queue manager name)
- Protocol = TCP/IP
- Network address = **NNN(90$$)** (where **NNN** is your partner's IP address or system name and **90$$** is your partner's port address)
- Transmission queue name = the same as the name of your partner's remote queue manager

```
DEF CHL(QMC##.QMC$$) CHLTYPE(SDR) REPLACE +
TRPTYPE(TCP) +
CONNAME('NNN(90$$)') +
XMITQ(QMC$$)
```

__ b. A definition of a message channel (type = receiver). The attributes should match to sender channel of partner team. Type

DEF CHL(QMC$$.QMC##) CHLTYPE(RCVR) REPLACE +
TRPTYPE(TCP)

__ c. A transmission queue with the same name as the remote queue manager. Type

DEF QL(QMC$$) REPLACE +
USAGE(XMITQ)

where $$ is your partner's team or machine number.

__ d. A dead letter queue named **DLQ**. Type **DEF QL(DLQ) REPLACE**

__ e. Change the queue manager to use the just created queue as its Dead Letter Queue. Type **ALTER QMGR DEADQ(DLQ)**

__ 3. Use runmqsc to create the WebSphere MQ objects.

Step 2: Configure and activate the required TCP listener function

Start the WebSphere MQ listener listening on 90##. Type

runmqlsr -t TCP -p 90## -m QMC##

where QMC## is your queue manager name.

Step 3: Test and start the connection

__ 1. Ping the message channel from the sender end in order to test the channel definitions, using the runmqsc command: **PING CHL(QMC##.QMC$$)**

Check for successful completion.

__ 2. Start the channel using the runmqsc command START CHANNEL and verify that it is working. Type **START CHL(QMC##.QMC$$)**

Step 4: Create the required application objects

__ 1. Prepare a second WebSphere MQ command file 'exer4b.txt' to define the application queues on your queue manager.

__ a. Delete and redefine the local queue **QL.A.** Type **DEF QL(QL.A) REPLACE**

__ b. Create a remote queue definition pointing to the local queue **QL.A** on your partner's queue manager using the name **QRMT$$**, where $$ is your partner's team or machine number.

Optionally, you can include the name of the transmission queue created in step one. Use the following commands:

DEF QR(QRMT$$) REPLACE +

RNAME(QL.A) RQMNAME(QMC$$) +

XMITQ(QMC$$)

__ 2. Use runmqsc to create the WebSphere MQ objects.

Step 5: Test distributed queueing

__ 1. Use the sample program amqsput to send messages to the queue **QL.A** on the partner queue manager. Type **amqsput QRMT$$ QMC##** (where **QMC##** is your queue manager name).

__ 2. Use amqsget or amqsbcg to check for successful arrival of messages from your partner. Type **amqsget QL.A QMC##** (where **QMC##** is your queue manager name) or **amqsbcg QL.A QMC##** (where **QMC##** is your queue manager name) or just check the **CURDEPTH** attribute of the target queue.

__ 3. If the messages do not arrive, investigate the possible causes and solve the problem together with your partner team. Check the following items:

__ a. Local transmission queue is empty?

__ b. Message channel is running?

__ c. Dead letter queue of target queue manager is empty?

__ d. Inspect the error logs in both queue managers.

End of exercise

Optional exercise instructions with hints

Step 1: Set up remote triggering

Use the sample program amqsreq to send request messages to the queue **QL.A** on the partner queue manager **QMC##**. The target queue should be enabled for triggering so that the sample program amqsech is started automatically and generates reply messages which are subsequently received by amqsreq.

__ 1. Reactivate the trigger function for **QL.A** to handle request messages send to your QL.A now by the partner team. Use the following commands:

ALTER QL(QL.A) +

TRIGGER +

TRIGTYPE(FIRST) +

INITQ(QL.INITQ) +

PROCESS(PR.ECHO)

__ 2. Restart the trigger monitor. Type **runmqtrm -q QL.INITQ -m QMC##** (where **QMC##** is your queue manager name)

__ 3. Put request messages on **QL.A** in your partner team queue manager. Type **amqsreq QRMT$$ QMC## QM.REPLY** (where QMC## is your queue manager name)

__ 4. Check for a reply of each request message.

Step 2: Use the channel initiator

__ 1. Create a new WebSphere MQ command file 'exer4c.txt' to define and modify WebSphere MQ objects to setup automatic channel operation using a channel initiator.

__ 2. Enable the transmission queue for triggering.

ALTER QL(QMC$$) +

TRIGGER +

TRIGTYPE(FIRST) +

INITQ(SYSTEM.CHANNEL.INITQ)

__ 3. Set the disconnect interval of the channel to 30 seconds. Type **ALTER CHANNEL(QMC##.QMC$$) CHLTYPE(SDR) + DISCINT(30)**

__ 4. Stop and restart the sender channel using runmqsc. Type the following:

STOP CHANNEL(QMC##.QMC$$)
START CHANNEL(QMC##.QMC$$)

__ 5. Wait until the channel is terminated by the DISCINT. Type **DISPLAY CHSTATUS(QMC##.QMC$$)**

__ 6. Put some messages on the queue **QRMT$$** using amqsput. Type **amqsput QRMT$$ QMC##** (where QMC## is your queue manager name)

__ 7. The channel should restart automatically.

__ 8. If the messages do not arrive, investigate the possible causes and solve the problem together with your partner team.

__ 9. Put some messages on the queue **QRMT$$** using amqsreq. Type **amqsreq QRMT$$ QMC## QM.REPLY** (where QMC## is your queue manager name)

__ 10. Both channels should restart automatically.

__ 11. Prove that by checking the channel status on both sides. (Be sure that the DISCINT has not elapsed.) Type **DISPLAY CHSTATUS(QM*) all**

__ 12. If the messages do not arrive, investigate the possible causes and solve the problem together with your partner team.

End of exercise

Exercise review and wrap-up

Having completed this exercise, you should be able to:

- Create the objects required for distributed queueing
- Configure and refresh the inet daemon or started an MQ listener
- Start message channels manually
- Test the message flow using sample applications
- Use triggering in a distributed environment
- Use the channel initiator to start messages

Exercise 5. Dead letter queue handling

Estimated time

00:30

What this exercise is about

In this exercise, you experience why dead letter queues should be used. In addition, you will configure Websphere MQ to handle messages that arrive on the dead letter queue by using the dead letter queue handler.

What you should be able to do

At the end of this exercise, you should be able to:

- Use runmqsc to configure the queue manager to use a dead letter queue
- Set up amqsdlh to handle dead letter messages

Introduction

This exercise should be done in teams of two with one person responsible for QMC## and one person responsible for QMC$$.

To complete this exercise you should be familiar with the following:

- amqsbcg - This program is invoked from a command prompt in exactly the same way as amqsput. It accepts two input parameters, the name of the dead letter queue and the name of queue manager which owns it. This program will output both the message descriptor and content fields of each message on the queue.
- runmqdlq - This is a command to start the dead letter queue handler which monitors and handles messages on a dead letter queue. It can take its input interactively from the terminal or from a rules table file.

Requirements

- Functioning channel connections between two queue managers, denoted as QMC## for the local queue manager and QMC$$ for the remote or partner's queue manager

- Both queue managers set up to use a dead letter queue
- Sample programs amqsbcg and amqsput
- A text editor

Exercise instructions

In this exercise, QMC## refers to your queue manager, where ## is the number of your team and $$ is the number of your partner or partner team.

__ 1. Use local queues on QMC## named QL.A and QL.B

__ 2. Use the remote queue definition on QMC$$ called QRMT## that targets QL.A on QMC##

__ 3. Put disable QL.A on QMC##

__ 4. Put a message on QRMT## on QMC$$

__ 5. Locate the message on the dead letter queue on QMC##

__ 6. Examine the dead letter queue header using amqsbcg

__ a. Locate the 4 byte reason code (bytes 9-12)

__ b. Use mqrc to find the meaning of this reason code

Windows

The MQ Explorer browse messages functionality can be used instead of amqsbcg. Note: If you use amqsbcg, then the message is represented in *little endian* format and will need to be interpreted correctly for use with mqrc.

__ 7. Create a DLQ handler rules table file that will forward the DLQ message destined for QL.A on QMC## to QL.B on QMC## but strips the DLQ Header

__ 8. Start the DLQ Handler. Use '&' to run the handler in the background.

Windows

For Windows you cannot use '&' to run the command in the background, so open a second command window to continue the exercise.

__ 9. Put another message on QRMT## on QMC$$

__ 10. See if the DLQ Handler handled the new message, (the message should now be found on QL.B on QMC##)

End of exercise

Exercise instructions with hints

__ 1. Use local queues on QMC## named QL.A and QL.B

__ 2. Use the remote queue definition on QMC$$ called QRMT## that targets QL.A on QMC##

__ 3. Put disable QL.A on QMC##. Type

runmqsc QMC##

alter qlocal(QL.A) put(disabled)

__ 4. Put a message on QRMT## on QMC$$. Type **amqsput QRMT## QMC$$.**

__ 5. Locate the message on the dead letter queue on QMC##. Type **display queue(DEAD_LETTER_QUEUE_NAME) curdepth**

Check the current depth parameter to see if there is a message on the queue.

__ 6. Examine the dead letter queue header using amqsbcg. Type **amqsbcg DEAD_LETTER_QUEUE_NAME QMC##**

__ a. Locate the 4 byte reason code (bytes 9-12). In this case the code should be 00000803

__ b. Use `mqrc` to find the meaning of this reason code. Type **mqrc 0x00000803**.

This should give you the reason of `MQRC_PUT_INHIBITED`

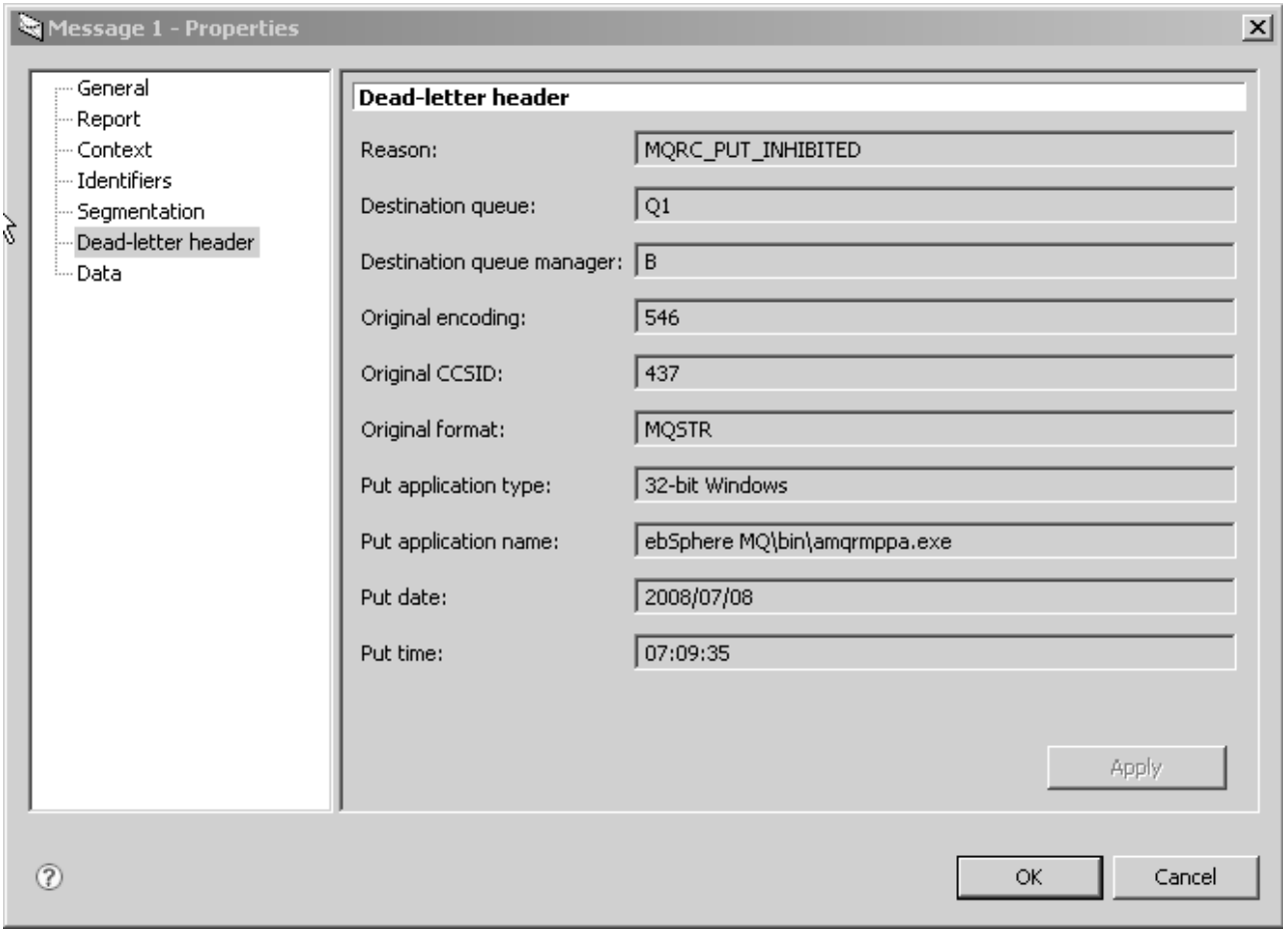
Windows

If using `amqsbcg`, the 4 byte reason code will be represented as 03080000, which should be interpreted as 00000803

For the MQ Explorer use the following instructions:

- Right-click the dead letter queue in QMC## and select browse messages
- Right-click the relevant message and select properties
- Click Dead-letter header from the left hand menu

This will show the dead letter header in the standard format



__ 7. Create a DLQ handler rules table file that will forward the DLQ message destined for QL.A on QMC## to QL.B on QMC## but strips the DLQ Header. Open a new text file and type in the following rules

```
inputqm(QMC##) inputq(DEAD_LETTER_QUEUE_NAME)
   REASON(MQRC_PUT_INHIBITED) action(fwd) fwdq(QL.B)
   fwdqm(QMC##) +  header(no)
```

__ 8. Start the DLQ Handler. Use the following according to your system type:

UNIX

```
runmqdlq DEAD_LETTER_QUEUE_NAME QMC## < rules_tbl_filename &
```

Windows

For Windows you cannot use ‘&' to run the command in the background, so open a second command window to continue the exercise.

__ 9. Put another message on QRMT## on QMC$$. Type **amqsput QRMT## QMC$$**

__ 10. See if the DLQ Handler handled the new message, (the message should now be found on QL.B on QMC##). Type the following commands:

runmqsc QMC##

display ql(QL.B) curdepth

End of exercise

Exercise review and wrap-up

Having completed this exercise, you should be able to:

- Examine a dead letter header to determine the reason the message arrived on the dead letter queue
- Write a rules table as input for runmqdlq to handle dead letter messages

Exercise 6. WebSphere MQ clustering setup

Estimated time

01:00

What this exercise is about

The aim of this exercise is to set up a simple queue manager cluster.

What you should be able to do

At the end of the exercise, you should be able to:

- Use runmqsc to create a queue manager cluster
- Test a cluster environment using WebSphere MQ utilities

Introduction

Each student or team creates two new queue managers to form a queue manager cluster. One will be a full repository; the other a non-repository member.

Once the cluster of two queue managers is working, it will be connected to another student or team cluster, forming a cluster of 4 queue managers with two repositories.

The instructor will allocate a two-digit cluster name suffix to each pair of students or teams.

Requirements

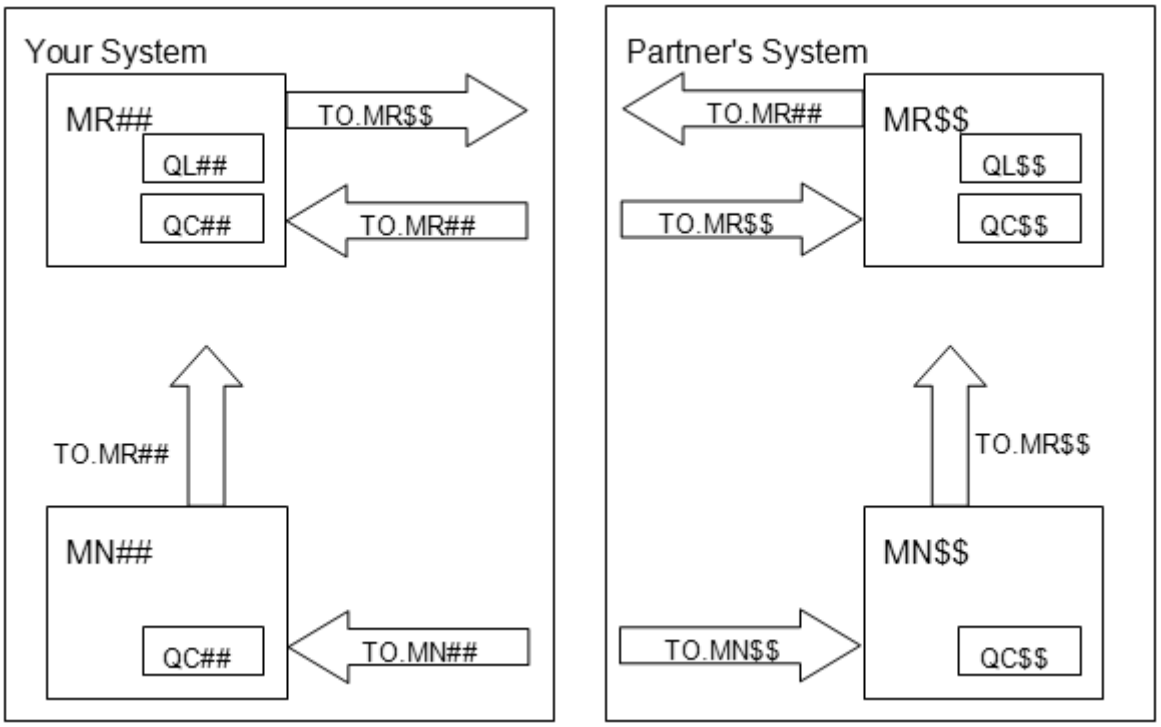
- Sample programs amqsput, amqsget, and amqsbcg (which are invocable from any position within the file system hierarchy).
- A text editor.

Exercise overview

The following is the naming convention for this exercise

- ## = Your "Student ID" (Team number, Machine number, Logon suffix)
- $$ = Your Partner's "Student ID"
- && = Cluster Number assigned by instructor

Here is how your cluster will look:



- "MR##" and "MR$$" are full repository queue managers.
- "MN##" and "MN$$" are normal queue managers that are not full repositories.
- "QC##" is a local queue defined on your two queue managers and advertised across the whole cluster.
- "QL##" is a local queue defined only on your full repository queue manager and advertised across the whole cluster.
- "QC$$" is a local queue defined on your partner's two queue managers and advertised across the whole cluster.
- "QL$$" is a local queue defined only on your partner's full repository queue Manager and advertised across the whole cluster.

Arrows pointing away from a queue manager are CLUSSDR channels.

Arrows pointing toward a queue manager are CLUSRCVR channels.

WM20x - Cluster lab worksheet

Cluster name = CLUS __ __

Table 1:

Your regular queue manager	Your full repository
Name: MN __ __ (Your student ID)	Name: MR __ __ (Your student ID)
CLUSRCVR: TO.MN __ __ (Your own normal queue manager)	CLUSRCVR: TO.MR _____ _____ (Your own full repository)
CONNAME for CLUSRCVR (your system): IP Address: ____.____.____.____ or DNS name: _____ Port: 9 0 _____ _____ (Your student ID)	CONNAME for CLUSRCVR (Your system): IP Address: ____.____.____.____ or DNS name: _____ Port: 9 1 _____ _____ (Your student ID)
CLUSSDR: TO. MR ___ ___ (Your own full repository)	CLUSSDR: TO.MR _____ _____ (Your partner's full repository)
CONNAME for CLUSSDR (your system): IP Address: ____.____.____.____ or DNS name: _____ Port: 9 1 _____ _____ (Your student ID)	CONNAME for CLUSSDR (partner system): IP Address: ____.____.____.____ or DNS name: _____ Port: 9 1 _____ _____ (Partner's student ID)
One of your cluster-wide local queues: QC _____ _____ (Your student ID)	One of your cluster-wide local queues: QC _____ _____ (Your student ID)
	Your other cluster-wide local queue: QL _____ _____ (Your student ID) (Only defined on your full repository)

Exercise instructions

- Perform the following steps from one queue manager at a time.
- Stop all previously created queue managers on your system.
- Stop all previously-running listeners on your system.

Step 1: Create full repository queue manager

__ 1. Create a new queue manager MR## to be used as the cluster repository.

__ 2. Start that queue manager.

__ 3. In a new session window, start the Listener for your queue manager MR## on port 91## using the MQ listener.

__ 4. Prepare MQSC command file "EX6MR.txt" to define the cluster connection objects required for your queue manager. The commands needed are as follows:

  __ a. Define one cluster receiver channel pointing to this queue manager (TO.MR##).

  __ b. Alter the queue manager to be a repository including the cluster name.

  __ c. Define a local queue called "DLQ" and tell the queue manager to use it as its dead letter queue.

__ 5. Run the command file through runmqsc on MR##.

Step 2: Create normal cluster member queue manager

__ 1. Create a new queue manager MN## to be used in a queue manager cluster.

__ 2. Start that queue manager.

__ 3. In a new session window, start the Listener for your queue manager MN## on port 90## using the WebSphere MQ listener.

__ 4. Prepare a WebSphere MQ command file "EX6MN.txt" to define the cluster connection objects required for your queue manager. The objects needed should include the following:

  __ a. One cluster receiver channel pointing to this queue manager (TO.MN##). Be sure to include the conname and cluster parameters.

  __ b. One cluster sender channel pointing to a full repository queue manager in your cluster (TO.MR##). Be sure to include the CONNAME and cluster parameters.

  __ c. Define a local queue called "DLQ" and tell the queue manager to use it as its dead letter queue.

__ 5. Run the command file through runmqsc on MN##

Step 3: Synchronize the cluster

__ 1. Wait about 30 seconds to allow the cluster members to exchange information.

__ 2. Use the runmqsc command DIS CLUSQMGR(*) on both systems to see the results of the cluster synchronization.

__ 3. What is the CURDEPTH on the SYSTEM.CLUSTER.REPOSITORY.QUEUE on both Queue Managers?

__ 4. Run amqsbcg to see what the messages look like on those queues.

Step 4: Set up the cluster queues

__ 1. Create a local queue called QC## on each of your queue managers. Include the "CLUSTER" parameter.

__ 2. Create a Local Queue called QL## only on your full repository Queue Manager. (such as MR##) Include the "CLUSTER" parameter.

__ 3. Wait about 30 seconds to allow the cluster members to exchange information.

__ 4. What is now the CURDEPTH on the SYSTEM.CLUSTER.REPOSITORY.QUEUE?

__ 5. Use the runmqsc DIS QC(*) command on both Queue Managers to display information about the new cluster-wide queues. Do you see what you expected?

Step 5: Test clustering

__ 1. Use amqsput connected to your MN## queue manager to send a message to QL## that's defined on your full repository queue manager.

__ 2. Use the runmqsc DIS QC(*) command on both queue managers to display information about the new cluster-wide queues. What is the different?

__ 3. Use amqsget connected to your MR## queue manager to retrieve the message from QL##.

Step 6: Connect to the cluster of your partner

__ 1. Modify your MQSC command file "EX6MR.txt" to define one cluster sender channel pointing to your partner's full repository queue manager in your cluster (TO.MR$$).

__ 2. Wait 30 seconds then use the runmqsc DIS CLUSQMGR(*) command to see what shows up.

__ 3. From your non-repository queue manager (MN##), run amqsput to put several messages to QC$$. Append a sequence number to each message.

__ 4. Get your partner to check which queue and queue manager the messages arrive on.

__ 5. Set DEFBIND(NOTFIXED) for your QC## cluster queues on both of your queue managers. Alter both your queue managers to set CLWLUSEQ to ANY. Wait until your partner does the same.

__ 6. Now put several messages again to your partner's QC$$ queue using your own MN## queue manager.

__ 7. On which instances of the destination queue do the messages arrive?

__ 8. Disable puts on your QC## queue on your MR## Queue Manager only. Wait until your partner does the same.

__ 9. Again put several messages to your partner's QC$$ queue using your own MN## queue manager.

__ 10. Disable puts on your QC## queue on your MN## queue manager. Wait until your partner does the same.

__ 11. Again put several messages to your partner's QC$$ queue.

__ 12. Explain the error indication you get.

End of exercise

Exercise Instructions with hints

Step 1: Create full repository queue manager

__ 1. Create a new queue manager MR## to be used as the cluster repository.

__ 2. Start that queue manager.

__ 3. In a new session window, start the Listener for your queue manager MR## on port 91## using the MQ listener.

__ 4. Prepare MQSC command file "EX6MR.txt" to define the cluster connection objects required for your queue manager. The commands needed are as follows:

__ a. Define one cluster receiver channel pointing to this queue manager (TO.MR##).

__ b. Alter the queue manager to be a repository including the cluster name.

__ c. Define a local queue called "DLQ" and tell the queue manager to use it as its dead letter queue. Enter the following:

```
DEFINE CHANNEL(TO.MR##) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) +
CONNAME('hostname(91##)') CLUSTER(CLUS&&)
ALTER QMGR REPOS(CLUS&&)
DEFINE QL(DLQ) REPLACE
ALTER QMGR DEADQ(DLQ)
```

__ 5. Run the command file through runmqsc on MR##. Type **runmqsc MR## < EX6MR.txt**.

Step 2: Create normal cluster member queue manager

__ 1. Create a new queue manager MN## to be used in a queue manager cluster.

__ 2. Start that queue manager.

__ 3. In a new session window, start the Listener for your queue manager MN## on port 90## using the WebSphere MQ listener.

__ 4. Prepare a WebSphere MQ command file "EX6MN.txt" to define the cluster connection objects required for your queue manager. The objects needed should include the following:

__ a. One cluster receiver channel pointing to this queue manager (TO.MN##). Be sure to include the conname and cluster parameters

__ b. One cluster sender channel pointing to a full repository queue manager in your cluster (TO.MR##). Be sure to include the conname and cluster parameters.

__ c. Define a local queue called "DLQ" and tell the queue manager to use it as its dead letter queue. Enter the following:

```
DEFINE CHANNEL(TO.MN##) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) +
CONNAME('hostname(90##)') CLUSTER(CLUS&&)
```



```
DEFINE CHANNEL(TO.MR##) CHLTYPE(CLUSSDR) TRPTYPE(TCP) +
CONNAME ('hostname(91##)') CLUSTER(CLUS&&)
DEFINE QL(DLQ) REPLACE
ALTER QMGR DEADQ(DLQ)
```

__ 5. Run the command file through runmqsc on MN##. Type **runmqsc MN## < EX6MN.txt**.

Step 3: Synchronize the cluster

__ 1. Wait about 30 seconds to allow the cluster members to exchange information.

__ 2. Use the runmqsc command DIS CLUSQMGR(*) on both systems to see the results of the cluster synchronization.

__ 3. What is the CURDEPTH on the SYSTEM.CLUSTER.REPOSITORY.QUEUE on both queue managers? Type **DISPLAY QL(SYSTEM.CLUSTER.REPOSITORY.QUEUE)**

The CURDEPTH of each queue should be greater than 0 as this queue holds system messages, for example the DISPLAY CLUSQMGR returns cluster information about queue managers in a cluster and are stored in this queue.

__ 4. Run amqsbcg to see what the messages look like on those queues.

Step 4: Set up the cluster queues

__ 1. Create a local queue called QC## on each of your queue managers. Include the "CLUSTER" parameter. Type **DEFINE QLOCAL(QC##) CLUSTER(CLUS&&)**

__ 2. Create a local queue called QL## only on your full repository queue manager. (such as MR##) Include the "CLUSTER" parameter. Type **DEFINE QLOCAL(QL##) CLUSTER(CLUS&&)**

__ 3. Wait about 30 seconds to allow the cluster members to exchange information.

__ 4. What is now the CURDEPTH on the SYSTEM.CLUSTER.REPOSITORY.QUEUE? The CURDEPTH value should be more than was recorded in Step 3.3

__ 5. Use the runmqsc DIS QC(*) command on both queue managers to display information about the new cluster-wide queues. Do you see what you expected? The QL## queue is only shown in the display for the full repository queue manager MR## on which it was defined as it is not yet known to the other queue managers in the cluster.

Step 5: Test clustering

__ 1. Use amqsput connected to your MN## queue manager to send a message to QL## that is defined on your full repository queue manager. Enter the following:

amqsput QL## MN##

hello

<Enter>

__ 2. Use the runmqsc DIS QC(*) command on both queue managers to display information about the new cluster-wide queues. What is the different?

The display of the cluster queues for MN## now shows the QL## queue which had been defined as a local queue to MR## in Step 4.2. With clustering, there is no need to specify/define remote queues, and the test PUT of a message in this step was made to a local queue on another QMGR which instigated the search for the queue within the cluster. Once found, the message is PUT to the queue and that QMGR will now be aware of that queue.

__ 3. Use amqsget connected to your MR## queue manager to retrieve the message from QL##. Type the following:

amqsget QL## MR##

message <hello>

no more messages

Step 6: Connect to your partner cluster

$$ is the team number of the partner.

__ 1. Modify your MQSC command file "EX6MR.txt" to define one cluster sender channel pointing to your partner's full repository queue manager in your cluster (TO.MR$$). Type

DEFINE CHANNEL(TO.MR##) CHLTYPE(CLUSSDR) TRPTYPE(TCP) +
CONNAME('hostname(91$$)') CLUSTER(CLUS&&)

__ 2. Wait 30 seconds then use the runmqsc DIS CLUSQMGR(*) command to see what shows up.

__ 3. From your non-repository queue manager (MN##), run amqsput to put several messages to QC$$. Append a sequence number to each message. Type the following:

amqsput QC$$ MN##

Hello

__ 4. Get your partner to check which queue and queue manager the messages arrive on.

__ 5. Set DEFBIND(NOTFIXED) for your QC## cluster queues on both of your queue managers. Alter both your queue managers to set CLWLUSEQ to ANY. Wait until your partner does the same. Type

ALTER QL(QC##) DEFBIND(NOTFIXED)

ALTER QMGR CLWLUSEQ(ANY)

__ 6. Now put several messages again to your partner's QC$$ queue using your own MN## queue manager.

__ 7. On which instances of the destination queue do the messages arrive?

__ 8. Disable puts on your QC## queue on your MR## queue manager only. Wait until your partner does the same. Type **ALTER QL(QC##) PUT(DISABLED)**

__ 9. Again put several messages to your partner's QC$$ queue using your own MN## queue manager.

__ 10. Disable puts on your QC## queue on your MN## queue manager. Wait until your partner does the same.

__ 11. Again put several messages to your partner's QC$$ queue.

__ 12. Explain the error indication you get. The error you should receive is MQRC 2268 MQRC_CLUSTER_PUT_ INHIBITED

End of exercise

Exercise review and wrap-up

At the end of this exercise, you should be able to:

- Create clusters
- Define all required WebSphere MQ objects for queue manager clusters
- Test and configure clusters
- Manage workload in clusters

Exercise 7. Object security administration with the OAM

Estimated time

01:00

What this exercise is about

In this exercise, you use the OAM utilities to set the ACL on a queue and the utilities to see the effect of attempting to breach security.

What you should be able to do

At the end of this exercise, you should be able to:

- Use setmqaut to create an ACL on a queue
- Use dspmqaut to display the ACL on a queue
- Test security using WebSphere MQ utilities

Introduction

In this exercise you will use a non-privileged user ID to access WebSphere MQ objects, experience access failures, examine related diagnostics and apply the correct rules. You will use the WebSphere MQ Explorer and the setmqaut command to apply OAM rules.

Requirements

- The appropriate WebSphere MQ V7 product installed on the system.
- A text editor.
- IH03 (RFHUtil) SupportPac installed
- MS0P “WebSphere MQ Events and Statistics Plug-in” SupportPac to examine WebSphere MQ security events
- In general, the WebSphere MQ Explorer can be used in these labs even for the AIX and Solaris systems. Ensure the SYSTEM.ADMIN.SVRCONN channel is defined and your listener

is running. Also ensure the Aventail VPN software is running and connected.

- WebSphere MQ Explorer functionality cannot be used to perform the non-privileged user tasks.

Exercise instructions

The testing methodology

Non-privileged user IDs for testing access:

UNIX

On AIX there are user IDs named user1.. usernn defined which are paired with the team number team1 .. teamnn. Start a second PCOMM session, log in as usernn/usernn

On Solaris there are paired user IDs named wm203unn defined for use. Start a second session, log in as wm203unn/wm203unn

Windows

On Windows, there is a user ID called testuser. To start a command window running as testuser, do this from a dos prompt: runas /user:testuser cmd

You will be prompted for a password which is "testuser" also.

These user IDs are referred to in this exercises as your “user” user ID. Administration user IDs are your regular or student IDs. This is referred to as your administration user ID.

The first exercise will be to log in as a user and make sure it cannot access a queue like QL.A, that is, it receives a 2035 error when accessed using a sample application.

__ 1. Using your administration user ID, enable authority events on your queue manager. This is one way to monitor authority violations.

__ 2. Start a session/window as a non-privileged user ID.

Using a sample program, attempt to put a message to queue QL.A on your team queue manager QMC##.

__ 3. Find a record of the violation using the MS0P Events and Statistics SupportPac. For the UNIX platforms, you will have to connect your WebSphere MQ Explorer in Windows to administer your remote queue manager. For Windows queue managers only, a security violation message is also written to the WebSphere MQ Error logs.

View the *Queue Manager Events* in the WebSphere MQ Explorer. Find the last event that shows the access violation.

Windows

Using your administration user ID, look at the queue manager error logs:

Windows: C:\Program Files\IBM\WebSphere MQ\Qmgrs\<qmgr-

ame>\errors\AMQERR01.LOG

Error message example:

(Note: The error message is produced on Windows only)

```
24/07/2008 22:29:22 - Process(8332.57) User(MUSR_MQADMIN)
program(amqzlaa0.exe)
AMQ8077: Entity 'testuser' has insufficient authority to access object
'QL.A'.
EXPLANATION:
The specified entity is not authorized to access the required object. The
following requested permissions are unauthorized: put/inq
ACTION:
Ensure that the correct level of authority has been set for this entity
against the required object, or ensure that the entity is a member of a
privileged group.
```

__ 4. Using WebSphere MQ Explorer or setmqaut, add connection authority for group users to your team queue manager. Use your administration user ID for this task.

__ 5. Using your administration user ID, give group users (staff on Solaris) general API access (open, get, put, inq, close and so forth) to QL.A on your queue manager.

__ 6. With your user user ID, use amqsbcg on QL.A to verify the access is now allowed.

__ 7. With your user user ID, place a message on QL.A ready for the next part of the exercise.

Testing Context Security.

__ 1. The IH03 SupportPac which includes the RFHUtil utility is installed on the Windows image. For testing on a local Windows queue manager use RFHUtil directly. For testing against a remote AIX or Solaris queue manager use the client version of RFHUtil: rfhutilc. Make sure you set your MQSERVER environment variable correctly beforehand.

__ 2. With your user user ID, start rfhutil or rfhutilc and get a message from QL.A. This will load all RFHUtil with all the message data and headers.

__ 3. Still using RFHUtil on the loaded message, set the identity context on the main tab and enter Administrator as the identity on the MQMD tab. Put this message back QL.A. Did it work?

__ 4. Set *Identity Context* is not set by default. Give the "users" group (or group “staff” on Solaris) access to do this using WMQ Explorer or setmqaut using your administrator ID.

__ 5. Now retry the "Write Q" in rfhutilc. Does it work now? No it doesn't. You need to perform an extra setmqaut to make this work. There is a undocumented authority required to use set message authority. Using WebSphere MQ Explorer or setmqaut, give users setid access to the queue manager.

__ 6. Now retry the put using RFHUtil with set identity. It should work now.

__ 7. Use amqsbcg (under your administrator ID) or the WebSphere MQ Explorer browse QL.A and look for the userIdentifer field of the MQMD in the newly put message.

Pub/Sub access controls

__ 1. Create a PubSub topic called sports.results.cricket with topic string: sports/results/cricket using WebSphere MQ Explorer or runmqsc (under your administrator ID).

__ 2. From the "user" session, use the amqssub command to subscribe to sports/results/cricket. This should fail with a 2035 authorization error.

__ 3. Now using either the WebSphere MQ Explorer or the setmqaut command, give group "users" subscription access to this topic.

__ 4. Retry the amqssub command. It should not fail with 2035 now.

__ 5. Similarly, try out publication access control.

Generic Profiles

__ 1. Using WebSphere MQ Explorer remove the specific profile that allows users in group "users" (or group “staff” on Solaris) accessing QL.A.

__ 2. Create a new generic profile called QL.*.TEST that allows "users" to access (browse, put, get, set, inq)

__ 3. Define a queue called QL.A.TEST.

__ 4. As the user user ID, try to access QL.A now, you should get 2035.

__ 5. Now using your user user ID, try to access QL.A.TEST. It should work returning 2033 - No messages available. This is not an access failure.

__ 6. Inspect the profiles that apply to QL.A.TEST, either using the WebSphere MQ Explorer or using dmpmqaut.

__ 7. How many times did you issue the MQSC command: refresh security in this exercise?

End of exercise

Exercise instructions with hints

Step 1: The testing methodology.

__ 1. Using your administration user ID, enable authority events on your queue manager. This is one way to monitor authority violations. Use the following according to your system:

Windows

Using WMQ Explorer, right-click queue manager name on left, Properties: Event: Authority Events: Enable.

UNIX

echo "alter qmgr authorev(enabled)" | runmqsc QMCnn

__ 2. Start a session/window as a non-privileged *user* user ID. Use the following according to your system:

Windows

runas /user:testuser cmd enter password at the prompt. Password is the same as the user ID.

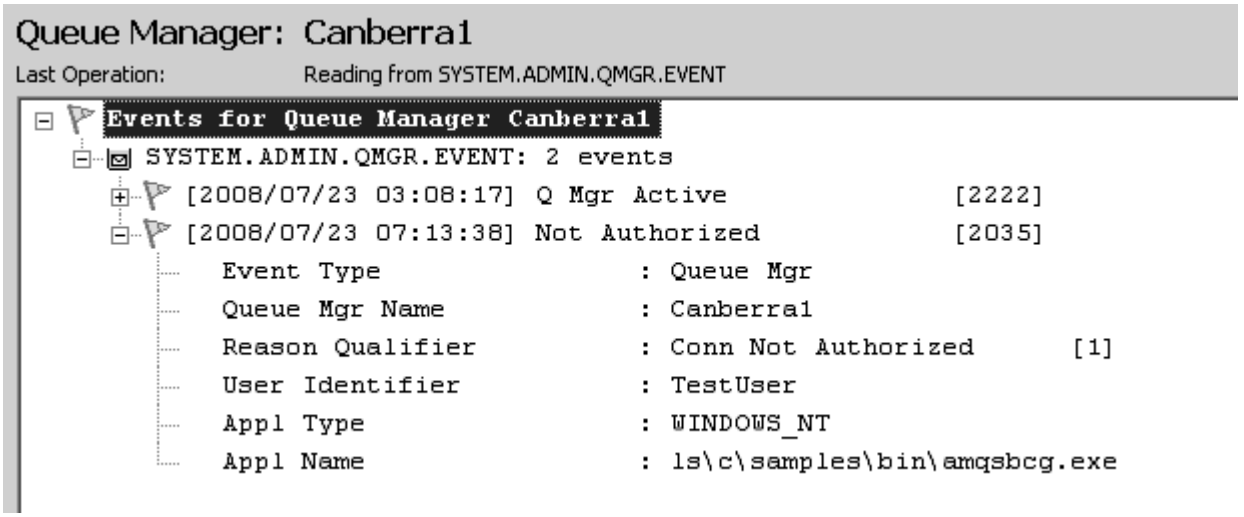
UNIX

Using PCOMM**, File-> Run the same** will duplicate the current session. Log in as the *user##* on AIX or *wm203u##* on Solaris. Password is the same as the user ID.

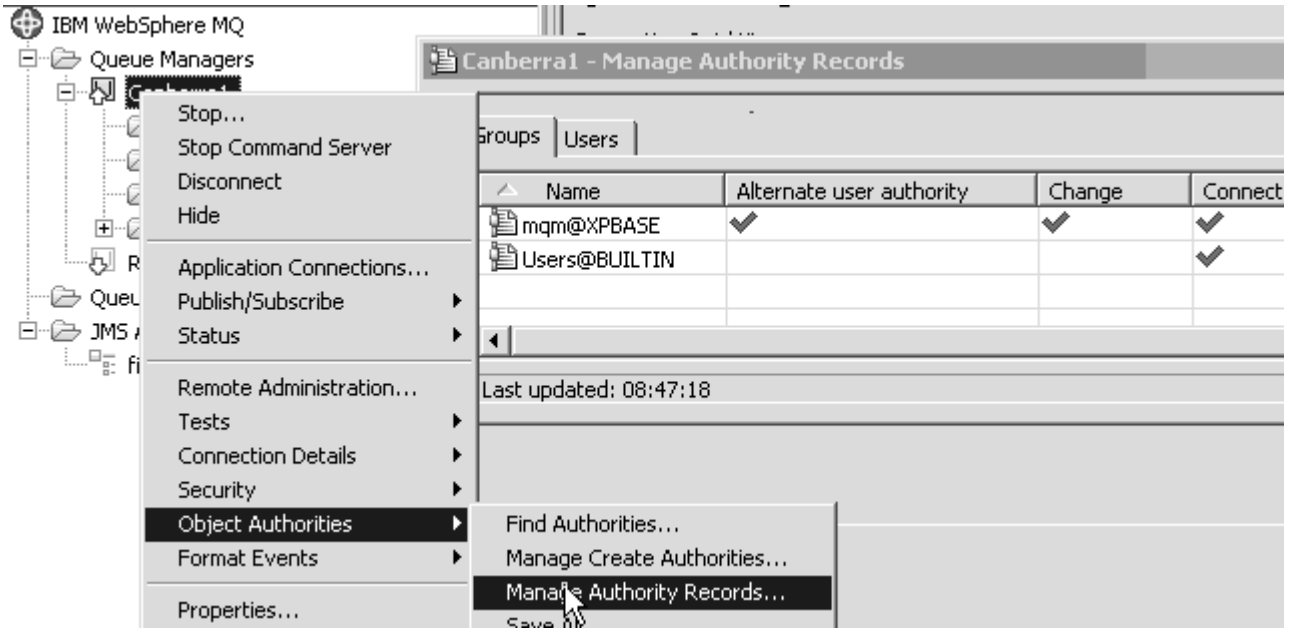
Using a sample program, attempt to put a message to queue QL.A on your team queue manager QMC##.

__ 3. Find a record of the violation using the MS0P *Events and Statistics* SupportPac. For the UNIX platforms, you will have to connect your WebSphere MQ Explorer in Windows to administer your remote queue manager. For Windows queue managers only, a security violation message is also written to the WebSphere MQ Error logs.

View the *Queue Manager Events* in the WebSphere MQ Explorer. Find the last event that shows the access violation. On Windows, right-click the queue manager name on the left, chose "Format Events" -> "Event Messages"



__ 4. Using WebSphere MQ Explorer or setmqaut, add connection authority for group *users* (or group “staff” on Solaris) to your team queue manager. Use your administration user ID for this task.



UNIX

(AIX) setmqaut -m QMCnn –t qm –g users +connect

(Solaris) setmqaut -m QMCnn -t qm -g staff +connect

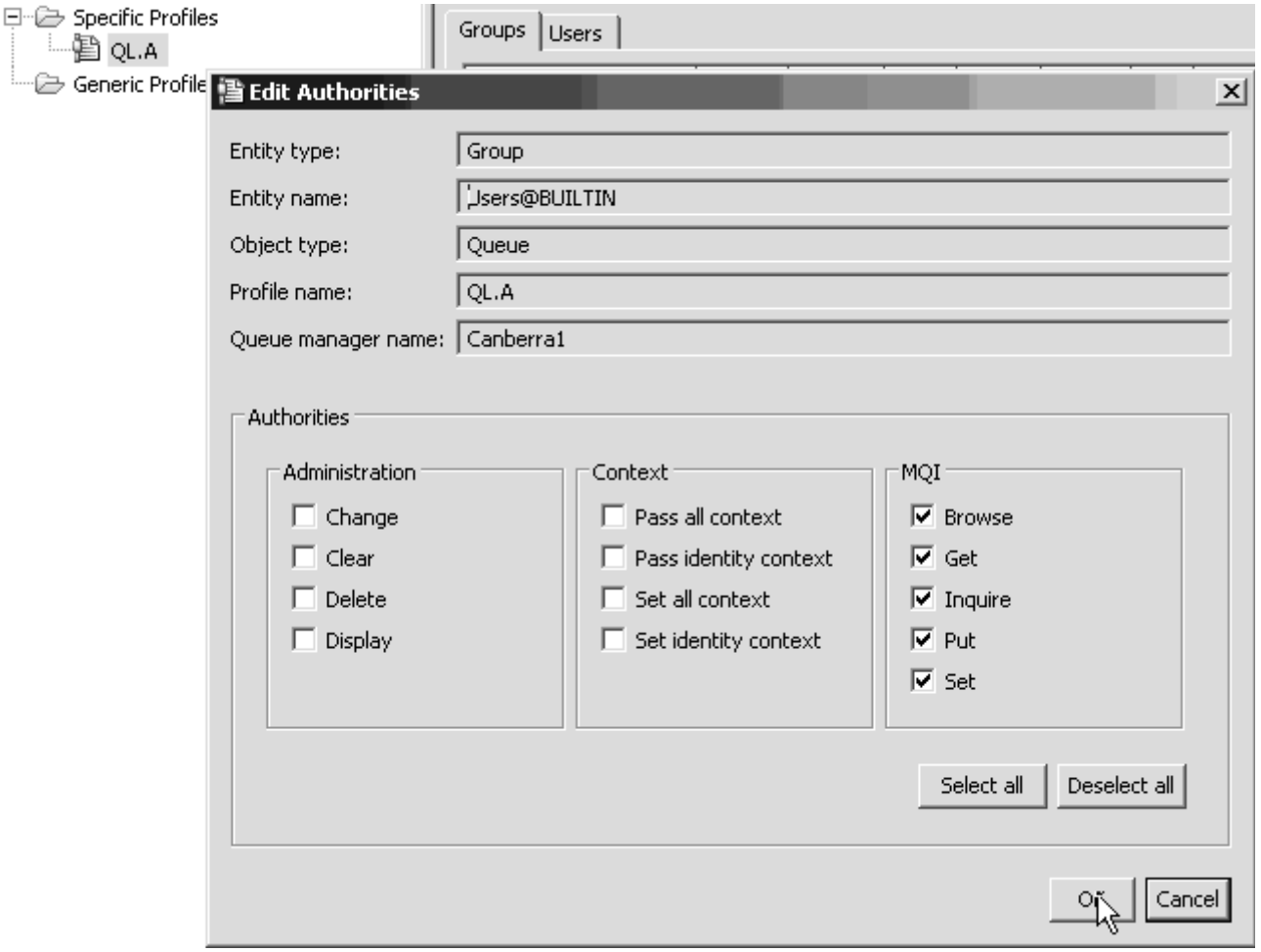
__ 5. Using your administration user ID, give group *users* (*staff* on Solaris) general API access (connect, open, get, put, inq, close and so forth) to QL.A on your queue manager. Use the following according to your system:

UNIX

(AIX) setmqaut -m QMCnn -t q -n QL.A -g users +allmqi
(Solaris) setmqaut -m QMCnn -t q -n QL.A -g staff +allmqi

Windows

Using WMQ Explorer add allmqi authority:



__ 6. With your *user* user ID, Use amqsbcg on QL.A to verify the access is now allowed. Type **amqsbcg QL.A AMC##**

__ 7. With your *user* user ID, place a message on QL.A ready for the next part of the exercise. Type **amqsput QL.A QMCnn**

Step 2: Testing context security.

__ 1. The IH03 SupportPac which includes the RFHUtil utility is installed on the Windows image. For testing on a local Windows queue manager use RFHUtil directly. For testing against a remote AIX or Solaris queue manager use the client version of RFHUtil: rfhutilc. Make sure you set your MQSERVER environment variable correctly beforehand.
set MQSERVER=SYSTEM.DEF.SVRCONN/TCP/ip-addr-of-server(port)

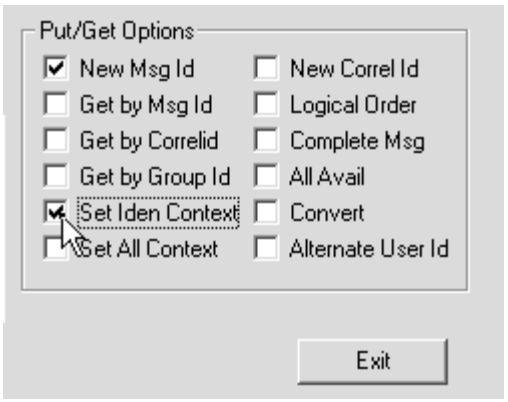
__ 2. With your *user* user ID, start rfhutil or rfhutilc and get a message from QL.A. This will load all RFHUtil with all the message data and headers.

Make sure MQSERVER is set before running RFHUtilc. In the Queue Name box, enter **QL.A,** click **Read Queue**. This does an MQGET on QL.A.

__ 3. Still using RFHUtil on the loaded message, set the identity context on the main tab and enter **Administrator** as the identity on the MQMD tab. Put this message back QL.A. Did it work?

Remember context is metadata describing the origin of the message. Applications can use the context data to authenticate messages, so changing the context data might well be something you would prevent being changed arbitrarily.

Set the user ID of the putter from the test user ID from something with more power, say Administrator. Do this by checking the "set identity context" Put/Get option at the bottom RHS of the "main" tab.



Then switch to the MQMD tab. Notice the user ID field. Overtype with Administrator, Note only 12 characters are accepted. Switch back to the main tab and click **write q** to put the message back on the queue. What happens? It fails with a 2036 when trying to open the queue.

__ 4. Set Identity Context is not set by default. Give the "users" group (or group “staff” on Solaris) access to do this using WMQ Explorer or setmqaut using your administrator ID. Type **setmqaut -m QMCnn -t q -n QL.A -g users +setid**

__ 5. Now retry the "Write Q" in rfhutilc. Does it work? No it does not. You need to perform an extra setmqaut to make this work. There is a undocumented authority required to use set message authority. Using WebSphere MQ Explorer or setmqaut, give *users* (or group “staff” on Solaris) **setid** access to the queue manager. Type **setmqaut -m QMCnn -t qmgr -g users +setid**

__ 6. Now retry the put using RFHUtil with set identity. It should work now

__ 7. Use amqsbcg (under your administrator ID) or the WebSphere MQ Explorer browse QL.A and look for the userIdentifer field of the MQMD in the newly put message.

Step 3: Publish/subscribe access controls

__ 1. Create a PubSub topic called **sports.results.cricket** with topic string: sports/results/cricket using WebSphere MQ Explorer or runmqsc (under your administrator ID).

Using WebSphere MQ Explorer, right-click **Topics; New, Topic.**

Enter a topic name **sports.results.cricket** click **Next**. Enter a Topic String: **sports/results/cricket**

Click **Finish**.

__ 2. From the "user" session, use the amqssub command to subscribe to sports/results/cricket. Type **amqssub sports/results/cricket QMC##**

This will fail with a 2035 authorization error

__ 3. Now using either the WebSphere MQ Explorer or the setmqaut command, give group "users" (or group “staff” on Solaris) subscription access to this topic. Type **setmqaut -m QMCnn -t topic -n sports.results.cricket –g users +sub**

__ 4. Retry the amqssub command. It should not fail with 2035 now.

__ 5. Similarly, try out publication access control. Type **Setmqaut –m QMCnn –t topic –n sports.results.cricket –g users +pub**

Step 4: Generic profiles

__ 1. Using WebSphere MQ Explorer remove the specific profile that allows users in group "users" (or group “staff” on Solaris) accessing QL.A. Type **setmqaut -m QMCnn -t q -n QL.A -g users -remove**

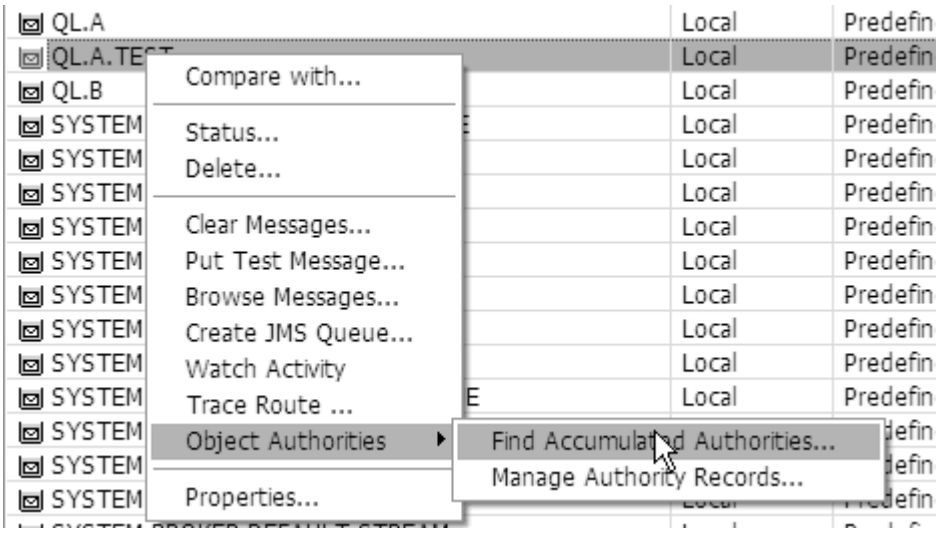
__ 2. Create a new generic profile called QL.*.TEST that allows "users" (or group “staff” on Solaris) to access (browse, put, get, set, inq). Type **setmqaut -m QMC## -t q -n QL.*.TEST -g users +allmqi**

__ 3. Define a queue called QL.A.TEST. Type **define ql(QL.A.TEST)**

__ 4. As the *user* user ID, try to access QL.A now, you should get 2035. Type **amqsput QL.A QMC##**

__ 5. Now using your *user* user ID, try to access QL.A.TEST. It should work returning 2033 - No messages available. This is not an access failure. Type **amqsget QL.A.TEST QMC##**

__ 6. Inspect the profiles that apply to QL.A.TEST using the WMQ



Inspect the profiles that apply to QL.A.TEST using dmpmqaut. Type **dmpmqaut -m QMCnn-t q -g users -n QL.A.TEST**

profile: QL.*.TEST

- **object type: queue**

entity: Users@BUILTIN

entity type: group

authority: get browse put inq set

Note: Solaris users use group “staff”.

__ 7. How many times did you issue the MQSC command: **refresh security** in this exercise? None.

End of exercise

Exercise review and wrap-up

Having completed this exercise, you should be able to:

- Use **setmqaut** to create an ACL on a queue
- Use **dspmqaut** to display the ACL on a queue
- Test security using MQ utilities

Exercise 8. Client attachment

Estimated time

01:00

What this exercise is about

In this exercise, you configure your system to act as a client connected to a partner's system. You use two different methods (MQSERVER environment variable and CLNTCONN channel definition) in order to gain experience with each. You encounter security issues and will use the MCAUSER parameter to overcome this problem.

What you should be able to do

At the end of this exercise, you should be able to:

- Use runmqsc to create a SVRCONN channel to support client connections
- Set up a client connection using the MQSERVER environment variable
- Set up a client connection using a CLNTCONN channel definition
- Explain the security requirements of WebSphere MQ clients

Introduction

You work in teams of two for the first step. The instructor is the server for the second step.

To put, browse, and get messages on your client system, use the sample programs. The following is a brief description of each of the programs you may need to use.

- amqsputc - This program is invoked the same way as amqsput and has the same parameter structure. But it connects as a WebSphere MQ client instead of a direct connection to the WebSphere MQ server.
- amqsbcgc - This program is invoked the same way as amqsbcg and has the same parameter structure but it connects as a WebSphere MQ client.

- amqsgetc - This program is invoked the same way as amqsget and has the same parameter structure but it connects as a WebSphere MQ client.

Requirements

Sample programs amqsputc, amqsbcg, amqsbcgc and amqsgetc are invocable from any position within the file system hierarchy.

Exercise instructions

In this exercise your system provides two functions:

- Your queue manager is the server for the WebSphere MQ clients of your partner teams.
- The client sample programs on your machine connects by the WebSphere MQ client function to queue managers of your partner teams.

__ 1. Server queue manager setup.

__ a. Reuse the queue manager used in the exercise 4 (probably QMC##) to be used as a server queue manager.

__ b. On that queue manager define a SVRCONN channel to make your queue manager connectable by WebSphere MQ Clients.

- Use QMC##_CLNT as the channel name, where ## is your partner's team number
- Protocol is TCP

__ c. Be sure that an appropriate listener function is active for the server queue manager.

__ 2. Client setup (method 1)

Use the MQSERVER environment variable to provide a client-connection channel definition to be able to connect to your partner team server queue manager.

__ 3. Test the client connection (setup method 1).

__ a. Ensure your partner team has completed the server setup.

__ b. Use amqsputc to put messages on the local queue QL.A on the server.

__ c. Use amqsbcgc to browse the messages on the server queue.

__ d. Use amqsgetc to get the messages from the server queue.

__ 4. Server queue manager setup using Auto-definition of channels.

Enable Channel Auto Definition in your queue manager so all teams are able to connect to your queue manager.

__ 5. Client setup (method 2)

Use your own queue manager to build a client channel definition table to enable a WebSphere MQ client to connect to each queue manager in the classroom which has enabled channel auto definition.

Create two client connection channel entries to connect to your partner queue manager. Use CLNT_A and CLNT_B as the channel names.

__ 6. Test the client connection (setup method 2).

__ a. Ensure your partner team has completed the server setup.

__ b. On the client system ensure the following environment variables are pointing to the just created client channel definition table. Be sure to unset the MQSERVER.

MQCHLLIB=….

MQCHLTAB=…

__ c. Use amqsputc again to put a message to QL.A on your partner's queue manager and ensure that the operation is completed successfully. Verify that the new server connect channel CLNT_A is now defined on your partner's queue manager.

__ d. Stop your channel CLNT_A and verify that your partner has done the same. Then use amqsputc to put a message to your partner's QL.A. Verify that the new server connect channel CLNT_B is now defined on your partner's queue manager, and that the message successfully arrived on the queue.

__ 7. Security ramifications of client connections.

__ a. Did you have to provide a user ID and or password to access your partner queue manager?

__ b. In the absence of any security exits, the WebSphere MQ client passes the client's logged-on user ID to the server using a field in the channel descriptor where it is used for access control. What's the problem with this?

End of exercise

Optional exercises

The purpose of this exercise is perform administration on a remote queue manager using runmqsc and the command server. You reuse the queue manager and channel connections established in exercise 4. This exercise may be difficult to perform in the UNIX environment of the classroom, as it requires that your queue manager be the default queue manager.

__ 1. Reconnect to the remote queue manager

   __ a. On both sides, make sure the channels are using a transmission queue with the same name as the appropriate remote queue manager

   __ b. The channels need to be active (or triggered) for this to work.

__ 2. Test remote administration

   __ a. Change the managing queue manager to run as a default queue manager

   __ b. On the managing queue manager, issue runmqsc to manage the other queue manager

   __ c. Use the following command to check if you are connected to the target (managed) queue manager. Type DISPLAY QMGR

   __ d. Issue some simple commands like DISPLAY Q(*).

   __ e. Change the target queue manager's description and change QL.A to disable puts

   __ f. On the managed queue manager, check using runmqsc that these changes have occurred.

End of exercise

Exercise instructions with hints

In this exercise your system provides two functions:

- Your queue manager is the server for the WebSphere MQ clients of your partner teams.
- The client sample programs on your machine connects by the WebSphere MQ client function to queue managers of your partner teams.

__ 1. Server queue manager setup.

__ a. Reuse the queue manager used in the exercise 4 (QMC##) to be used as a server queue manager.

__ b. On that queue manager define a SVRCONN channel to make your queue manager connectable by WebSphere MQ Clients.

- Use QMC##_CLNT as the channel name, where ## is your partner's team number
- Protocol is TCP. Type **DEFINE CHL(QMC##_CLNT) CHLTYPE(SVRCONN) TRPTYPE(TCP)**

__ c. Be sure that an appropriate listener function is active for the server queue manager

__ 2. Client setup (method 1)

Use the MQSERVER environment variable to provide a client-connection channel definition to be able to connect to your partner team server queue manager. Use the following commands according to your system type:

UNIX

```
export MQSERVER=QMC##_CLNT/TCP/QMC##(90##)
```

Windows

```
SET MQSERVER=QMC##_CLNT/TCP/QMC##(90##)
```

__ 3. Test the client connection (setup method 1)

__ a. Ensure your partner team has completed the server setup

__ b. Use **amqsputc** to put messages on the local queue QL.A on the server. Type **amqsputc QL.A QMC##**

__ c. Use **amqsbcgc** to browse the messages on the server queue. The value of "Reply-to-qmgr"in the MQMD shows you, which server queue manager was connected and has created the message.

__ d. Use **amqsgetc** to get the messages from the server queue

__ 4. Server queue manager setup using Auto-definition of channels

Enable Channel Auto Definition in your queue manager so all teams are able to connect to your queue manager. Type **ALTER QMGR CHAD(ENABLED)**

__ 5. Client setup (method 2)

__ a. Use your own queue manager to build a client channel definition table to enable a WebSphere MQ client to connect to each queue manager in the classroom which has enabled channel auto definition.

__ b. Use CLNT_A and CLNT_B as the channel names. Use the following commands:

```
DEF CHL(CLNT_A) CHLTYPE(CLNTCONN) REPLACE +
TRPTYPE(TCP) +
CONNAME('QMC##(90##)') +
QMNAME(QMC##)
DEF CHL(CLNT_B) CHLTYPE(CLNTCONN) REPLACE +
TRPTYPE(TCP) +
CONNAME('QMC##(90##)') +
QMNAME(QMC##)
(## = other team number)
```

__ 6. Test the client connection (setup method 2)

__ a. Ensure your partner team has completed the server setup.

__ b. On the client system ensure the following environment variables are pointing to the just created client channel definition table. Be sure to unset the MQSERVER.

```
MQCHLLIB=….
MQCHLTAB=…
```

UNIX

Default location on the creating queue manager:

```
export MQCHLLIB=/var/mqm/qmgrs/<qmgrname>/@ipcc
export MQCHLTAB=amqclchl.tab
export MQSERVER=
```

Windows

Windows: Default location on the creating queue manager:

```
SET MQCHLLIB=..\mqm\qmgrs\<qmgrname>\@ipcc
SET MQCHLTAB=amqclchl.tab
SET MQSERVER=
```

__ c. Use **amqsputc** again to put a message to QL.A on your partner's queue manager and ensure that the operation is completed successfully. Verify that the new server connect channel CLNT_A is now defined on your partner's queue manager. Type **amqsputc QL.A QMC##**

__ d. Stop your channel CLNT_A and verify that your partner has done the same. Then use **amqsputc** to put a message to your partner's QL.A. Verify that the new server connect channel CLNT_B is now defined on your partner's queue manager, and that the message successfully arrived on the queue. Type

```
runmqsc yourqmgrname
 stop channel(CLNT_A)
 end
 amqsputc QL.A QMC##
```

__ 7. Security ramifications of client connections

__ a. Did you have to provide a user ID and or password to access your partner queue-manager? No

__ a. In the absence of any security exits, the WebSphere MQ client passes the client logged-on user ID to the server using a field in the channel descriptor where it is used for access control. What is the problem with this?

The problem with this approach is that *spoofing*, or pretending to be someone you are not is trivial. If you have administrative control of the client system, then it is a trivial matter to define a user ID, say mqm or MUSR_MQADMIN, then use this user ID to connect to a remote queue manager with full access to that queue manager. Clearly this is unsatisfactory.

End of exercise

Optional exercises with hints

The purpose of this exercise is perform administration on a remote queue manager using runmqsc and the command server. You reuse the queue manager and channel connections established in exercise 4. This exercise may be difficult to perform in the UNIX environment of the classroom, as it requires that your queue manager be the default queue manager.

__ 1. Reconnect to the remote queue manager

   __ a. On both sides, make sure the channels are using a transmission queue with the same name as the appropriate remote queue manager

   __ b. The channels need to be active (or triggered) for this to work.

__ 2. Test remote administration

   __ a. Change the **managing** queue manager to run as a **default** queue manager

   __ b. On the **managing** queue manager, issue **runmqsc** to manage the other queue manager. Type **runmqsc** -w 15 *remote-qmgrname* (where -w 15 is the wait time for responses).

   __ c. Use the following command to check if you are connected to the target (managed) queue manager.
   DISPLAY QMGR

   __ d. Issue some simple commands like DISPLAY Q(*).

   __ e. Change the **target** description of the queue manager and change QL.A to disable puts

   __ f. On the managed queue manager, check using **runmqsc** that these changes have occurred.

End of exercise

Exercise review and wrap-up

Having completed this exercise, you should be able to:

- Use runmqsc to create a SVRCONN channel to support client connections
- Setup a client connection using the MQSERVER environment variable
- Setup a client connection using a CLNTCONN channel definition
- Discuss the security ramifications of MQ clients

Exercise 9. Backup and restore of WebSphere MQ object definitions

Estimated time

00:30

What this exercise is about

In this exercise, you will look at the usage of the MS03 SupportPac for backup and restore of WebSphere MQ object definitions. You will also look at backup of security definitions using amqoamd or the WebSphere MQ Explorer.

What you should be able to do

At the end of this exercise, you should be able to:

- Use MS03 to download the object definitions of a queue manager
- Use runmqsc to upload those definitions to another queue manager

Introduction

To complete this exercise you should be familiar with the following:

- MS03 - This SupportPac interrogates the attributes of all the objects defined to a queue manager (local or remote), and saves them to a file, in a format that is suitable for use with runmqsc. It is therefore possible to use this SupportPac to save the definitions of objects known to a queue manager and subsequently re-create that queue manager.
- amqoamd - This is a command to dump the OAM security definitions. Using the –s option, will format the output as setmqaut commands, such that the file can be used to re-create the security definitions.

Requirements

- Active Internet connection to download MS03

Exercise instructions

Downloading, Installing and using the MS03 SupportPac

__ 1. Download MS03 from the Websphere MQ SupportPac site.

UNIX

Since there is no Web interface, follow the instructions below to download the SupportPac directly using ftp.

```
mkdir ms03
cd ms03
ftp 207.25.253.40
user: anonymous
password: <your_email_address>
cd /software/integration/support/supportpacs/individual
bin
get ms03_unix.tar.Z
bye
```

__ 2. Unzip and install the MS03 SupportPac in your home directory on UNIX, or "My Documents" on Windows.

__ 3. Run the saveqmgr command using the -m, -q and -f options.

- -m <qmgr>: name of the local queue manager to connect
- -q: quiet mode, do not issue warnings about unknown PCF attributes
- -f <file>: allows the output file to be named.

If -f is specified without a file name, output is named <qmgrname>.MQS

__ 4. Examine the resulting file, and answer the following questions

__ a. Are SYSTEM.* objects included in the dump?

__ b. How can you suppress generation of the SYSTEM.* objects?

__ c. Is the REPLACE option present in the generated definitions?

__ d. What does REPLACE do?

Restoring the object definitions

You will now simulate the running of this generated file. This would be done if you needed to redefine all your WebSphere MQ objects. In this case you will not actually replace any definitions because some of the SYSTEM queues are not empty, and redefining these objects will effectively remove all cluster, channel state and authorities stored in the SYSTEM queues.

__ 1. Use runmqsc to check if there are SYSTEM queues which are not empty

__ 2. Simulate the restore function without actually replacing the definitions. To do this use runmqsc with verify mode. Use the -v option to run in verify mode.

Note

Ensure that you use the -v option.

Dumping security definitions

__ 1. Use the amqoamd utility to dump the security definitions. Use the -s option to format the output as a series of setmqaut commands.

Windows

Try using the MQ Explorer to dump the OAM definitions for QMC##

__ 2. Examine the output from the security definitions file

Find the setmqaut entries for QL.A

__ 3. Run dmpmqaut -m QMC## -n QL.A -t q. This will dump a formatted view of the authorities in place for the "q" object QL.A

__ a. Note the format of this output

__ b. Consider how the different format leads to different uses

End of exercise

Exercise instructions with hints

Step 1: Downloading, installing and using the MS03 SupportPac

__ 1. Download MS03 from the Websphere MQ SupportPac site. Refer to the following according to your system type:

Windows

Go to http://www-306.ibm.com/software/integration/wmq/support/

Choose “Download” from the “Websphere MQ Support” links box

Choose “SupportPacs” from the “Featured Websphere MQ Downloads”

Choose Product: WebSphere MQ

Find MS03 and select the link

Select the download package link for Windows

Since there is no Web interface, follow the instructions below to download the SupportPac directly using ftp

```
mkdir ms03
cd ms03
ftp 207.25.253.40
```

user: anonymous

password: <your_email_address>

cd /software/integration/support/supportpacs/individual

bin

get ms03_unix.tar.Z

bye

__ 2. Unzip and install the MS03 SupportPac in your home directory on UNIX, or *“My Documents”* on Windows.

Windows

Copy the MS03_WIN.ZIP file to a temporary directory and uncompress. The Windows executable file is named SAVEQMGR.EXE

UNIX

Uncompress the compressed tar file

```
uncompress ms03_unix.tar.Z  OR
gunzip ms03_unix.tar.Z
```

Now untar the file

```
tar –xvf ms03.tar
```

The AIX executable file is named saveqmgr.aix

The Solaris executable file is named saveqmgr.solaris

__ 3. Run the saveqmgr command using the –m, -q and –f options.

- -m <qmgr>: name of the local queue manager to connect
- -q: quiet mode, do not issue warnings about unknown PCF attributes
- -f <file>: allows the output file to be named.

If –f is specified without a file name, output is named <qmgrname>.MQS

Windows

```
saveqmgr -m QMC## -q -f
```

UNIX

```
./saveqmgr.aix -m QMC## -q -f
./saveqmgr.solaris –m QMC## -q -f
```

__ 4. Examine the resulting file, and answer the following questions:

__ a. Are SYSTEM.* objects included in the dump? Yes

__ b. How can you suppress generation of the SYSTEM.* objects? Use the –s option of saveqmgr

__ c. Is the REPLACE option present in the generated definitions? Yes

__ d. What does REPLACE do? The replace and noreplace options control whether any existing definitions will be replaced with the new ones. The WebSphere MQ default is NOREPLACE.

Step 2: Restoring the object definitions

You will now simulate the running of this generated file. This would be done if you needed to redefine all your WebSphere MQ objects. In this case you will not actually replace any definitions because some of the SYSTEM queues are not empty, and redefining these objects will effectively remove all cluster, channel state and authorities stored in the SYSTEM queues.

__ 1. Use runmqsc to check if there are SYSTEM queues which are not empty. Type **runmqsc QMC##**

display ql(SYSTEM.*) where(curdepth gt 0)

__ 2. Simulate the restore function without actually replacing the definitions. To do this use runmqsc with **verify** mode. Use the –v option to run in verify mode.

Note

Ensure that you **use the -v option**. Type **runmqsc -v QMC## < QMC##.MQS**

Step 3: Dumping security definitions

__ 1. Use the amqoamd command to dump the security definitions. Use the –s option to format the output as a series of setmqaut commands. Type **amqoamd -s -m QMC##**.

Windows

In the Windows Explorer, right-click the queue manager name and select “object authorities” -> “save all”

__ 2. Examine the output from the security definitions file. Find the setmqaut entries for QL.A

__ 3. Run `dmpmqaut –m QMC## -n QL.A –t q`.

This will dump a formatted view of the authorities in place for the “q” object QL.A

__ a. Note the format of this output

__ b. Consider how the different format leads to different uses

amqoamd –s outputs the security definitions as a list of setmqaut commands. These commands can then be used to re-create the security definitions on a new queue manager. In contrast, dmpmqaut, dumps a formatted view of the authorities, which is useful for finding all the authorities set against a particular websphere MQ object, however this output cannot be used to re-create the security definitions without reformatting.

End of exercise

Exercise review and wrap-up

Having completed this exercise, you should be able to:

- Locate, download and install the MS03 SupportPac
- Use MS03 to download a queue manager's object definitions
- Use runmqsc to upload those definitions to another queue manager
- Dump the OAM security definitions for a queue manager

Exercise 10.Trace route utility

Estimated time

0:30

What this exercise is about

In this exercise, you create a network of two queue managers connected using channels in a linear fashion, with remote queue definitions that will cause a message to be passed from one to another and back again. You inject a test message using the traceroute utility and observe the results.

What you should be able to do

At the end of this exercise, you should be able to:

- Use the traceroute utility to troubleshoot production problems with messages going astray
- Interpret the output from traceroute in order to make configuration corrections

Introduction

In this exercise the WebSphere MQ trace route utility will be employed to diagnose some typical situations where message processing is halted across a distributed network.

Requirements

This exercise depends upon the success of the distributed queuing exercise. At a minimum, it expects functioning channels between two queue managers, denoted as QMC## for the local queue manager and QMC?? for the remote or partner's queue manager. If your queue manager is not correctly setup for distributed queuing with a partner queue manager, see your instructor for assistance. A script of commands is available to complete configuration.

If you are using a Windows image, you could use two queue managers in the same Windows image or a partner's queue manager if connectivity can be established. If you define or use a second queue

manager on the same Windows image, remember to create all the necessary channels and transmission queues.

Exercise instructions

In this exercise, QMC## refers to your queue manager where ## is the number of your ID or team. ?? is the number of your partner or partner team

__ 1. Observe the trace route output of the normal delivery of a message using a remote queue.

__ a. Place a message on QRMT?? on your local queue manager. Have your partner verify the message arrived. Use **amqsput** or the WebSphere MQ Explorer to put the message on the remote queue.

__ b. Now run the following trace route command and observer the response:
`dspmqrte -m QMC## -q QRMT?? -v outline`

__ c. You should see the final operation of the last activity is **Discard**. This means the trace route message reached it's final destination. The **Feedback** of this last operation should always be checked. If it is:
Feedback: NotDelivered
this is normal, as the trace route message is never delivered to actual target queues as this could adversely affect applications that are not expecting a message like this.

__ d. Examine each activity of the trace route output and the process names associated with each activity.

__ 2. Now, observe the trace route output when a channel is stopped.

__ a. Stop the channel QMC##.QMC??

__ b. Run the same trace route command. If you like, you can specify a shorter wait time by appending “-w 10” for a 10 second timeout wait.

__ c. What do you observe as the final activity? Does this look like an erroneous outcome?

```
Assuming local qmgr is 31 and remote is 32:
------------------------------------------------------------------------
Activity:
 ApplName: 'dspmqrte                    '

 Operation:
  OperationType: Put
  QMgrName: 'QMC31                                              '
  QName: 'QRMT32                                              '
  ResolvedQName: 'QMC32                                                '
  RemoteQName: 'QL.A                                                '
  RemoteQMgrName: 'QMC32                                                '
------------------------------------------------------------------------
```

Result is not erroneous, just incomplete. You need to draw the conclusion the message is being held up.

__ d. What conclusion can you draw from this outcome?

Since this is a transmission queue, conclude the channel is not processing messages for whatever reason. This is where you need to focus your investigation.

__ 3. Next, observe the trace output of an incorrect remote queue name in the remote queue definition.

__ a. Restart the sender channel.

__ b. Alter the QMC?? remote queue definition to point to a non-existent target queue called XYZZY:

```
alter qr(QMC??) rname(XYZZY)
```

__ c. Perform another trace route.

__ d. What do you observer this time? How similar is this to the output of the known working trace taken in Step 1 above? What is the FeedBack value this time?

```
Activity:
 ApplName: 'amqrmppa                    '

 Operation:
  OperationType: Receive
  QMgrName: 'QMC32                                          '
  RemoteQMgrName: 'QMC31                                    '
  ChannelName: 'QMC31.QMC32         '
  ChannelType: Receiver

 Operation:
  OperationType: Discard
  QMgrName: 'QMC32                                          '
  Feedback: UnknownObjectName
```

This is the same as the working trace but for the final feedback code. UnknownObjectName instead of NotDelivered.

__ e. Where might you expect to find this message now? The dead letter queue if one is defined.

__ 4. Now, observe what happens when a message ends up on the default XMITQ. This can happen if a remote queue definition incorrectly specifies the remote queue manager name (rqmname) in the queue remote definition.

__ a. Change the remote queue definition to use a fictitious remote queue manager named Ralph like this:

```
alter qr(QMC??) rqmname(RALPH)
```


__ b. Perform another trace route

__ c. What happens in the last operation of the last activity?

```
Activity:
 ApplName: 'amqrmppa                    '

 Operation:
  OperationType: Receive
  QMgrName: 'QMC32                                          '
  RemoteQMgrName: 'QMC31                                    '
  ChannelName: 'QMC31.QMC32         '
  ChannelType: Receiver

 Operation:
  OperationType: Put
  QMgrName: 'QMC32                                          '
  QName: 'XYZZY                                           '
  ResolvedQName: 'DEFAULT.XMITQ                               '
  RemoteQName: 'XYZZY                                         '
  RemoteQMgrName: 'RALPH                                      '.
```

__ d. why is this a **Put** not a **Discard** ? The message has not yet reached the destination specified in the resolved qremote defintion. The trace route utility expects more data to be returned from the RALPH queue manager.

Optional exercises

Perform some traces of clustered queues to investigate how load balancing with clustered queues operates.

End of exercise

Exercise review and wrap-up

Having completed this exercise, you should be able to:

- Use the traceroute utility to troubleshoot production problems with messages going astray
- Interpret the output from traceroute in order to make configuration corrections.

Exercise 11.Performing an WebSphere MQ trace

Estimated time

00:30

What this exercise is about

In this exercise, you configure and run a trace of message activity on a particular local queue. Sample programs are used to send and receive message from the queue and then the trace output is located in the filesystem and interpreted. The trace function can also be controlled using MQ Explorer.

What you should be able to do

At the end of this exercise, you should be able to:

- Set up a WebSphere MQ trace
- Analyze the output from a trace

Introduction

The trace facility in WebSphere for MQ provides an ideal method for use in problem determination or as a program development aid. Service personnel may request that a scenario be recreated and traced for solving error or unexpected output situations. The commands used for controlling trace are:

- `strmqtrc` – to start a trace
- `endmqtrc` – to end a trace
- `dspmqtrc` – to format a trace file (UNIX only)

Requirements

- The sample programs amqsput and amqsget are invocable from any position within the file system hierarchy.

Exercise instructions

Step 1: Start an MQ trace

Start an MQ trace. Refer to the following according to the system type:

Windows

From a command prompt issue the following command:

```
strmqtrc -m QMC##
```

or Using MQ Explorer:

__ a. Start the WebSphere MQ Explorer from the Start menu.

__ b. In the Navigator View, right-click the **WebSphere MQ** tree node, and select **Trace...**. The Trace Dialog is displayed.

__ c. Click Start.

UNIX: From a command prompt issue the following command:

```
strmqtrc -m QMC##
```

The strmqtrc command enables tracing. The command has optional parameters that specify the level of tracing. In your exercise you specify the name of the queue manager using the '-m' parameter and QMGR name.

Step 2: Run a scenario to PUT and GET a message to a queue

__ 1. In this step you are going to write some sample messages to the local queue QL.A on queue manager QMC##.

Windows

From a command prompt issue the following command:

```
amqsput QL.A QMC##
```

…write a message

<enter>

UNIX

From a command prompt issue the following command:

```
amqsput QL.A QMC##
```

…write a message

<enter>

__ 2. Now retrieve the messages from the local queue QL.A on queue manager QMC##.

 **Windows**

From a command prompt issue the following command:

```
amqsget QL.A QMC##
```

UNIX

From a command prompt issue the following command:

```
amqsget QL.A QMC##
```

Step 3. Stop the MQ trace

Stop the MQ trace. Refer to the following according to your system type:

 **Windows**

From a command prompt issue the following command:

```
endmqtrc -m QMC##
```

Using MQ Explorer:

__ a. Start the WebSphere MQ Explorer from the Start menu.

__ b. In the Navigator View, right-click the **WebSphere MQ** tree node, and select **Trace...**. The Trace Dialog is displayed.

__ c. Click **Stop**.

UNIX

From a command prompt issue the following command:

```
endmqtrc -m QMC##
```

The endmqtrc command ends tracing. In your exercise you specify the name of the queue manager using the '-m' parameter and QMGR name.

Step 4. Locate the trace files

Windows

Windows: The default trace directory is C:\Program Files\IBM\WebSphere MQ\trace. The trace files are readable without formatting and are called AMQppppp.TRC (where ppppp is the process identifier or pid which created the file).

Navigate to the directory:

```
C:\Program Files\IBM\WebSphere MQ\trace
```

View the trace output and see the PUT and GET operations.

UNIX

Files associated with trace are created in a fixed location in the file tree which is /var/mqm/trace. These files need to be formatted before they can be read.

Navigate to the directory:

```
/var/mqm/trace
```

From a command prompt, format the trace files using the command:

```
dspmqtrc *.TRC
```

The trace formatter program converts binary files named AMQppppp.TRC (where ppppp is the process identifier or pid which created the file) into readable files named AMQppppp.FMT.

Browse the formatted output files and see the PUT and GET operations.

End of exercise

Exercise 12.JMS administration (optional)

Estimated time

00:30-01:00

What this exercise is about

In this exercise, you use the JMSAdmin utility or the JMS administration facilities in the WebSphere MQ Explorer to set up a connection factory and a queue connection factory within a JNDI namespace for use by JMS.

What you should be able to do

At the end of this exercise, you should be able to:

- Use utilities to administer the Java Naming and Directory Interface (JNDI) namespace to configure WebSphere MQ as a JMS provider

Introduction

JMS Administration for the WebSphere MQ administrator involves using the JMSAdmin utility or the WebSphere MQ Explorer and coming to terms with the concept of the JNDI namespace. In this straightforward exercise, the student will create a file-system based JNDI namespace, define a queue connection factory and destination objects. They will then run a JMS Producer sample to place a JMS format message onto a destination and retrieve that message using a sample JMS Consumer application.

Requirements

- WebSphere MQ installed on one of Windows, AIX or Solaris
- It is assumed there exists a queue manager called QMC## where ## is the student's team number. Further, it is assumed there is a local queue defined called QL.A.

- The following WebSphere MQ supplied samples programs are used:
  - JmsJndiProducer

  Usage:

```
JmsJndiProducer -i initialContext -c connectionFactory -d destination
```

  Generates a random “lucky-number” and places a message on the specified destination. Displays JMS RFH2 data.

- JmsJndiConsumer

  Usage:

```
JmsJndiConsumer -i initialContext -c connectionFactory -d destination
```

  Pulls a JMS message of the QCF/Destination specified and displays the RFH2 data and the “lucky-number” generated by JmsJndiProducer.

Exercise instructions

In this exercise, the *home* directory refers to the standard UNIX concept of the home directory $HOME. On Windows, a new folder called C:\JMSContext can be used instead.

__ 1. Create a JMS namespace to hold your JMS administered definitions.

__ a. Create a directory that will be your JNDI FSContext namespace repository. Call it **JMSContext** in your home directory. Refer to the following according to your system type:

Windows

Using Windows Explorer, create a new folder called C:\JMSContext. Now skip to step 3.

UNIX

Create an empty directory called JMSContext in your home directory.

__ 2. Copy the sample JMSAdmin.config and JMSAdmin startup script file to your home directory.

UNIX

Depending on which UNIX system you are running:

- AIX: These files are located under /usr/mqm/java/bin
- Solaris: These files are located under /opt/mqm/java/bin

In both cases, the JMSAdmin startup script is called JMSAdmin.

__ 3. Configure JMS Administration for your file-system JNDI namespace.

Windows

Using the WebSphere MQ Explorer, right-click **JMS Administered Objects** and select **Add Initial Context**. In the dialog that follows, select **File System** and choose your newly created JMSContext directory for the **bindings directory.** Click **Next**, then make sure **Connect Immediately on Finish** is selected. Then **Finish**. Now skip to step 5 below.

UNIX

Ensure your JMSAdmin files are write-enabled: **chmod u+w JMSAdmin***

__ a. Edit JMSAdmin.config in your home directory.

__ b. Note: lines starting with the hash '#' character are comments.

__ c. Ensure only the FSContext Initial Context Factory is uncommented. For example:

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
```

__ d. Set the Provider URL to point to your filesystem JNDI namespace. Substitute <HOME> with your home directory.

```
PROVIDER_URL=file://<HOME>/JMSContext/
```

UNIX

Use "echo $HOME" to discover your home directory.

__ e. Save and close this file.

__ 4. Run JMSAdmin.

__ a. Set the Java environment:

UNIX

Source the following file. Do this by using the "dot" command: . *file-name*

- AIX: /usr/mqm/java/bin/setjmsenv64
- Solaris: /opt/mqm/java/bin/setjmsenv

__ b. Edit the JMSAdmin script (if needed – see below).

UNIX

AIX: Add the full path to the java command: /usr/java14_64/bin/java

Solaris: not necessary

__ c. Save and close.

__ d. Run JMSAdmin.

UNIX

./JMSAdmin

__ e. Ensure the JMSAdmin prompt “InitCtx>“ appears.

__ 5. Define a Queue Connection Factory.

UNIX

From the “InitCtx>” prompt define a Queue Connection Factory administered object for your queue manager QMC##
Name it **QCF_QMC##**

Type **define qcf(QCF_QMC##) qm(QMC##)**

Windows

Using WebSphere MQ Explorer right-click **Connection Factories -> New -> Connection Factory …**
Call your new connection factory **QCF_QMC##**

Set the type as **Queue Connection Factory**.

Transport **Bindings**.

When the properties screen appears, choose **Connection** on the left. Then in **Base Queue Manager** specify your queue manager name **QMC##**.

Finish.

__ 6. Define a destination administered object for QL.A.

UNIX

From the “InitCtx>” prompt define a JMS Queue administered object for your queue.Queue is a subtype of Destination.
Name it **DEST_QL.A**.

Type **define q(DEST_QL.A) qu(QL.A)**.

Windows

Using WebSphere MQ Explorer right-click **Destinations -> New -> Destination …**
Call you new destination **DEST_QL.A**.

Set the type as **Queue**.

On the **change properties** dialog, set the **Queue** to QL.A.

Finish.

__ 7. Run the sample JMS/JNDI Consumer sample.

__ a. Locate the JMS samples:

UNIX

AIX: /usr/mqm/samp/jms/samples

Solaris: /opt/mqm/samp/jms/samples

Windows

You will need to run these from a DOS prompt.

C:\Program Files\IBM\WebSphere MQ\tools\jms\samples

__ b. Run the JmsJndiProducer sample with the following arguments:
-i file://<your-jndi-directory> -c <your-QCF> -d <your-destination>

UNIX

AIX: Java is located under /usr/java14_64/bin

Solaris: Default Java is ok. Set this environment variable before you run the java samples:
export LD_LIBRARY_PATH=/opt/mqm/java/lib

Windows

Java is located under “C:\Program Files\IBM\WebSphere MQ\Java\jre\bin\java”. This should already be in the path.

Make sure Java is found before proceeding.

Note also that you need to specify “.class”: JmsJndiProducer.**class** in the above command.

Here is a sample run:

```
java JmsJndiProducer -i file://C:/WMQ/JMSContext -c QCF_AJGMQ1 -d
DEST_Q1
Initial context found!
 Sent message:
JMSMessage class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:414d5120414a474d513120202020202065df824820006402
JMSTimestamp: 1216595003250
JMSCorrelationID: null
JMSDestination: queue:///Q1?targetClient=1
JMSReplyTo: null
JMSRedelivered: false
JMSXAppID: ereAdapters\jre\bin\java.exe
JMSXDeliveryCount: 0
JMSXUserID: andrewg
JMS_IBM_PutApplType: 11
JMS_IBM_PutDate: 20080720
JMS_IBM_PutTime: 23032325
JmsJndiProducer: Your lucky number today is 219
  SUCCESS
```

__ 8. Run the sample JMS/JNDI Consumer sample

__ a. Run the JmsJndiConsumer sample with the following arguments:
-i file://<your-jndi-directory> -c <your-QCF> -d <your-destination>

__ b. Verify that your JMS Consumer sample received the lucky number generated by the JMS producer.

End of exercise

Exercise review and wrap-up

At the end of this exercise, you should be able to:

- Use utilities to administer the JNDI namespace to configure MQ as a JMS provider