

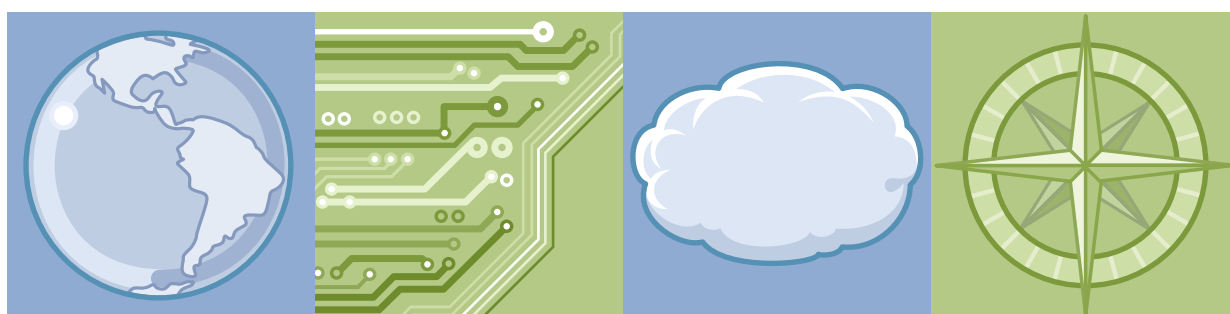


# IBM Training

Student Exercises

## IBM MQ V8 Application Development (Linux Labs)

Course code WM508 ERC 1.0



IBM Cloud  
Middleware

## Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

AIX®	Bluemix™	CICS®
developerWorks®	IMS™	MVS™
Notes®	PartnerWorld®	Passport Advantage®
Redbooks®	Tivoli®	WebSphere®
z/OS®		

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

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

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

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

VMware and the VMware “boxes” logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the “Marks”) of VMware, Inc. in the United States and/or other jurisdictions.

Other product and service names might be trademarks of IBM or other companies.

### June 2016 edition

The information contained in this document has not been submitted to any formal IBM test and is distributed on an “as is” basis without any warranty either express or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will result elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

© Copyright International Business Machines Corporation 2016.

This document may not be reproduced in whole or in part without the prior written permission of IBM.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

Trademarks .....	vii
Exercises description .....	ix
<b>Exercise 1. Working with IBM MQ to find your message .....</b>	<b>1-1</b>
Section 1: Determine the status of queue managers in your server .....	1-6
Section 2: Ensure that both queue managers have a stated dead-letter queue, and channel authentication is set to disabled .....	1-7
Section 3: Review the queue and channel definitions for MQ01 .....	1-8
Section 4: Define queues and channel objects in queue manager MQ01 .....	1-9
Section 5: Examine the objects that you defined in queue manager MQ01 .....	1-11
Section 6: Put and get messages from the queues .....	1-12
Section 7: Get the message put to the alias queue .....	1-14
Section 8: Define queues and channel objects for queue manager MQ03 .....	1-15
Section 9: Anticipate the places where your message might be found .....	1-17
Section 10: Put messages to remote queue TO.MQ03 in queue manager MQ01 .....	1-17
Section 11: Start channel MQ01.MQ03 on queue manager MQ01 .....	1-18
Section 12: Put messages to remote queue TO.MISSING.Q .....	1-19
Section 13: Check whether local queue NOT.THERE exists in MQ03 .....	1-20
Section 14: Check the IBM MQ error logs .....	1-21
Section 15: Alter alias queue FREQ.CHANGED.Q to point to queue TO.MQ03 .....	1-27
<b>Exercise 2. Getting started with IBM MQ development .....</b>	<b>2-1</b>
Section 1: Compile and run a C program .....	2-5
Section 2: Define and review queues EX2PERS, EX2NONPERS, and alias queue TO.PERS .....	2-7
Section 3: Examine the putmsg.c and cmqc.h header code .....	2-9
Section 4: Put a message to non-persistent alias queue TO.PERS in queue manager MQ01 .....	2-10
Section 5: Change program twoqput.c to open and put the same message to a second queue. ....	2-12
Section 6: Add the MQOPEN for the second queue to the program .....	2-15
Section 7: Add the MQPUT for the second queue .....	2-17
Section 8: Add the MQCLOSE for the second queue .....	2-18
Section 9: Test the updated twoqput application with the MQMD Persistence attribute set to MQPER_PERSISTENCE_AS_Q_DEF .....	2-19
Section 10: Test the updated twoqput application with the MQMD Persistence attribute set to MQPER_PERSISTENT .....	2-21
Section 11: Does the MQMD Persistence attribute set to MQPER_PERSISTENT supersede the alias queue persistence? .....	2-22
Section 12: Resume the test with the MQMD Persistence attribute set to MQPER_PERSISTENT .....	2-23
Section 13: Test the updated twoqput application with the MQMD Persistence attribute set to MQPER_NOT_PERSISTENT .....	2-23
Section 14: Review the connection authentication initial value in the cmqc.h header ...	2-25
Section 15: Display the queue manager connection authentication settings .....	2-27
Section 16: Test the authentication code with the MQSAMP_USER_ID variable .....	2-28
Section 17: Repeat the test but do not provide a valid password .....	2-29

Section 18: If the MQCNO version number is set to 1, what happens? . . . . . 2-29

**Exercise 3. Working with MQOPEN and queue name resolution, MQPUT, and MQMD fields** . . . . . **3-1**

Section 1: Review the MQCONNX and MQOPEN queue manager name use in qnmres.c 3-6

Section 2: Check or start queue managers MQ01 and MQ03, and channel MQ01.MQ03 3-7

Section 3: Test the qnmres program with the MQOPEN MQOD queue name set to EX3.SAME.QNAME and the MQOPEN MQOD queue manager name set to blanks . . . . . 3-8

Section 4: Alter program qnmres to set the MQOD queue manager name to MQ03 . . . . 3-9

Section 5: Test the qnmres program with the MQOPEN MQOD queue name set to EX3.SAME.QNAME, and the MQOPEN MQOD queue manager name set to MQ03 . . . . 3-9

Section 6: Review the crt dynq.c program source . . . . . 3-11

Section 7: Run the first test of the program by using the option to allow the queue manager to generate the full dynamic queue name . . . . . 3-12

Section 8: Change and test program crt dynq to create a dynamic queue name with a specified prefix . . . . . 3-13

Section 9: Alter crt dynq and test to use a user-generated queue name . . . . . 3-16

Section 10: Update program exprep.c (copied from putmsg.c) to set expiry and request a COA and an expiry report. . . . . 3-19

Section 11: Review, set up, and test . . . . . 3-21

Section 12: Put messages in queue EX3.CAPX.REQ and observe the expiration behavior when the lowest expiry value is set on the queue definition . . . . . 3-22

Section 13: Put messages in queue EX3.REQUEST and observe the expiration behavior when an expiry value is not set on the queue definition . . . . . 3-24

Section 14: Review the report messages by using the amqsbcg formatted browse sample . . . . . 3-25

Section 15: Review the report messages by using IBM MQ Explorer . . . . . 3-26

**Exercise 4. Correlating requests to replies** . . . . . **4-1**

Section 1: Back up application ex4req.c . . . . . 4-5

Section 2: Review application ex4req.c . . . . . 4-5

Section 3: Type the code necessary to create a report message and preserve the original message MsgId and CorrelId in the report message . . . . . 4-6

Section 4: Compile and test the ex4req application . . . . . 4-7

Section 5: Confirm that the MsgId of the request and report messages match . . . . . 4-8

Section 6: Backup application ex4repl.c . . . . . 4-9

Section 7: Put a few messages to queue EX4REPL . . . . . 4-10

Section 8: Review the base ex4repl.c source . . . . . 4-10

Section 9: Make necessary additions to ex4repl.c . . . . . 4-11

Section 10: Baseline what messages you have now . . . . . 4-13

Section 11: Compile and test the ex4repl application . . . . . 4-14

Section 12: Check the MsgId and CorrelId values . . . . . 4-15

Section 13: Review the ex4match.c code sections already implemented . . . . . 4-18

Section 14: Set up the queue object descriptor names, and queue manager names . . . 4-19

Section 15: Set up the MQGET for queue EX4REPL to get messages by the correlation identifier . . . . . 4-20

Section 16: Compile and test the ex4match application . . . . . 4-20

**Exercise 5. Commit and backout review** . . . . . **5-1**

Section 1: Locate the sample commit and backout application . . . . . 5-2

Section 2: Observe the manual tracking of resources outside IBM MQ . . . . . 5-3

Section 3: Review the MQCMIT and MQBACK function calls . . . . .	5-4
<b>Exercise 6. Asynchronous messaging review . . . . .</b>	<b>6-1</b>
Section 1: Check the callback function . . . . .	6-3
Section 2: Check the asynchronous messaging-related declarations . . . . .	6-4
Section 3: Review the standard MQCONNX and MQOPEN calls . . . . .	6-4
Section 4: Find and review the callback code . . . . .	6-4
Section 5: Find and review the MQCTL invocation to start consumption of messages . . .	6-5
Section 6: Review how to end message consumption . . . . .	6-6
Section 7: Check for the standard IBM MQ ending function calls . . . . .	6-6
Section 8: Start the asynchronous callback sample amqscbf . . . . .	6-7
<b>Exercise 7. Working with an IBM MQ client . . . . .</b>	<b>7-1</b>
Section 1: Compile program putmsg.c to run as an IBM MQ client application . . . . .	7-5
Section 2: Check your client channel definition table (CCDT) CLNTCONN type channel .	7-6
Section 3: Start the test of the putmsgc application by using the CCDT definition . . . . .	7-8
Section 4: Check the channel status from queue manager MQ01 . . . . .	7-8
Section 5: Resume the putmsgc test . . . . .	7-9
Section 6: Use a sample application to remove the message from the MQ01 EX7.LOCALQ queue . . . . .	7-10
Section 7: Set up your client to connect to queue manager MQ03 by using the MQSERVER environment variable . . . . .	7-10
Section 8: Test the putmsgc application by using the MQSERVER environment variable . . . . .	7-12
Section 9: Modify an existing IBM MQ application to use the MQCONNX call to set up connectivity to queue manager MQ01 . . . . .	7-13
Section 10: Code change 1: Add extra header file that is required for MQCD structure .	7-14
Section 11: Code change 2: Declare the MQCD structure . . . . .	7-14
Section 12: Code change 3: Define hardcoded variables . . . . .	7-16
Section 13: Code change 4: Set the MQCD fields, point the MQCNO to the MQCD structure, and set the queue manager name for the MQCONNX call . . . . .	7-16
Section 14: Save changes, and compile your code . . . . .	7-18
Section 15: Test the codeconn application . . . . .	7-19
Section 16: Confirm that the message put with the codeconn application was put to queue manager MQ01 . . . . .	7-20
Section 17: Use putmsgc to put another message to queue EX7.LOCALQ . . . . .	7-21
<b>Exercise 8. Working with publish/subscribe basics . . . . .</b>	<b>8-1</b>
Section 1: Back up the source code for both applications in this exercise . . . . .	8-2
Section 2: Review and change the ex8sub.c application . . . . .	8-3
Section 3: Compile and do a preliminary test of ex8sub.c . . . . .	8-4
Section 4: Review and make necessary changes to the ex8pub.c copy of putmsg.c . . . .	8-5
Section 5: Compile and test the publisher . . . . .	8-7
Section 6: Use the runmqsc utility to check information about the managed storage that the subscription MQSUB creates . . . . .	8-10
Section 7: Use IBM MQ Explorer to check your subscription. . . . .	8-10
<b>Exercise 9. Connecting IBM MQ Light applications to IBM MQ applications . . . . .</b>	<b>9-1</b>
Section 1: Start queue manager MQ15 . . . . .	9-6
Section 2: Confirm that the AMQP capability is enabled in the queue manager . . . . .	9-6
Section 3: Start and check the AMQP service and channel MQ15.AMQP . . . . .	9-8

Section 4: Set up your windows session to work with the IBM MQ Light clients . . . . . 9-12

Section 5: Use the recv.js IBM MQ Light sample to subscribe to the default topic . . . . . 9-13

Section 6: Open a second terminal window and check the subscription . . . . . 9-14

Section 7: Use the second terminal window to publish the send.js IBM MQ Light client sample default message . . . . . 9-15

Section 8: Set up to exchange messages between IBM MQ applications and IBM MQ Light applications . . . . . 9-16

Section 9: Create an IBM MQ Light subscription to topic “Sports/soccer” . . . . . 9-17

Section 10: Use IBM MQ to publish to the IBM MQ Light “Sports/soccer” subscription . . 9-18

Section 11: Publish messages to both IBM MQ applications and IBM MQ Light subscribers at the same time . . . . . 9-19

Section 12: Use the IBM MQ Light sample client recv.js to receive messages that a queue-based application puts . . . . . 9-21

Section 13: Use a queue-based IBM MQ application to receive messages that the IBM MQ Light send.js client publishes . . . . . 9-23

# 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, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

AIX®	Bluemix™	CICS®
developerWorks®	IMS™	MVS™
Notes®	PartnerWorld®	Passport Advantage®
Redbooks®	Tivoli®	WebSphere®
z/OS®		

Intel is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

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

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

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

VMware and the VMware “boxes” logo and design, Virtual SMP and VMotion are registered trademarks or trademarks (the “Marks”) of VMware, Inc. in the United States and/or other jurisdictions.

Other product and service names might be trademarks of IBM or other companies.





# Exercises description

The exercises in this course play a major role in reinforcing the lecture material. This narrative expands on the stated exercise objectives:

- **Exercise 1: Work with IBM MQ to find your message**

In this exercise you learn how to:

- Work with different types of queues.
- Follow the trajectory of a message that goes to a remote queue manager.
- Determine “where is my message” when you test in an IBM MQ environment.
- Find the IBM MQ queue manager error logs.
- Become familiar with the dead-letter queue. Learn how to find the reason that a message was placed there.

- **Exercise 2: Getting started with IBM MQ development**

This exercise provides experience with critical items, such as:

- Learn to compile IBM MQ programs
- Learn how to initialize IBM MQ structures
- Understand how changes that are made in development override attributes defined in IBM MQ objects
- Determine the behavior of persistence in different scenarios
- Observe what happens when you overlook to set the version number of an IBM MQ structure
- Learn about connection authentication configuration in the queue manager
- Learn how to include connection authentication in your code

- **Exercise 3: Working with MQOPEN and queue name resolution, MQPUT, and MQMD fields**

This exercise helps you:

- Understand how the code of different fields in the MQOPEN influences queue name resolution
- Code report options
- Create dynamic queues
- Work with message expiry

- **Exercise 4: Correlating requests to replies**

This exercise shows how to:

- Create a request and save details to retrieve the reply later
- Reply to a request and preserve information that is needed to match the reply
- Select a specific message from a queue

- **Exercise 5: Commit and backout review**

This exercise provides a view of transaction code, and shows you:

- Which resources are controlled with IBM MQ function calls
- Which resources must be manually tracked with your code

- **Exercise 6: Asynchronous messaging review**

This exercise:

- Walks through an asynchronous message application and shows how the IBM MQ function calls interact
- Includes a test of the asynchronous application to demonstrate how the code works

- **Exercise 7: Working with an IBM MQ client**

This exercise teaches you:

- How to compile an application with the IBM MQ client libraries
- The different ways your client can be configured to connect to a queue manager
- Which connection method takes precedence
- How to create the connection to the queue manager in your code

- **Exercise 8: Working with publish/subscribe basics**

In this exercise you work with the MQI for integrated publish/subscribe to:

- Learn to use the MQSUB call to create a subscriber application
- Convert a point-to-point application to publish messages

- **Exercise 9: Exchanging messages between IBM MQ applications and IBM MQ Light applications**

While Exercise 9 does not have dependencies on previous labs, it is advisable to complete Exercise 8 first. Exercise 8 establishes the publish/subscribe background helpful for Exercise 9.

In this exercise, you work IBM MQ Light applications and IBM MQ applications.

- Review how a V8.0.0.4 queue manager is configured to enable AMQP
- Exchange messages between IBM MQ integrated publish/subscribe applications and IBM MQ Light applications

- Exchange messages between IBM MQ queued applications and IBM MQ Light applications

In the exercise instructions, you can check off the line before each step as you complete it to track your progress.

## Exercise solutions

In this course, you alter IBM MQ application code as required in most of the exercises.

The `LabFiles` directory structure shows one source and one solution directory for each lab (except for units that use IBM MQ samples for source). The presence of solution source is documented in the **Exercise Requirements** section of each lab.

You find source in the solution directory for:

- One solution file for Exercise 2
- One solution file for Exercise 7
- Two solution files for Exercise 8

The remaining solution directories are expected to be empty.



# Exercise 1. Working with IBM MQ to find your message

## What this exercise is about

This exercise explains how to test your application by showing you where and how to look for messages that you need to put or get with your code. It also provides basic familiarity with the technology for which you are developing code. In addition, it shows you how to define various types of queues, which, if you are given the authority to define in a test system, reduces your dependency on infrastructure services.

## What you should be able to do

After completing this exercise, you should be able to:

- Determine the status of queue managers in a server
- Start a queue manager
- Use the runmqsc utility and command scripts to create IBM MQ objects and check results
- Put messages to local and alias queues and determine whether the messages arrived at the intended destination
- Determine the trajectory and possible stops of a message put to a remote queue
- Start a sender channel and check the channel status
- Check the queue manager error logs
- Determine where your message is
- Examine the dead-letter queue and identify the reason that a message was placed in the queue

## Introduction

The purpose of the exercise is to teach you to use IBM MQ to work with messages for the applications you develop. It gives you experience by using IBM MQ to exchange messages between two queue managers, and helps you answer the question “where is my message”.

## Requirements

Queue manager MQ01, and IBM MQ object definition scripts  
ex1.MQ01.mqsc and ex1.MQ03.mqsc at `<dir>/LabFiles/Lab1/source`.

## Platform-dependent notes for all exercises

In this exercise guide, you work with either a Linux or Windows VMware image.



### Stop

As you log on and work with the VMware image, ***disregard and ignore any operating system or application request to either register or update the software.***

When you log into the VMWare image, use ID **root** and password **web1sphere**.

Throughout the guide, it includes separate instructions for Linux and Windows where needed. It also includes ***frequently occurring tasks***, which are provided one time in this section. In the sections of this course, the tasks are mentioned specifically, for example, “Edit file”. ***Refer to this section for operating-system-specific instructions for these repeating tasks.***

In all exercises, unless otherwise stated, press the Enter key at the end of each command.



### Linux

#### **Frequently occurring Linux tasks:**

##### ***Open a terminal command window***

To open a terminal command window and set it to the correct user and path, follow these instructions, ***and*** the instructions in section ***Change to the mqadmin8 user.***

1. The terminal icon is the second icon to the right of the “Red Hat” icon.
2. Click the terminal icon and wait until the command terminal opens. The terminal comes up, and the command prompt is prefixed with [root@rhel6base ~].
3. ***You must complete the next task, “Change to the mqadmin8 user”, next in this Linux text box.***

##### ***Change to the mqadmin8 user***

When you open a new terminal command window in Linux, since you are logged on as the root user, the window opens under the root ID. The root ID is not authorized to run certain IBM MQ tasks, and does not have the path set to use all the IBM MQ sample programs and utilities. ***DO NOT SET ANY PATHS MANUALLY; the script takes care of it.***

You must change to the `mqadmin8` user to have the path set correctly, and have authorization to work with the queue managers.

4. Type `su - mqadmin8` and press the Enter key.
5. The profile for the IBM MQ user runs. Results are:  

```
PATH set at
/usr/lib64/qt-3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/s
bin:/sbin:/home/mqadmin8/bin:/opt/mqm/samp/bin:/opt/node/node-v0.12.9-lin
ux-x64/bin
```

You are in your home directory at `/home/mqadmin8`

6. The directory prefix should now be `mqadmin8@rhel6base ~]`.

Directory `LabFiles` can be found in the `mqadmin8` user home directory.

### List the directory

- Type `ls -l` and press the Enter key. The letters in the command are lowercase *alpha-L*, not the number 1.
- Sometimes the list of files is too large. If you know part of the name of the file you need, you can use double commands such as:

```
ls -l /opt/mqm/inc | grep amq
```

The expanded command limits the output to those file names that contain string `amq`.

### Determine your current directory

- To determine your current directory in a terminal command window, type `pwd` and press the Enter key. The response is your current directory.

### View or edit a file:

You have different options to view or edit a file, and these options depend on your comfort level with each tool.

- You can use `gedit` to view and edit most text files such as the IBM MQ logs, or the `mqsc` definition files. To use `gedit`, look for the icon that shows a sheet or paper and a pencil that is located directly to the left of the Firefox icon.
- To view files by using the `vi` editor, use the browse-only version if you need to prevent making inadvertent changes. Type `view filename` from the directory where the file is located.
- If you are proficient with the `vi` editor, you can opt to use `vi` to edit files.  
Type `vi filename` from the directory where the file is located.

### Change directories to the queue manager error files

- From the terminal command prompt, type:

```
cd /var/mqm/qmgrs/<qmname>/errors
```

where you replace `<qmname>` with the queue manager name.

### Make a backup copy of the source (before you change anything)

- Ensure that you are in the `/home/mqadmin8/LabFiles/Lab#/source` directory, where you replace the `#` with the exercise number you are working with. For example, if you are working on Exercise 1, then it is: `/home/mqadmin8/LabFiles/Lab1/source`
- Type `cp sourcename.c sourcename.c.bkp` and press the Enter key, where `sourcename` is the name of the program.

### Compile code

- In the course environment, the code is in directory `/home/mqadmin8/LabFiles/Lab#/source` where `#` is the Exercise number.

- The course environment has the path set for the IBM MQ libraries. To compile programs, move to the code directory and type:

```
gcc -I/opt/mqm/inc -lmqm -o executable_name program_name.c
```

where *program\_name.c* is the source, and *executable\_name* is the name of the compiled executable program.



## Windows

---

### Frequently occurring Windows tasks include the following tasks:

- Open a terminal command window:

You can open a terminal command window by either:

- Clicking the command prompt icon on your Windows taskbar
- Using **Start > All Programs > Accessories > Command Prompt**

- View or edit a file:

notepad++ is the suggested editor, although other options are available. notepad++ shows line numbers, which are referenced in some of the exercises. It also formats the files.

**Warning:** If you get any requests to upgrade notepad++, ignore them.

- Change directories to the queue manager error files.

- From the terminal command prompt, type:

```
cd C:\ProgramData\IBM\MQ\qmgrs\<qmname>\errors where you replace  
<qmname> with the queue manager name.
```

- Use Windows Explorer to find directory

```
C:\ProgramData\IBM\MQ\qmgrs\<qmname>\errors where you replace <qmname>  
with the queue manager name whose logs you need to view.
```

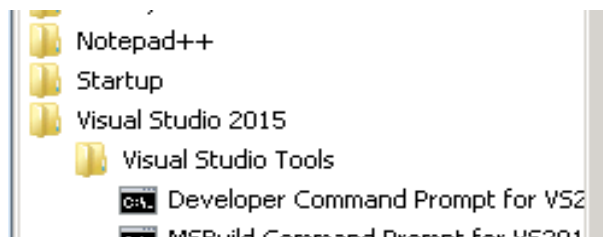
- Make a backup copy of the source (before you change anything).

- Ensure that you are in the *C:\LabFiles\Lab#\source* directory, where you replace the # with the exercise number you are working with. For example, if you are working Exercise 1, then you should be at: *C:\LabFiles\Lab1\source*
- Type *copy sourcename.c sourcename.c.bkp* and press the Enter key, where *sourcename* is the name of the program.

- Compile code.



- Open a visual studio command prompt by going to **Start > All Programs > Visual Studio 2015 > Visual Studio Tools > Developer Command Prompt for VS2015.**



- Change to the source directory for the exercise you are working with.
- File `copy_edit.txt` at `C:\LabFiles\Lab#\source` can be used to copy and paste the compile command. You need to replace the `program_name.c` placeholder with the name of your C source.
- Type the command as shown, or change and use the `copy_edit.txt` file to paste the compile command:

```
cl program_name.c C:\PROGRA~1\IBM\WEBSPH~1\Tools\Lib\mqm.lib
```

where `program_name.c` is the name of the program source.

## Exercise 1 instructions

### Section 1: Determine the status of queue managers in your server



#### Important

The first part of this exercise is intended to provide the student experience by using the IBM MQ script processor, `runmqsc`, and work with command-line utilities.

Do not use IBM MQ Explorer until requested.

1. Open a terminal command session as indicated by the instructions for your operating system.
2. Type the command to examine the queue managers in the server:

```
dspmqr
```

The expected results are:

```
QMNAME(MQ15) STATUS(Ended normally)
```

```
QMNAME(MQ01) STATUS(Ended normally)
```

```
QMNAME(MQ03) STATUS(Ended normally)
```

3. Start queue manager `MQ01` and queue manager `MQ03` by typing the commands that are shown. **Do not start MQ15.**

```
strmqm MQ01
```

```
strmqm MQ03
```

Expected results:

Start messages that are similar to the `MQ01` results in the display should appear for each queue manager.

```
The queue manager is associated with installation 'Installation1'.
```

```
5 log records accessed on queue manager 'MQ01' during the log replay phase.
```

```
Log replay for queue manager 'MQ01' complete.
```

```
Transaction manager state recovered for queue manager 'MQ01'.
```

```
WebSphere MQ queue manager 'MQ01' started using V8.0.0.4.
```

- \_\_ 4. Check the messages that are displayed for each queue manager and ensure that the queue managers started without errors. Resolve any problems or request assistance as needed before you proceed to the next step.

## Section 2: Ensure that both queue managers have a stated dead-letter queue, and channel authentication is set to disabled



### Note

#### Dead-letter queue:

When a queue manager is created, the dead-letter queue needs to be specified. Course queue managers MQ01 and MQ03 should have the dead-letter queue declared by using the `SYSTEM.DEAD.LETTER.QUEUE`. However, if the dead-letter queue is not declared, other steps in this exercise do not work correctly.

If the dead-letter queue is not set for either queue manager MQ01 or MQ03, alter the queue manager object and set it as shown in the steps that follow.

#### Channel authentication:

Channel authentication (CHLAUTH) is a powerful feature in IBM MQ. In the IBM MQ administration courses, *it is advisable to keep CHLAUTH enabled and rules set*. Setting CHLAUTH rules is an administrative topic. For purposes of this course, the time is dedicated to development topics, and channel authentication is set to `disabled`. However, disabling CHLAUTH **should not be construed** as a good practice.

Failure to disable CHLAUTH causes **blocked channel** errors in some exercises, particularly the work with IBM MQ Clients. **Ensure that channel authentication is disabled as shown in the steps that follow.**

- \_\_ 5. Type `runmqsc MQ01` and press the Enter key.
- \_\_ 6. Display the dead-letter queue configuration for MQ01 by typing `dis qmgr deadq` and pressing the Enter key. The expected results are:

```
dis qmgr deadq chlauth
  1 : dis qmgr deadq chlauth
AMQ8408: Display Queue Manager details.
  QMNAME(MQ01)                CHLAUTH(DISABLED)
  DEADQ(SYSTEM.DEAD.LETTER.QUEUE)
```

- \_\_ 7. If the `DEADQ` attribute of the queue manager is set as shown, proceed to the next step. If the `DEADQ` attribute of the queue manager is blank, declare the dead-letter queue by typing `alter qmgr deadq(SYSTEM.DEAD.LETTER.QUEUE)` and pressing the Enter key.
- \_\_ 8. If the `CHLAUTH` attribute of the queue manager is set as shown, proceed to the next step. If the `CHLAUTH` attribute of the queue manager is set to `ENABLED`, disable `CHLAUTH` by typing `alter qmgr CHLAUTH(DISABLED)` and pressing the Enter key.
- \_\_ 9. Exit `runmqsc` for queue manager MQ01 by typing `end` and pressing the Enter key.

- \_\_\_ 10. **Repeat the process** in this section for **both** the `DEADQ` and `CHLAUTH` queue manager attributes by following the same steps for queue manager `MQ03`.
- \_\_\_ 11. Before you continue, ensure that both queue managers have:
  - A declared dead-letter queue, and
  - Channel authentication disabled

### **Section 3: Review the queue and channel definitions for MQ01**

- \_\_\_ 12. Move to the `LabFiles` directory by using the instructions for your operating system, or by using the `LabFiles` directory shortcut towards the lower left of your VMware display.
- \_\_\_ 13. Move to the `Lab1` directory, then to the `source` subdirectory. You **should be in the** `<dir>/LabFiles/Lab1/source` directory.
- \_\_\_ 14. List the directory by using the instructions appropriate for your operating system.
- \_\_\_ 15. Ensure that the command script file `ex1.MQ01.mqsc` is in the directory.
- \_\_\_ 16. Browse file `ex1.MQ01.mqsc` by using the editor applicable to your operating system. **Review the objects that you are about to define.** The commands in the file should look as displayed:

```

*****
*** MQSC definitions for queue manager MQ01 - exercise 1          ***
*****
* Local queue
DEF QLOCAL(EX1.LOCALQ)
*
* One more local queue to use as target for QALIAS
DEF QLOCAL(UNDER.COVER.QUEUE)
*
* Alias queue
DEF QALIAS(FREQ.CHANGED.Q) TARGETTYPE(Queue) TARGET(UNDER.COVER.Queue)
*
* Local queue of usage type transmit for channel to MQ03 - not triggered
DEF QLOCAL(MQ03) USAGE(XMITQ)
*
* Remote queue to MQ03 - uses XMITQ and name of MQ03 queue
DEF QREMOTE(To.MQ03) XMITQ(MQ03) RQMNAME(MQ03) RNAME(FROM.MQ01)
*
* One more remote queue. The queue in the RNAME attribute
* is missing from MQ03
DEF QREMOTE(To.MISSING.Q) XMITQ(MQ03) RQMNAME(MQ03) RNAME(NOT.THERE)
*
* Sender channel to MQ03
DEF CHL(MQ01.MQ03) CHLTYPE(SDR) trptype(TCP) XMITQ(MQ03) +
  CONNAME('localhost(1623)')

```

- \_\_ 17. Close script file `ex1.MQ01.mqsc`. **Ensure that you are still in the**  
**<dir>/LabFiles/Lab1/source directory.**

#### **Section 4: Define queues and channel objects in queue manager MQ01**

- \_\_ 18. You start the `runmqsc` utility by using a script file. If you have any errors, you must be sure to capture the results.



#### **Important**

**Be careful that you use the correct queue manager name.**

Type the command exactly as shown, and press the Enter key:

```
runmqsc MQ01 < ex1.MQ01.mqsc > ex1.MQ01.txt
```

- \_\_\_ 19. Check the results by editing file `ex1.MQ01.txt`. **You must see** messages “No commands have a syntax error” and “All valid MQSC commands were processed” at the end of the results:

```

* : *****
  : * Local queue
  1 : DEF QLOCAL(EX1.LOCALQ)
AMQ8006: WebSphere MQ queue created.
  : *
  : * One more local queue to use as target for QALIAS
  2 : DEF QLOCAL(UNDER.COVER.QUEUE)
AMQ8006: WebSphere MQ queue created.
  : *
  : * Alias queue
  3 : DEF QALIAS(FREQ.CHANGED.Q) TARGTYPE(QUEUE) TARGET(UNDER.COVER.QUEUE)
AMQ8006: WebSphere MQ queue created.
  : *
  : * Local queue of usage type transmit for channel to MQ03 - not triggered
  4 : DEF QLOCAL(MQ03) USAGE(XMITQ)
AMQ8006: WebSphere MQ queue created.
  : *
  : * Remote queue to MQ03 - uses XMITQ and name of MQ03 queue
  5 : DEF QREMOTE(TO.MQ03) XMITQ(MQ03) RQMNAME(MQ03) RNAME(FROM.MQ01)
AMQ8006: WebSphere MQ queue created.
  : *
  : * One more remote queue. The queue in the RNAME attribute
  : * is missing from MQ03
  6 : DEF QREMOTE(TO.MISSING.Q) XMITQ(MQ03) RQMNAME(MQ03) RNAME(NOT.THERE)
AMQ8006: WebSphere MQ queue created.: *
  : * Sender channel to MQ03
  7 : DEF CHL(MQ01.MQ03) CHLTYPE(SDR) trptype(TCP) XMITQ(MQ03) +
  :   CONNAME('localhost(1623)')
AMQ8014: WebSphere MQ channel created.
  : *

7 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.

```

- \_\_\_ 20. The script files that are provided should not contain any errors. If errors are found, make any needed corrections before proceeding.

**Stop**

**If your definitions completed as expected, do not run these steps.** These alphabetic-numbered steps are for use only if you had problems with the definitions. **If your script ran as expected, proceed to the next section.**

- \_\_ a. Open a `runmqsc` session by typing `runmqsc MQ01` and pressing the Enter key. You are placed in the interactive IBM MQ command processor.
- \_\_ b. You define one object at a time, but you need to type definitions for only those objects that failed to be defined.
- \_\_ c. Type the name of the object that failed to be defined, omitting the number shown on the left of the display. For example, if a problem is found with `UNDER.COVER.QUEUE`, type `DEF QLOCAL(UNDER.COVER.QUEUE)` and **omit** the “2 :” that is shown on the display.

`runmqsc` should provide positive feedback on each step, such as:

```
AMQ8006: WebSphere MQ queue created.
```

- \_\_ d. However, if you need to reenter the channel definition, you can continue typing so it wraps around to a new line if needed. It ensures that you type the channel `CONNNAME` attribute exactly as shown **in this step**, including the quotation marks where placed:

```
CONNNAME('localhost(1621)')
```

- \_\_ e. To exit `runmqsc`, type `end` and press the Enter key.

## **Section 5: Examine the objects that you defined in queue manager MQ01**

For convenience, you might refer to the definitions results shown in the previous display. Answers are at the end of this exercise under heading **First group of questions**.

- \_\_ 21. How many local queues (QLOCAL) did you define? \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_
- \_\_ 22. What is the difference between local queue MQ03 and the other local queues?  
\_\_\_\_\_
- \_\_ 23. If you place a message in the alias queue `FREQ.CHANGED.Q`, where can you expect to find the message?  
\_\_\_\_\_
- \_\_ 24. The remote queues and the sender channel point to a transmission queue (XMITQ). What local queue is used as the transmission queue? \_\_\_\_\_

\_\_\_ 25. In a remote queue definition, the RNAME attribute points to the name of a local queue in the remote queue manager. Two remote queues (QREMOTE) are defined. What are the names of the two local queues that are expected to be in the remote queue manager?

\_\_\_\_\_

\_\_\_\_\_

\_\_\_ 26. When a channel is running, messages put to a remote queue are expected to flow to the remote queue manager. If the channel is not running, messages put to a remote queue wait in the transmission queue until the channel starts. If messages are put to both remote queues TO.MQ03 and TO.MISSING.Q and the channel is stopped, where do you expect to find the messages that you put on the remote queues? (This question arises since remote queues do not hold messages.)

\_\_\_\_\_

## Section 6: Put and get messages from the queues



### Information

In this exercise, you use sample program `amqspout` to put messages to queues. For the labs, `amqspout` takes two parameters:

- The queue name, and
- The queue manager name

After the program initializes, it confirms that it is waiting for messages to be typed by returning:

```
Sample AMQSPUT0 start
target queue is queue_name
```

When you see this message, `amqspout` is waiting for you to type in some messages.

You type one line and press the Enter key to send one message.

To stop sending messages, you send a blank message by pressing the Enter key twice after the last message you need to send.

**Remember the instructions in this text box. You use sample programs that work in a similar way many times throughout the course.**

\_\_\_ 27. Ensure that queue `EX1.LOCALQ` is empty by listing the queue depth attribute (`CURDEPTH`) of the queue. For this part of the exercise, you use the `runmqsc` utility.

- \_\_\_ a. From your terminal window, start a `runmqsc` session by typing `runmqsc MQ01` and pressing the Enter key. You are now in interactive mode of the IBM MQ script command processor.



- \_\_ b. Display the attribute that keeps count of the messages in the queue by typing:

```
DIS Q(EX1.LOCALQ) CURDEPTH
```

Expected results are shown, and the number of messages should be 0 in CURDEPTH:  
AMQ8409: Display Queue details.

```
QUEUE(EX1.LOCALQ)                                TYPE(QLOCAL)
CURDEPTH(0)
```

- \_\_ c. Type `end` and press the Enter key to exit `runmqsc`.

- \_\_ 28. Put one message to queue `EX1.LOCALQ` in queue manager `MQ01` by using the `amqspout` sample program.

- \_\_ a. Type `amqspout EX1.LOCALQ MQ01` and be **careful to press Enter one time only**.

The result should be:

```
Sample AMQSPUT0 start
target queue is EX1.LOCALQ
```

- \_\_ b. When you see this prompt, the program waits for input. Type one line and press the Enter key one time to put one message to the queue. You can type any text, for example: `message1`

- \_\_ c. Press the Enter key for a second time to end `amqspout`. The result should be:

```
Sample AMQSPUT0 end
```

- \_\_ 29. Check the message count for queue `EX1.LOCALQ` with `runmqsc`. Type `runmqsc MQ01` and press the Enter key. You should now be in the IBM MQ script command utility.

- \_\_ a. Type `DIS Q(EX1.LOCALQ) CURDEPTH` and press the Enter key.

Results should be:

```
AMQ8409: Display Queue details.
QUEUE(EX1.LOCALQ)                                TYPE(QLOCAL)
CURDEPTH(1)
```

- \_\_ b. Leave the messages in the queue. Type `end` and press the Enter key to exit `runmqsc`.

- \_\_ 30. Get your message from `EX1.LOCALQ` by using the `amqsget` sample program. `amqsget` does a destructive read of the queue, that is, it removes the messages from the queue. The syntax is similar to `amqspout`. Type the command `amqsget EX1.LOCALQ MQ01` and press the Enter key one time. `amqsget` runs for 15 seconds and then ends. (*You can use Ctrl-C if you do not want to wait for `amqsget` to end.*)

```
amqsget EX1.LOCALQ MQ01
```

Expected reply:

```
Sample AMQSGET0 start
message <one message>
no more messages
Sample AMQSGET0 end
```

- \_\_ 31. Put a message to alias queue `FREQ.CHANGED.Q`.

- \_\_ a. Type `amqspout FREQ.CHANGED.Q MQ01` and press the Enter key.

- \_\_\_ b. Type some text such as `message2` and press the Enter key two times to end `amqspu`.
- \_\_\_ 32. Alias queues do not store messages. You need to determine where `FREQ.CHANGED.Q` is pointing to, so you can check whether the message arrived. Do not worry about checking the name of the target queue; you find out in the next step.
  - \_\_\_ a. Open a `runmqsc` session by typing: `runmqsc MQ01`
  - \_\_\_ b. Confirm the target of messages that are put to alias queue `FREQ.CHANGED.Q` by typing the command `DIS Q(FREQ*) TARGET` exactly as shown and pressing the Enter key. If you use an asterisk, you do not need to type the full queue name.

Results should be:

```
AMQ8409: Display Queue details.
      QUEUE(FREQ.CHANGED.Q)                TYPE(QALIAS)
      TARGET(UNDER.COVER.QUEUE)
```

- \_\_\_ c. The previous display confirmed that the message put to alias queue `FREQ.CHANGED.Q` should be in local queue `UNDER.COVER.QUEUE`. Display the message count in `UNDER.COVER.QUEUE` by typing:
 

```
dis q(UNDER.COVER.QUEUE) curdepth
```

Results should be:

```
AMQ8409: Display Queue details.
      QUEUE(UNDER.COVER.QUEUE)            TYPE(QLOCAL)
      CURDEPTH(1)
```



**Note**

In this exercise, you used “DIS Q” to display local and alias queues. It is important to differentiate that if you need to change the attributes of a queue, the correct type of queue must be used, such as:

```
ALTER QLOCAL(queue_name)
CLEAR QLOCAL(queue_name)
```

The correct queue type name is needed to define queues.

**Section 7: Get the message put to the alias queue**



**Note**

While alias queues do not hold messages, when you work with the queue manager **local to the alias queue**, that is, the queue manager where the alias queue is defined, you can get messages from the alias queue. However, you cannot get messages from a remote queue.

\_\_ 33. Get the message from the alias queue by typing: `amqsget FREQ.CHANGED.Q MQ01`

Expected reply:

Sample `AMQSGETO` start

message <one message>

`^C` **← In this example, Ctrl-C was used to break out of the `amqsget` wait.**

## Section 8: Define queues and channel objects for queue manager MQ03



### Warning

**Do not start the channel for which you are about to complete the definitions until explicitly instructed**

\_\_ 34. Move to your `LabFiles/Lab1/source` directory by using the instructions for your operating system.

\_\_ 35. Ensure that the command script file `ex1.MQ03.mqsc` is in the directory. The contents of this file get displayed when you check the result of your definitions.

\_\_ 36. You start the `runmqsc` utility by using a script file. You must be sure to capture the results to ensure that all commands completed successfully. Also, check that you use the correct queue manager name. Type the command exactly as shown:

```
runmqsc MQ03 < ex1.MQ03.mqsc > ex1.MQ03.txt
```

\_\_ 37. Check the results by editing file `ex1.MQ03.txt`. **You must see** messages “No commands have a syntax error” and “All valid MQSC commands were processed” at the end of the results for the two `mqsc` objects defined.

\_\_ 38. The **name** of the local queue, or QLOCAL, that you defined in queue manager MQ03, called FROM.MQ01, should match the RNAME attribute of QREMOTE TO.MQ03, defined in queue manager MQ01. **Do not go back through the instructions to check the names.** Check by typing the command that is shown in a `runmqsc` session **for queue manager MQ01**. The command and expected output are both shown:

```
runmqsc MQ01
5724-H72 (C) Copyright IBM Corp. 1994, 2015.
Starting MQSC for queue manager MQ01.
```

```
dis q(TO.MQ03) RNAME XMITQ
  1 : dis q(TO.MQ03) RNAME XMITQ
AMQ8409: Display Queue details.
  QUEUE(TO.MQ03)                                TYPE(QREMOTE)
  RNAME(FROM.MQ01)                              XMITQ(MQ03)
```

- \_\_\_ 39. You confirmed that the local queue you defined in queue manager MQ03 matches the queue that is expected when you put messages to the TO.MQ03 remote queue. In the command as given, you also listed the name of the transmission queue that is associated with TO.MQ03.



### Reminder

When channels run, they take messages from the transmission queues associated in the channel definition, and transfer them to the remote queue manager. If the channel is not running, messages wait in the transmission queue.

***Do not start the channel until explicitly instructed.***

## Section 9: Anticipate the places where your message might be found

The diagram in this section illustrates the possible stops for your message.

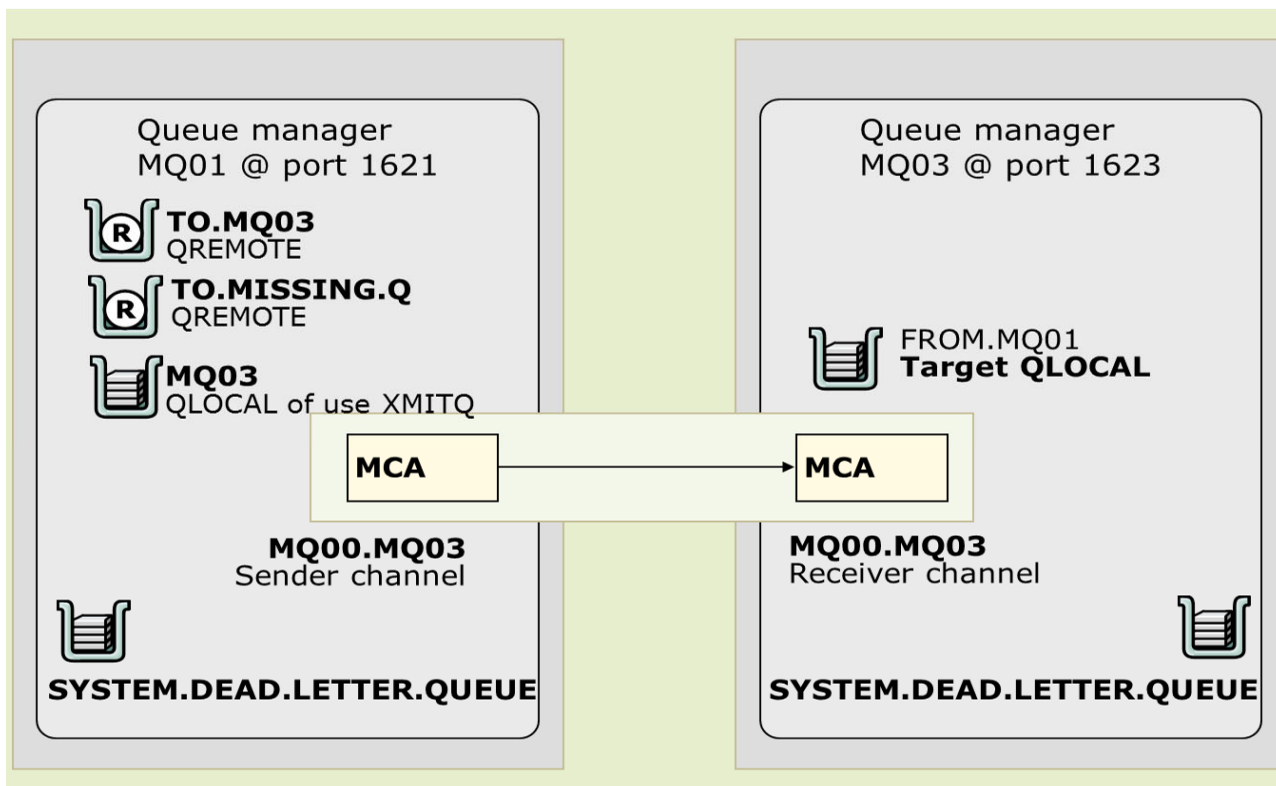


Figure 1-1. Trajectory of a message put to a remote queue

\_\_\_ 40. Review the definitions in the diagram and answer the questions. *Answers are at the end of this exercise under heading **Second group of questions**.*

\_\_\_ a. You put a message to remote queue TO.MQ03 in queue manager MQ01. You remember that remote queues do **not** hold messages. If the channel **is not** running, at what queue manager and queue do you expect your message to be?

\_\_\_\_\_.

\_\_\_ b. If the channel is running, at what queue manager and queue do you expect your message to be?

\_\_\_\_\_.

\_\_\_ c. Assume that the queue manager is not restarted and the message you put is persistent. Look at the queues in the diagram. If your message is not in queue MQ03 at MQ01 or at FROM.MQ01 at MQ03, where else would you check for your message?

\_\_\_\_\_.

\_\_\_\_\_.

## Section 10: Put messages to remote queue TO.MQ03 in queue manager MQ01

\_\_\_ 41. Do not start the channel.

- \_\_ 42. Put a message to queue remote TO.MQ03 in queue manager MQ01.
- \_\_ a. Type `amqsput TO.MQ03 MQ01` and press the Enter key.
  - \_\_ b. Type some text such as `message3` and press the Enter key two times to end `amqsput`.
- \_\_ 43. Since the channel is not running, check the message count in transmission queue MQ03 to determine whether the message is there. Open a `runmqsc` session for queue manager MQ01 and type:

```
dis q(MQ03) curdepth
  1 : dis q(MQ03) curdepth
```

Expected results:

```
AMQ8409: Display Queue details.
  QUEUE(MQ03)                                TYPE(QLOCAL)
  CURDEPTH(1)
```



### Important

The channels that are defined from MQ01 to MQ03 are a **sender-receiver** pair. When you use a sender-receiver channel pair, the sender is started. Ensure that you start **the sender channel from the MQ01** queue manager as indicated.

## Section 11: Start channel MQ01.MQ03 on queue manager MQ01

- \_\_ 44. Use the existing MQ01 `runmqsc` session, or if needed, reopen a `runmqsc` session for queue manager MQ01 by typing `runmqsc MQ01` and pressing the Enter key.
- \_\_ 45. Start the channel by typing `start channel(MQ01.MQ03)` and press the Enter key. Expected result is:
- ```
  1 : start channel(MQ01.MQ03)
  AMQ8018: Start WebSphere MQ channel accepted.
```
- \_\_ 46. While you are still in the `runmqsc` session, check the status of the channel by typing:

```
dis chs(MQ01.MQ03)
```

Expected result:

```
2 : dis chs(MQ01.MQ03)
AMQ8417: Display Channel Status details.
CHANNEL(MQ01.MQ03)                CHLTYPE(SDR)
CONNNAME(127.0.0.1(1623))          CURRENT
RQMNAME(MQ03)                      STATUS(RUNNING)
SUBSTATE(MQGET)                    XMITQ(MQ03)
```

- \_\_ 47. Type `end` and press the Enter key to exit `runmqsc` for queue manager `MQ01`.
- \_\_ 48. The channel is running. The message channel agent is expected to pick up the message from transmission queue `MQ03` in the `MQ01` queue manager, and forward it to queue manager `MQ03`. The message is then placed in its destination queue, queue local to `MQ03`, `FROM.MQ01`. Open a `runmqsc` session for queue manager `MQ03` and check for messages in queue `FROM.MQ01`. The command and results are shown:

```
runmqsc MQ03
```

```
5724-H72 (C) Copyright IBM Corp. 1994, 2015.
Starting MQSC for queue manager MQ03.
```

```
dis q(FROM.MQ01) curdepth
1 : dis q(FROM.MQ01) curdepth
AMQ8409: Display Queue details.
QUEUE(FROM.MQ01)                TYPE(QLOCAL)
CURDEPTH(1)
```

- \_\_ 49. Close the `runmqsc` session by typing `end` and pressing the Enter key.

## Section 12: Put messages to remote queue `TO.MISSING.Q`



### Note

When remote queue `TO.MISSING.Q` was defined in queue manager `MQ01`, the intended target `QLOCAL` on queue manager `MQ03`, queue `NOT.THERE`, was deliberately omitted. You now test what happens when you put messages to queue `TO.MISSING.Q` in queue manager `MQ01`.

- \_\_ 50. Open a `runmqsc` session for queue manager `MQ01` by typing `runmqsc MQ01` and pressing the Enter key.

- \_\_ 51. Check that the channel is still running by typing: `DIS CHS(MQ01.MQ03)`
- \_\_ 52. If the channel is no longer in RUNNING status, start it again by typing  
`START CHL(MQ01.MQ03)`
- \_\_ 53. Exit `runmqsc` by typing `end` and pressing the Enter key.



**Note**

In most organizations, channels are “triggered”. It is not necessary to continue starting the channels (after the first time they are started) because the channel automatically restarts when a message arrives in the transmission queue. The transmission queue is what is triggered.

**For the objectives of this development course, the channels are not triggered.**

- \_\_ 54. Put a message to queue remote `TO.MQ03` in queue manager `MQ01`. Type  
`amqsput TO.MISSING.Q MQ01` and press the Enter key.
- \_\_ 55. Type some text such as `message4` and press the Enter key two times to end `amqsput`.

**Section 13: Check whether local queue *NOT.THERE* exists in MQ03**

- \_\_ 56. Open a `runmqsc` session for queue manager `MQ03` by typing `runmqsc MQ03` and pressing the Enter key.
- \_\_ 57. Display any queues in queue manager `MQ03` that start with string `NOT` by typing the command that is shown and pressing the Enter key:  
`DIS Q(NOT*)`



**Note**

You confirmed that the expected target queue was not defined on queue manager `MQ03`. To continue the search for where your message went, you now check whether the message is in *transmission queue* `MQ03`, of *queue manager* `MQ01`.

- \_\_ 58. Open IBM MQ Explorer. Click the IBM MQ Explorer icon in your VMware image and wait a few moments until it starts.
- \_\_ 59. Expand the *Queue Managers* heading to see all queue managers.
- \_\_ 60. Place your cursor on *queue manager* `MQ01`.
- \_\_ 61. Click heading *Queues*, and locate *queue* `MQ03`.
- \_\_ 62. Place your cursor over queue `MQ03`. Confirm whether column **Current queue depth** for queue `MQ03` has any messages. For this test, **Current queue depth** is expected to have **zero messages**.



**Important**

Ensure that you checked column **Current queue depth**, which is the fourth column from the left of the queue name, for the message count.

If the channel is still running, the channel has *queue* MQ03 open for input (*getting messages from the local queue manager*) and for output (*sending messages to the remote queue manager*). In the case that the channel is running, the *Open input count* and *Open output count* columns show the count of open handles, which are both 1 in the display.

| Queue name                  | Queue type | Open input count | Open output count | Current queue depth | Priority |
|-----------------------------|------------|------------------|-------------------|---------------------|----------|
| AMQ.MQEXPLORER.56B3E9A42000 | Local      | 1                | 1                 | 0                   | AL       |
| EX1.LOCALQ                  | Local      | 0                | 0                 | 1                   | AL       |
| FREQ.CHANGED.Q              | Alias      |                  |                   |                     | AL       |
| MQ03                        | Local      | 1                | 1                 | 0                   | AL       |

**Note**

A note about IBM MQ Explorer. Note the two icons at the upper right of the IBM MQ Explorer graphic. These icons represent toggle switches. The first icon allows the view of objects whose names start with **SYSTEM**. If, when you start looking for **SYSTEM.DEAD.LETTER.QUEUE** you do not find any queue with a name that starts with the word **SYSTEM**, that is system queues, toggle the first icon and the system queues are displayed.

The second icon shows dynamic queues. For example, the queue name that starts with **AMQ** at the start of the display is a dynamic queue that is used by IBM MQ Explorer. If you need to see dynamic queues in later exercises and they do not show, you can use this second icon to the upper right.

**Section 14: Check the IBM MQ error logs**

\_\_\_ 63. Minimize IBM MQ Explorer and return to your command terminal window.



## Stop

IBM MQ error information is kept in different logs. When you need to find **queue manager-related errors**, look in the queue manager logs, under the `<install>/qmgrs/<qmname>/errors` directory, by substituting the queue manager name that you need to investigate for the `<qmname>` placeholder. These files contain information pertinent to the queue managers.

Another `errors` directory, which is at the same level as the `qmgrs` directory, exists. This directory holds **IBM MQ software errors**, and does not provide any help to resolve *queue manager-related problems*. This IBM MQ software directory contains information that you might exchange with the IBM support team if you suspect an IBM MQ problem.

In this course, you work with the **queue manager-related errors** exclusively, at the `<install>/qmgrs/<qmname>/errors` directory.

\_\_ 64. Change directories to the IBM MQ error log files for **queue manager MQ03**. Use the instructions applicable to the operating system you are working with.

Replace `<qmname>` with: **MQ03**



## Linux

From the terminal command prompt, type:

```
cd /var/mqm/qmgrs/<qmname>/errors
```

Where you replace `<qmname>` with the queue manager name.



## Windows

You can access the queue manager error files in two ways:

- From the terminal command prompt, type:

```
cd C:\ProgramData\IBM\MQ\qmgrs\<qmname>\errors
```

Where you replace `<qmname>` with the queue manager name.

- Use Windows Explorer to find directory

`C:\ProgramData\IBM\MQ\qmgrs\<qmname>\errors` where you replace `<qmname>` with the queue manager name whose logs you need to view.

\_\_ 65. Open file `AMQERR01.LOG` for viewing by using the view or edit method for your operating system.

- \_\_ 66. Proceed to the end of the file. View the error log from the bottom to the top of the error file.
- \_\_ a. The queue manager error log contains many entries. You are after an entry that shows:  
AMQ9544: Messages not put to destination queue.
  - \_\_ b. If you see message AMQ9545: Disconnect interval expired followed by a message, which indicates that the channel is stopped, this message is not a problem. It is possible that some time elapsed before you continued this exercise, and the channel closed normally since it was not being used. You might not see the AMQ9545 entry unless you pause before you get to this step.
  - \_\_ c. Scroll backwards to the next entry in the log until you find message AMQ9544. An example of the AMQ9544 message is shown:

```
----- amqrmrca.c : 1051 -----
02/08/2016 01:21:36 PM - Process(439675.1) User(mqadmin8) Program(runmqchl)
                        Host(rhel6base) Installation(Installation1)
                        VRMF(8.0.0.4) QMgr(MQ01)
```

AMQ9544: Messages not put to destination queue.

EXPLANATION:

During the processing of channel 'MQ01.MQ03' one or more messages could not be put to the destination queue and attempts were made to put them to a dead-letter queue. **The location of the queue is 2, where 1 is the local dead-letter queue and 2 is the remote dead-letter queue.**

ACTION:

Examine the contents of the dead-letter queue. Each message is contained in a structure that describes why the message was put to the queue, and to where it was originally addressed. Also look at previous error messages to see if the attempt to put messages to a dead-letter queue failed. The program identifier (PID) of the processing program was '439675'.

- \_\_ d. **Always check** the date and time stamp. It should be close to the date and time when you put a message to queue remote TO.MISSING.Q on MQ01.
  - \_\_ e. Write the time and date stamp of message AMQ9544: here. You might need it later:  
\_\_\_\_\_.
  - \_\_ f. Look at the sentence that is highlighted in bold in the display, which refers to which queue manager dead-letter queue that the message was sent to. In this message, the terms "local" and "remote" apply if the dead-letter queue of the **originating queue manager is 1**, or the dead-letter queue of the *target queue manager is 2*.
  - \_\_ g. Close the error log.
- \_\_ 67. Return to IBM MQ Explorer.

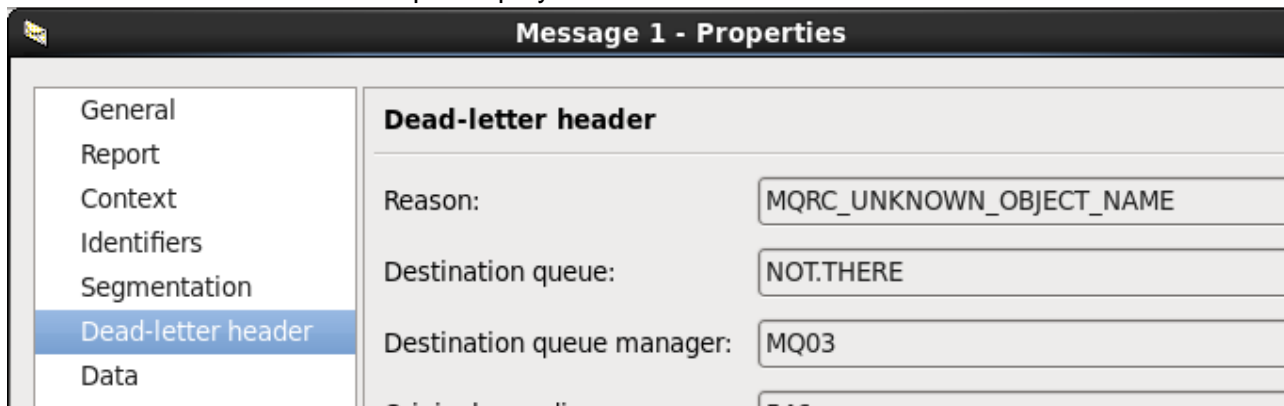
- \_\_\_ 68. Click queue manager MQ03 on the IBM MQ Explorer Navigation pane. Expand the “+” menu under queue manager MQ03 if necessary.
- \_\_\_ 69. Click **Queues**.
- \_\_\_ 70. From the list of queues, you must now scroll down by using the bar on the right of the queue list. Find `SYSTEM.DEAD.LETTER.QUEUE`. If you do not see the dead-letter queue, you might need to use the toggle switch that is on the upper right of IBM MQ Explorer to allow `SYSTEM` named objects to display.
- \_\_\_ 71. Place your cursor over `SYSTEM.DEAD.LETTER.QUEUE`. Notice that one message is in the fourth column, Current queue depth.
- \_\_\_ 72. Right-click `SYSTEM.DEAD.LETTER.QUEUE` and select **Browse messages**. A pop-up pane that is named `Message Browser` appears.



**Hint**

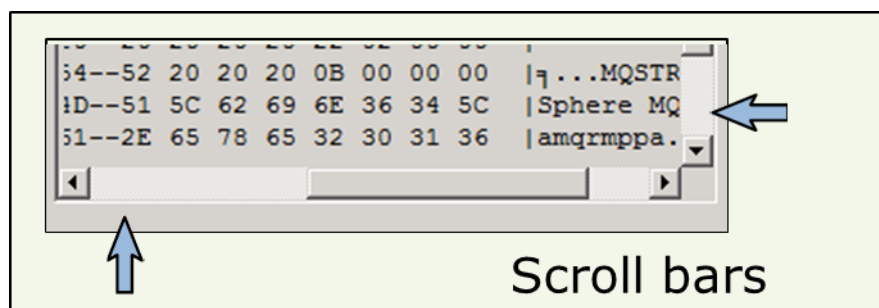
If you have more than one message in the dead-letter queue, check the time and date stamp that you captured from the error log. Select the message with the closest time stamp right before the `AMQ9544` message. This statement is repeated a couple of times in the lab because it is important to look at the current issue, not something that happened earlier.

- \_\_\_ 73. Place your cursor on the message, and double-click. A pop-up display pane with a title `Message n - Properties` appears, where `n` is usually 1, or the message number selected if other than 1. An example display is shown:



- \_\_\_ 74. Place your cursor on the `Dead-letter header` entry label. Look at the Reason field. This field shows an interpreted message code, `MQRC_UNKNOWN_OBJECT_NAME`, which is reason code 2085. The second formatted field provides the name of the “unknown object”, which is local queue `MQRC_UNKNOWN_OBJECT_NAME`, whose definition was deliberately omitted.

- \_\_\_ 75. A different way to look at the reason code is to check the Data entry label. In the lecture for the next unit, you learn about structures such as the dead-letter header. You take an early view of one of these headers. Click the `Data` label in the same pane. You see the unformatted dead-letter header.
- \_\_\_ a. Look at the diagram that follows these steps for a view of the message data. For this display, it was necessary to scroll the data portion to the right to show the structure identifier, `DLH`. It has scroll bars to the right and bottom of the data display. The figure that follows shows the scroll bars:



- \_\_\_ b. The data is in hex. To the right of the structure identifier is a version number and the reason code. The reason code, which is decimal 2058, appears in hex, or `x'0825'`.
- \_\_\_ c. However, in the operating systems that you use for the lab exercises the data is in `Little Endian` format, so it shows as `x'2508'`.
- \_\_\_ d. You can also see the name of the missing queue after the reason code, and the message data you typed (in this example `message 4`) at the end of the character text portion of the message.

| Dead-letter header | Format:             | MQDEAD                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Data               | Encoding:           | 546                                                                                                                                                                                                                                                                                                                                                                                                               |
|                    | Message data:       | DLH <span style="border: 1px solid black; padding: 0 2px;">00</span><br><span style="border: 1px solid black; padding: 0 2px;">01</span>                                                                                                                                                                                                                                                                          |
|                    | Message data bytes: | <pre> 00--25 08 00 00 4E 4F 54 2E   DLH ...%...NOT. 20--20 20 20 20 20 20 20 20   THERE 20--20 20 20 20 20 20 20 20   20--20 20 20 20 4D 51 30 33   MQ03 20--20 20 20 20 20 20 20 20   20--20 20 20 20 22 02 00 00   "... 54--52 20 20 20 06 00 00 00   7...MQSTR .... 61--20 20 20 20 20 20 20 20   amqrmppa 20--20 20 20 20 32 30 31 36   2016 31--33 35 37 39 6D 65 73 73   020818213579mess --   age 4 </pre> |

- \_\_\_ 76. Click **Close** in each of the pop-up displays to close all boxes except IBM MQ Explorer.
- \_\_\_ 77. Minimize IBM MQ Explorer and return to your terminal command window.

\_\_ 78. The sample program `amqsbcbg` can also be used to do a non-destructive get, so the messages are not removed from a queue. `amqsbcbg` reads the message, formats the MQMD header, and dumps the data portion of the message in hex. Type the `amqsbcbg` request as shown:

```
amqsbcbg SYSTEM.DEAD.LETTER.QUEUE MQ03
```

\_\_ 79. Check the output. Bypass the formatted message descriptor and start your review at the hex dump below the `*** Message ***` header. In the output, find:

- \_\_ a. The dead-letter structure header, DLH.
- \_\_ b. The name of the missing queue.
- \_\_ c. The operating systems that are used for the lab exercises use Little Endian encoding. The reason code, in Little Endian format, is `x'2508'` instead of `x'0825'` for decimal 2058 reason code.

```
[mqadmin8@rhel6base ~]$ amqsbcbg SYSTEM.DEAD.LETTER.QUEUE MQ03

AMQSBG0 - starts here
*****
MQOPEN - 'SYSTEM.DEAD.LETTER.QUEUE'

MQGET of message number 1, CompCode:0 Reason:0
****Message descriptor****
* For purposes of this exercise, message descriptor is omitted from this display *

**** Message ****
length - 181 of 181 bytes

00000000: 444C 4820 0100 0000 2508 0000 4E4F 542E      'DLH ....%...NOT.'
00000010: 5448 4552 4520 2020 2020 2020 2020 2020      'THERE'
00000020: 2020 2020 2020 2020 2020 2020 2020 2020      ' '
00000030: 2020 2020 2020 2020 2020 2020 4D51 3033      ' MQ03'
00000040: 2020 2020 2020 2020 2020 2020 2020 2020      ' '
00000050: 2020 2020 2020 2020 2020 2020 2020 2020      ' '
00000060: 2020 2020 2020 2020 2020 2020 2202 0000      ' "...'
00000070: B804 0000 4D51 5354 5220 2020 0600 0000      '....MQSTR ....'
00000080: 616D 7172 6D70 7061 2020 2020 2020 2020      'amqrmppa'
00000090: 2020 2020 2020 2020 2020 2020 3230 3136      ' 2016'
000000A0: 3032 3038 3138 3231 3335 3739 6D65 7373      '020818213579mess'
000000B0: 6167 6520 34                                     'age 4'

No more messages
MQCLOSE
```

## Section 15: Alter alias queue `FREQ.CHANGED.Q` to point to queue `TO.MQ03`

- \_\_ 80. Open a `runmqsc` session for queue manager `MQ01` by typing `runmqsc MQ01` and pressing the Enter key.
- \_\_ 81. Change the target of alias queue `FREQ.CHANGED.Q` to point to remote queue `TO.MQ03` by typing:

```
ALTER QALIAS(FREQ.CHANGED.Q) TARGET(TO.MQ03)
```

Expected results:

```
1 : ALTER QALIAS(FREQ.CHANGED.Q) TARGET(TO.MQ03)
AMQ8008: WebSphere MQ queue changed.
```

- \_\_ 82. Exit `runmqsc` by typing `end` and pressing the Enter key.
- \_\_ 83. Put a message to alias queue `FREQ.CHANGED.Q`.
  - \_\_ a. Type `amqspout FREQ.CHANGED.Q MQ01` and press the Enter key.
  - \_\_ b. Type message text `This is the last message for Exercise 1` and press the Enter key two times to end `amqspout`.
- \_\_ 84. Return to IBM MQ Explorer.
- \_\_ 85. Under Queue managers, select **MQ01**.
- \_\_ 86. Place your cursor on the Channels label, taking care that you are still under queue manager **MQ01**.
- \_\_ 87. Locate channel `MQ01.MQ03`. If the Overall channel status column shows as `Running`, **proceed to the next numbered step**. If the channel shows inactive under the Overall channel status column, start the channel by:
  - \_\_ a. Placing your cursor over sender channel name `MQ01.MQ03`. Make sure that the channel type is `Sender`. **If the channel type is receiver, you are in the wrong queue manager.**
  - \_\_ b. Right-click and select `Start channel`. After a few moments, the channel status might briefly change to `Initializing`, but it should soon turn to `Running` as shown on the display:

| Channels                      |              |                        |
|-------------------------------|--------------|------------------------|
| Filter: Standard for Channels |              |                        |
| Channel name                  | Channel type | Overall channel status |
| MQ01.MQ03                     | Sender       | Running                |

- \_\_\_ 88. In the last few steps, you changed alias queue `FREQ.CHANGED.Q` to point to remote queue `TO.MQ03`. You also ensured that channel `MQ01.MQ03` was in running status.

Where do you expect the last message put to `FREQ.CHANGED.Q`, which contained text `This is the last message for Exercise 1`, to be? *The answer is at the end of this exercise under heading **Last question**.*

---



### Reminder

When you use the Message Browser in IBM MQ Explorer, sometimes it is necessary to scroll with the bar along the hex part of the message to see the character data portion of the message dump.

- \_\_\_ 89. Use IBM MQ Explorer to confirm your prediction of where the queue manager and queue where the last message was placed. From IBM MQ Explorer:
- \_\_\_ a. Click your predicted queue manager name.
  - \_\_\_ b. Click **queues**, and place the cursor on your predicted queue. Remember you might need to scroll up or down to find the queue name.
  - \_\_\_ c. Right-click the queue name and select **Browse messages**.
  - \_\_\_ d. Double-click the right message and select the Data label. Scroll to the right as needed to see the message data you typed: `This is the last message for Exercise 1`.
- \_\_\_ 90. If you find your message, **you completed Exercise 1**. Click **Close** to close all the extra display boxes.
- \_\_\_ 91. If you did not find your message, look at the answers in the next section to determine whether your prediction was correct.
- \_\_\_ 92. If your prediction was correct and you cannot find the last message, check alias queue `FREQ.CHANGED.Q` to ensure that no errors were present when you changed the queue name. You can use either a `runmqsc` session for queue manager `MQ01`, or IBM MQ Explorer to check the alias queue.
- \_\_\_ 93. If you still cannot find your message, obtain assistance from your instructor.



## Exercise 1 answers.

Questions are repeated. **Answers are in bold.**

### First group of questions:

- How many local queues (QLOCAL) did you define?
  - Three QLOCAL type queues: EX1.LOCALQ, UNDER.COVER.Q, and MQ03
- What is the difference between local queue MQ03 and the other local queues?
  - **MQ03 is a transmission queue. The USAGE attribute is set to XMITQ.**
- If you place a message in the alias queue FREQ.CHANGED.Q, where can you expect to find the message?
  - **The message is in QLOCAL UNDER.COVER.Q.**
- The remote queues and the sender channel point to a transmission queue (XMITQ). What local queue is used as the transmission queue?
  - **MQ03**
- In a remote queue definition, the RNAME attribute points to the name of a local queue in the remote queue manager. Two remote queues (QREMOTE) are defined. What are the names of the two local queues that are expected to be in the remote queue manager?
  - **Remote queue TO.MQ03 points to queue FROM.MQ01 in queue manager MQ03.**
  - **Remote queue TO.MISSING.Q expects to find a queue that is named NOT.THERE in queue manager MQ03. However, NOT.THERE is deliberately omitted. Do not define the missing queue.**
- When a channel is running, messages put to a remote queue are expected to flow to the remote queue manager. If the channel is not running, messages put to a remote queue wait in the transmission queue until the channel starts. If messages are put to both remote queues TO.MQ03 and TO.MISSING.Q and the channel is stopped, where do you expect to find the messages that you put on the remote queues? (This question arises since remote queues do not hold messages.)
  - **If the channel is not running, messages are held in the transmission queue that is associated with the remote queue definition. For this exercise, both remote queues use transmission queue MQ03.**

### Second group of questions

- Review the definitions in the diagram and answer the questions. *Answers are at the end of this exercise under heading **Second group of questions**.*
- You put a message to remote queue TO.MQ03 in queue manager MQ01. You remember that remote queues do **not** hold messages. If the channel **is not** running, at what queue manager and queue do you expect your message to be?

- Queue manager MQ01, queue MQ03 (transmission queue for both the channel and the TO.MQ03 remote queue definition)
- If the channel is running, at what queue manager and queue do you expect your message to be?
  - Queue manager MQ03, queue FROM.MQ01
- Assume that the queue manager was not restarted and the message you put is persistent. If your message is not MQ03 at MQ01 or at FROM.MQ01 at MQ03, where else would you check for your message?
  - In this case, you need to check the dead-letter queue in each queue manager. You might need to check the queue manager logs at `<install_data_dir_path>/qmgrs/QMNAME/errors`, where `<install_data_dir_path>` is the platform-dependent directory where the data for your IBM MQ configuration and logs is kept.

If you find the message in the dead-letter queue, you might use IBM MQ Explorer or any other tool that formats the dead-letter queue header, and look up the reason why the message was rejected. You can find the reason by typing `mqrc xxxxx` and pressing the Enter key. Replace the `xxxxx` placeholder with the reason code that is found in the dead-letter header. You can also look up the reason code in the IBM MQ Reference manual.

The message might be in the dead-letter queue of the remote queue manager, in the dead-letter queue of the originating (local) queue manager, or even returned to the transmission queue, depending on the situation. The error logs usually show a message that explains where the message went, and why.

**Do not forget the dead-letter queue that is associated with a queue manager is not always SYSTEM.DEAD.LETTER.QUEUE. To confirm the correct queue that the queue manager currently uses, open a runmqsc session, and type the command: DIS QMGR DEADQ**



### Example

```
DIS QMGR DEADQ
1 : DIS QMGR DEADQ
AMQ8408: Display Queue Manager details.
QMNAME(MQ03)
DEADQ(SYSTEM.DEAD.LETTER.QUEUE)
```

## Last question

Where do you expect the last message put to `FREQ.CHANGED.Q`, which contained text `This is the last message for Exercise 1`, to be?

\_\_\_\_\_ The message should be in queue `FROM.MQ01` of queue manager `MQ03`. \_\_\_\_\_

### End of exercise

## Exercise review and wrap-up

In this exercise, you learned to complete basic tasks to help you test your programs and find your messages. You also learned to do some basic troubleshooting by checking the IBM MQ queue manager logs, and by locating the failing reason code in the dead-letter queue header. You were able to:

- Determine the status of queue managers in a server
- Start a queue manager
- Use the runmqsc utility and command scripts to create IBM MQ objects and check results
- Put messages to local and alias queues and determine whether the messages arrived at the intended destination
- Determine the trajectory and possible stops of a message put to a remote queue
- Start a sender channel and check the channel status
- Check the queue manager error logs
- Determine where your message is
- Examine the dead-letter queue and identify the reason that a message was placed in the queue



---

## Exercise 2. Getting started with IBM MQ development

### What this exercise is about

In this exercise, you start work with IBM MQ development by learning how to make basic changes, and compile an MQI program. You use named constants to determine the superseding IBM MQ behavior when the same object attribute and MQI attribute use different values. In the last section of the exercise, you experience the outcome of an incorrect version number in a structure. You learn about the missed version number by learning how to code connection authentication in the MQCONN call. You also learn how to check the queue manager connection authentication settings.

### What you should be able to do

After completing this exercise, you should be able to:

- Compile and test a copy of the put message sample program
- Review the cmqc.h structure and the initialization values for the message descriptor structure
- Review selected default values of a local queue definition
- Add MQOPEN, MQPUT, and MQCLOSE calls to an application
- Change persistence attributes in a program by using named constants
- Determine the outcome of persistence behavior when the queue definition attributes and the MQI attributes use different values
- Check the queue manager environment to determine the connection authentication settings
- Set a variable to test connection authentication with a program
- Determine the results of not setting the correct version number in a structure

### Introduction

In this exercise, you start by compiling an IBM MQ C program. You learn how to check the contents of the cmqc.h header file to find information and items necessary for your code. After you add processing of a second queue to a program, you learn the importance of controlling certain behaviors such as persistence by using your code, instead of deferring it to the queue definition. In the last section of the exercise, you learn the relationship between the queue manager connection authentication configuration and your connection

authentication code. You learn about key MQCNO structure fields that affect the processing of connection authentication.

When you work with persistence, your focus is on checking the number of messages in the queues before and after a restart of the queue manager.

When you work with connection authentication, your focus is on observing how your inputs to the code correlate with the queue manager connection authentication configuration.

## Requirements

- Queue manager MQ01
- IBM MQ object definition scripts `ex2.MQ01.mqsc`
- Source for `putmsg.c` and `twoqput.c` at `<dir>\LabFiles\Lab2\source`
- Solution source for `twoqput.c` at `<dir>\LabFiles\Lab2\solution`

## Platform-dependent notes

Repeating platform-dependent tasks are detailed in Exercise 1. Some of the tasks are repeated one time for convenience. If you need to review details that are repeated frequently, refer to Exercise 1.



### Linux

#### Compiling code (repeated from Exercise 1)

- In the course environment, the code is in directory `/home/mqadmin8/LabFiles/Lab#/source`
- The course environment has the path set for the IBM MQ libraries. To compile programs, move to the code directory and type:

```
gcc -I/opt/mqm/inc -lmqm -o executable_name program_name.c
```

where `program_name.c` is the source, and `executable_name` is the name of the executable program.

#### Making a backup copy of the source

- Ensure that you are in the `/home/mqadmin8/LabFiles/Lab#/source` directory.
- Type `cp sourcename.c sourcename.c.bkp` and press the Enter key, where `sourcename` is the name of the program.



### Windows

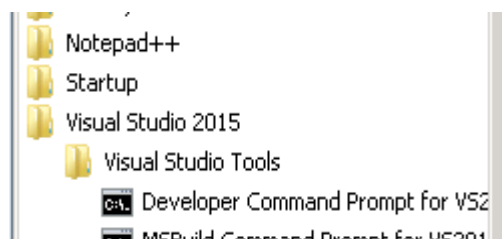
#### Frequently occurring Windows tasks:

##### Make a backup copy of the source (before you change anything)

- Ensure that you are in the `C:\LabFiles\Lab#\source` directory, where you replace the # with the exercise number you are working with. For example, if you are working Exercise 1, then you should be at `C:\LabFiles\Lab1\source`.
- Type `copy sourcename.c sourcename.c.bkp` and press the Enter key, where `sourcename` is the name of the program.

##### Compile code

- Open a visual studio command prompt by going to **Start > All Programs > Visual Studio 2015 > Visual Studio Tools > Developer Command Prompt for VS2015**.



- **Change to the source directory for the exercise you are working with.**
- File `copy_edit.txt` at `C:\LabFiles\Lab#\source` can be used to copy and paste the compile command. You need to replace the `program_name.c` placeholder with the name of your C source.
- Type the command as shown, or change and use the `copy_edit.txt` file to paste the compile command:

```
cl program_name.c C:\PROGRA~1\IBM\WEBSPH~1\Tools\Lib\mqm.lib
```

where `program_name.c` is the name of the program source.



### Important

**All work in Exercise 2 uses queue manager MQ01.**



### Hint

Throughout the exercises, you can recall past commands that were typed on your terminal command session by using the up arrow in your keyboard. You can then alter part of the command as needed and resubmit.

To save time, it is suggested that, **for those of you working on Windows**, after you open the Developer Command Prompt for VS 2015, you leave that window open. When you start a new exercise, change directories as needed from the same window. You can also recall the CL command with the up arrow, and change the source name as needed, rather than typing or copying the entire command.



## Part 1: Compile and use the MQI

### Section 1: Compile and run a C program



#### Note

Program `putmsg.c` is a slightly modified copy of sample program `amqsput0.c`. You use `putmsg.c` to try your first compile, and later you alter and recompile the code.



#### Linux

**Before you work with `putmsg.c`**, make a backup copy by typing: `cp putmsg.c putmsg.c.bkp`

- \_\_ 1. Change to the `LabFiles/Lab2/source` directory where the `putmsg.c` code is located according to your platform-specific instructions.
- \_\_ 2. Compile `putmsg.c` and place the results in module `putmsg` as shown in your platform-specific instructions.

```
gcc -I/opt/mqm/inc -lmqm -o putmsg putmsg.c
```



#### Windows

**Before you work with `putmsg.c`**, make a backup copy by typing: `copy putmsg.c putmsg.c.bkp`

- \_\_ 3. Open a visual studio command prompt by going to **Start > All Programs > Visual Studio 2015 > Visual Studio Tools > Developer Command Prompt for VS2015**.
- \_\_ 4. Change to the `C:\LabFiles\Lab2\source` directory.
- \_\_ 5. File `copy_edit.txt` at `C:\LabFiles\Lab2\source` can be used to copy and paste the compile command. You need to replace the `program_name.c` placeholder with the name of your C source.

- \_\_ 6. Type the command as shown, or change and use the `copy_edit.txt` file to paste the compile command:

```
cl putmsg.c C:\PROGRA~1\IBM\WEBSPH~1\Tools\Lib\mqm.lib
```

**Expected results are:**

```
putmsg.c
Microsoft (R) Incremental Linker Version 14.00.23506.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:putmsg.exe
putmsg.obj
C:\PROGRA~1\IBM\WEBSPH~1\Tools\Lib\mqm.lib
```

- \_\_ 7. Test that the compiled source runs by typing `putmsg` without any parameters. **You can type the test commands in the VS Developer Command Prompt window.**

**If the usage information is displayed, the program compiled correctly.**

```
putmsg
```

Expected result:

```
Lab source putmsg start
Required parameter missing - queue name
```



**Hint**

To save time, you might want to leave the Developer Command Prompt window open. Remember to change directories as needed for the different exercises.

## Part 2: Work with code to override object attributes



### Information

The next set of steps demonstrates the importance of specifying the required persistence in the code, rather than relying on the queue definitions. Many other attributes, besides persistence, exhibit the same behavior where the coded attributes supersede the object definition.

The table shows six combinations of *persistence named values* and *persistence queue attributes*. You define local EX2NONPERS queue with default initial values of no persistence, and local queue EX2PERS with persistence.

You change source `twoqput.c` as needed to have the program write to both queues at the same time. After you have the code to put messages to both queues, you use it one time to test with the “persistence per queue definition” MQMD initial default. Then, you change the code again to have the MQMD `Persistence` set to use the persistent and non-persistent named values. At the end, you test all six iterations of persistence queue attribute and code combinations.

Every time that you put messages to both queues with one persistence code value, you restart the queue manager to confirm results.

In addition, you define a non-persistent alias queue whose target is a persistent queue. With this definition, you demonstrate the case where the attributes of the **original object**, that is, the alias queue, supersedes the attributes of target objects. You test the alias queue first with the `putmsg` program.

| Persistence MQI code / queue attribute  | DEFPSIST (NO) | DEFPSIST (YES) |
|-----------------------------------------|---------------|----------------|
| <code>MQPER_PERSISTENCE_AS_Q_DEF</code> | Lost          | Kept           |
| <code>MQPER_PERSISTENT</code>           | Kept          | Kept           |
| <code>MQPER_NOT_PERSISTENT</code>       | Lost          | Lost           |

### Section 2: Define and review queues EX2PERS, EX2NONPERS, and alias queue TO.PERS

- \_\_ 8. From the terminal command prompt window, change to your `LabFiles/Lab2/source` directory.
- \_\_ 9. Create the queues for this exercise in queue manager MQ01 by using the `mqsc` script `ex2.MQ01.mqsc`. Type your command as shown. **Ensure that you selected the correct queue manager and script before you proceed:**

```
runmqsc MQ01 < ex2.MQ01.mqsc > ex2.out.txt
```

Edit the results file `ex2.out.txt` and review the results. The results at the end of the file should indicate that all valid MQSC commands were processed as shown in the

display. No errors are expected. If you find an error, make any corrections before you proceed. Request assistance if necessary.

```

*** MQSC definitions for queue manager MQ01 exercise 2 ***
: *****
: * Persistent local queue
1 : DEF QLOCAL(EX2PERS) DEFPSIST(YES)
AMQ8006: WebSphere MQ queue created.
: *
: * Non-Persistent local queue
2 : DEF QLOCAL(EX2NONPERS) <=== DEFPSIST defaults to NO
AMQ8006: WebSphere MQ queue created.
: *
: * Non-Persistent alias queue
3 : DEF QALIAS(TO.PERS) TARGTYPE(Queue) +
:   TARGET(EX2PERS) DEFPSIST(NO)
AMQ8006: WebSphere MQ queue created.
: *

3 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.

```

- \_\_ 10. Review the three queue definitions:
- \_\_ a. Queue EX2PERS has the DEFPSIST attribute set to YES. If a message is put to this queue with the Persistence field of the message descriptor set to MQPER\_PERSISTENCE\_AS\_Q\_DEF, that message is stored on this queue as a persistent message.
  - \_\_ b. Queue EX2NONPERS has the DEFPSIST attribute set to NO. If a message is put to this queue with the Persistence field of the message descriptor set to MQPER\_PERSISTENCE\_AS\_Q\_DEF, that message is stored on this queue as a non-persistent message.
  - \_\_ c. Queue TO.PERS is an alias queue with the DEFPSIST attribute set to NO, with a target queue of EX2PERS with the DEFPSIST attribute set to YES. If a message is put to this alias queue with the Persistence field of the message descriptor set to MQPER\_PERSISTENCE\_AS\_Q\_DEF, that message is stored on the queue EX2PERS as a non-persistent message.
- \_\_ 11. Open a runmqsc session for queue manager MQ01 to confirm that queue EX2NONPERS is defined with the persistence attribute set to NO. Type runmqsc MQ01 and press the Enter key.
- \_\_ 12. Type DIS Q(EX2NONPERS) DEFPSIST and press the Enter key.

```
dis q(EX2NONPERS) defpsist
```

Expected results are:

```
1 : dis q(EX2NONPERS) defpsist
AMQ8409: Display Queue details.
        QUEUE(EX2NONPERS)                                TYPE(QLOCAL)
        DEFPSIST(NO) <=== confirms that default queue attribute is not persistent
```

- \_\_ 13. You can see all initial default values in a queue definition by typing `dis q(QNAME) ALL` and pressing the Enter key. If you try this display, you can use the Reference documentation in your desktop to learn what the different attributes represent.
- \_\_ 14. Type `end` to exit the `runmqsc` session.

### Section 3: Examine the `putmsg.c` and `cmqc.h` header code

- \_\_ 15. Move to your `LabFiles/Lab2/source` directory by following the instructions for the operating system you are working with.
- \_\_ 16. Edit to browse program `putmsg.c`.
  - \_\_ a. Find string `MQMD`. You should see a line of code as shown:
 

```
MQMD    md = {MQMD_DEFAULT};    /* Message Descriptor */
```
  - \_\_ b. In the same line, `md` is initialized with the initial default values for the `MQMD` structure, kept in a named variable structure named `MQMD_DEFAULT`.
- \_\_ 17. Move to the installation include file directory according to the instructions for the operating system you are working with, and **browse** file `cmqc.h`.
  - \_\_ a. Find the `MQMD_DEFAULT` named-value structure.
  - \_\_ b. Check the initial value that is used for the persistent attribute of the `MQMD` structure. Partial results should be:

```
#define MQMD_DEFAULT {MQMD_STRUC_ID_ARRAY},\
                    MQMD_VERSION_1,\
                    MQRO_NONE,\
                    MQMT_DATAGRAM,\
                    MQEI_UNLIMITED,\
                    MQFB_NONE,\
                    MQENC_NATIVE,\
                    MQCCSI_Q_MGR,\
                    {MQFMT_NONE_ARRAY},\
                    MQPRI_PRIORITY_AS_Q_DEF,\
                    MQPER_PERSISTENCE_AS_Q_DEF,\
                    ... ..
```

\_\_ 18. Confirm that the “persistence as queue definition” is maintained in the code by checking for the `md` label until you get to the `MQPUT` to ensure that no changes are done to the `md.Persistence` value.



**Information**

Before you proceed to change the code, check what happens when you use:

- An alias queue that is defined with no persistence, `DEFPSIST(NO)`
- Its target queue that is defined with persistence, `DEFPSIST(YES)`
- An application with `MQPER_PERSISTENCE_AS_Q_DEF` in the Persistence attribute of the MQMD structure

**Section 4: Put a message to non-persistent alias queue TO.PERS in queue manager MQ01**

\_\_ 19. You confirmed that program `putmsg.c` uses the `MQPER_PERSISTENCE_AS_Q_DEF` option in the MQMD. Use `putmsg` to put a message to alias queue `TO.PERS` by typing:

```
putmsg TO.PERS MQ01
```

Expected results:  
 Lab source `putmsg` start  
 target queue is `TO.PERS`

\_\_ 20. Type some text, such as “this message does not survive a restart” and press the Enter key two times to end the `putmsg` application.

- \_\_ 21. The target for alias queue `TO.PERS` is `EX2PERS`. Check that the message is in the queue by using the non-destructive get sample `amqsbcg` as shown. **Take special care that you do use `amqsbcg`, as any other sample application might remove the message.** Type the command as shown in the first line of the text box and press the Enter key.

```
amqsbcg EX2PERS MQ01
```

Results should be (partial display with relevant data shown):

```
MQGET of message number 1, CompCode:0 Reason:0
```

```
****Message descriptor****
```

```
StrucId : 'MD ' Version : 2
```

```
Report : 0 MsgType : 8
```

```
Expiry : -1 Feedback : 0
```

```
Encoding : 546 CodedCharSetId : 1208
```

```
Format : 'MQSTR '
```

```
Priority : 0 Persistence : 0 <=== 0 denotes message is not persistent
```

```
MsgId : X'414D51204D513031
```

```
...
```

```
... (remainder of formatted MQMD bypassed for purposes of this step)
```

```
...
```

```
**** Message **** Message data displays the text that you typed
```

```
length - 39 of 39 bytes
```

```
00000000: 7468 6973 206D 6573 7361 6765 2064 6F65 'this message doe'
```

```
00000010: 7320 6E6F 7420 7375 7276 6976 6520 6120 's not survive a '
```

```
00000020: 7265 7374 6172 74 'restart '
```

- \_\_ 22. You have one message in queue `EX2PERS`. `EX2PERS` is set to `DEFPSIST(YES)`, so the `putmsg` application puts a persistent message to this queue. However, the alias queue that uses `EX2PERS` as its target, `TO.PERS`, has the `DEFPSIST` attribute set to `NO`. Restart the queue manager and check the message by typing both the `endmqm` and `strmqm` commands shown. Wait a few seconds after the queue manager stops before you type the command to start it. Type the commands as shown in the text box:

```
endmqm MQ01
```

Expected results are:

```
Quiesce request accepted. The queue manager will stop when all outstanding work is complete.
```

`strmqm MQ01`

Expected results are:

```
WebSphere MQ queue manager 'MQ01' starting.  
The queue manager is associated with installation 'Installation1'.  
5 log records accessed on queue manager 'MQ01' during the log replay phase.  
Log replay for queue manager 'MQ01' complete.  
Transaction manager state recovered for queue manager 'MQ01'.  
WebSphere MQ queue manager 'MQ01' started using V8.0.0.4
```

23. Check the messages in queue EX2PERS by using `amqsbcg`. It is assumed that no other messages were in queue EX2PERS at this stage of the lab. **What you need to check is that the message that you put by using queue TO.PERS is gone. Ignore any other message other than the one you put before the restart of the queue manager.**  
Expected results are:

`amqsbcg EX2PERS MQ01`

```
AMQSBCG0 - starts here  
*****  
MQOPEN - 'EX2PERS'  
  
No more messages  
MQCLOSE  
MQDISC
```

### Section 5: Change program `twoqput.c` to open and put the same message to a second queue.



#### Important

Source `twoqput.c` needs to be changed to add processing of a second queue. After the change, when you put to a first queue whose queue name you provide as an input parameter, the same message is also put to queue `EX2NONPERS`.

`twoqput.c` is an adaptation of the `amqsput0.c` sample program. To ensure that the application works for both queues, some buffer and string manipulation of the message data was necessary. These changes are general programming tasks (versus IBM MQ programming), and are done for you.

You code the `MQOPEN`, `MQPUT`, and `MQCLOSE` function calls for the second queue. Both queues use the same queue manager connection to open them, which means that the same connection handle works for both queues. However, you need different object and return code declarations for



the second queue, which are already coded for you. You use these declarations in the MQOPEN, MQPUT, and MQCLOSE for the second queue.

**Be careful to differentiate which code artifacts can be used for both queues, and which ones need separate resources.**



### Warning

**Before you proceed with the rest of the exercise, ensure that you made a backup copy of the `twoqput.c` source by using your operating system-dependent instructions that were given at the start of this exercise!**

**If time becomes a challenge**, directory `LabFiles/Lab2/solution` contains a completed solution file, which can be used to copy the code.

- \_\_\_ 24. Locate source that is named `twoqput.c` in your `LabFiles/Lab2/source` directory, and open it for edit.
- \_\_\_ 25. The source is marked with the string `ex2` where changes are made, or need to be made, to add the second queue to the program. Find the first instance of the `ex2` marker and review the declarations added for the second queue. You might want to refer to this section when coding your function calls. This section is for review. **No changes are made here.**



### Hint

**You might want to copy and paste these declarations so that you do not need to be scrolling in your code.**

#### **For Linux**

Open a second command prompt. Copy and paste the definitions into a `vi` editor session. Save them to a temporary file. Use the “cat” command to display your file in the second command prompt session.

#### **For Windows**

Copy and paste these definitions to a new Notepad++ file, so you can refer to them easily when you code the MQOPEN and MQPUT call.

You then toggle the two Notepad++ tabs to check the declarations, without having to scroll. Notepad++ also presents you with a choice of declarations to select when you are coding your function calls.

```

/* ***ex2 declarations*** */
47  MQOD      npod = {MQOD_DEFAULT}; /* ex2 second q object desc */
48  MQHOBJ    Hnpobj;                /* ex2 second queue handle */
49  MQLONG    npOC;                   /* ex2 open return code) */
50  MQLONG    npReas;                 /* ex2 open reason code */
51  MQLONG    npputOC;               /* ex2 put return code) */
52  MQLONG    npputReas;             /* ex2 put reason code */
53  char      npq[] = "EX2NONPERS";  /* ex2 2nd q name for MQOPEN */
54  MQLONG    savelen = 0;           /* ex2 save length of msg for put */
55  char      npbuff[65535];        /* ex2 copy buffer after fgets */
56  int       loopctr = 0;          /* ex2 used to copy full buffer */
57  /* end of ex2 declarations */

```

- \_\_\_ a. Review the first four definitions, which are required for the second queue.
  - \_\_\_ b. You need to isolate the return codes from the queue, so the return and reason were also duplicated.
  - \_\_\_ c. You use the `npq` initialization variable to provide the name of the second queue. While the original queue name is obtained from input parameters, the second queue name is hardcoded in the `EX2NONPERS` variable.
  - \_\_\_ d. `npbuff` is a string that is used to save the message that was typed in for the first queue. You use `npbuff` for the second `MQPUT`.
  - \_\_\_ e. `savelen` keeps the original length of the input message.
  - \_\_\_ f. `loopctr` is not directly related to IBM MQ code; it is used to adapt the existing code to add the second queue.
- \_\_\_ 26. Do a find for the original `MQCONN` call. You need the same connection handle for the `MQOPEN` and `MQPUT` calls. Record the name of the connection handle here for later use: \_\_\_\_\_ . Do not worry about the ID and password code for now; you do not use it in this part of the lab.
- \_\_\_ 27. Find comment label `Open the target message queue for output` and review the `MQOPEN` call and the options set.
- \_\_\_ 28. Proceed to where the `O_options` (open options) are set. Note the format that is used when you need to provide multiple options. You use the same options in your open call, so make note of the `O_options` parameter name; **you reuse the same parameter** as set in this call.

```
O_options = MQOO_OUTPUT          /* open queue for output      */
           | MQOO_FAIL_IF QUIESCING /* but not if MQM stopping */
           ;
```

\_\_ 29. Continue to review the MQOPEN call as an example to complete the MQOPEN for the second queue. Note how the fields in the structure are addressed.



### Hint

If you need to confirm the name or initialized values of any fields, or the name of any field names in the object descriptor structure (MQOD) or any other structure:

- Open a second terminal and browse file `/opt/mqm/inc/cmqc.h`
- Browse for the structure and initializing values

For example, for the object descriptor structure, search for “Object desc”. You should see the object descriptor structure.

The initializing values for a structure are usually kept in the named value with the structure name followed by `_DEFAULT`. For MQOD, it is `MQOD_DEFAULT`. You can browse or scroll forward to find the initial values; for MQOD, they are directly following the structure definition in `cmqc.h`.

## Section 6: Add the MQOPEN for the second queue to the program

In your MQOPEN work for the second queue, you:

- Reuse the connection handle
- Use the newly declared object descriptor
- Reuse the open options
- Replace the last three MQOPEN parameters with the newly declared fields.

\_\_ 30. Proceed to the section titled MQOPEN for queue EX2NONPERS. You can search for that title or scroll forward through the code.

- \_\_ a. Look at the string-copy to name `npod.ObjectName`, which is shown in bold. This copy is done for you. Note how it is using the object descriptor instance that was defined for the second queue, followed by the object descriptor name field.
- \_\_ b. **Uncomment the MQOPEN call**, and add the needed parameters.

**Hint**

If you need help with the syntax for the function calls, you can refer to either the course material, or to the code provided in the solution folder for this exercise.

```

/* MQOPEN for queue EX2NONPERS in ex2                                     */
/* The queue name is already copied to the object descriptor */
/* A line to display the name of the second queue is added */
strcpy(npod.ObjectName, npq, (size_t)MQ_Q_NAME_LENGTH);
printf("second target queue is %s\n", npod.ObjectName);

/* Complete your MQOPEN here by using ex2 declarations                */
/* where needed. Return code checking is done for you                */
/* below the MQOPEN call. Note use of ex2 declarations                */
/* Uncomment the MQOPEN call and adjust comment lines                */

/* MQOPEN */                  /* connection handle NO CHANGE */
/* ex2 object descriptor for queue */
/* open options NO CHANGE */
/* ex2 object handle */
/* ex2 MQOPEN completion code */
/* ex2 reason code */

```

- \_\_\_ 31. The code that is used to check the status of the MOPEN call is completed for you. This status checking code should use the return and reason codes field that is defined for this exercise, which you should use in your MQOPEN call.
- \_\_\_ 32. **You should be in the C:\LabFiles\Lab2\source directory.**
- \_\_\_ 33. Compile the partially changed code to eliminate any problems by typing:

**Linux**

```
gcc -I/opt/mqm/inc -lmqm -o twoqput twoqput.c
```

**Windows**

- \_\_\_ 34. If not already done, open a visual studio command prompt by going to **Start > All Programs > Visual Studio 2015 > Visual Studio Tools > Developer Command Prompt for VS2015.**

- \_\_ 35. File `copy_edit.txt` at `C:\LabFiles\Lab2\source` can be used to copy and paste the compile command. You need to replace the `program_name.c` placeholder with the name of your C source.
- \_\_ 36. Type the command as shown, or change and use the `copy_edit.txt` file to paste the compile command:

```
cl putmsg.c C:\PROGRA~1\IBM\WEBSPH~1\Tools\Lib\mqm.lib
```

- \_\_ 37. If errors surface with the code added, make corrections before you proceed.



### Information

**Do not change persistence now.** If you see the comment that is shown here, ignore it now.

```
* ex2 This is where you change the persistence options */
```

After your MQPUT to the two queues is working, and you complete the first test that uses the “persistent as queue definition” default setting, then you return to change persistence. Now you focus on adding the calls to process the second queue.

## Section 7: Add the MQPUT for the second queue



### Warning

The line numbers that are shown in the displays are approximate and might differ in the actual file.

- \_\_ 38. Locate the `Uncomment` the MQPUT string and **add the code for the new queue**.
- \_\_ a. Remove the comment text around the MQPUT function call.
  - \_\_ b. Reuse the original connection handle.
  - \_\_ c. Use the object handle defined for this exercise.
  - \_\_ d. Reuse the message descriptor **and** put message options.
  - \_\_ e. Use the `saveLen` field for the message length.
  - \_\_ f. Use the `npbuff` saved message buffer for the data.
  - \_\_ g. **Ensure that you use** the **separate** `npputOC` and `npputReas` completion and reason code fields that are defined for the MQPUT in the new declarations for this exercise.

```

/* ex2 Uncomment the MQPUT and add the remaining parameters      */
320  /* starting here   */
321  /*  MQPUT  */                /* connection handle          */
322                                /* ex2 object handle        */
323                                /* message descriptor      */
324                                /* put message options     */
325                                /* ex2 saved message length */
326                                /* ex2 message buffer       */
327                                /* ex2 put completion code */
328                                /* ex2 put reason code */

```

- \_\_ 39. Recompile the code as instructed in the MQOPEN section, and if necessary, make any corrections before you proceed.

### Section 8: Add the MQCLOSE for the second queue

- \_\_ 40. Open the twoqput.c source for edit and search for string Uncomment for MQCLOSE until you find the section of code shown:

```

/* ex2 Uncomment the MQPUT and add the remaining parameters      */

370  /* ex2 MQCLOSE for second queue      */
371  /* add your MQCLOSE whre indicated   */
372  /* Status handling is coded for you  */
373  if (npoc != MQCC_FAILED)
374  {
375      /* ex2 Uncomment the MQCLOSE call function and -      */
376      /* ex2 start adding your code to close second queue here */
377      /* MQCLOSE */                /* connection handle          */
378                                /* ex2 object handle        */
379                                /* closing options unchanged */
380                                /* ex2 completion code         */
381                                /* ex2 reason code             */
382

```

- \_\_ 41. Code the MQCLOSE call for the second queue.
- \_\_ a. Use the same connection handle as in the MQOPEN call.
  - \_\_ b. Use the object handle defined for the second queue.

- \_\_ c. Use the same closing options.
  - \_\_ d. Use the completion and reason codes that are used in the `MQOPEN` call.
- \_\_ 42. Recompile the code as instructed in the `MQOPEN` section and, if necessary, make any corrections before you proceed.

### **Section 9: Test the updated twoqput application with the MQMD Persistence attribute set to MQPER\_PERSISTENCE\_AS\_Q\_DEF**



#### **Important**

The updated program is expected to place the first message that is typed in the second queue, EX2NONPERS, and also in queue EX2PERS. For the tests in this section, and the sections that follow, type **one message** and press the Enter key to end the program.

If you type more than one message, queue EX2PERS receives all messages that are typed, but EX2NONPERS gets the first message. For purposes of this exercise, type one message.

**However, this first test is also checking the success of the changes that you made to the program to add the second queue. If the program is not working as described, you need to make corrections before proceeding.**



#### **Note**

You changed the program to write to both the queue that is defined as persistent, EX2PERS, and the queue that is defined as non-persistent, EX2NONPERS, in the same program. You need to run the program three times to test with each iteration. You revisit the table that was presented earlier in this exercise. This first test is for the first row, `MQPER_PERSISTENCE_AS_Q_DEF`. You write to both the persistent and the non-persistent queue.

| Persistence MQI code / queue attribute  | DEFPSIST (NO) | DEFPSIST (YES) |
|-----------------------------------------|---------------|----------------|
| <code>MQPER_PERSISTENCE_AS_Q_DEF</code> | <b>Lost</b>   | <b>Kept</b>    |
| <code>MQPER_PERSISTENT</code>           | <b>Kept</b>   | <b>Kept</b>    |
| <code>MQPER_NOT_PERSISTENT</code>       | <b>Lost</b>   | <b>Lost</b>    |

- \_\_ 43. Open IBM MQ Explorer. Select the MQ01 queue manager.

\_\_ 44. Select the Queues menu and locate the EX2PERS and EX2NONPERS queues.

| Queue name     | Queue | Open in | Open | Current c |
|----------------|-------|---------|------|-----------|
| EX1.LOCALQ     | Local | 0       | 0    | 0         |
| EX2NONPERS     | Local | 0       | 0    | 0         |
| EX2PERS        | Local | 0       | 0    | 0         |
| FREQ CHANGED Q | Alias |         |      |           |

\_\_ 45. For each queue, look under the Current queue depth column. Make sure that queue manager MQ01 is selected. If the queue count is greater than zero on any of the queues:

- \_\_ a. Place your cursor on the queue name and right-click.
- \_\_ b. Select **Clear messages**.
- \_\_ c. Check that the Current queue depth was reset to zero.

\_\_ 46. When both EX2 queues have zero messages, run program `twoqput` by typing:

```
twoqput EX2PERS MQ01
```

Expected results are:  
 Lab source putmsg start  
 target queue is EX2PERS  
 second target queue is EX2NONPERS

\_\_ 47. Type one message such as `Message with code pers per q def` and press the Enter key two times to exit the program.

\_\_ 48. Return to IBM MQ Explorer and check that there is at least one message in queue `EX2PERS` and one message in queue `EX2NONPERS`. You might see other messages in queue `EX2PERS`.



**Stop**

This run is your first run of the program after the changes. ***If the code is not working as described, make necessary corrections, and return to the start of this section until your code runs as expected. Ensure that the queues are back to zero messages.***

\_\_ 49. Both queues were put with the code option to use the persistence of the queue definition. One queue is defined as persistent, and the other is not. If the queue manager is restarted, what happens to the messages?



- \_\_ 50. Return to your terminal command window. Stop the queue manager by typing `endmqm MQ01` and wait a few seconds for the queue manager for the end process to complete.
- \_\_ 51. Remaining in your terminal command window, type `strmqm MQ01` to start the queue manager.
- \_\_ 52. Return to IBM MQ Explorer.

**Note**

It might take a few moments for IBM MQ Explorer to refresh its view and show queue manager MQ01 restarted. You can request a refresh of the IBM MQ Explorer view by clicking the Refresh icon. The Refresh icon is at the upper-right corner of IBM MQ Explorer.



- \_\_ 53. Return to the Queues view for queue manager MQ01. The code is using the same persistence as the queue definition. The expected results are:
  - \_\_ a. Queue `EX2PERS`, defined with attribute `DEFPSIST(YES)`, has one (or more) messages.
  - \_\_ b. Queue `EX2NONPERS`, with initial default attribute `DEFPSIST(NO)`, has no messages.
  - \_\_ c. If your results are different, review your code and your queue definitions and resolve the problem.
- \_\_ 54. Return to IBM MQ Explorer, locate queue `EX2PERS` in queue manager MQ01, and clear messages by right-clicking the queue name and selecting **Clear Messages**.
- \_\_ 55. **Ensure that the Current queue depth for each EX2PERS and EX2NONPERS is zero before you continue.**

### **Section 10: Test the updated twoqput application with the MQMD Persistence attribute set to MQPER\_PERSISTENT**

- \_\_ 56. If your `twoqput.c` is working correctly, make a backup copy before you continue.
- \_\_ 57. Open `twoqput.c` for edit.
- \_\_ 58. Locate the first MQPUT function call.
- \_\_ 59. Directly before the first MQPUT function call, type code to **change** `md.Persistence` to `MQPER_PERSISTENT` as shown:

```
md.Persistence = MQPER_PERSISTENT;
```



## Reminder

The message descriptor is an input and output field. The `md.Persistence` set in the first MQPUT call might change. Repeat the addition to set persistence before the second MQPUT call.

- \_\_ 60. Find the second MQPUT call, and add the same code right above the MQPUT.
- \_\_ 61. Save and recompile `twoqput.c`.
- \_\_ 62. Run program `twoqput` by typing:

```
twoqput EX2PERS MQ01
```

Expected results are:

```
Lab source putmsg start
target queue is EX2PERS
second target queue is EX2NONPERS
```

- \_\_ 63. Type one message such as `Message with mqper_persistent` coded and press the Enter key two times to exit the program.

## **Section 11: Does the MQMD Persistence attribute set to `MQPER_PERSISTENT` supersede the alias queue persistence?**



## Information

Earlier in this exercise you put a message to queue `TO.PERS`, which uses queue `EX2PERS` as its target, and confirmed that the alias queue definition supersedes the target queue definition. What happens when the code specifies persistence?

- \_\_ 64. Run program `twoqput` by typing the command that is shown.

**Warning**

**Ensure that you use queue name TO.PERS for this alias queue test.**

```
twoqput TO.PERS MQ01
```

Expected results are:

```
Lab source putmsg start
```

```
target queue is TO.PERS
```

```
second target queue is EX2NONPERS
```

- \_\_ 65. Type one message such as `Message put to alias queue` and press the Enter key two times to exit the program.

### **Section 12: Resume the test with the MQMD Persistence attribute set to MQPER\_PERSISTENT**

- \_\_ 66. Return to IBM MQ Explorer and verify that **two messages** are in each queue, `EX2PERS` and `EX2NONPERS`. The second message is the one put with the alias queue.
- \_\_ 67. Return to your command input terminal and restart the queue manager by first typing:  

```
endmqm MQ01
```

 Wait a few moments, and then type: `strmqm MQ01`
- \_\_ 68. Return to IBM MQ Explorer. It might be necessary to wait a few moments or refresh the view.
- \_\_ 69. When queue manager `MQ01` is available, check the messages. Since you made the messages persistent in the code, it is expected that the messages remained in each queue. The Current queue depth for queues `EX2PERS` and `EX2NONPERS` should be 2.
- \_\_ 70. If your results are not as expected, resolve the problem before you proceed. Request assistance if necessary.
- \_\_ 71. **Clear the messages from both queues as previously instructed before you proceed.**

### **Section 13: Test the updated twoqput application with the MQMD Persistence attribute set to MQPER\_NOT\_PERSISTENT**

- \_\_ 72. Open `twoqput.c` for edit.
- \_\_ 73. Go to the location of the first MQPUT.
- \_\_ 74. Change the `md.Persistence` setting to make the messages non-persistent by setting the persistence attribute to `MQPER_NOT_PERSISTENT` as shown:

```
md.Persistence = MQPER_NOT_PERSISTENT;
```

- \_\_ 75. Repeat the change for the second MQPUT call.
- \_\_ 76. Save and recompile `twoqput.c`.
- \_\_ 77. Run program `twoqput` by typing:

```
twoqput EX2PERS MQ01
```

Expected results are:

```
Lab source putmsg start
target queue is EX2PERS
second target queue is EX2NONPERS
```

- \_\_ 78. Type one message such as `Message` with `mqper_not_persistent` coded and press the Enter key two times to exit the program.
- \_\_ 79. **You do not test with the alias queue again.** You should be able to deduce the outcome.
- \_\_ 80. Return to IBM MQ Explorer and verify that one message is in each queue, `EX2PERS` and `EX2NONPERS`.
- \_\_ 81. Return to your command input terminal and restart the queue manager by first typing:  
`endmqm MQ01`  
Wait a few moments, and then type: `strmqm MQ01`
- \_\_ 82. Return to IBM MQ Explorer. It might be necessary to wait a few moments or refresh the view.
- \_\_ 83. When queue manager MQ01 is available, check the messages. Since you made the messages not persistent in the code, it is expected that the messages did not survive the restart of the queue manager. The `Current queue depth` for `EX2PERS` and `EX2NONPERS` should be 0.
- \_\_ 84. If your results are not as expected, resolve the problem before proceeding. Request assistance if necessary.

**You completed the second section of Exercise 2.**

## Part 3: MQCONNX and connection authentication



### Warning

Failure to set the required version number in a structure can lead to behavior in a program that might be difficult to correct when you do not realize that the problem is an incorrect version number.

While this part of the exercise teaches how to code connection authentication, it also shows you what happens when you overlook the version number in a structure.



### Information

The IBM MQ authentication service is accessed by using the `MQCONNX` function call and the connection security parameters in the `MQCSP` structure. The `MQCONNX` connection options structure (`MQCNO`) holds a pointer to the `MQCSP` structure.

The `MQCNO` structure contains a version number parameter that must be set in order for the `MQCSP` information to be considered. Without the version number, the `MQCSP` is ignored.

- Know the underlying queue manager settings to successfully test connection authentication in your code.
- Be familiar with the initial default values of the connection security parameters (`MQCSP`) structure.
- Work with the IBM MQ administrator to ensure that the password is protected when connecting to a remote queue manager.

The connection options structure `MQCNO` points to the security structure `MQCSP`.

For this part of the exercise, your focus is connection authentication. It is not necessary to check the messages that are put to the queue.

### ***Section 14: Review the connection authentication initial value in the `cmqc.h` header***

- \_\_ 85. Open file `cmqc.h` according to the instructions for the operating system you are using.
- \_\_ 86. Find string `MQCNO_DEFAULT` to see the connection options default values related to connection authentication. Partial initialization values are shown:

```

#define MQCNO_DEFAULT {MQCNO_STRUC_ID_ARRAY},\
                    MQCNO_VERSION_1,\
                    MQCNO_NONE,\
                    ... ..

```

- \_\_ 87. Review the `MQCNO_VERSION_1` setting. To use connection authentication, this value must be changed to `MQCNO_VERSION_5` or the `MQCSP` structure processing is ignored.
- \_\_ 88. Close the `cmqc.h` file
- \_\_ 89. Open program `putmsg.c` for browse.
- \_\_ 90. Review the code comments on how to code connection authentication for the `MQCONN` call. These comments and code can be located, approximately, between lines 65 and 85 of the `putmsg.c` source. **The code and comments are also included in the display that follows.**



**Information**

Line numbers in the code are approximate.

**No action is necessary now; you review this section, but do not change any code.**

When you need to include connection authentication in your code:

- Look at the course `putmsg.c` or IBM MQ sample `amqsput0.c`, from which `putmsg.c` was created:
  - a. The code uses variable `MQSAMP_USER_ID` to input the User ID.
  - b. It prompts for the password.
- Lines 38 – 39: The `MQCNO` and `MQCSP` structures are declared and initialized with the default structure names.
- Lines 89 – 90: In the `MQCNO` structure, you:
  - a. Set the pointer to the `MQCSP` structure.
  - b. Set the `cno.Version` to `MQCNO_VERSION_5` so that the `MQCSP` structure is acknowledged. **Make note of this line; you work with it later in the exercise.**
- Line 91: In the `MQCSP` structure, you:
  - a. Set `AuthenticationType` to `MQCSP_AUTH_USER_ID_AND_PWD` so that you can use name and password authentication.

- b. Provide a pointer and the corresponding length to the user ID and password strings.

```

38     MQCNO    cno = {MQCNO_DEFAULT}; /* connection options          */
39     MQCSP    csp = {MQCSP_DEFAULT}; /* security parameters        */
85  /*****
86  /* Set the connection options to use the security structure and */
87  /* set version information to ensure the structure is processed.*/
88  /*****
89     cno.SecurityParmsPtr = &csp;
90     cno.Version = MQCNO_VERSION_5;
91
92     csp.AuthenticationType = MQCSP_AUTH_USER_ID_AND_PWD;
93     csp.CSPUserIdPtr = UserId;
94     csp.CSPUserIdLength = strlen(UserId);

106     csp.CSPPasswordPtr = Password;
107     csp.CSPPasswordLength = strlen(csp.CSPPasswordPtr);

```

\_\_ 91. Close putmsg.c.

## Section 15: Display the queue manager connection authentication settings

- \_\_ 92. Open or return to a command terminal window and open a runmqsc session by typing:  
runmqsc MQ01
- \_\_ 93. Type the `dis qmgr connauth` and `dis authinfo` commands exactly as shown in the text box to see the current connection authentication settings of the queue manager.

```
dis qmgr connauth
```

Expected results:

```

1 : dis qmgr connauth
AMQ8408: Display Queue Manager details.
QMNAME(MQ01)
CONNAUTH(SYSTEM.DEFAULT.AUTHINFO.IDPWOS)

```

```
dis authinfo(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) CHCKCLNT CHCKLOCL
```

Expected results:

```

2 : dis authinfo(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) CHCKCLNT CHCKLOCL
AMQ8566: Display authentication information details.
AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS)
AUTHTYPE(IDPWOS)                CHCKCLNT(REQDADM)
CHCKLOCL(OPTIONAL)

```

Your results should resemble the text box. In the lecture, you learned that the `OPTIONAL` setting means that if no ID is provided, no authentication is done – but if an ID is provided, then a valid password must be entered. The sample programs and `putmsg` are set up to prompt for the password when the `MQSAMP_USER_ID` variable is used.

**If `CHCKCLNT` and `CHCKLOCL` are both `OPTIONAL`, do not be concerned.**

\_\_ 94. Type `end` and press the Enter key to exit the `runmqsc` session.

### Section 16: Test the authentication code with the `MQSAMP_USER_ID` variable

\_\_ 95. Set the `MQSAMP_USER_ID` variable to the ID you are using for your operating system.



Linux

```
export MQSAMP_USER_ID=mqadmin8
```



Windows

```
set MQSAMP_USER_ID=administrator
```

\_\_ 96. Run the `putmsg` program to put one message to queue `EX2PERS` as shown. Before you are able to type any text, you are prompted provide the password:

```
putmsg EX2PERS MQ01
```

Expected result:  
Lab source `putmsg` start  
Enter password:

\_\_ 97. Type the password `weblsphere` and press the Enter key as shown.

**After you complete the test, keep this command prompt window open!**

```
Enter password: weblsphere  
target queue is EX2PERS
```

\_\_ 98. Type any text and press the Enter key two times to end the `putmsg` program.



## Section 17: Repeat the test but do not provide a valid password

\_\_ 99. Type `putmsg EX2PERS MQ01` and press the Enter key.

\_\_ 100. When prompted for the password, type `xyz` or any other invalid password as shown:

```
putmsg EX2PERS MQ01
```

Expected result:

```
Lab source putmsg start
```

Enter password: `xyz`

Expected result:

```
MQCONNX ended with reason code 2035
```

\_\_ 101. If you need help in determining the meaning of reason code 2035, type `mqrcc 2035` as shown:

```
mqrcc 2035
2035 0x000007f3 MQRC_NOT_AUTHORIZED
```

\_\_ 102. Leave the `MQSAMP_USER_ID` variable set.

## Section 18: If the `MQCNO` version number is set to 1, what happens?

\_\_ 103. From the same command terminal window where the `MQSAMP_USER_ID` variable is set, edit program `putmsg.c` and locate the line of code where the `MQCNO` version is set to 5.

\_\_ 104. Comment out the line of code that sets the `MQCNO` version to `MQCNO_VERSION_5` as shown. **After you recompile, the `cno.Version` uses the initial default value, `MQCNO_VERSION_1`.**

```
/*      cno.Version = MQCNO_VERSION_5; */
```

\_\_ 105. Save `putmsg.c` and recompile.

**Stop**

When you test `putmsg` after you recompile, you are still prompted for a password due to the `MQSAMP_USER_ID` variable setting.

**Do not provide a password;** press the Enter key at the password prompt.

\_\_ 106. From the same terminal window where the `MQSAMP_USER_ID` is set, test the `putmsg` code now by typing `putmsg EX2PERS MQ01` as shown, but **do not provide a password**. **Press the Enter key at the Password prompt.**

```
putmsg EX2PERS MQ01
```

Expected return:

Lab source `putmsg` start

Enter password: **<== press the Enter key without typing a password**

Expected return:

target queue is EX2PERS

**Note**

When the `MQCNO` `version` was set to 1, connection authentication was ignored.

`MQSAMP_USER_ID` causes the sample program to prompt for a password, but as expected, the program continued to accept input without a valid password.

Remember, structures might require a version number to be set for a feature to work. A missed structure number might result in time that is wasted in “problem” determination.

**End of exercise**

## Exercise review and wrap-up

In this exercise, you learned to:

- Compile an IBM MQ C program
- Check the cmqc.header to check names of fields in structures to obtain initialization constants for structures, and find numerous items that are needed to work with the MQI
- Review selected default values of a local queue definition
- Change a program that puts to one queue to add an MQOPEN, MQPUT, and MQCLOSE to process a second queue
- Change persistence in a program by using named constants
- Recognize the importance of setting persistence in the application instead of deferring it to the queue definition
- Determine queue manager connection authentication settings
- Add connection authentication structures and code to the MQCONN call
- Alter the connection options structure to disable connection authentication processing
- Recognize the importance of setting the version of a structure, so certain features are processed



## Exercise 3. Working with MQOPEN and queue name resolution, MQPUT, and MQMD fields

### What this exercise is about

This exercise reinforces topics that are related to the MQOPEN and MQPUT calls. In the first part of the exercise, you learn about queue name resolution by coding a program to put a message to a remote queue manager without using a remote queue. You learn how to create dynamic queues with different naming options. You then learn how to use the Report field to request confirmation on arrival and expiry messages. You also learn how to work with the IBM MQ V8.0.0.4 expiry cap object attribute.

### What you should be able to do

After completing this exercise, you should be able to:

- Code various combinations of queue manager and queue name in the object descriptor to test and confirm how queue name resolution takes place
- Code report options and review the results in the reply-to queue
- Create and display a dynamic queue where the queue manager determines the name
- Create and display a dynamic queue by specifying a partial prefix of the queue name
- Create and display a dynamic queue by specifying the exact queue name
- Request a confirm-on-arrival with data report message
- Set the expiry attribute in a message and request an expiry report
- Set the expiry attribute in a message for a queue with a lower expiry cap value in the queue definition

### Introduction

This exercise has three distinct parts. The first part of this exercise is critical in understanding how the parameters used during the MQOPEN call control queue name resolution. Next, you learn to create dynamic queues and use different naming techniques. In Part 3 of this exercise, you learn to request report messages. As part of the report messages, you also test expiry with a queue altered with the IBM MQ V8.0.0.4 cap expiry feature.

## Requirements

- Queue manager MQ01 at IBM MQ V8.0.0.4
- Source for `qnmres.c`, `crtdyn.c`, and `putmsg.c` at `<dir>/LabFiles/Lab3/source`

## Platform-dependent notes

Repeating platform-dependent tasks are detailed in Exercise 1. Some of the tasks are repeated one time for convenience. If you need to review details that are repeated frequently, refer to Exercise 1.



### Linux

#### Compiling code (repeated from Exercise 1)

- In the course environment, the code is in directory `/home/mqadmin8/LabFiles/Lab#/source` where `#` is the number of the exercise with which you are working.
- The course environment has the path set for the IBM MQ libraries. To compile programs, move to the code directory and type:

```
gcc -I/opt/mqm/inc -lmqm -o executable_name program_name.c
```

where `program_name.c` is the source, and `executable_name` is the name of the executable program.

#### Making a backup copy of the source

- Ensure that you are in the correct `/home/mqadmin8/LabFiles/Lab#/source` directory
- Type `cp sourcename.c sourcename.c.bkp` and press the Enter key, where `sourcename` is the name of the program.



### Windows

#### Compiling code (repeated from Exercise 1)

- In the course environment, the code is in directory `C:\LabFiles\Lab#\source` where `#` is the number of the exercise with which you are working.
- Open a visual studio command prompt by going to **Start > Programs > Visual Studio 2015 > Visual Studio Tools > Developer Command Prompt for VS2015**.
- Change to the `C:\LabFiles\Lab#\source` directory.
- File `copy_edit.txt` at `C:\LabFiles\Lab#\source` can be used to copy and paste the compile command. You need to replace the `program_name.c` placeholder with the name of your C source.
- Type the command as shown, or change and use the `copy_edit.txt` file to paste the compile command:

```
cl source_name.c C:\PROGRA~1\IBM\WEBSPH~1\Tools\Lib\mqm.lib
```

- **Leave the Developer Command Prompt window open so you can reuse it. These instructions are not repeated in their entirety!**

#### Making a backup copy of the source

- Ensure that you are in the correct `C:\LabFiles\Lab#\source` directory

Type `copy sourcename.c sourcename.c.bkp` and press the Enter key, where `sourcename` is the name of the program.



**Reminder**

Remember to always back up the source code before you change any code.



## Part 1: Queue name resolution

In earlier exercises, you used alias and remote queues. In this exercise, you change attributes in the MQOD structure to observe the behavior of queue name resolution. You see how the values you set in the MQOD structure result in selection of a locally defined or remotely defined queue. Later in the exercise you use code to work with model and dynamic queues.

For the first part of the exercise, you work with queue managers MQ01 and MQ03. Local queue `EX3.SAME.QNAME` is predefined in each queue manager. These queues are non-clustered, local queues.

You learn where your message goes, depending on the attribute you set for the `MQOPEN`.

**All connections (`MQCONN` calls) for all tests in this exercise use queue manager MQ01 as the second parameter that you provide to the program.**

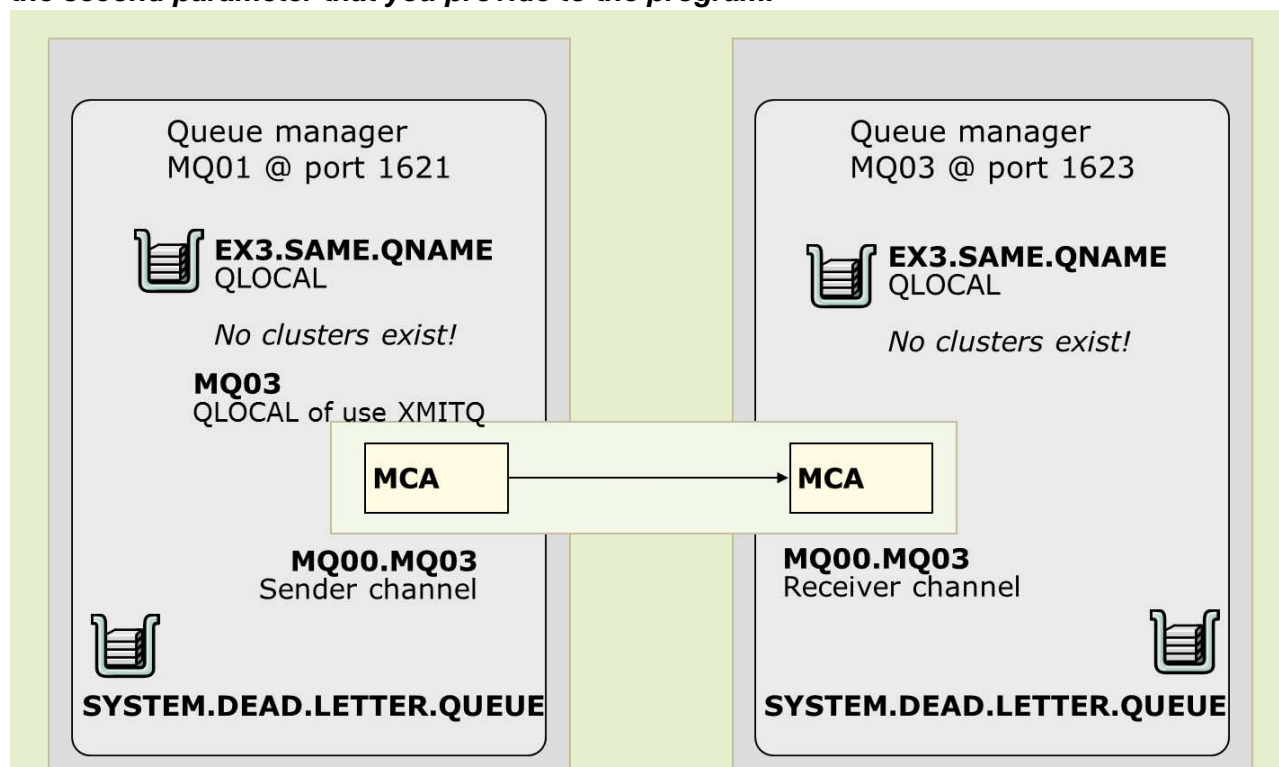


Figure 3-1. Local queue `EX3.SAME.QNAME` exists in MQ01 and MQ03

- \_\_ 1. Log on to your VMware image and open two terminal command windows.
- \_\_ 2. On both terminals, move to directory `<dir>/LabFiles/Lab3/source` where the `<dir>` prefix depends on the operating system you are working with.
- \_\_ 3. Back up your copy of program `qnmres.c` to a file named `qnmres.bkp.c`.
- \_\_ 4. Open `qnmres.c` for edit. The code should look familiar; it is a modified copy of sample `amqsput0.c`.

## Section 1: Review the MQCONNX and MQOPEN queue manager name use in qnmres.c



### Important

In this section you **do not change** the code; you review and compile.

- \_\_ 5. Locate string `Ex3` taking care to use exact case and review the contents of the declaration as shown. You should advance to the declaration shown, located approximately in line 51 of the source. You use this variable to specify the **queue manager name** in the `MQOD` for the `MQOPEN` call. Its initial value is blanks. **Do not change this line for the first compile.**

```
51 char    OpenQMN[50] = "";                /*Ex3 open qmgr name    */
```

- \_\_ 6. Find the next occurrence of string `Ex3`, which should take you to approximately line 102.
- \_\_ 7. Look at the code that follows. `QMName` is taken from the program input parameter and set in the `MQCNO` structure for the `MQCONNX` call.
- \_\_ 8. Repeat the search for the `Ex3` string so that you are at approximately line 137. You see where the queue manager name is set in the `MQOD` for the `MQOPEN` call. **Do not change any code.**

```
137 /* Ex 3 changes */
138     strncpy(od.ObjectQMName, OpenQMN, (size_t) MQ_Q_MGR_NAME_LENGTH);
139     printf("target queue manager is %s\n", od.ObjectQMName);
```

- \_\_ 9. Compile `qnmres.c` by using the method appropriate for the operating system you are using.



### Linux

```
gcc -I/opt/mqm/inc -lmqm -o qnmres qnmres.c
```



## Windows

You should be:


- At an open developer command window.
- At the source directory for the lab that you are working with. For this lab, you should be at `C:\LabFiles\Lab3\source`.

If you need help setting up the Developer Command Windows, refer to the notes at the start of this exercise.

To compile, type:

```
cl qnmres.c C:\PROGRA~1\IBM\WEBSPH~1\Tools\Lib\mqm.lib
```

## Section 2: Check or start queue managers MQ01 and MQ03, and channel MQ01.MQ03

- \_\_\_ 10. Display the status of the queue managers MQ01 and MQ03 in your VMware image by typing `dspmq` and pressing the Enter key. **Do not start MQ15.**
- \_\_\_ 11. If the status of MQ01 or MQ03 is `Ended`, start the `Ended` queue manager by typing `strmqm queuemanagername` and pressing the Enter key. Replace `queuemanagername` with the queue manager name that shows the `Ended` status. You might need to repeat the start command for both MQ01 and MQ03.
- \_\_\_ 12. Start a `runmqsc` session **for the MQ01** queue manager by typing: `runmqsc MQ01`
- \_\_\_ 13. Start the `MQ01.MQ03` sender channel by typing `start channel(MQ01.MQ03)` and pressing the Enter key.
- \_\_\_ 14. Type `end` and press the Enter key to exit the `runmqsc` session.
- \_\_\_ 15. Open IBM MQ Explorer. Under the Queue Managers label in the left navigation pane, you should see that both MQ01 and MQ03 started, as denoted by the up green arrow icon.
- \_\_\_ 16. Under queue manager MQ01, click the Channels label. On the channel listing in the right pane, confirm that channel `MQ01.MQ03` is started by looking for the up green arrow icon. It might take a few moments to show the channel active. You can also click the refresh  icon to obtain the channel status update.
- \_\_\_ 17. Locate queues `EX3.SAME.QNAME` on both the MQ01 and MQ03 queue managers. **You check the queue depth, or number of messages in these queues, throughout the next two sections.**
- \_\_\_ 18. Ensure the current queue depth for queue `EX3.SAME.QNAME` is zero in both queue managers.
- \_\_\_ 19. If the starting number of messages is not zero, clear the queue by right-clicking in queue name `EX3.SAME.QNAME` and selecting **Clear Messages**. **Be careful that you select Clear Messages, and no other option.**

- \_\_ 20. If you did not need to clear messages from the queues, **proceed to the next section**.
- \_\_ 21. If you cleared messages from the queues, press Accept or OK the confirmation panels to finish clearing the messages so that you start the next section with both queues empty.

### Section 3: Test the `qnmres` program with the `MQOPEN MQOD` queue name set to `EX3.SAME.QNAME` and the `MQOPEN MQOD` queue manager name set to blanks



#### Important

This information is detailed here. In subsequent steps, the instruction requests to “test the program by using specific queue and queue manager name.”

Program `qnmres` works like the `amqsput0.c` sample program. To start the program:

- Type the program name
- Type the queue name
- Type the **connecting** queue manager name
- Press the Enter key one time

Program `qnmres` **hardcodes** the queue manager name for the `MQOPEN`.

When you start the program, it responds by confirming:

- `qnmres EX3.SAME.QNAME MQ01 <=== Program execution, MQ01 for MQCONN`
- Lab source `qnmres start <=== Program responds with parameters used for MQOPEN`
- `target queue is EX3.SAME.QNAME <=== MQOPEN queue name`
- `target queue manager is <=== MQOPEN value of OpenQMN variable; it might be blank`

You then type one message. Each time that you press the Enter key, you send the line that you typed as a message.

To end the program, press the Enter key two times consecutively, without data.



#### Reminder

Remember, the local queue that you use to test `qnmres`, `EX3.SAME.QNAME`, exists in both `MQ01` and `MQ03`.

- \_\_ 22. Test the first test with the blank queue manager name.
- \_\_ a. Type `qnmres EX3.SAME.QNAME MQ01` and press the Enter key.
- \_\_ b. Type `Message 1` and press the Enter key two times to end `qnmres`.

- \_\_ 23. Check the contents of queue `EX3.SAME.QNAME` in both queue managers by using IBM MQ Explorer. What queue manager did the message resolve to, `MQ01` or `MQ03`?
- \_\_\_\_\_.



### Information

The correct result should be `MQ01`. When the queue manager name that is used in the `MQOPEN` call is left blank, the queue manager looks for a local queue that matches the queue name provided in the `MQOD`.

## Section 4: Alter program `qnmres` to set the `MQOD` queue manager name to `MQ03`

- \_\_ 24. Open program `qnmres.c` for edit.
- \_\_ 25. Find the `MQOPEN` queue manager name declaration: `OpenQMN`
- \_\_ 26. Alter the initialization value for the `OpenQMN` string from blanks to `"MQ03"`, taking care to type the new value inside the quotation marks. The new line should look as shown:

```
51  char    OpenQMN[50] = "MQ03";           /*Ex3 open qmgr name    */
```

- \_\_ 27. Recompile `qnmres.c` by using the method appropriate for your operating system. If you need help with the compile, refer to the instructions provided earlier in this exercise.

## Section 5: Test the `qnmres` program with the `MQOPEN MQOD` queue name set to `EX3.SAME.QNAME`, and the `MQOPEN MQOD` queue manager name set to `MQ03`

- \_\_ 28. Test the second compilation of `qnmres.c` with value `MQ03` hardcoded in the `MQOD` queue manager name.
- \_\_ a. Type `qnmres EX3.SAME.QNAME MQ01` and press the Enter key.
- \_\_ b. Type `Message 2` and press the Enter key two times to end `qnmres`.
- \_\_ 29. Check the contents of queue `EX3.SAME.QNAME` in both queue managers by using IBM MQ Explorer. **Do not consider the one message in MQ01 for the previous test.** What queue manager did the message resolve to, `MQ01` or `MQ03`? Answer in the information box.
- \_\_\_\_\_.
- \_\_ 30. Stop queue manager `MQ03` by typing: `endmqm MQ03`
- \_\_ 31. Leave the two terminal command windows open.



## Information

The correct result should be `MQ03`. When a queue manager name is provided to the `MQOPEN`, the queue manager looks for a **local transmission queue** that matches the **queue manager name** specified in the `MQOD`.

You completed the first part of this exercise.

In this first section of the exercise, you tested the difference of specifying a queue manager name in the `MQOPEN` call.

In previous exercises, you used a locally defined remote queue to place messages to a remote queue manager. In this lab, you used a transmission queue, with a name that matched the `MQOPEN` supplied queue manager name, to send messages to a remote queue manager without the need of a local remote queue.

## Part 2: Creating dynamic queues

In this part of the exercise, you check the code and change the values that are used in the `MQOD DynamicQName` attribute to create a dynamic queue with different naming options:

- Use an asterisk (“\*”) in the `MQOD DynamicQName` to allow the queue manager to generate the entire unique queue name.
- Use a partial name followed by the “\*” in the `MQOD DynamicQName` to establish a prefix for the first part of the queue name, and allow the queue manager to generate the remainder of the queue name.
- Use a complete queue name **without** an “\*” in the `MQOD DynamicQName` to have the code create the queue name in its entirety.

The program that you check and alter, is another copy of the `amqsput0.c` sample. The parameters that follow the program name, `crtdynq`:

- Provide the queue name `SYSTEM.DURABLE.MODEL.QUEUE` as the first parameter
- Provide the queue manager name `MQ01` as the second parameter
- Alter the `DynQN` character string declaration with the needed prefixes or full queue name

To expedite completion of the exercise, much of the work that is needed to adapt the program to use a hardcoded dynamic queue name was completed. You review the work that was required, and later change the dynamic queue name value.



### Hint

To expedite your work, you can use the up arrow in your workstation to recall repetitive commands in your terminal command prompt.

### Section 6: Review the `crtdynq.c` program source

- \_\_ 32. Open the `crtdynq.c` source for edit.
- \_\_ 33. Search for string `Ex3` to find the declaration that is made for the variable that holds the name of the dynamic queue, `DynQN`. The definition is approximately at line 52.

```
52 char    DynQN[48] = "*";        /* Ex3 part 2 dynamic queues */
```

- \_\_ 34. For the first test iteration, **do not change** to this value.

- \_\_\_ 35. Search for string `Ex3` again to locate the place where the `DynQN` is used to set the `DynamicQName` value in the `MQOD`. This statement should be around line 143.

**You might need to repeat the search more than one time to reach line 143.**

**Ensure that this `strcpy` statement is not commented out in your source.**

```
143 strcpy(od.DynamicQName,DynQN);
```



### Information

The `MQOD DynamicQName` field is set for the first iteration in this exercise.

The name of the model queue to open, `SYSTEM.DURABLE.MODEL.QUEUE`, is provided in the `crtdynq` program invocation parameter.

You now need to check that the queue is deleted in the `MQCLOSE` statement by checking the options that are provided.

- \_\_\_ 36. Do a find for string: `MQCLOSE`
- \_\_\_ 37. Check above the `MQCLOSE` call to ensure that the option to delete the dynamic queue with purge of the messages in it is specified, as shown. This code should be at approximately line 256.

```
256 C_options = MQCO_DELETE_PURGE;
```

- \_\_\_ 38. **If all the expected items are confirmed, close the source file, and compile `crtdynq.c`** by using the compile method appropriate for the operating system that you are using.

## **Section 7: Run the first test of the program by using the option to allow the queue manager to generate the full dynamic queue name**

**This test demonstrates the first iteration on how the queue name is generated.**

- \_\_\_ 39. Run program `crtdynq` by typing as shown and pressing the Enter key **one time only** following the command:

```
crtdynq SYSTEM.DURABLE.MODEL.QUEUE MQ01
```



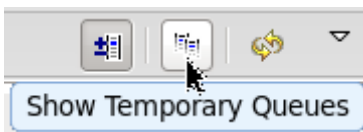
**Stop**

Be **careful not to stop the program by accidentally typing a blank line** until instructed. Since the `MQCLOSE` uses the delete-purge option, the queue goes away when the `MQCLOSE` function runs.

- \_\_ 40. Type some text and **press the Enter key one time; do not press the Enter key again until instructed.**
- \_\_ 41. Open IBM MQ Explorer, and look at the list of queue names for queue manager MQ01. Towards the top of the queue list, look for a queue with a name composed of a series of alphanumeric characters, which are similar but not equal to: `56C39A62206E5602`

**Troubleshooting**

If you cannot see the dynamic queue, ensure that the toggle switch that allows dynamic queues to display in IBM MQ Explorer is set correctly. To check for the icon, check the upper-right corner of IBM MQ Explorer for the icon selected by the arrow.

**Note**

When the `MQOD DynamicQName` field is provided with an "\*", the queue manager generates the entire queue name.

- \_\_ 42. Minimize IBM MQ Explorer.
- \_\_ 43. Press the Enter key to stop `crtdynq`.
- \_\_ 44. After stops, check IBM MQ Explorer again. The dynamic queue should be gone.

### **Section 8: Change and test program `crtdynq` to create a dynamic queue name with a specified prefix**

- \_\_ 45. Open `crtdynq.c` for edit.
- \_\_ 46. Search for the `Ex3` string to locate the dynamic queue name variable `DynQN` towards the start of the source code.
- \_\_ 47. Change the initialization value of the variable from "\*" to "BRIEF.\*". Ensure that you:
  - \_\_ a. Have a period that is followed by an asterisk at the end.

- \_\_ b. Use the quotation marks as shown.
- \_\_ 48. Recompile `crtdynq.c` by using the process appropriate for your operating system.
- \_\_ 49. Run program `crtdynq` by typing:  
`crtdynq SYSTEM.DURABLE.MODEL.QUEUE MQ01`



**Stop**

Be **careful not to stop the program by accidentally typing a blank line** until instructed. Since the `MQCLOSE` uses the delete-purge option, the queue goes away when the `MQCLOSE` is run.

**This test demonstrates the second iteration on how the queue name is generated.**

- \_\_ 50. Type some text and **press the Enter key one time; do not press the Enter key again until instructed.**
- \_\_ 51. Return to one of the command prompt windows and start a `runmqsc` session by typing:  
`runmqsc MQ01`
- \_\_ 52. You know the starting characters for the dynamic queue name. Display the dynamic queue by typing:

```
dis q(BRI*) DEFTYPE
```

Expected results should be similar, but not equal to the results shown:

```
5 : dis q(BRI*) DEFTYPE
AMQ8409: Display Queue details.
        QUEUE(BRIEF.56C39A62206DB802)          TYPE(QLOCAL)
        DEFTYPE(PERMDYN)
```

- \_\_ 53. **Keep the `runmqsc` session open until instructed to end.**



**Note**

Observe the details of this display:

- You see that the queue name was generated by using the provided prefix, along with a random alphanumeric string.
- Look at the type of queue created; while the dynamic queue exists, it is a `QLOCAL`.
- Observe the definition type, or `DEFTYPE` attribute of the queue. The `DEFTYPE` attribute of `PERMDYN`, or permanent dynamic, was inherited from the model queue that was provided to the `MQOPEN` call. In this case, it was `SYSTEM.DURABLE.MODEL.QUEUE`.

\_\_ 54. Still in the `runmqsc` session, type the command shown. Your results are similar, but not identical to the corresponding display:

```
dis q(A*) DEFTYPE
Expected results should be similar, but not equal to the results shown:
  2 : dis q(A*) DEFTYPE
AMQ8409: Display Queue details.
      QUEUE(AMQ.MQEXPLORER.56C39A6220002503)
      TYPE(QLOCAL)                                DEFTYPE(TEMPDYN)
```

\_\_ 55. In the text box that follows, **review the note for this last display**.



### Note

Observe the details of this display:

- This queue is used by your IBM MQ Explorer session.
- The type of queue that is created, as expected, is also a `QLOCAL`.
- It is similar to the previous case, where the queue name is generated by using the provided prefix, along with a random alphanumeric string.
- Notice a difference in the definition type, or `DEFTYPE` attribute of the queue. In this case, the `DEFTYPE` is `TEMPDYN`. The `DEFTYPE` attribute was inherited from the model queue used by the IBM MQ Explorer application in the `MQOPEN` call.
- In the next step, you see how the `DEFTYPE` of the dynamic queue makes a difference when using IBM MQ Explorer.

The IBM MQ Explorer application uses prefix “`AMQ.MQEXPLORER.*`” for the temporary dynamic queue it creates to interface with the queue manager.

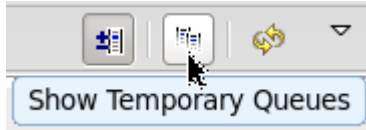
Be aware of one other detail on creating dynamic queues. If you do not provide any value in the `MQOD DynamicQName` field, the value in the `MQOD_DEFAULT` initialization structure precedes the queue (assuming that you used the `MQOD_DEFAULT` structure in your program). The `MQOD_DEFAULT` default value is “`AMQ.*`” for distributed platforms, and “`CSQ.*`” for z/OS.

**The use of the `MQOD_DEFAULT` initial value and omitting the setting of a new value string in the `MQOD DynamicQName` field is the third way of naming a dynamic queue.** This method is similar to using a prefix, but in this case you use the IBM MQ default prefix, “`AMQ.*`”.

In the first two cases, the `MQOD DynamicQName` had a value set. In this third case, the `MQOD DynamicQName` uses the `MQOD_DEFAULT` initial value.

\_\_ 56. Type `end` and press the Enter key to exit the `runmqsc` session.

- \_\_ 57. Open IBM MQ Explorer, and look at the list of queue names for queue manager MQ01. You should see the same queue displayed in the `runmqsc` session, prefixed by string `BRIEF`. This name should be similar to, but with different ending characters from:  
`BRIEF.56C39A62206DB802`
- \_\_ 58. But where is the `AMQ.MQEXPLORER.*` named queue? If you recall, the IBM MQ Explorer queue was a temporary dynamic queue. A toggle switch that is on the upper-right section of IBM MQ Explorer can provide a view of the temporary queues. When you move your cursor over the icon, it should display as shown:



- \_\_ 59. If you cannot see the IBM MQ Explorer temporary dynamic queue, click the `Show Temporary Queues` icon. You should now see the IBM MQ Explorer queue.
- \_\_ 60. Minimize IBM MQ Explorer.
- \_\_ 61. In your `crtdynq` session, press the Enter key to end `crtdynq`.

**Section 9: Alter `crtdynq` and test to use a user-generated queue name**

- \_\_ 62. Open `crtdynq.c` for edit.
- \_\_ 63. Search for the `Ex3` string to locate the dynamic queue name variable `DynQN`.
- \_\_ 64. Change the initialization value of the variable from `"*`", to `"BRIEF.EX3.Q"`. **Ensure that the name you type in does not have a `"*`" at the end.**
- \_\_ 65. Recompile `crtdynq.c` by using the method appropriate to the operating system that you are working with.

***This test demonstrates the third iteration on how the queue name is generated.***

- \_\_ 66. Run program `crtdynq` by typing:

```
crtdynq SYSTEM.DURABLE.MODEL.QUEUE MQ01
```



**Stop**

Be **careful not to stop the program by accidentally typing a blank line until instructed**. Since the `MQCLOSE` uses the delete-purge option, the queue goes away when the `MQCLOSE` is run.

- \_\_ 67. Type some text and **press the Enter key one time; do not press the Enter key again until instructed**.
- \_\_ 68. Return to IBM MQ Explorer and check the queues under MQ01.
- \_\_ 69. You should find queue name `BRIEF.EX3.Q`.
- \_\_ 70. In your `crtdynq` session, press the Enter key to end `crtdynq`.

- \_\_\_ 71. Return to IBM MQ Explorer and look for the queue now. `BRIEF.EX3.Q` should no longer appear. You might need to press the Refresh to get the updates reflected in IBM MQ Explorer.

**Proceed to Part 3.**

## Part 3: Using message expiry and report options

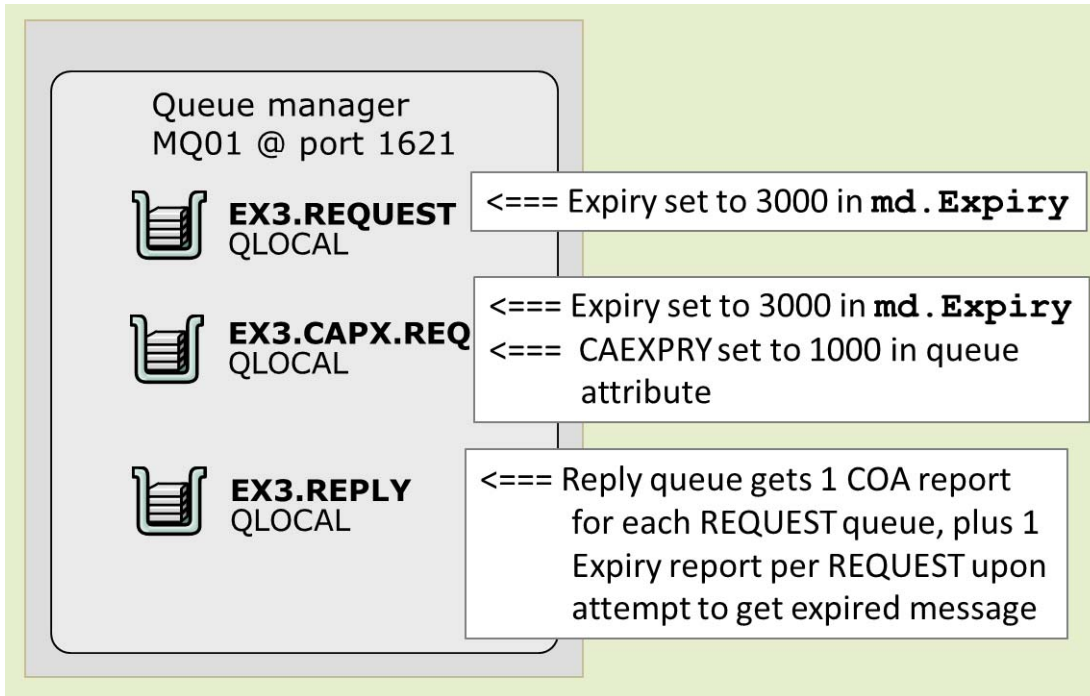


Figure 3-2. Request queues and reply queue for Exercise 3

In this exercise, you request report options for confirm-on-arrival with data, and you request an expiry report.

### ***The request and reply queues are predefined.***

You use two local queues to put requests, and one queue to receive COA report messages:

- Queue `EX3.REQUEST` has no special attributes set in its object definition. **You set an expiry of 30 seconds in your code.**
- Queue `EX3.CAPX.REQ` is defined with a 1-minute expiry. For messages placed in queue `EX3.CAPX.REQ`, since the queue `CAEXPRY` has a smaller value than the expiry set in the code, **the `CAEXPRY` 1-minute expiry value overrides the 3 minutes in the code.** The capped message expiry feature works **opposite** to other MQI features, where the code attribute overrides the queue parameter specification for the same attribute. In this case, if the defined capped message attribute is less than the coded Expiry attribute, the object attribute overrides the coded MQI attribute.
- You see a COA report message that arrives on the `EX3.REPLY` queue for each of the request queues (`EX3.REQUEST` and `EX3.CAPX.REQ`).
- The expiry report message is sent to the `EX3.REPLY` queue when you attempt to browse the message in either of the request queues after the expiry time is reached.
- The message expires upon the first browse or get of the queue after the expiration period elapses.

**Important**

The time and date time of the expiry report message are associated with the time at which you did the browse or get.

If you subtract the COA report put time from the expiry report time stamp, the results should be close to the expiry period. How close depends on how quickly after the end of the expiry time you browsed or got the message.

**Note**

The capped message expiry feature is available with the IBM MQ V8.0.0.4 distribution and later. Refer to IBM Knowledge Center for z/OS availability requirements.

You start by working on the source code.

### ***Section 10: Update program `exprep.c` (copied from `putmsg.c`) to set expiry and request a COA and an expiry report.***

- \_\_ 72. If you are not at the `<dir>/LabFiles/Lab3/source` directory for the operating system you are using, move to that directory.
- \_\_ 73. Copy `putmsg.c` source to a new file called `exprep.c`. `putmsg` is your backup source for this part of the exercise.
- \_\_ 74. Open `exprep.c` for edit.
- \_\_ 75. Find string `pass` (for password) in the `exprep.c` source. You should be approximately in line 54 of the code.
- \_\_ 76. Position your cursor to type the next line below the declaration for `QMName[50]`.
- \_\_ 77. Add the declaration for a 50-character array field that is named `ReplyQName[50]` and set its initial value to `EX3.REPLY` as shown. **Do not include the line number**; it is shown for reference.

```
55 char    ReplyQName[50] = "EX3.REPLY"; /* Ex3 reply queue          */
```

- \_\_ 78. Save the `ReplyQName` declaration in your code.
- \_\_ 79. Find string `md.Msg` by using the exact case in the string. The string should be approximately in code line 238, right before the `MQPUT`.

- \_\_ 80. Alter the program to:
- \_\_ a. Add comments to indicate that these changes are from Exercise 3.
  - \_\_ b. Set the `md.Expiry` field to 3000.
  - \_\_ c. Request a confirm on arrival report with data.
  - \_\_ d. Request an expiry report.

An example of the required code is shown in the text box that follows this step.

**Ensure that you type the changes to the `MQMD` fields directly above the `MQPUT` to mitigate risk of having them reset in other parts of the code.**



### Example

```

241  /* Ex3 md changes for report and expiry */ <= Your code
242      md.Expiry = 3000;
243      md.Report = MQRO_COA_WITH_DATA | MQRO_EXPIRATION;
244      strncpy(md.ReplyToQ, ReplyQName, MQ_Q_NAME_LENGTH);
245  /* end of Ex3 changes */ <= Your code
246
247      MQPUT(Hcon, * connection handle */ <= Existing code

```

- \_\_ 81. Compile the program by typing:



### Linux

```
gcc -I/opt/mqm/inc -lmqm -o exprep exprep.c
```



### Windows

- Ensure that you are at a Developer Command Prompt, and that you are positioned at directory `C:\LabFiles\Lab3\source` in the Developer Command Prompt.
- Type:

```
cl exprep.c C:\PROGRA~1\IBM\WEBSPH~1\Tools\Lib\mqm.lib
```



## Section 11: Review, set up, and test

Your test consists of putting one message in each of the request queues, `EX3.REQUEST` and `EX3.CAPX.REQ`, and then checking as the reports arrive in the reply queue, `EX3.REPLY`. The COA reports should arrive shortly after you put them to each queue. The expiry reports arrive after you attempt to get the expired messages.

If you take care to put one message to each request queue, you should have four reply messages at the end of the test: two COA reports and two expiry reports. If you inadvertently put extra messages, you receive one extra COA report per message put, and one extra expiry report for the first attempt to get an expired message.

- \_\_ 82. Open a `runmqsc` session for queue manager `MQ01` from a command terminal window and display queue `EX3.CAPX.REQ` to review the `CAPEXPY` setting by typing:

```
dis q(EX3.CAPX.REQ) custom
```

Expected results are:

```
1 : dis q(EX3.CAPX.REQ) custom
AMQ8409: Display Queue details.
        QUEUE(EX3.CAPX.REQ)                TYPE(QLOCAL)
        CUSTOM(CAPEXPY(1000))
```

- \_\_ 83. Review the results, **type *end* and press the Enter key to exit `runmqsc`**. Notice that `CAPEXPY` is a name-value pair of the `CUSTOM` attribute, available with IBM MQ V8.0.0.4 and later.
- \_\_ 84. Set up your IBM MQ Explorer in a way that you can easily view the `EX3`-named queues. You can use the scroll bar to the right of the queue list for queue manager `MQ01`, as shown:

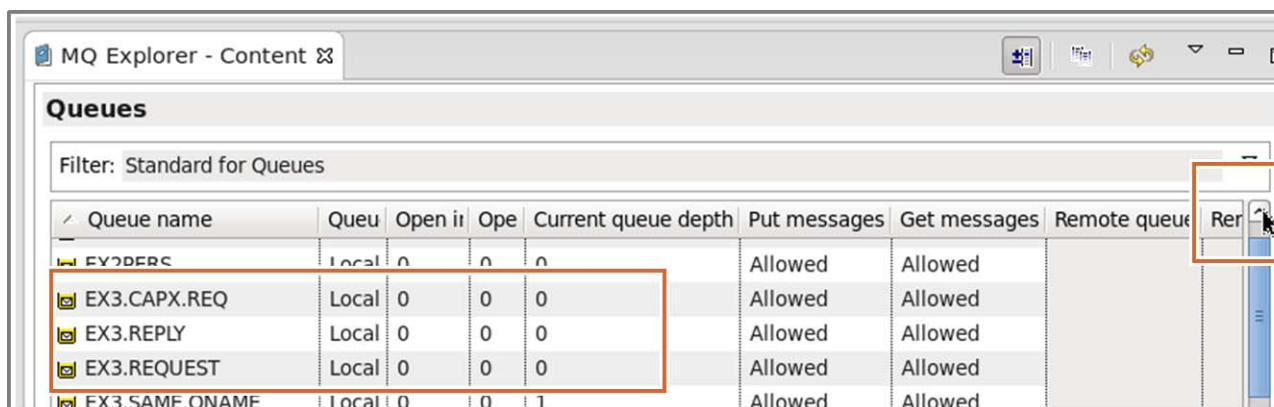


Figure 3-3. IBM MQ Explorer setup for report message testing

**Note**

Program `exprep.c` is a copy of `putmsg.c`. To use the program, you start like `putmsg.c`, by:

- Providing the program name, target queue, and queue manager name in one line and pressing the Enter key
- Typing one line and pressing the Enter key one time to send the message
- Pressing the Enter key on an empty line to end the program

**You did not change the program name**, so upon execution and ending, it is expected that you see “Lab source putmsg start” and “Course PUTMSG end”. **Leave the display names of the program unchanged.** The focus of this exercise is to test the capabilities that are used.

**All work in this section uses queue manager MQ01.**

**Warning**

When you use `exprep.c` to put your messages, **you start the expiry timer.**

**Work expeditiously to browse the messages before the expiry period runs out.**

- You have 3 minutes for the message in queue `EX3.REQUEST` (your coded value).
- You have 1 minute for queue `EX3.CAPX.REQ` (the queue `CAPEXPY` value).

## **Section 12: Put messages in queue `EX3.CAPX.REQ` and observe the expiration behavior when the lowest expiry value is set on the queue definition**

- \_\_\_ 85. Maximize the IBM MQ Explorer window so you can see all three queues used in this part of the exercise near the top of the queue list.
- \_\_\_ 86. Use a command terminal session to put one message to queue `EX3.CAPX.REQ` by typing `exprep EX3.CAPX.REQ MQ01` and pressing the Enter key one time.
- \_\_\_ 87. Type `messagea` and press the Enter key two times to end the program.
- \_\_\_ 88. Look at IBM MQ Explorer. Under queue manager `MQ01`, you should see one message in queue `EX3.CAPX.REQ` and one COA report message in queue `EX3.REPLY`.
- \_\_\_ 89. Right-click queue `EX3.CAPX.REQ` and select **Browse messages**.
- \_\_\_ 90. Double-click the one message, and look at the Expiry field. You see that it is decreasing from the initial 1000 set by the `CAPEXPY` value.
- \_\_\_ 91. Close the Message properties pane and the Message browser pane.



### Important

You need to repeat the following step quickly so that you get a time stamp in the report message that is close to the actual expiry for this message. The time stamp in the expiry report is the time stamp that you last browsed before the expiry timer ran out. You want to get this close to the actual expiry time.

- \_\_\_ 92. Repeat the process to browse queue `EX3.CAPX.REQ` until the Browse messages pane shows that no messages are available for browsing, as shown:

There are no messages available for browsing. Press the refresh button for new messages.

- \_\_\_ 93. After you see the information that no more messages are available for browsing, close message browser.
- \_\_\_ 94. Look at the count of messages. The count should have 0 messages in queue `EX3.CAPX.REQ`. The count should have two report messages in queue `EX3.REPLY`: one COA report message and one expiration report message.
- \_\_\_ 95. Browse queue `EX3.REPLY`. You should see two messages. Review the time stamps of the messages. If you browsed the queue frequently as instructed, the time difference between the two time stamps (one from the COA when the message was put, the other from the expiry when the message expired) should be approximately 1 minute apart, as shown:

| Position | Put date/time           | Use |
|----------|-------------------------|-----|
| 1        | May 31, 2016 3:20:08 PM | Adr |
| 2        | May 31, 2016 3:22:18 PM | Adr |

- \_\_\_ 96. Close the Message browser pane.
- \_\_\_ 97. Clear the messages in the `EX3.REPLY` queue by right-clicking **EX3.REPLY** and selecting **Clear messages**.
- \_\_\_ 98. Select **Clear** to proceed and then close the confirmation pane.



### Information

In the first test, your code requested a 3-minute expiry. The `CAPEXPY` value of queue `EX3.CAPX.REQ` specified a 1-minute expiry. This test confirmed how the queue `EX3.CAPX.REQ` `CAPEXPY` value superseded the Expiry value in your code.

If the `CAPEXPY` value of queue `EX3.CAPX.REQ` is higher than the Expiry value in the code, then the Expiry value in the code would be used.

**Note**

If you put more than one message per request queue, and returned to browse both queues after the expiry time elapsed, the reply queue should have extra messages.

### **Section 13: Put messages in queue `EX3.REQUEST` and observe the expiration behavior when an expiry value is not set on the queue definition**

- \_\_\_ 99. Put one message to queue `EX3.REQUEST` by typing `exprep EX3.REQUEST MQ01` and pressing the Enter key one time.
- \_\_\_ 100. Type `messageb` and press the Enter key two times to end the program.
- \_\_\_ 101. Look at IBM MQ Explorer. You should see one message in queue `EX3.REQUEST` and one message (the COA report message) in queue `EX3.REPLY`.
- \_\_\_ 102. Use IBM MQ Explorer to browse the message at queue `EX3.REQUEST`.
- \_\_\_ 103. Double-click the message, and use the scroll bar to see the decrease in the Expiry time.
- \_\_\_ 104. Close the Message browser pane.

**Note**

If you want to see the Expiry timer decrease, you must exit the Message browser pane and repeat the browse messages.

- \_\_\_ 105. Continue to exit the Message browser pane, and repeat the browse messages process for queue `EX3.REQUEST` at 15-second intervals until you see the “no messages available for browsing” message in the Browse messages pane.
- \_\_\_ 106. Look at the count of messages. You should see 0 messages in queue `EX3.REQUEST`. You should see two report messages in queue `EX3.REPLY`: one COA report message and one expiration report message.
- \_\_\_ 107. Browse queue `EX3.REPLY`. You should see two messages. Review the time stamps of the messages. If you browsed the queue frequently as instructed, the time difference between the two time stamps (one from the COA when the message was put, the other from the expiry when the message expired) should be approximately 3 minutes. The time difference might be higher depending on the frequency in which you repeated the browse of the `EX3.REPLY` queue.
- \_\_\_ 108. Close the Browse messages pane.

**Information**

In the second test, your code requested a 3-minute expiry, and no `CAPEXPY` value was set on the queue. This test confirms the behavior of traditional message expiry, without a capped expiry limit set on the queue.

## Section 14: Review the report messages by using the `amqsbcg` formatted browse sample

**Note**

Most application developers have access to IBM MQ Explorer. However, in case you are not able to use IBM MQ Explorer, this section shows you how to look at the report messages by using the `amqsbcg` sample program in a command line.

\_\_ 109. Leave IBM MQ Explorer open.

\_\_ 110. Return to a command terminal prompt and browse all the messages from the reply queue into a file by typing the command shown and then pressing the Enter key:

```
amqsbcg EX3.REPLY MQ01 > reports.txt
```

\_\_ 111. Browse the **first** message of the `reports.txt` file and locate the formatted Message Type (MsgType) and Feedback fields for the first message. Write them here as you need them for the next steps:

Feedback: \_\_\_\_\_ MsgType: \_\_\_\_\_

\_\_ 112. If you started with an empty `EX3.REPLY` queue, the first report is expected to be a COA report. You requested to get the first 100 bytes of the message. Scroll down to the end for the first formatted message, and see your message data. The data is displayed in both hexadecimal and text formats.

\_\_ 113. Browse the **last** message of the `reports.txt` file and locate the formatted Message Type (MsgType) and Feedback fields for the last message. Write them here as you need them for the next steps:

Feedback: \_\_\_\_\_ MsgType: \_\_\_\_\_

\_\_ 114. While still on the formatted MQMD for the last report message, locate the origin context and capture Put Application Type and Put Application Name in the Origin Context section. Record the names:

PutApplType: \_\_\_\_\_ PutApplName: \_\_\_\_\_

\_\_ 115. Close the `reports.txt` file.

- \_\_ 116. You might remember the meaning of some of the values you recorded from the lecture material. However, you confirm now by looking up the values. **Open the cmqc.h header file** at the platform-dependent directory.
- \_\_ 117. Do a find for string "Message Types". If you recorded 4 in your review of the report messages, you confirm that 4 in the message type is for MQMT\_REPORT.
- \_\_ 118. Find string "Feedback Values". The information that you recorded should match as follows:
  - 259 is MQFB\_COA
  - 258 is MQFB\_EXPIRATION
- \_\_ 119. Search for string "Application Types".
  - The application type 7 is MQAT\_QMGR, which confirms that the queue manager sent the report messages.
  - The application name should be MQ01, the queue manager you are using.

**Section 15: Review the report messages by using IBM MQ Explorer**

- \_\_ 120. Return to IBM MQ Explorer.
- \_\_ 121. Right-click queue EX3.REPLY and select **Browse Messages**. The Message Browser pane is displayed:



**Note**

You should see two messages in your display; disregard the extra messages in this figure.

| Message browser          |                         |                 |                      |        |
|--------------------------|-------------------------|-----------------|----------------------|--------|
| Queue Manager Name: MQ01 |                         |                 |                      |        |
| Queue Name: EX3.REPLY    |                         |                 |                      |        |
| Position                 | Put date/time           | User identifier | Put application name | Format |
| 1                        | Mar 2, 2016 11:46:34 AM | mqadmin8        | MQ01                 | MQSTR  |
| 2                        | Mar 2, 2016 11:46:58 AM | mqadmin8        | MQ01                 | MQSTR  |
| 3                        | Mar 2, 2016 11:48:45 AM | mqadmin8        | MQ01                 | MQSTR  |
| 4                        | Mar 2, 2016 11:48:45 AM | mqadmin8        | MQ01                 | MQSTR  |

Figure 3-4. IBM MQ Explorer Message browser

- \_\_ 122. Double-click any of the messages. The information in IBM MQ Explorer shows the field names in a descriptive manner, rather than showing the actual field name.
- \_\_ 123. The General menu shows the type of message, which is spelled out as Report.

\_\_ 124. From the left menu, select **Report**, and the type of report is displayed.

\_\_ 125. From the left menu, select **Context**. Look at the application type and application name.

***You completed Exercise 3.***

**End of exercise**

## Exercise review and wrap-up

In this exercise, you learned to:

- Code different combinations of queue manager and queue name in the object descriptor to test and confirm how queue name resolution takes place
- Code report options in an MQPUT call and review the results in the reply-to queue
- Create and display a dynamic queue by letting the queue manager specify the name
- Create and display a dynamic queue by specifying a partial prefix of the queue name
- Create and display a dynamic queue by specifying the exact queue name
- Request a confirm-on-arrival with data report message
- Set the expiry attribute in a message and request an expiry report
- Set the expiry attribute in a message for a queue with a lower expiry cap value in the queue definition
- Contrast the expiry behavior when the capped message expiry attribute is set on a queue



## Exercise 4. Correlating requests to replies

### What this exercise is about

In this exercise, you learn how to work with the common task of correlating a reply message with a request message.

### What you should be able to do

After completing this exercise, you should be able to:

- Code or modify an application to generate a confirm-on-arrival (COA) report message that preserves the original message identifier (MsgId)
- Code or modify an application reply to a request message by setting up the correlation identifier of the reply message to the message identifier of the request message
- Use a formatted message descriptor display to check the correct setting of your message and correlation identifier fields
- Alter or code an application to get a reply message from a queue with a correlation identifier that matches the message identifier of its corresponding request message

### Introduction

Earlier in this course you learned:

- How to get messages in browse, or non-destructive mode, which leaves the messages in the queue
- How to get messages in destructive mode, where the MQGET removes the messages from the queue
- To use programs to get messages from a queue in first-in, first-out order
- How to set a wait for the MQGET

In this exercise, you learn how to select a specific message from a queue, a common task when developing IBM MQ applications.

### Requirements

- Queue manager MQ01
- Basel source modules `ex4req.c`, `ex4repl.c`, and `ex4match.c` at `<dir>/LabFiles/Lab4/source`
- Predefined local queues `EX4REQ`, `EX4REPT`, and `EX4REPL`

## Exercise description



### Important

Review this description carefully before commencing work on this exercise.

A common task when developing IBM MQ applications is the need to correlate a request message with its reply message. Many scenarios exist where messages need to be correlated:

- A user sends a request and waits for the message; that is, the same application that sent the request waits and processes the reply.
- A requester does not wait for a message, but rather sends a reply, and a later application checks for responses. In this case, a method is necessary to keep the correct information that needs to be used to match the reply to the request.

This exercise takes the second approach; the replies are correlated in a different application from the request. You work with three applications and three queues, as depicted in the diagram.

The requests and replies consist of partial names of football clubs. You send the first part of the name and receive the second part of the name as a reply. For purposes of the lab, the valid requests are limited to four teams, detailed later.

***In this exercise, you modify three applications.***

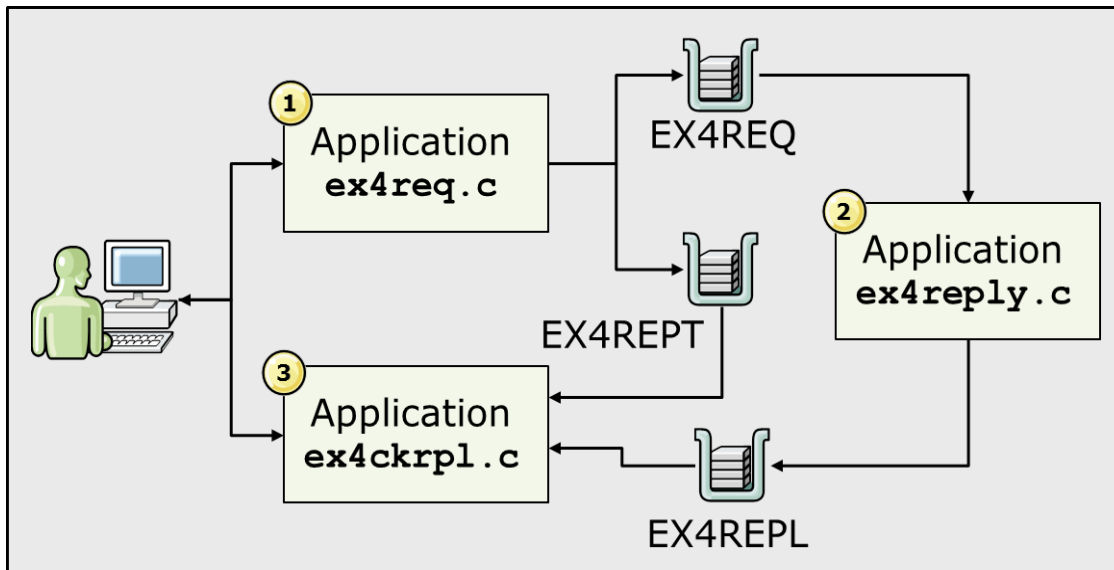


Figure 4-1. Architecture of Exercise 4

1. Program `ex4req.c` takes the queue name and queue manager name as input parameters and takes the message contents from text you type on the terminal.
  - To get a valid reply, the text that is typed ***must be limited*** to the first part of selected names of football teams.

- The queue name that is provided **must be** `EX4REQ` and the queue manager `MQ01`. It works similarly to `putmsg`, and requires a blank line to signal an end to the program.
  - `ex4req.c` also generates a confirm-on-arrival report message to track the message identifier of the request in queue `EX4REPT`. It ensures that the message identifier of the report message is the same as the message identifier of the original request message.
  - It sends one message per line typed.
  - **The valid input per line that is typed must be** either `ASTON`, `BAYERN`, `ORLANDO`, or `REAL`.
2. Program `ex4repl.c` gets messages from queue `EX4REQ`, checks the input text in the message, and matches `ASTON` with `VILLA`, `BAYERN` with `MUNICH`, `ORLANDO` with `CITY`, and `REAL` with `MADRID`.
- The program sends the reply, which consists of the matched text, as a message to queue `EX4REPL`.
  - This program copies the `MsgId` field of the request message to the `CorrelId` field of the reply message.
3. Program `ex4match.c` gets messages from queue `EX4RPT`, where a report copy of the original request message that contains the original `MsgId` value is kept. The program then uses the saved `MsgId` in the report message to get the reply message from queue `EX4REPL` by `CorrelId`.



### Warning

The design of the applications in this exercise is for the sole purpose of reinforcing coding details such as preservation or generation of new `MsgID` and `CorrelId` fields in report messages. Other actions that are taken by the three applications in this exercise also concentrate on showing specific points: for example, the convention of moving the `MsgID` of the request message into the `CorrelId` of the reply message for later retrieval.

The design of the three applications is not intended to show robustness. They are kept simple to demonstrate the topics that are tested.

***No part of this code is intended to denote a “best practice”.***

## Exercise instructions

### Platform-dependent notes

Repeating platform-dependent tasks are detailed in Exercise 1. Some of the tasks are repeated one time for convenience. If you need to review details that are repeated frequently, refer to Exercise 1.



#### Linux

---

##### **Compiling code (repeated from Exercise 1)**

- In the course environment, the code is the `/home/mqadmin8/LabFiles/Lab#/source` directory, where `#` is a placeholder for the number of the exercise you are working on.
- The course environment has the path set for the IBM MQ libraries. To compile programs, move to the code directory and type:

```
gcc -I/opt/mqm/inc -lmqm -o executable_name program_name.c
```

where `program_name.c` is the source and `executable_name` is the name of the executable program.

##### **Making a backup copy of the source**

- Ensure that you are in the `/home/mqadmin8/LabFiles/Lab#/source` directory
- Type `cp sourcename.c sourcename.c.bkp` and press the Enter key, where `sourcename` is the name of the program.



#### Windows

---

##### **Compiling code (repeated from Exercise 1)**

- In the course environment, the code is in directory `C:\LabFiles\Lab#\source` where `#` is the number of the exercise with which you are working.
- Open a visual studio command prompt by going to **Start > Programs > Visual Studio 2015 > Visual Studio Tools > Developer Command Prompt for VS2015**.
- Change to the `C:\LabFiles\Lab#\source` directory.
- File `copy_edit.txt` at `C:\LabFiles\Lab#\source` can be used to copy and paste the compile command. You need to replace the `program_name.c` placeholder with the name of your C source.
- Type the command as shown, or change and use the `copy_edit.txt` file to paste the compile command:

```
cl source_name.c C:\PROGRA~1\IBM\WEBSPH~1\Tools\Lib\mqm.lib
```

- **Leave the Developer Command Prompt window open so you can reuse it. These instructions are not repeated in their entirety!**

**Making a backup copy of the source**

- Ensure that you are in the correct `/home/mqadmin8/LabFiles/Lab#/source` directory.

Type `copy sourcename.c sourcename.c.bkp` and press the Enter key, where `sourcename` is the name of the program.

**Modify application ex4req.c****Reminder**

Remember to always back up the exercise source code before you change any code.

**Section 1: Back up application ex4req.c**

1. You should be in directory `<dir>/LabFiles/Lab4/source`. Use the method appropriate for the operating system you are using to make a copy of source `ex4req.c` by copying the source to `ex4reqBkp.c`.

**Section 2: Review application ex4req.c**

2. After you back up `ex4req.c`, open application `ex4req.c` for edit by using your application-dependent editor.
3. Locate the first occurrence of label “Exercise 4” and review the hardcoded queue and queue manager name to put the COA report messages. You should see:

```
/* Exercise 4 */
char ReplyQueueName[] = "EX4REPT";
char ReplyQMName[] = "MQ01";
```

**Reminder**

Queue `EX4REPT` is where you keep a copy of the message, to later obtain the `MsgId` value to match to your reply `CorrelId`.

### Section 3: Type the code necessary to create a report message and preserve the original message `MsgId` and `CorrelId` in the report message

- \_\_ 4. Locate the next “Exercise 4” marker. You should be right above comment “Type your code here”, and see the `MQPUT` function call as shown. Scroll to the bottom of the next step for an example of the code.

```
{
  /* Exercise 4 */
  /* Type your code here */

  MQPUT(Hcon,                               /* connection handle */

```

- \_\_ 5. Type the code necessary to achieve the objectives that are detailed in the following steps. **An example of the code that is required is displayed** in the box that follows this set of steps and substeps. **Do not copy and paste the code directly from the Exercises guide**, as you might introduce special characters that cause problems.
- \_\_ a. Set the `ReplyToQ` in the `MQMD` by using the `ReplyQueueName` variable that was reviewed in an earlier step. This variable represents the queue where the report message is sent.
  - \_\_ b. Set the `ReplyToQMGr` in the `MQMD` by using the `ReplyQMName` variable that was reviewed in an earlier step. This variable represents the queue manager where the report message is sent.
  - \_\_ c. Request a confirm on arrival (COA) Report message with full data. Ensure to preserve the `MsgId` and `CorrelId` of the original message in the report message.
  - \_\_ d. Set the `MQMD` message type (`MsgType`) to be a request message by using the `MQMT_REQUEST` named variable.



## Example

```
strncpy(md.ReplyToQ, ReplyQueueName, (size_t)MQ_Q_NAME_LENGTH);
strncpy(md.ReplyToQMgr, ReplyQMName,
(size_t) MQ_Q_MGR_NAME_LENGTH);
md.Report = MQRO_COA_WITH_FULL_DATA |
MQRO_PASS_MSG_ID |
MQRO_PASS_CORREL_ID;
md.MsgType = MQMT_REQUEST;
```

## Section 4: Compile and test the ex4req application

- \_\_\_ 6. Compile application `ex4req.c` by using the method appropriate for the operating system you are using for the labs. **Name the executable: `ex4req`**



### Linux

```
gcc -I/opt/mqm/inc -lmqm -o ex4req ex4req.c
```



### Windows

- Ensure that you are at a Developer Command Prompt, and that you are positioned at directory `C:\LabFiles\Lab4\source` in the Developer Command Prompt.
- Type:

```
cl ex4req.c C:\PROGRA~1\IBM\WEBSPH~1\Tools\Lib\mqm.lib
```

- \_\_\_ 7. Test `ex4req` by typing the name of the module, followed by the request queue name, `EX4REQ`, queue manager name `MQ01`, and typing the Enter key one time. An example is shown in the next box.

**Do NOT type any messages yet. Proceed to the next step after you reach the “target queue us `EX4REQ`” message.**

```
ex4req EX4REQ MQ01
```

Expected reply: **⏪ Do not press the Enter key!**

```
Program ex4req start  
target queue is EX4REQ
```



### Reminder

Each message input to `ex4req` must be **one of** `ASTON`, `BAYERN`, `ORLANDO`, or `REAL`. Any other text that is typed returns an “Invalid team name requested” response later in this exercise. `ex4req` **does not** validate the data that is typed.

- \_\_\_ 8. Type one message, by using any one of the four documented names. Type **exactly** `ASTON` for the first message. Press the Enter key two times to end `ex4req`.

```
target queue is EX4REQ  
ASTON
```

```
Program ex4req end
```

## Section 5: Confirm that the `MsgId` of the request and report messages match

- \_\_\_ 9. Use the `amqsbcg` sample to format the `MQMD` of **both** the `EX4REQ` and `EX4REPT` queues, and pipe the display to show the `MsgId` line. You can accomplish this task as shown in the following text boxes, depending on the operating system you are using. You type the command two times, one time for `EX4REQ` and then the second time for `EX4REPT`.
- \_\_\_ 10. Check that `MsgId` fields match. If they match, proceed to the next step. If they do not match, check your code.
- \_\_\_ 11. The `CorrelId` is expected to be zeros and does not need to be checked now. It is checked later in last program of this exercise.



**Linux**

```
$ amqsbcg EX4REQ MQ01 | grep MsgId
```

Expected result:

```
MsgId : X'414D51204D5130312020202020202020202087261E5702950A20' <==Check
```

```
$ amqsbcg EX4REPT MQ01 | grep MsgId
```

Expected result

```
MsgId : X'414D51204D5130312020202020202020202087261E5702950A20' <==Check
```

```
C:\>amqsbcg EX4REQ MQ01| find /n "MsgId"
```

Expected reply:

```
[17] MsgId : X'414D51204D5138342020202020202020202393265720002205' <==Check
```

```
C:\>amqsbcg EX4REPT MQ01| find /n "MsgId"
```

Expected reply:

```
[17] MsgId : X'414D51204D5138342020202020202020202393265720002205' <==Check
```

**Warning**

While the message ID in your message is different from the message ID shown in the display, it is expected that the `MsgId` value in **your** corresponding EX4REQ and EX4REPT messages match. If these fields do not match, program `ex4match.c` **does not find any matching replies** in the last part of this exercise. It is important that these `MsgId` values are carefully compared.

## Modify application ex4repl.c

### Section 6: Backup application ex4repl.c

- \_\_\_ 12. Use the method appropriate for the operating system you are using to make a backup copy of source `ex4repl.c` by copying the source to `ex4replBkp.c`.

## Section 7: Put a few messages to queue EX4REPL



### Important

Why is this step necessary? It is not, but you want to ensure that you are selecting the right reply from among several messages, not just the next message. If you have one reply in the queue, you get the next message, and you do not prove that you selected a specific message.

- \_\_\_ 13. Use sample program `amqspout` to send three messages to queue `EX4REPL`. Sample `amqspout` was the base for `putmsg` and works the same; you provide the queue name and queue manager name, then a blank line to stop sending messages. It does not matter what text is in the messages. Remember to press Enter two times after the last message typed to end the `amqspout` application. An example `amqspout` run is shown in the next box.



### Example

```
amqspout EX4REPL MQ01
```

Expected reply and input:

```
Lab source putmsg start
target queue is EX4REPL
decoy msg 1
decoy msg 2
decoy msg 3
```

```
Sample AMQSPUT0 end
```

## Section 8: Review the base `ex4repl.c` source

- \_\_\_ 14. Make a backup copy of source `ex4repl.c`.
- \_\_\_ 15. Find the first occurrence of string "Exercise 4" in the `ex4repl.c` source; it should be at approximately line 62.
- \_\_\_ a. In this segment of the code, you see the declarations that are made for this program.
- \_\_\_ b. Write down the name of the queue and queue manager name **variables** that you need to set the fields for the `MQPUT` later in this exercise.
- Queue manager variable name: \_\_\_\_\_
- Reply queue variable name: \_\_\_\_\_
- \_\_\_ 16. Directly below the declarations, find the `MQCONN` call for this program. Subsequent IBM MQ function calls use the same connection handle, `Hcon`.

Repeat the search for string “Exercise 4”. This section sets up the MQOPEN options. **Some of the options are updated “left over” from the original `amqsecha.c` code.** Remember, line numbers that are shown in the examples are **approximate**.

- \_\_ a. Remember that structures have a version number; if you overlook updating the structure’s version number, the features that are supported at the higher version number are ignored. The `MQMO_NONE` option is available with `MQGMO` structure version 2 and higher.
- \_\_ b. The remaining options were kept from the original code sample, but the wait interval was changed to 0. For purposes of this lab, the remaining `MQGMO` options are not needed.

```

133  /* Exercise 4 - review only section */
134
135  gmo.Version = MQGMO_VERSION_2;      /* Avoid need to reset Message */
136  gmo.MatchOptions = MQMO_NONE;      /* ID and Correlation ID after */
137                                     /* every MQGET */                      */
138  gmo.Options = MQGMO_ACCEPT_TRUNCATED_MSG
139          | MQGMO_CONVERT              /* receive converted messages */
140          | MQGMO_WAIT                 /* wait for new messages */
141          | MQGMO_NO_SYNCPOINT;       /* No syncpoint */
142  gmo.WaitInterval = 0;               /* no need to wait */

```

### Section 9: Make necessary additions to `ex4repl.c`

- \_\_ 17. Repeat the find of string “Exercise” until you find the section with a comment that starts with “Type your code ...”
- \_\_ 18. Type the four lines of required code in the section marked. **A sample of the code that is required is shown in the next text box.**
  - \_\_ a. Set the `MQMD` queue and queue manager name as required for the `MQPUT1` call to respond to the reply queue.
  - \_\_ b. Set the `MQMD` field for the `MQPUT1` call so that no report messages are sent from `ex4repl.c`.
  - \_\_ c. Set the `MQMD` message type to indicate that this message is a reply message.

```

/* Exercise 4 */
    /* Type your code to copy queue and queue manager name here */
    /* Type code to disable sending any Report messages */

strncpy(odR.ObjectName, ReplQName, MQ_Q_NAME_LENGTH);
strncpy(odR.ObjectQMgrName, QMgrName, MQ_Q_MGR_NAME_LENGTH);
md.Report = MQRO_NONE;
md.MsgType = MQMT_REPLY;

```

- \_\_ 19. Repeat the search for string “Exercise 4”, or slowly scroll down until you find a section with a comment that indicates “copy the incoming MQMD MsgId to the outgoing MQMD CorrelId.” Type code to do the actions in the substeps. An example of the code can be found in the text box directly below the substeps.
- \_\_ a. Copy the message identifier of the incoming message to the correlation identifier of the outgoing message.
  - \_\_ b. Clear the message identifier of the outbound message to ensure that all messages have a unique message identifier.
  - \_\_ c. It is assumed that you know to omit the line number.

```

225     /* Exercise 4 */
226     /* Type your code to copy the incoming MQMD MsgId to the */
227     /* outgoing MQMD CorrelId. Clear the outgoing MsgId      */
228
229     memcpy(md.CorrelId, md.MsgId, sizeof(md.MsgId));
230     memcpy(md.MsgId, MQMT_NONE, sizeof(md.MsgId));

```

- \_\_ 20. Save your changes; leave your source file open.



### Important

You are now done with your code for `ex4repl.c`. **Do not change any other part of the `ex4repl.c` application.**

## Section 10: Baseline what messages you have now



### Information

**It is not imperative** to track the number of messages in each queue, but you might find it helpful to do a baseline **before you run** `ex4repl`. (An example that shows two ways to see the number of messages in each queue follows this box.)

**Before you run** `ex4repl`, you should have:

- One request message in queue `EX4REQ` (unless you typed more than one request, in which case each message typed has one request message)
- One report message in queue `EX4REPT` (unless you typed more than one request, in which case each request message typed has one report message)
- The number of messages you put by using `amqspout` in queue `EX4REPL` (these messages are throw-away messages later; none of them match any of the request messages)

**After you run** `ex4repl`, there should be:

- No messages in queue `EX4REQ` (`ex4repl` does a destructive `MQGET`)
- Same number of messages in `EX4REPT` as before you ran `ex4repl`
- One extra message in queue `EX4REPL` as compared to before you ran `ex4repl` (unless you typed more than one request, in which case each request message typed earlier has one reply message)



### Hint

#### **Method 1:**

Use the `runmqsc` utility.

**runmqsc MQ01 <== you type**

... Starting MQSC for queue manager MQ01.

**dis q(EX4R\*) curdepth <== you type**

```

1 : dis q(EX4R*) curdepth
AMQ8409: Display Queue details.
      QUEUE(EX4REPL)                TYPE(QLOCAL)
      CURDEPTH(3)
AMQ8409: Display Queue details.
      QUEUE(EX4REPT)                TYPE(QLOCAL)
      CURDEPTH(1)
AMQ8409: Display Queue details.
      QUEUE(EX4REQ)                 TYPE(QLOCAL)
      CURDEPTH(1)
end
2 : end <== you type
```

**Method 2:**

Use IBM MQ Explorer, and look at the Queues view for queue manager MQ01.

**Section 11: Compile and test the ex4repl application**

- \_\_ 21. Compile source `ex4repl.c` to executable `ex4repl` by using the instructions appropriate for the operating system you are using.
- \_\_ 22. The `ex4repl` application does not require any parameters. You type `ex4repl` in your terminal command screen. The test and expected reply as shown in the box assume that you typed one request message.

**ex4repl**

```

Program ex4repl start
Connecting to queue manager MQ01
Opening queue EX4REQ
Incoming message is: <ASTON>
Reply returned is <VILLA>
All EX4REQ messages processed
Program ex4repl end
```

- \_\_ 23. If your results resemble the display, proceed to the next step. If your results are not as expected, resolve the problem before you continue. Request assistance from the instructor if necessary.

## Section 12: Check the MsgId and CorrelId values

- \_\_ 24. Display messages in queues EX4REPT and EX4REPL by using the method appropriate to the operating system you are using. Remember, EX4REPT has a copy of your original request message. You need to:
- \_\_ a. Display the message identifier (MsgId) field for the message in the EX4REPT queue.
  - \_\_ b. Display the correlation identifier (CorrelId) field for the message in the EX4REPL queue.
  - \_\_ c. Ensure that the CorrelId of the EX4REPL message matches the MsgID of the message in queue EX4REPT.



### Warning

The message and correlation identifier comparison is a critical milestone for this exercise. If your values do not match, the third application, `ex4match`, is not able to find a matching reply. If these values do not match, *fix it now before you proceed* to the next program in this exercise.



### Reminder

Remember that EX4REPL **does** have extra messages in it. **Some of the messages in queue EX4REPL have zero values in the CorrelId field.** However, one CorrelId field should match the MsgId field of the message in the EX4REPT queue.



### Linux

```
$ amqsbcg EX4REPT MQ01 | grep MsgId
```

Expected result:

```
MsgId : X'414D51204D5130312020202020202020202020202087261E5702950A20' <==Check
```

```
$ amqsbcg EX4REPL MQ01 | grep CorrelId
```

Expected result:

```
CorrelId : X'414D51204D5130312020202020202020202020202087261E5702950A20' <==Check
```





**This Notes box is a reminder for your future reference; do not make any other changes.**

```

****Message descriptor****
StrucId  : 'MD  '  Version : 2
  Report   : 0  MsgType  : 2
  Expiry   : -1  Feedback : 0
  Encoding : 546  CodedCharSetId : 1208
  Format    : 'MQSTR  '
  Priority  : 0  Persistence : 0
  MsgId    : X'414D51204D5130312020202020202020202087261E5702322E20'
  CorrelId : X'414D51204D5130312020202020202020202087261E5702950A20'
  BackoutCount : 0
  ReplyToQ      : 'EX4REPT              '
  ReplyToQMgr   : 'MQ01                  '
  ** Identity Context
  UserIdentifier : 'mqadmin8              '
  AccountingToken :
    X'033530320006'
  ApplIdentityData : '                    '
  ** Origin Context
  PutApplType     : '6'
  PutApplName     : 'ex4repl              '
  PutDate        : '20160502'  PutTime : '15513263'
  ApplOriginData : '                    '
  GroupId        : X'00'
  MsgSeqNumber   : '1'
  Offset         : '0'
  MsgFlags       : '0'
  OriginalLength : '-1'
  **** Message ****
length - 5 of 5 bytes
00000000: 5649 4C4C 41                                'VILLA'

```

**Proceed to the next section to work with the last application in this exercise, `ex4match.c`.**

## Modify application `ex4match.c`



### Information

In this part of the exercise, you match request messages to their corresponding reply messages.

The `ex4match` application is a modification of the `amqsget0.c` sample, changed to get messages from an extra queue and match the request and reply.

**`ex4match` processes one request per execution.**

ex4match:

- Does not take any parameters. You type in the terminal command `ex4match` window **after** successfully running `ex4req` and `ex4repl`.
- Does a destructive MQGET of one message from the EX4REPT queue.
- Sets up a second destructive MQGET for queue EX4REPL by using the MsgId from the EX4REPT message to match the CorrelId of the EX4REPL message.
- Displays the request string followed by the reply string, for example, ASTON VILLA.
- If you typed several request messages, run one iteration of `ex4match` for each request message you typed.

### Section 13: Review the `ex4match.c` code sections already implemented



#### Important

This section is for review. You do not change any code.

- \_\_\_ 26. Review the code already added to `ex4match.c`. Your review should be similar to the text box directly below this step. *Remember, line numbers that are shown are **approximate**.*
- \_\_\_ a. Search the `ex4match.c` source for string “Ex4”.
  - \_\_\_ b. The extra object descriptor, `MQMD` structure, and `MQGMO` structure are named with the letters “Rpt” following the `od`, `md`, and `gmo` corresponding declaration prefixes.
  - \_\_\_ c. Extra fields are also added for the different return and reason codes.
  - \_\_\_ d. Review the highlighted declaration setup to hold the hardcoded values of the queue manager and queues to connect to. **You use these variables in the following step.**

```

31  /*  Declare MQI structures needed                                */
32  MQOD      od = {MQOD_DEFAULT}; /* Object Descriptor          */
33  MQOD      odRpt = {MQOD_DEFAULT}; /* Ex4 Object Descriptor     */
34  MQMD      md = {MQMD_DEFAULT}; /* Message Descriptor        */
35  MQMD      mdRpt = {MQMD_DEFAULT}; /* Ex4 Message Descriptor    */
36  MQGMO     gmo = {MQGMO_DEFAULT}; /* get message options       */
37  MQGMO     gmoRpt = {MQGMO_DEFAULT}; /* Ex4 get message options   */
38  MQCNO     cno = {MQCNO_DEFAULT}; /* connection options        */

48  MQLONG    CompCodeRpt;
49  MQLONG    OpenCode; /* MQOPEN completion code */
50  MQLONG    OpenCodeRpt;
51  MQLONG    Reason; /* reason code */
52  MQLONG    ReasonRpt;

61
62  /*  Exercise 4  */
63  char replQNm[] = "EX4REPL";
64  char reptQNm[] = "EX4REPT";
65  char QMNm[] = "MQ01";
66

```

### Section 14: Set up the queue object descriptor names, and queue manager names

- \_\_ 27. Scroll down until you find string “Exercise 4”. You should be at approximately line 70, directly before the comment that starts with “Type code to set up ...”.
- \_\_ 28. Use the names of the variables that you reviewed to set up the object descriptor queue names, and the queue manager name for the function calls. **An example of the code is shown in the box directly following this set of steps.**
- \_\_ a. Copy the hardcoded queue manager name to the `QMName` parameter used in the `MQCONN` function call.
- \_\_ b. Use the variable that holds the name of the `EX4REPT` queue to set up the queue name for object descriptor “`odRpt`”.
- \_\_ c. Use the variable that holds the name of the `EX4REPL` queue to set up the queue name for object descriptor “`od`”.

```

70  /* Exercise 4 */
71  /*Type code to set up the name of the queue manager for the MQCONN here */
72  /*Type code to set up the queue names in both object descriptors here */
73
74  strncpy(odRpt.ObjectName, reptQNm, MQ_Q_NAME_LENGTH);
75  strncpy(od.ObjectName, replQNm, MQ_Q_NAME_LENGTH);
76  strncpy(QMName, QMNm, MQ_Q_MGR_NAME_LENGTH);

```

### Section 15: Set up the MQGET for queue EX4REPL to get messages by the correlation identifier

- \_\_ 29. Search forward for string “Exercise 4” until you find comment /\*Exercise 4 - set up EX4REPL MQGET by CorrelId\*/. This comment should be *approximately* at line 233 of the ex4match.c source.
- \_\_ 30. Type the code necessary to get a specific message by correlation ID. An example of the code that is needed is provided in the text box directly following this set of substeps.
  - \_\_ a. Ensure that the MQGMO version is set to version 2. Failure to set the MQGMO version causes the match option to be ignored.
  - \_\_ b. Set the MQGMO match options to match correlation identifier.
  - \_\_ c. Copy the MQMD message identifier from the MQREPT message to the MQMD correlation identifier of the MQREPL message.

```

233 /* Exercise 4 - set up the EX4REPL MQGET by CorrelId */
234 /* Type the code needed to match by correlation identifier here */
235
236 /*****
237      gmo.Version = MQGMO_VERSION_2;
238      gmo.MatchOptions = MQMO_MATCH_CORREL_ID;
239      memcpy(md.CorrelId, mdRpt.MsgId, sizeof(md.MsgId));

```

### Section 16: Compile and test the ex4match application

- \_\_ 31. Compile the ex4match.c application into executable ex4match by using the instructions applicable to the operating system you are using.
- \_\_ 32. Test the application by typing ex4match without any parameters. The application gets the first message in queue EX4REPT. Search EX4REPL for a matching message. The expected test results are shown in the text box that follows this step.

`ex4match`

Expected results:

Program ex4match start

Matching first EX4REPT message to corresponding message in EX4REPL

Request/Reply ASTON VILLA

Program ex4match end

\_\_ 33. If you received the results that are shown, you successfully completed Exercise 3. If you did not get the expected results, resolve the problem or request assistance from your instructor.

***You completed Exercise 3.***

**End of exercise**

## Exercise review and wrap-up

During this exercise, you:

- Modified an application to generate a confirm-on-arrival (COA) report message that preserves the original message identifier (MsgId)
- Modified an application to reply to a request message by setting up the correlation identifier of the reply message to the message identifier of the request message
- Used a formatted message descriptor display to check the correct setting of your message and correlation identifier fields
- Altered an application to get a reply message from a queue with a correlation identifier that matches the message identifier of its corresponding request message

## Exercise 5. Commit and backout review

### What this exercise is about

This exercise reinforces the topics in the commit and backout unit by reviewing the sample program `amqsxag0.c`.

### What you should be able to do

After completing this exercise, you should be able to:

- Locate an example of a commit and backout application in the IBM MQ installation
- Include the backout structure declaration and initialization in your application
- Code the `MQBEGIN` function call
- Design and code the flow of tasks that are required for a commit or backout application
- Code the `MQCMIT` and `MQBACK` function calls

### Introduction

In this exercise, you review a sample application to reinforce the material that is presented in the corresponding unit. You do not need to compile or test the application. You review how the `MQBEGIN` function call is coded, how different indicators are set in the program logic to check the progress of database updates, and how either `MQCMIT` or `MQBACK` is selected.

### Requirements

- IBM MQ sample application `amqsxag0.c`

## Exercise instructions

### Section 1: Locate the sample commit and backout application

- \_\_\_ 1. Open one command prompt terminal window.
- \_\_\_ 2. Locate and open IBM MQ sample `amqsxag0.c`. This application is found on a different directory based on the operating system you selected for your exercises.



#### Linux

`amqsxag0.c` can be found in directory: `/opt/mqm/samp/xatm`



#### Windows

`amqsxag0.c` can be found in directory:

`C:\Program Files\IBM\WebSphere MQ\tools\c\Samples\xatm`

- \_\_\_ 3. Browse the program description, and scroll down to the function prototypes of the SQL routines. You find these routines started throughout `amqsxag0.c`. You also find different return codes that are checked through the application flow.
- \_\_\_ 4. Scroll until you find the comment `/* MQI structures*/`. You see a new declaration for the IBM MQ begin options structure, initialized by its corresponding initial value default structure.

```

/* MQI structures  */
/*****/

MQOD od = {MQOD_DEFAULT};           /* object descriptor          */
MQMD md = {MQMD_DEFAULT};           /* message descriptor         */
MQGMO gmo = {MQGMO_DEFAULT};        /* get message options       */
MQBO bo = {MQBO_DEFAULT};           /* begin options              */
    
```

- \_\_\_ 5. If you look at the other declarations below the MQI structures, you also find declarations for return codes that are checked throughout the progress of the application.
- \_\_\_ 6. Scroll slowly and you pass the code segment where the input parameters are processed: the `MQCONN` function call, setting of some options for the `MQOPEN`, and the `MQOPEN` call.
- \_\_\_ 7. Proceed until you find documentation string `/* Get messages from the message queue, loop until there is a failure */`. The code to review is displayed in the following text box.



## Section 2: Observe the manual tracking of resources outside IBM MQ

```

while (rc == OK)
{
    /*****
        /* Set flags so that we can back out if something goes wrong and not
        /* lose the message.
    *****/
    gotMsg = FALSE;
    committedUpdate = FALSE;
    /*****
        /* Start a unit of work
    *****/
    MQBEGIN (hCon, &bo, &compCode, &reason);

```

- \_\_ 8. The while loop continues while the flag `rc` continues to have the value `OK`. This flag is used to track of the IBM MQ function call returns. Note the other details in this code:
- \_\_ a. The developer manages the next two flags, `gotMsg` and `committedUpdate`, which apply to the results of the database routines. Database status is not tracked automatically. Tracking results of IBM MQ and database resources lies within the application developer.
  - \_\_ b. After setting all tracking flags, the `MQBEGIN` function call is run. Aside from the return code and return reason parameters at the end of the parameter list, this call uses the connection handle and the begin options, or `MQBO` structure.
  - \_\_ c. You might want to review some of the specific errors that are checked following the `MQBEGIN` function call.
- \_\_ 9. Find the additional manual synchronization check below the comment that starts with `/* The number of transactions to the two databases...*/`. The code that is referenced is displayed in the text box:

```

/* The number of transactions to the two databases should be          */
/* identical, stop if not. */
/*****/

    if (rc == OK)
    {
        if (temp != transactions)
        {
            printf("Databases are out of step !\n");
            rc = NOT_OK;                /* stop looping          */
        } /* endif */
    } /* endif */

```

\_\_\_ 10. `amqsxag0.c` processes two databases. Depending on the application, the developer might also need to manually check whether the outcome is as expected before reaching the end of the logic to determine whether to commit or back out the change. This observation is not an IBM MQ-specific coding detail. However, the key point is to ensure, before you code the `MQCMIT`, that all resources are ready and as expected.

**Section 3: Review the `MQCMIT` and `MQBACK` function calls**

- \_\_\_ 11. Scroll to find the `MQCMIT` and `MQBACK` calls. For this application, both calls are included in the while loop. Also, notice the continued use of the status flags throughout the code.
- \_\_\_ 12. Review the parameters used by the `MQCMIT` and `MQBACK` calls. Both calls use the same parameters:
  - The connection handle
  - The function return code and return reason

**You completed Exercise 5.**

**End of exercise**

## Exercise review and wrap-up

In this exercise, you:

- Reviewed how to include the backout structure declaration and initialization in your application
- Reviewed how to code the `MQBEGIN` function call
- Located the example of a commit and backout application in the IBM MQ installation
- Looked at the details to consider when you design and code an application that coordinates IBM MQ and database resources
- Reviewed how to code the `MQCOMMIT` and `MQBACK` function calls



## Exercise 6. Asynchronous messaging review

### What this exercise is about

This exercise reinforces and extends the asynchronous messaging concepts and mechanics that are covered in the lecture. Rather than writing code, you follow the actions that are taken in sample program `amqcbf0.c`, and identify key portions of the process. You analyze the code and run the precompiled binary file to see asynchronous messaging in action.

### What you should be able to do

After completing this exercise, you should be able to:

- Describe the mechanics and component exchanges of asynchronous messaging applications
- Explain the role of the MQCB function call and its required parameters
- Differentiate between the IBM MQ MQCB callback function and the application callback module represented by the MQCB\_FUNCTION parameter definition
- Describe how to initiate and end consumption of messages with the MQCTL function call
- Explain how to code and exchange information in an application callback function

### Introduction

The objective of this exercise is to reinforce the mechanics of asynchronous message consumption, rather than adding the code yourself. Asynchronous message consumption can pose a challenging task. This exercise extends the concepts that are learned in the lecture, and serves as an example on which to base your own development.

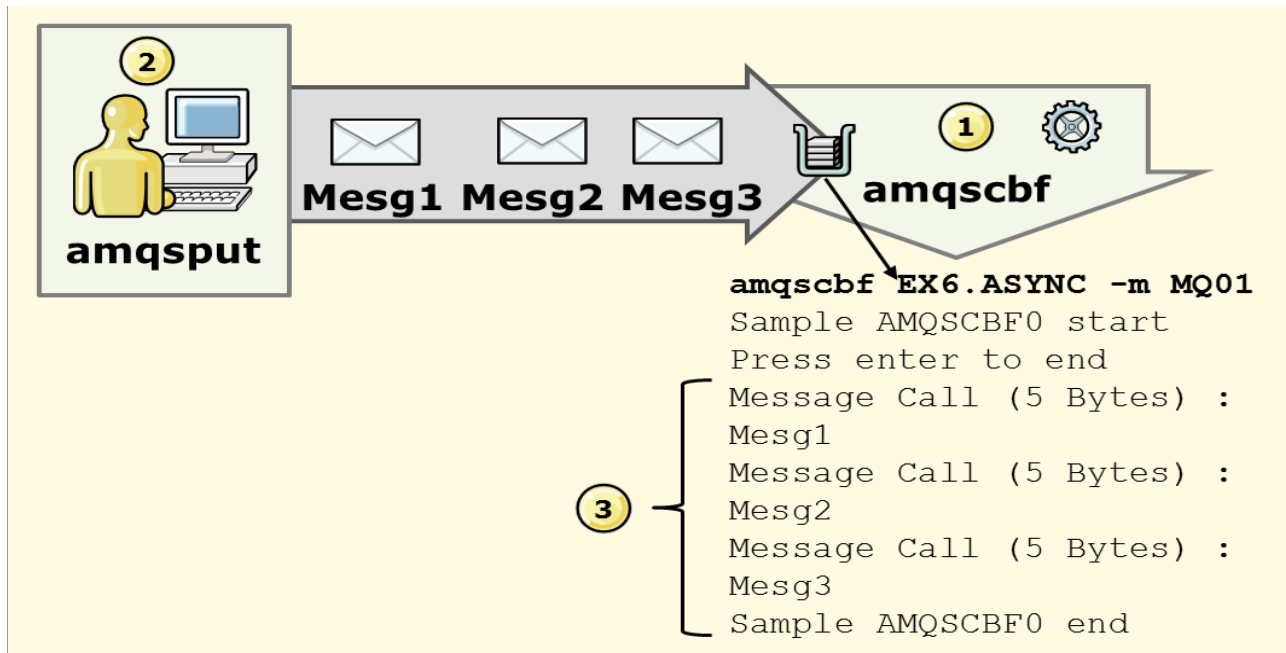
### Requirements

- Queue manager MQ01 with local queue `EX6.ASYNC` defined
- IBM MQ sample applications `amqscbf0.c` and `amqsput0.c`
- One terminal command session to run the callback sample
- One terminal command session to run the put message sample

# Exercise instructions

## Preface

In this exercise, you focus on reinforcing the mechanics of how the asynchronous message consumption is coded by reviewing sample program `amqscbf0.c`. Following the review, you start the precompiled `amqscbf` binary to processes that arrive at queue `EX6.ASYNC`, then send messages to the same queue by using sample `amqspout`, and observe the behavior. In the sample callback function for `amqscbf0.c`, the messages are printed to the console. In your program, you would have your business logic process your message. The diagram depicts the test that you run in this lab.



## Part 1: Asynchronous messaging code review

- \_\_ 1. Open two command prompt terminal screens.
- \_\_ 2. Use one terminal screen to find sample code `amqscbf0.c` according to your operating system location, and open it for browse.



Linux

`amqscbf0.c` is in the `/opt/mqm/samp/` directory.

**Windows**

amqscbf0.c is in the C:\Program Files\IBM\WebSphere MQ\Tools\c\Samples directory.

**Information**

All code line numbers that are provided are approximate.

\_\_ 3. Scroll through the start of the program and review the documentation.

**Section 1: Check the callback function**

- \_\_ 4. Continue scrolling to approximately line 85, where you find the comment `/* FUNCTION: MessageConsumer */`. This part of the code is the callback function.
- \_\_ a. Compare the syntax of this code with the documentation for `MQCB_FUNCTION`. Remember, `MQCB_FUNCTION` is not a function call, but rather the syntax that the queue manager expects when it calls this code.
  - \_\_ b. Note how the `CallType` from the callback context (`MQCBC`) structure is checked.
  - \_\_ c. For sample `amqscbf0.c`, the `MessageConsumer` function prints the contents of the message up to 200 bytes.
  - \_\_ d. Note the pointer to a type `MQCBC` structure named `pContext`.

**User-written, started callback function in sample program `amqscbf0.c` (to distinguish from actual `MQCB` function call):**

```

85  /* FUNCTION: MessageConsumer                                     */
    86  /* PURPOSE : Callback function called when messages arrive     */
    87  /******
    88  void MessageConsumer(MQHCONN  hConn,
    89                      MQMD     * pMsgDesc,
    90                      MQGMO     * pGetMsgOpts,
    91                      MQBYTE    * Buffer,
    92                      MQCBC     * pContext) <= MQCBC type structure pointer.
    ... ..
    96
    97  switch(pContext->CallType) <= CallType from MQCBC structure type pointer.
    98  {
    99  case MQCBCT_MSG_REMOVED:
   100  case MQCBCT_MSG_NOT_REMOVED:

```

## Section 2: Check the asynchronous messaging-related declarations

- \_\_ 5. Scroll to the start of the main section and review the two structure declarations that are associated with asynchronous consumption, the callback descriptor and control options structures:

### Declaration of structures that are used for asynchronous messaging:

```
146  MQCBD   cbd = {MQCBD_DEFAULT}; /* Callback Descriptor          */
147  MQCTLO  ctlo= {MQCTLO_DEFAULT}; /* Control Options
```

## Section 3: Review the standard MQCONN and MQOPEN calls

- \_\_ 6. Search for string `MQCONN`. You should be at the `MQCONN` invocation around code line 267. This code shows the standard connection to the queue manager.
- \_\_ 7. Continue to scroll until you find the `MQOPEN` call. This code shows the standard open for the queue from which the callback function gets messages. So far, the code looks as usual.

## Section 4: Find and review the callback code

- \_\_ 8. Find the documentation comment “Register a consumer” at approximately line 307. This code is also displayed in the text box directly following this numeric step, and might be in the following page.
- \_\_ a. This portion of code registers the callback and specifies the function that the callback starts.
- \_\_ b. Notice how the name of the callback function, `MessageConsumer`, is set in the `CallbackFunction` field of the `MQCBD` structure in line 310 of the display, ahead of the `MQCB` call invocation.
- \_\_ c. Notice that the `gmo.Options` are also set before the `MQCB` call. A key difference of asynchronous messaging is that no `MQGET` call is coded. The `MQCTL` call gets the messages when its `Operation` field is set to `MQOP_START` or `MQOP_START_WAIT`. The `gmo.Options` are set in the `MQCB` call since the `MQCTL` call uses the same thread from the `MQCB` call.
- \_\_ d. Check how the `Operations` field is now replaced with the actual defined value for the required operation, in this case `MQOP_REGISTER`.
- \_\_ e. Next, you see the use of the callback descriptor, or `cbd` field, and later the `gmo` structure.



**MQCB IBM MQ callback function setup and invocation:**

```

/*****/
306      /*                                     */
307      /*  Register a consumer                 */
308      /*                                     */
309
/*****/
310      cbd.CallbackFunction = MessageConsumer;
311      gmo.Options = MQGMO_NO_SYNCPOINT;
312
313      MQCB(Hcon,
314          MQOP_REGISTER,
315          &cbd,
316          Hobj,
317          &md,
318          &gmo,
319          &CompCode,
320          &Reason);
321      if (CompCode == MQCC_FAILED)
322      { ... .. }

```

**Section 5: Find and review the MQCTL invocation to start consumption of messages**

- \_\_ 9. Carefully scroll until you see the invocation for the MQCTL function call.
- \_\_ 10. The MQCB function call registered and set up the process so that the MessageConsumer callback function is started when messages arrive. Next, you need the MQCTL function with an MQOP\_START or MQOP\_START\_WAIT operation to initiate the consumption of messages and “hand these messages over” to the MessageConsumer callback function. Review the MQCTL call.

**MQCTL IBM MQ control callback function set to start consumption by having the Operations field set to MQOP\_START:**

```

332      MQCTL(Hcon,
333          MQOP_START,
334          &ctlo,
335          &CompCode,
336          &Reason);

```

## Section 6: Review how to end message consumption

- \_\_ 11. In the callback function that you code for your application, you need to include code to end the consumption of messages. In the sample program, the end of the program is started by pressing the Enter key. **For purposes of this exercise, the process that is used to signal an end to the program in the sample is not important**, what matters is how to use the MQCTL function to end consumption. Scroll until you find the second MQCTL call and review the operation that is used. The second CTL should be near line 357.

**The MQCTL IBM MQ control callback function is set to stop consumption of messages by having the `Operations` field set to `MQOP_STOP`:**

```

352  /*****
353      /*
354      /* Stop consumption of messages
355      /*
356
357  /*****
357      MQCTL(Hcon,
358          MQOP_STOP,
359          &ctlo,
360          &CompCode,
361          &Reason);

```

## Section 7: Check for the standard IBM MQ ending function calls

- \_\_ 12. The next logical step is to code an MQCLOSE on queue `EX6.ASYNC`, and MQDISC on queue manager `MQ01`. However, the sample `amqscbf0.c` code has a reported error, in that it omitted the MQCLOSE. If you look at the program description, the MQCLOSE is included, but the function call is omitted.



### Warning

The omission of the `MQCLOSE` call has an open problem report (APAR), and the regular maintenance processes are to rectify it.

- \_\_ 13. Scroll and check the standard MQDISC.

You finished the code review portion of the exercise. Continue to Part 2 of the exercise to see this code in action.

## Part 2: Test the asynchronous messaging sample

You should have two command prompt windows open.

The local queue that is used for this test, `EX6.ASYNC`, is predefined.

## Section 8: Start the asynchronous callback sample `amqscbf`



### Warning

The syntax to invoke this sample program is different from the put and get samples. Take care to type the commands exactly as shown.



### Windows

If you have problems when you look for the `amqscbf` binary in your Windows environment, change to the `bin64` directory as shown, and run `amqscbf` from that directory:

```
cd C:\Program Files\IBM\WebSphere MQ\Tools\c\Samples\Bin64
```

- \_\_\_ 14. Start the `amqscbf` sample program by using the instructions in the box that follows this step, and move to the second window. **Do not press Enter after you start `amqscbf`.** If you press the Enter key a **second time**, `amqscbf` stops. Start `amqscbf` by typing the command that is provided, and press the Enter key **one time only** to activate the callback sample. Use queue `EX6.ASYNC` and queue manager `MQ01`.

```
amqscbf EX6.ASYNC -m MQ01
```

Expected response:

```
Sample AMQSCBF0 start
```

Press Enter to end.

- \_\_\_ 15. `amqscbf` registered the callback. The `MQCB` function uses the `MessageConsumer` callback function, which has the task to print messages as they are received. **Leave this screen open where you can view it, and move to the second terminal command screen.**
- \_\_\_ 16. In the second command prompt screen, put three messages to queue `EX6.ASYNC` by using the instructions for the `amqspout` sample program that is provided in the next box. **As you type each message, observe how the registered callback in the first screen intercepts the arrival of the messages.** Remember to press Enter on a blank line to end `amqspout` after you type the messages.

```
amqsput EX6.ASYNC MQ01
```

Expected response:

```
Sample AMQSPUT0 start  
target queue is EX6.ASYNC
```

```
This is the first message  
This is the second message <=== Messages you type  
This is the third message
```

```
Sample AMQSPUT0 end
```

\_\_ 17. As you type the messages, each message should display on the first screen, where `amqscbf` is running with the callback. Expected results are shown in the next box, but your output data will vary depending on the text you type.

**Messages display in the screen where `amqscbf` is running as you type them in `amqsput`.**

```
.. ... ..  
Press enter to end  
Message Call (25 Bytes) :  
This is the first message  
Message Call (26 Bytes) :  
This is the second message  
Message Call (25 Bytes) :  
This is the third message
```

**<=== In the next step, you end `amqscbf` by pressing Enter on  
<=== the first screen**

```
Sample AMQSCBF0 end
```

\_\_ 18. After you see your messages on the `amqscbf` screen, press Enter to end `amqscbf`.  
You completed Exercise 6.

## End of exercise

## Exercise review and wrap-up

In the first part of the exercise, you used a sample program to review the sequence of calls and information that is exchanged to complete an asynchronous messaging program.

In the second part of the exercise, you tested the code that you reviewed to see the results of asynchronous message consumption in action.



# Exercise 7. Working with an IBM MQ client

## What this exercise is about

This exercise shows you how to compile a program with the IBM MQ client libraries. You also learn how to connect the IBM MQ client to the queue manager in three different ways, which types of connections supersede others, and how to connect with code without configuring the IBM MQ client.

## What you should be able to do

After completing this exercise, you should be able to:

- Compile an existing program as an IBM MQ client
- Review a client connection channel definition and test connectivity to a queue manager by using the client channel definition table
- Configure and test connectivity to the queue manager by using the MQSERVER environment variable
- Alter the client program to use the MQCONN call to establish connectivity to the queue manager

## Introduction

The objective of this exercise is to provide experience with IBM MQ clients in areas you face when developing a client application. You learn to compile a client application, but most important, you learn how to set up your test environment. When you work with an IBM MQ client, you might not have an administrator to set it up, as an IBM MQ client does not have a queue manager. You must know how to connect to a queue manager from your client. You also work with an MQCONN call as a connectivity option.

## Requirements

- `putmsg.c` base source at `<dir>/LabFiles/Lab7/source`.
- `codeconn.c` solution source at `<dir>/LabFiles/Lab7/solution`.
- Queue manager `MQ01` with predefined server connection (`SVRCONN`) channel `EX7.CLNTCONN`.
- IBM MQ Client with predefined client connection (`CLNTCONN`) channel `EX7.CLNTCONN`.
- Queue manager `MQ03` with server connection channel `EX7.MQSERVER`.

- Local queue EX7.LOCALQ predefined in each queue manager, MQ01, and MQ03. EX7.LOCALQ **is not a clustered queue**.
  - For purposes of this lab exercise, both queue managers must have channel authentication (CHLAUTH) disabled. **If unsure**, refer to the section on how to check the queue manager channel authentication, provided towards the start of Exercise 1.
-



## Environment description

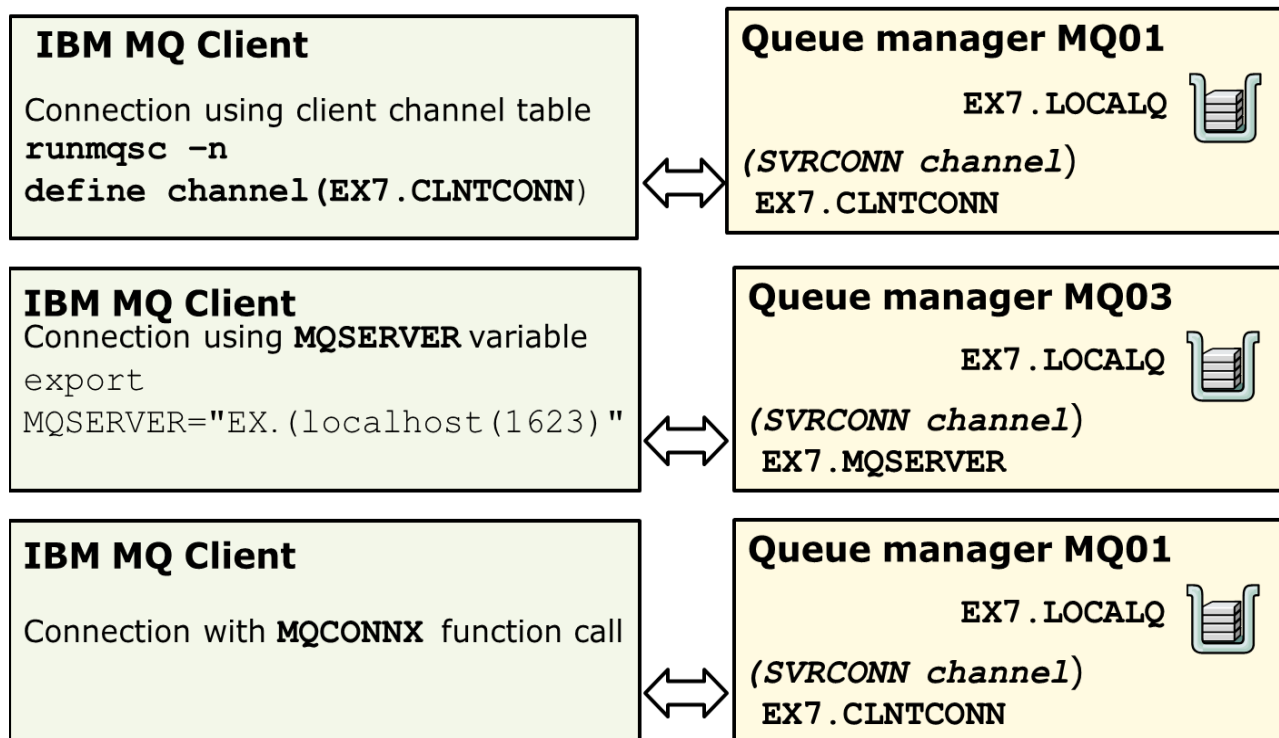


Figure 7-1. IBM MQ client connectivity options



### Important

Take time to read through the narrative for this exercise so you understand the concepts presented.

In this exercise:

- You compile an existing program as an IBM MQ client to compare and check how the same code works in the IBM MQ client as in the IBM MQ server.
- The sequence of the next steps *is important* because it shows you how different types of channel connections supersede each other. Which connection is used might be important when trying to determine where your message was sent.
- To complete the exercise correctly, use the **same** command prompt terminal screen for all work, unless explicitly directed otherwise, so that you can observe what connection types supersede other connection types.

This exercise uses two queue managers. Each queue manager has a queue with the same name defined, `EX7.LOCALQ`. However, server connection channels in the two queue managers have different names.

**What queue manager receives the message depends on the client channel used at the time of your test.**

1. After you compile the test program, you use the client channel definition table (CCDT) connection to send a message to the queue manager. Since the CCDT channel connects to queue manager MQ01, the message that is sent should be found in the `EX7.LOCALQ` queue defined in the MQ01 queue manager.
  2. You now configure the MQSERVER environment variable to connect to queue manager MQ03. An MQSERVER connection supersedes a CCDT connection, so after you configure your MQSERVER variable, messages that are put to queue `EX7.LOCALQ` are found in queue manager MQ03.
  3. Last you modify the program so that you specify the connectivity in the MQCONNX call. You use the information for queue manager MQ01 in MQCONNX, so any calls put to queue `EX7.LOCALQ` return to queue manager MQ01.
-

## Exercise instructions

### Section 1: Compile program `putmsg.c` to run as an IBM MQ client application

In an earlier lab, you worked with program `putmsg.c`. You now use the same code, but prepare your program with the IBM MQ client libraries so `putmsg.c` runs as an IBM MQ client.



#### Stop

In this exercise you use:

- One terminal command prompt to test the different connectivity options. Ensure that you use this terminal for all tests to the queue managers.
- One terminal command prompt to use the `runmqsc` utility, since some of the tests have you use `runmqsc` while an application is active in the other screen.
- If you are using Windows, one Developer Command Prompt for VS2015 screen, also at the source directory for Lab 7.

In this exercise you use two different implementations of the `runmqsc` utility:

- The “traditional” `runmqsc` utility, which you start followed by a queue manager name.
- The client `runmqsc` utility, which you start followed by a `-n`.

\_\_ 1. Except for the Developer Command Prompt for VS2015 if you use Windows, **close all other** command prompt terminal screens that you might have open. **It is critical to start with a fresh command prompt window. If any unexpected environment variables remain, the results of the lab might be invalid.**

\_\_ 2. Open a new command prompt terminal screen.



#### Linux

Change to the IBM MQ administrator user by typing:

```
su - mqadmin8
```

Ensure that you see the response message “You are in your home directory at `/home/mqadmin8`” after you change users.

\_\_ 3. Proceed to the `LabFiles/Lab7/source` directory, by using the commands according to the operating system used.

\_\_ 4. Type the command as shown, taking care to use the different libraries other than the earlier `putmsg.c`, and naming your executable file: `putmsgc`



## Linux

It is not imperative that you rename the compiled module to `putmsgc`. The step is highlighting a convention that is used in the IBM MQ provided samples, which append a “c” to client modules. In the last part of this exercise, you omit adding a “c” to the name of the last program. The addition of the “c” to the name is optional. *Ensure that you are consistent in the name you use for your module and the name you use to test.*

```
gcc -I/opt/mqm/inc -lmqic -o putmsgc putmsg.c
```



## Windows

It is not imperative that you copy the source to `putmsgc.c`. The step is highlighting a convention that is used in the IBM MQ provided samples, which append a “c” to client modules. In the last part of this exercise, you omit adding a “c” to the name of the last program. *Ensure that you are consistent in the name you use for your client program and the name you use to test.*

====> **Copy `putmsg.c` to `putmsgc.c`**

- Ensure that you are at a Developer Command Prompt, and that you are positioned at directory `C:\LabFiles\Lab7\source` in the Developer Command Prompt.
- If you choose to use the `copy_ed_clt.txt` file, it is updated to use the client library, `mqic.lib`.
- Type:

```
cl putmsgc.c C:\PROGRA~1\IBM\WEBSPH~1\Tools\Lib\mqic.lib
```

## Section 2: Check your client channel definition table (CCDT) CLNTCONN type channel

5. Start a `runmqsc` session in the client. Remember to use the `runmqsc` command in the IBM MQ client. You follow the command with a `-n` as shown, followed by pressing the Enter key:

```
[mqadmin8@rhel6base source]$ runmqsc -n
```

**Expected results:**

```
5724-H72 (C) Copyright IBM Corp. 1994, 2015.
Starting local MQSC for 'AMQCLCHL.TAB'.
```

**Reminder**

When you use the `-n` command, you are using the *IBM MQ client version* of the `runmqsc` utility.

- \_\_ 6. You are now in the IBM MQ command processor. Type the command to show the client channel, as shown:

```
dis chl(*) conname
```

**Expected response:**

```
1 : dis chl(*) conname
AMQ8414: Display Channel details.
CHANNEL(EX7.CLNTCONN)                CHLTYPE(CLNTCONN)
CONNAME(localhost(1621))
```

- \_\_ 7. Check the connection information for this channel definition. What channel and port does this connection point to? \_\_\_\_\_
- \_\_ 8. Save the channel name that you recorded in the previous step. You use it in a later step.
- \_\_ 9. From your knowledge of what ports point to which queue manager in this course, which queue manager does this connection point to? \_\_\_\_\_

**Note**

Queue manager MQ01 is in your VMware image; it listens on port 1621. You can deduce that this CLNTCONN type channel, which is defined in your IBM MQ client, connects to the SVRCONN type channel of the same name, in queue manager MQ01.

- \_\_ 10. Exit the `runmqsc` client process by typing `end` and pressing the Enter key.

### Section 3: Start the test of the `putmsgc` application by using the CCDT definition



#### Note

Since the only connection from the IBM MQ client to the queue manager at this stage of the exercise uses the type `CLNTCONN` channel to port 1621, the message is expected to go to the `EX7.LOCALQ` queue that is defined in queue manager `MQ01`.

- \_\_\_ 11. Type the command as shown to put a message to local queue `EX7.LOCALQ`. You need to provide the queue name; the queue manager is not used when you test. The queue manager is implicit in channel definition that is in use.

```
putmsgc EX7.LOCALQ
```

Expected response:

```
Sample PUTMSG start
target queue is EX7.LOCALQ
```

- \_\_\_ 12. **Do not press the Enter key again until instructed. Leave the `putmsgc` program in a state of waiting for input.** If you pressed the Enter key, repeat the `putmsgc EX7.LOCALQ`.



#### Note

The client program is now waiting for input. Do not be concerned that the displayed name of the program is `PUTMSG` instead of `PUTMSGC`; you did not update the program before compiling it as an IBM MQ client application. For purposes of this exercise, the displayed name is not important.

If you receive the “Sample `PUTMSG` start”, then it means that your client program connected to the queue manager, and the client channel started. In the next section, you see how, for a client channel, it is not necessary to manually start the channel.

### Section 4: Check the channel status from queue manager `MQ01`

- \_\_\_ 13. Type `runmqsc MQ01` and press the Enter key to start an IBM MQ command script session. You are now in the traditional, queue manager version of the `runmqsc` utility.
- \_\_\_ 14. Type `dis chstatus(EX7.CLNTCONN)` and press the Enter key. Your results should resemble the text box. The channel should be in running status.

```
dis chstatus(EX7.CLNTCONN)
```

Expected results:

```
2 : dis chs(EX7.CLNTCONN)
```

```
AMQ8417: Display Channel Status details.
```

```
CHANNEL(EX7.CLNTCONN)
```

```
CHLTYPE(SVRCONN)
```

```
CONNAME(127.0.0.1)
```

```
CURRENT
```

```
STATUS(RUNNING) <==check! SUBSTATE(RECEIVE)
```

\_\_ 15. Type `end` and press the Enter key to exit `runmqsc`.

### Section 5: Resume the `putmsgc` test

\_\_ 16. Type the following message: message 1 uses `ccdt` definition

\_\_ 17. Press the Enter key two times to end the `putmsgc` application.

\_\_ 18. Confirm that the message was sent to queue manager `MQ01`. Open a `runmqsc` session for queue manager `MQ01` by typing `runmqsc MQ01` and pressing the Enter key.

\_\_ 19. Display the number of messages in queue `EX7.LOCALQ` in queue manager `MQ01` by typing:

```
dis q(EX7.LOCALQ) curdepth
```

Expected results:

```
1 : dis q(EX7.LOCALQ) curdepth
```

```
AMQ8409: Display Queue details.
```

```
QUEUE(EX7.LOCALQ)
```

```
TYPE(QLOCAL)
```

```
CURDEPTH(1)
```

\_\_ 20. The value in the `CURDEPTH` attribute of queue `EX7.LOCALQ` for queue manager `MQ01` should be 1. If you do not see a count of 1 in this display, resolve the problem before continuing. If necessary, request help from your instructor.

\_\_ 21. Type `end` and press the Enter key to exit `runmqsc`.

## Section 6: Use a sample application to remove the message from the MQ01 EX7.LOCALQ queue



### Reminder

The IBM MQ client, or MQI channels are bidirectional; that is, messages flow in both directions, to and from the IBM MQ client and the queue manager. You sent messages from the client to the queue manager. You now access the queue manager from the client to retrieve messages.

- \_\_ 22. `amqsgetc` is a client compilation of the `amqsget` sample application, assembled to use the client libraries, similar to your work with `putmsgc`. `amqsgetc` starts to retrieve messages, and continues for 15 seconds. Use `amqsgetc` to retrieve the message you just put in queue `EX7.LOCALQ` by typing the command shown:

```
amqsgetc EX7.LOCALQ
```

#### Expected result:

```
Sample AMQSGET0 start
message <message 1 uses the CCDT definition>
```

- \_\_ 23. After the message is displayed, you can either wait a few seconds for `amqsgetc` to end, or break out of the application by typing Ctrl-C.

## Section 7: Set up your client to connect to queue manager MQ03 by using the MQSERVER environment variable



### Information

You are setting up a configuration with the `MQSERVER` environment variable for two reasons:

- To show an alternative way to connect without a *client channel definition table*, or `CCDT`
- To confirm how the `MQSERVER` environment variable supersedes a `CCDT` connection

The `CCDT` connection was made to queue manager `MQ01`. The `MQSERVER` environment variable connects to queue manager `MQ03`. Remember, both queue managers have local queue defined named `EX7.LOCALQ`. When you test this next configuration, you check queue manager `MQ03` for the message put to the queue by IBM MQ client application `putmsgc`.



**Warning**

**To test how the `MQSERVER` environment variable supersedes the CCDT table, you must run this next test in the same command prompt terminal screen where you tested the first connection.**

Be careful to set up the `MQSERVER` environment variable exactly as shown per the operating system platform used, including quotation marks and parentheses as shown.

- \_\_ 24. Set up and confirm the `MQSERVER` environment variable to connect to queue manager MQ03, listening on port 1623, as shown per operating system used.

**Linux**

Set up by typing the export as shown. **Ensure that you include the quotation marks and correct port number.**

```
export MQSERVER="EX7.MQSERVER/TCP/localhost(1623) "
```

**Confirm the setup** by typing the echo command, and precede the variable with “\$” to show contents of variable, as shown:

```
echo $MQSERVER
```

The expected response is:

```
EX7.MQSERVER/TCP/localhost(1623)
```

**Windows**

Configure the `MQSERVER` variable by typing the set statement as shown. **Ensure that you include the correct port number.**

```
set MQSERVER=EX7.MQSERVER/TCP/localhost(1623)
```

**Confirm the setup** by typing the echo command, and enclose the variable with “%” to show contents of variable, as shown:

```
echo %MQSERVER%
```

The expected response is:

```
MQSERVER/TCP/localhost(1623)
```

## Section 8: Test the `putmsgc` application by using the `MQSERVER` environment variable



### Note

You now have two connections to find one of the two queue `EX7.LOCALQ` definitions:

- The `CLNTCONN` type channel in the `CCDT`, which connects to queue manager `MQ01`
- The `MQSERVER` variable, which connects to queue manager `MQ03`

The `MQSERVER` connection is expected to supersede the `CCDT` connection. Any messages put to queue `EX7.LOCALQ` should now go to queue manager `MQ03`.

\_\_ 25. Type the command as shown to put a message to local queue `EX7.LOCALQ`. You need to provide the queue name; **do not provide the queue manager name**.

The queue manager is implicit in the one existing channel definition in use. If you receive a return `2035` when you typed this command, refer to the Troubleshooting section at the end of this exercise.

```
putmsgc EX7.LOCALQ
```

Expected response:

Sample PUTMSG start

target queue is EX7.LOCALQ

Application is now waiting for messages to be typed.

\_\_ 26. Type message “message 2 using `MQSERVER` variable” and press the Enter key two times to end the application.

\_\_ 27. Confirm that the message was sent to queue manager `MQ03`. To expedite checking, rather than `runmqsc`, you can use the IBM MQ server sample `amqsget` to get the message specifically from queue manager `MQ03`, as shown in the text box.



### Note

The server version of the application, `putmsg` or `amqsput`, requires the queue manager name as an input parameter.

The client version of the applications `putmsgc`, `amqsputc`, and `amqsgetc`, require the queue name, **without** the queue manager name. For these superseding tests, **do not** include the queue manager name when you work with the client version of the application.

By typing the queue manager name, you can confirm that the message went to the queue manager you typed as the parameter for the application that runs the `MQGET` function call.

- \_\_ a. Be sure to supply the parameter with queue manager `MQ03` following the queue name.
- \_\_ b. The expected result is shown below the `amqsget` command.
- \_\_ c. You might want to press Ctrl-C to stop the wait for extra messages to arrive.

```
amqsget EX7.LOCALQ MQ03
```

Expected results:

```
Sample AMQSGET0 start
```

```
message <message 1 using MQSERVER variable>
```

```
^C
```

- \_\_ 28. If the message that you typed is found in queue manager `MQ03`, you can proceed to the next section.
- \_\_ 29. If the message went to queue manager `MQ01`, correct any problems before you continue. Request help from your instructor if necessary.



### Information

You observed how the `MQSERVER` environment variable took precedence over the `CCDT` (`CLNTCONN` definition). The initial `CCDT` connection was to queue manager `MQ01`. However, when you set the `MQSERVER` variable to queue manager `MQ03`, the connection to `MQ03` prevailed.

In the next section, you see how to use the `MQCONN` call with a channel descriptor (`MQCD`) structure in an application. An application that builds the channel description by using the `MQCD` does not use the `CCDT` or the `MQSERVER` variable. The application that uses the `MQCD` configures its own connectivity.

## **Section 9: Modify an existing IBM MQ application to use the `MQCONN` call to set up connectivity to queue manager `MQ01`**

- \_\_ 30. Copy the `putmsg.c` application to a file called: `codeconn.c`
- \_\_ 31. Open the `codeconn.c` source file as documented for the operating system you are using.

**Warning**

All references to specific code line numbers might not be exact; refer to the cited code. References to the cited code segments should be obvious. However, if in doubt, consult your instructor.

**The sections of this exercise where you add your code are prefixed with “Code change #” in the section name.**

### **Section 10:Code change 1: Add extra header file that is required for MQCD structure**

The MQCD structure that is required to create a client connection channel by using the MQCONN function is located in a separate header file, `cmqxc.h`. This file needs to be added to your source.

- \_\_ 32. Locate the include statements for the `cmqc.h` header file declaration, at approximately line 27 of the code.
  - \_\_ a. Add the `cmqxc.h` include statement directly below the `cmqc.h` statement. *Do not include the line number; start at the #.*
  - \_\_ b. Include comments that indicate you added this code. Your results should resemble the display:

```
27 #include <cmqc.h>
28 #include <cmqxc.h> /* Added for Exercise 7 */
```

### **Section 11:Code change 2: Declare the MQCD structure**

- \_\_ 33. Locate line 40 in the `codeconn.c` source.
  - \_\_ a. You should be looking at the MQCSP declaration as shown in the box.
  - \_\_ b. Insert a comment at the start of your changes that shows `/* Exercise 7 */` as shown in the display in the box.
  - \_\_ c. Type your channel definition (MQCD) structure declaration, as shown in the box. **Note the name of the structure that is used to initialize the MQCD for an IBM MQ client.**

```

40  MQCSP  csp = {MQCSP_DEFAULT}; /* security parameters          */
41  /* Exercise 7 */
42  MQCD  cltConn = {MQCD_CLIENT_CONN_DEFAULT}; /* declare MQCD structure
*/

```

- \_\_ 34. Open file `cmqxc.h` by looking at the operating system-specific location.
- \_\_ 35. Find string `MQCD_CLIENT` and scroll down to review the initialization values noted in the box that follows:

```

#define MQCD_CLIENT_CONN_DEFAULT {""},\
    MQCD_VERSION_6,\ <==MQCD version number.
    MQCHT_CLNTCONN,\ <==MQCD channel type, which is used like the CHLTYPE field in DEF
    CHANNEL
    MQXPT_TCP,\ <==MQCD transport type, which is used like the TRPTYPE field in DEF
    CHANNEL
    ... ..

```

- \_\_ 36. Close `cmqxc.h`.



### Warning

A couple of caveats:

- When you type the code into the source file, do not copy and paste from the lab guide to prevent introducing special characters that might cause errors that are difficult to resolve.
- Code samples are intended to show you how to code the function calls, not to be used as an application. If you choose to base your proof-of-concept application on a code sample, ensure that you either:
  - Set all the function call parameters directly above the specific function code, or
  - Remove extraneous code, such as code that processes input arguments, or code that blanks out fields used in function call structures or parameters

## Section 12: Code change 3: Define hardcoded variables

Hardcoded values are used for purposes of this exercise. When you code an application, you might use different methods to provide these values.

- \_\_ 37. Type the following declarations, which start in approximately line 56 of the `codeconn.c` source:

```
/* Exercise 7 declarations */
   char chlName[] = "EX7.CLNTCONN"; /* hard code channel */
   char connName[] = "localhost(1621)"; /* hard code connect info */
   char qmgrName[] = "MQ01"; /* hard code queue mgr name */
```

## Section 13: Code change 4: Set the MQCD fields, point the MQCNO to the MQCD structure, and set the queue manager name for the MQCONN call

- \_\_ 38. Locate string `csp.CSPUser`. The search should take you to the segment of code shown at approximately line 91 in your code, depending on how you spaced the declarations:

**This step is browse only. Correct instructions are provided in a later step.**

```
87 cno.SecurityParmsPtr = &csp;
88 cno.Version = MQCNO_VERSION_5; <=== Check value is 2 or greater
89 ==> this code does not get run!
90 csp.AuthenticationType = MQCSP_AUTH_USER_ID_AND_PWD;
91 csp.CSPUserIdPtr = UserId;
92 csp.CSPUserIdLength = strlen(UserId);
93 } <=== Any code before the bracket does not run. Must start below bracket!
94 /*****
```

- \_\_ 39. Scroll down until you are *below* the last bracket (any code above the bracket is not run), and *directly above* the `/*` Connect to queue manager `*/` text box. The code that you type appears to be duplicated, but earlier code does not run due to an `if` statement. Extra clarification of the same step is shown in the next text box:

```

csp.CSPPasswordLength = strlen(csp.CSPPasswordPtr);
    } <=== Last bracket
. . . . . Start to type your code here
/*****
/*
/*   Connect to queue manager
/*
/*
/*****
QMName[0] = 0;    /* default */
if (argc > 2)
    strncpy(QMName, argv[2], (size_t)MQ_Q_MGR_NAME_LENGTH);
MQCONNX(QMName,          /* queue manager
*/

```

- \_\_ 40. Type **the first five lines of code** as shown in the box below the alphabetic substeps. Start below the line that contains code: `csp.CSPUserIdLength = strlen(UserId)`. You should be above the `MQCONNX` call. **Be careful to start below the last } bracket, or code does not run. The positioning of your code is critical.**
- \_\_ a. Mark the start of your code with a commented line.
  - \_\_ b. The `MQCNO` structure version must be at 2 **or greater** in order for the `MQCD` to be acknowledged. This exercise uses a sample where these fields get set if certain parameters are supplied. You need to set the `MQCNO` version.
  - \_\_ c. Copy the channel name and connection name values to the `MQCD` structure
  - \_\_ d. Copy the queue manager name to the object descriptor.
  - \_\_ e. Point the `MQCNO` structure to the `MQCD` structure.



#### Hint

The code that is shown in the text box is included in the `codeconn.c` source located in the solution directory for this exercise.

```

111  /*****
112  /* Code added for Exercise 7                                     */
113  /*****
114  cno.Version = MQCNO_VERSION_5;
115  strncpy(cltConn.ChannelName, chlName, (size_t)MQ_CHANNEL_NAME_LENGTH);
116  strncpy(cltConn.ConnectionName, connName, (size_t)MQ_CONN_NAME_LENGTH);
117  strncpy(QMName, qmgrName, (size_t)MQ_Q_MGR_NAME_LENGTH);
118  cno.ClientConnPtr = &cltConn; /* point MQCNO to MQCD definition */
119  /* Optional Exercise 7 displays */
120  printf("Channel name set to %s\n", cltConn.ChannelName);
121  printf("Connname set to %s\n", cltConn.ConnectionName);
122  printf("MQCD Queue manager name set to %s\n", cltConn.QMgrName);
123  printf("Queue manager name set to %s\n", QMName);
124  printf("MQCD Version is %d\n", cltConn.Version);
125  printf("MQCD trptype is %d\n", cltConn.TransportType);
126  printf("MQCD chlname is %d\n", cltConn.ChannelType);
127  printf("MQCNO ClientConnPtr is set to %x\n", cno.ClientConnPtr);

```

## Section 14: Save changes, and compile your code

- \_\_ 41. Save your programming changes.
- \_\_ 42. Compile your new application by using the method appropriate for the operating system that you selected.



### Warning

When you compile codeconn.c, you might see the following compile warning. For purposes of this lab exercise, you can safely ignore the warning. It is expected.

The warning that is shown is for Windows. Linux might give a similar warning. **If your module is created in the compile, proceed with the next step and ignore the warning.**

```

codeconn.c
codeconn.c(120): warning C4477: 'printf' : format string '%08X' requires an
argu
ment of type 'unsigned int', but variadic argument 1 has type 'MQPTR'
Microsoft (R) Incremental Linker Version 14.00.23506.0
Copyright (C) Microsoft Corporation. All rights reserved.

```

```

/out:codeconn.exe ← Module is created
codeconn.obj

```





## Linux

**Ensure that the link library that is used is the client library; it is different from previous compiles.**

```
gcc -I/opt/mqm/inc -lmqic -o codeconn codeconn.c
```



## Windows

**Ensure that the link library that is used is the client library; it is different from previous compiles.**

- Ensure that you are at a Developer Command Prompt, and that you are positioned at directory `C:\LabFiles\Lab7\source` in the Developer Command Prompt.
- Type:

```
cl codeconn.c C:\PROGRA~1\IBM\WEBSPH~1\Tools\Lib\mqic.lib
```

## Section 15: Test the `codeconn` application



### Important

To demonstrate how different types of connections override other connections in the IBM MQ client, **it is important that you use the same command prompt terminal** that you used for the CCDT and MQSERVER tests.

- The CCDT connection was to queue manager MQ01.
- The MQSERVER connection changed to queue manager MQ03.
- The `codeconn` application, which built its own channel in the MQCONN call by using the MQCD structure, should go to queue manager MQ01.

- \_\_\_ 43. Test your `codeconn` client application by typing the command in the box. Results that are shown assume that the display statements were included in the code.
- \_\_\_ a. If you do not get the “target queue is EX7.LOCALQ” message displayed as shown in the next text box, look for an error code.
- \_\_\_ b. If you do encounter an error, ensure that you determine whether the error is in the MQCONN call, or in the MQOPEN call.
- \_\_\_ c. If the application stops by presenting the “target queue is...” message as shown in the text box, your code is waiting for the MQPUT. You need to type a message as directed in the **next numeric step**.

codeconn EX7.LOCALQ

Expected results:

Lab source putmsg start  
Channel name set to EX7.CLNTCONN  
Connname set to localhost(1621)  
MQCD Queue manager name set to  
Queue manager name set to MQ01 *<== Refers to queue manager set in MQOD*  
MQCD Version is 6  
MQCD trptype is 2  
MQCD chltype is 6  
MQCNO ClientConnPtr is set to 002DF070  
target queue is EX7.LOCALQ  
target queue manager is MQ01 *<== MQCD connection superseded MQSERVER*

\_\_ 44. If your program did not return errors and paused after the display “target queue is EX7.LOCALQ”, type the suggested message:

codeconn app message should go to MQ01

\_\_ 45. Press the Enter key two times to end your program. The expected result is:

codeconn app message should go to MQ01  
Course PUTMSG end



**Note**

Remember, the `codeconn.c` code is a copy of the `putmsg.c` source. For purposes of expediting this exercise, the displayed application name was left unchanged, so it shows `PUTMSG`

**Section 16: Confirm that the message put with the `codeconn` application was put to queue manager MQ01**

\_\_ 46. Use the `amqsget` program to confirm that the message from the application was put to the queue in MQ01.

- \_\_ a. Type the command as shown in the text box.
- \_\_ b. The expected results are shown in text box.

- \_\_ c. If you prefer to interrupt the wait for extra messages, you can type Ctrl-C to end `amqsget`.

```
amqsget EX7.LOCALQ MQ01
```

**Expected results:**

```
Sample AMQSGET0 start
message <codeconn app message should go to MQ01>
^C ← typed Ctrl-C to end wait for extra messages
```

### **Section 17: Use `putmsgc` to put another message to queue EX7.LOCALQ**

- \_\_ 47. Based on the completed earlier in this exercise, to which queue manager do you anticipate that this message goes to? Write your answer: \_\_\_\_\_
- \_\_ 48. Type the command as shown in the text box to put a message to local queue `EX7.LOCALQ`. Expected results are shown:

```
putmsgc EX7.LOCALQ
```

**Expected response:**

```
Sample PUTMSG start
target queue is EX7.LOCALQ
```

The application is now waiting for messages to be typed.

- \_\_ 49. Type message “where is this message going” and press the Enter key two times to end the application.
- \_\_ 50. What queue manager did you anticipate to be the target of this message? If you guessed `MQ03`, that should be correct because the `MQSERVER` variable is set to use `MQ03`. The `MQSERVER` variable supersedes the client connection table, which is set to `MQ01`.
- \_\_ 51. Confirm results by using the `amqsget` application, as shown in the box. Expected results are included in the display.

```
amqsget EX7.LOCALQ MQ03
```

Expected results:

Sample AMQSGET0 start

message <where is this message going?>

^C



### Information

If you clear the MQSERVER variable by setting it to blanks, the `putmsg` test returns to queue manager MQ01. The extra test is omitted to allow time to complete other labs.

***You completed Exercise 7.***

---

## Troubleshooting

### 2035 return code when testing a client

When you work with IBM MQ connections, you should have channel authentication rules in place. However, channel authentication rules are a topic for an IBM MQ administration course.

To expedite the work, CHLAUTH must be disabled for the queue managers in this course.

- \_\_ 1. Use the `runmqsc` utility to check the queue manager by typing `runmqsc #####` where `#####` is a placeholder for the queue manager name.
- \_\_ 2. Type the command `dis qmgr chlauth` followed by the Enter key in a `runmqsc` session to display the setting of channel authentication for the queue manager.
- \_\_ 3. If channel authentication (CHLAUTH) is enabled:
  - \_\_ a. Disable channel authentication by typing:  

```
alter qmgr chlauth(disabled)
```

and pressing the Enter key.
  - \_\_ b. Confirm that the command completed successfully with message:  

```
AMQ8005: WebSphere MQ queue manager changed
```
  - \_\_ c. Repeat the `putmsgc` test with MQSERVER; it should now work.
- \_\_ 4. If channel authentication is disabled, go to the queue manager error logs according to instructions for the operating system you use, and starting from the end of the file, look for the reason the connection failed.

## Exercise review and wrap-up

In this exercise you learned:

- How to compile IBM MQ client applications
- Three different ways to connect an IBM MQ client application to an IBM queue manager:
  - By using the client channel connection table (CCDT) and a `CLNTCONN` type channel defined on the client side with `runmqsc -n`
  - By using the `MQSERVER` environment variable
  - By using the `MQCD` structure and `MQCONN` call to code the channel definition in the client application
- How to code an `MQCONN` function call, and create the connection by using the `MQCD` structure
- How to check and set the different structure version numbers to ensure that the code works as expected
- How some connectivity options supersede other connectivity options, and the order in which IBM MQ client connections are superseded by testing and confirming the three connectivity scenarios

# Exercise 8. Working with publish/subscribe basics

## What this exercise is about

In this exercise, you learn how to use the integrated publish/subscribe API to complete the code in a subscriber application. As part of the exercise, you convert the `putmsg.c` program to publish messages to a topic instead of putting messages to a queue.

## What you should be able to do

After completing this exercise, you should be able to:

- Add code to an application to use the subscription descriptor and the `MQSUB` function call to create a subscription
- Explain how to pass the managed queue handle from the `MQSUB` function call to access the managed queue in a subsequent `MQGET` function call
- Convert an application that puts messages to a queue to publish messages to a topic
- Use IBM MQ Explorer to obtain more details about your subscription

## Introduction

In the first part of this exercise, you use integrated publish/subscribe and the native API to complete the code in a subscriber application. Next, you use a copy of the `putmsg.c` application and convert it to publish to a topic. You also review the IBM MQ Explorer publish/subscribe capabilities.

## Requirements

- Queue manager `MQ01`
- Partially coded subscriber application `ex8sub.c` at `<dir>/LabFiles/Lab8/source`
- A copy of the `putmsg.c` application, renamed `ex8pub.c` at `<dir>/LabFiles/Lab8/source`
- Solution source for both applications at `<dir>/LabFiles/Lab8/solution`

## Exercise instructions

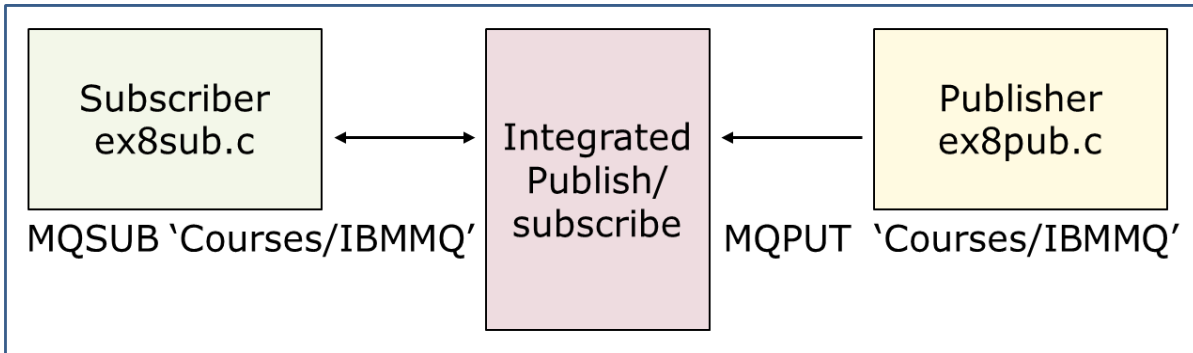


Figure 8-1. Subscriber and publisher applications for Exercise 8



### Information

In this exercise, application `ex8sub.c` subscribes and gets messages that are associated with a topic, and application `ex8pub.c` publishes to a topic.

You use two terminal command prompt screens. One screen is for the subscriber, which you start first, and the second screen is used to publish.



### Important

When you test these two applications, **be consistent about typing the topic to subscribe or publish to the same exact topic**, in quotation marks. If the topics do not match exactly, the publications do not find the corresponding subscription.

In this exercise, use topic “Course/IBMMQ” and queue manager `MQ01` as input parameters for both applications: `ex8sub` and `ex8pub`.

## Section 1: Back up the source code for both applications in this exercise

- \_\_\_ 1. Open a terminal command screen and proceed to the `<dir>/LabFiles/Lab8/source` directory.
- \_\_\_ 2. Make a backup copy of the two base source files for this exercise by using the copy method appropriate to the operating system you are using.
  - \_\_\_ a. Copy subscriber source member `ex8sub.c` to a backup file name, such as:  
`ex8subBkp.c`
  - \_\_\_ b. Copy publisher source member `ex8pub.c` to a backup file name, such as:  
`ex8pubBkp.c`



## Complete the code in a subscriber application



### Information

In this part of the exercise, you complete the key parts of the code that is required to use the `MQSUB` function call. You also follow the program to see how the `MQGET` uses the handle that is provided by the `MQSUB` call to use a managed queue in the application.

### Section 2: Review and change the `ex8sub.c` application

- \_\_\_ 3. Open application source `ex8sub.c` for edit by using the editor applicable to the operating system you are using for this course.
- \_\_\_ 4. Search for string “Exercise 8” one time. It should take you to the subscription descriptor definition, as shown in the text box. Review the definition and default value setting.

```
/* Exercise 8 No changes needed - Review only */
MQSD      sd = {MQSD_DEFAULT};    /* Subscription Descriptor */
```

- \_\_\_ 5. Search on string “Exercise 8” again. This search should take you to the `MQCONN` call at approximately line 93. `MQCONN` needs no changes; it is the same as for point-to-point IBM MQ messaging style, and returns connection handle `Hcon`.
- \_\_\_ 6. Repeat the search on string “Exercise 8”. You now need to type to complete the partial code. An example of the code is included in the text box that follows this set of steps. **After you type your code, stay in the same place in the source.** For the first part of your additions, set `MQSD` options as follows:
  - \_\_\_ a. Create a subscription where the queue manager handles storage of the messages.
  - \_\_\_ b. In addition, the options should specify that the subscription is not durable, and request that the call fails if the queue manager is quiescing.

```
sd.Options = MQSO_CREATE
             | MQSO_NON_DURABLE
             | MQSO_FAIL_IF QUIESCING
             | MQSO_MANAGED;
```

- \_\_ 7. Directly below the line where you set the subscription options, for the second part of your code, you need to complete the `MQSUB` call parameters. An example of the code is shown in the text box that follows these substeps.

```
MQSUB(Hcon, /* connection handle          */
      &sd, /* object descriptor for queue */
      &Hobj, /* object handle (output) */
      &Hsub, /* object handle (output) */
      &S_CompCode, /* completion code */
      &Reason); /* reason code */
```

- \_\_ 8. Save your code, but keep the source open.
- \_\_ 9. Search for the next occurrence of string “Exercise 8”. This search should bring you to the `MQGET` call at approximately line 220. Notice how the `Hobj` handle that is used in the `MQSUB` call is used for the `MQGET`. It represents the managed storage for the subscription-related messages.
- \_\_ 10. Search for the last occurrence of string “Exercise 8”. You should be right above the `MQCLOSE` function call. Note how the `MQCLOSE` is for the topic handle `Hsub`.
- \_\_ 11. Close the `ex8sub.c` application source file.

### Section 3: Compile and do a preliminary test of `ex8sub.c`

- \_\_ 12. Compile your `ex8sub.c` code according to the operating system you are using.



#### Linux

Ensure that you are positioned at directory `C:\LabFiles\Lab8\source`.

```
gcc -I/opt/mqm/inc -lmqm -o ex8subx ex8sub.c
```



#### Windows

- Ensure that you are at a Developer Command Prompt, and that you are positioned at directory `C:\LabFiles\Lab8\source` in the Developer Command Prompt.
- Type:

```
cl ex8sub.c C:\PROGRA~1\IBM\WEBSPH~1\Tools\Lib\mqm.lib
```

- \_\_ 13. If your results are satisfactory, proceed to the next step. If any errors are found, resolve any issues before you proceed to the next step.
- \_\_ 14. Although you do not have your publisher, you can test the `ex8req` subscriber and allow it to time out waiting for publications. To test, type the application named followed by the topic and queue manager as shown in the text box. The application times out after it waits for publications for 45 seconds.

```
ex8sub 'Courses/MQDev' MQ01
```

Expected reply:

```
Sample ex8sub.c start
Calling MQGET : 45 seconds wait time
no more messages
Sample ex8sub.c end
```

## Convert a copy of the `putmsg.c` application to publish to a topic



### Information

For this part of the exercise, the `putmsg.c` source is copied to source file `ex8pub.c`. The only additions to `ex8pub.c` are some comment lines that contain string “Exercise 8” to indicate either a place in the code where you need to change, or a place in the code to review.

Except for the commented lines, `ex8pub.c` is an exact copy of `putmsg.c`.

## Section 4: Review and make necessary changes to the `ex8pub.c` copy of `putmsg.c`

- \_\_ 15. Open application source `ex8pub.c` for edit by using the editor applicable to the operating system you are using for this course.
- \_\_ 16. Search for string “Exercise 8” one time. It should bring you to approximately source line 135. You should be right below comment “Use parameter as ...” and above comment “Open the target ...”
- \_\_ 17. Change the code between the original comments to take the following actions described. The text box that follows the steps shows an example of how the code should look after your changes:
  - \_\_ a. Remove the code that sets up the queue name and displays its name.
  - \_\_ b. Remove the two “if” statements that check for the number of input parameters, and remove all the actions taken by these “if” statements. Ensure that you remove all associated brackets.

- \_\_ c. Set the `objectString` pointer and length to the topic passed in the first application parameter, as shown in the next text box.
- \_\_ d. Display the contents of the topic string.

**The previous code segment should now look as shown in the text box:**

```

131  /* Use parameter as the name of the target queue          */
132  /*   */
133  /*****
134
135  /* Exercise 8 - remove the setup of the ObjectName      */
136 /* Replace the code to set up the object string with the topic string */
137 /* Remove the two "if" statements and actions for args 5 and 6 */
138
139  od.ObjectString.VSPtr=argv[1];
140  od.ObjectString.VSLength=(MQLONG)strlen(argv[1]);
141  printf("target topic is %s\n", od.ObjectString.VSPtr);
142
143
144  /*****
145  /*   */
146  /* Open the target message queue for output
  
```

- \_\_ 18. Save your code, and search for the next occurrence of the “Exercise 8” string. This search should take you right above the `MQOPEN` function call, at approximately line 163.



**Note**

In the next string search, **be careful where your cursor ends**. You might need to carefully scroll down a couple of lines, as you might still be looking at the line that shows `od.ObjectString.VSPtr=argv[1]`, but your search ended below that code, above the `MQOPEN`.

- \_\_ 19. Below the “Exercise 8” marker, but above the `MQOPEN`, type the code requested in the earlier substeps. The text box at the end of the substeps has an example of the code:
  - \_\_ a. Set the object descriptor structure version to version 4.
  - \_\_ b. Set the object descriptor object type to be a topic.

```

/* Exercise 8 - type your code below this comment line    */
    /* Set the object type and MQOD version                */
    od.Version = MQOD_VERSION_4;
    od.ObjectType = MQOT_TOPIC;

```

- \_\_ 20. Search for string “Exercise 8” again. You should now be above the `MQPUT` function call. When you use publish/subscribe, you use the `MQPUT` function to publish your messages. The difference is that the object to which it is publishing is a topic, not a queue. **No updates are necessary in this step**, only your review of the `MQPUT` function call.
- \_\_ 21. Save your changes and close the `ex8pub.c` application source file.

### Section 5: Compile and test the publisher

- \_\_ 22. Compile the application by using the method appropriate for the operating system that you are using.



#### Linux

```
gcc -I/opt/mqm/inc -lmqm -o ex8pub ex8pub.c
```



#### Windows

- Ensure that you are at a Developer Command Prompt, and that you are positioned at directory `C:\LabFiles\Lab8\source` in the Developer Command Prompt.
- Type:

```
cl ex8pub.c.c C:\PROGRA~1\IBM\WEBSPH~1\Tools\Lib\mqm.lib
```



#### Important

The `ex8pub` application is tested simultaneously with the `ex8sub` application. You use two terminal command prompt screens. `ex8sub` runs for 45 seconds. To test:

- On one terminal, start the subscriber, `ex8sub`. `ex8sub` ends after it waits for publications for 45 seconds.

- On the second terminal, start the publisher, `ex8pub`. Then, type some news about the courses to publish them. Type the Enter key two times after the last message you need to publish to end `ex8pub`.
- Return to the first terminal and view the subscriber that is receiving the publications.

Although you do not need to rush, you need to work expeditiously during the test so that you get to publish your news before the subscriber application ends.

- \_\_ 23. If you have one terminal command prompt screen open, open a second terminal.
- \_\_ 24. On your first terminal, start your subscriber by typing `ex8sub 'Courses/IBMMQ' MQ01`. An example is displayed in the text box.

```
ex8sub 'Courses/IBMMQ' MQ01
```

Expected results:

```
Sample ex8sub.c start
```

```
Calling MQGET : 45 seconds wait time
```

- \_\_ 25. Proceed to the second terminal screen and publish some messages by typing the sequence that is detailed in the substeps. An example is shown in the text box that follows.
- \_\_ a. Type `ex8pub 'Courses/IBMMQ' MQ01` and press the Enter key one time. ***Be careful. The topic that is requested must match the topic of the subscription exactly.***
- \_\_ b. After you see the application start and confirmation of your topic, type some news and press the Enter key one time after each publication.
- \_\_ c. To end `ex8pub`, press the Enter key two times.

```
ex8pub 'Courses/IBMMQ' MQ01
```

Expected return:

```
Lab source g start
```

```
target topic is Courses/IBMMQ
```

```
New courses <== Your typed news to publish
```

```
WM302 IBM MQ V8 Administration for z/OS
```

```
WM312 IBM MQ V8 Advanced Administration for z/OS
```

```
WM507 IBM MQ V8 Application Development <== press the Enter key two times
```

```
Course G end
```

**Note**

For purposes of this exercise, the displayed application name in the `ex8pub.c` source is not changed. Unless you decided to change the displayed names, the original application name displays at the start and end of the test.

- \_\_ 26. Return to the subscriber screen. You should now see the published messages that arrived at the subscriber. An example is shown in the text box display.

```
ex8sub 'Courses/IBMMQ' MQ01
```

```
Sample ex8sub.c start
Calling MQGET : 45 seconds wait time
message <New courses>
Calling MQGET : 45 seconds wait time
message <WM302 IBM MQ V8 Administration for z/OS>
Calling MQGET : 45 seconds wait time
message <WM312 IBM MQ V8 Advanced Administration for z/OS>
Calling MQGET : 45 seconds wait time
message <WM507 IBM MQ V8 Application Development>
Calling MQGET : 45 seconds wait time
no more messages
Sample ex8sub.c end
```

- \_\_ 27. **Leave both terminal command screens open.** You use them in the next section.

## Check information about your subscription

**Note**

When you used the `MQSO_NON_DURABLE` option in the `MQSUB` function, you created a non-durable subscription. In the `ex8sub` application, when the `MQCLOSE` function call that uses subscription object handle runs, the subscription is removed.

***In this section, you must complete the `runmqsc` and IBM MQ Explorer steps while `ex8sub` is running, or the subscription is removed.***

**Section 6: Use the runmqsc utility to check information about the managed storage that the subscription MQSUB creates**

- \_\_\_ 28. In one of the terminal screens that you have open, start a `runmqsc` session by typing:  
`runmqsc MQ01`  
 Leave this session open and move to the second terminal screen.
- \_\_\_ 29. In the second terminal screen, start the subscriber application by typing  
`ex8sub 'Courses/IBMMQ' MQ01` and pressing the Enter key.
- \_\_\_ 30. Return to the first terminal screen and in your `runmqsc` session, type the command  
`dis q(SYSTEM.MANAGED.NDUR*)` and press the Enter key. A **partial** display of the expected response is shown in the text box.

```

dis q(SYSTEM.MANAGED.NDUR*) all

      AMQ8409: Display Queue details.
      QUEUE(SYSTEM.MANAGED.NDURABLE.571E3C5E2017CD09)
      TYPE(QLOCAL)                                ACCTQ(QMGR)
      ALTDATE(2016-05-04)                          ALTTIME(12.27.50)
      ... ..
      CURDEPTH(0)                                  CUSTOM( )
      DEFBIND(OPEN)                                DEFPRTY(0)
      DEFPSIST(NO)                                 DEFRESP(SYNC)
      DEFREADA(YES)                               DEFSOPT(SHARED)
      DEFTYPE(TEMPDYN)
      DESCR(Model for managed queues for non durable subscriptions)
      DISTL(NO)                                    GET(ENABLED)
      ... ..
    
```

- \_\_\_ 31. Review your display. Since your subscription is the only one in the system, it is showing the dynamic queue that the queue manager for your subscription created.
  - \_\_\_ a. By default, the queue name prefix is `SYSTEM.MANAGED.NDURABLE` followed by a differentiating alphanumeric string.
  - \_\_\_ b. Notice that the definition type shows a temporary dynamic queue (`TEMPDYN`), but for the time the queue exists, it is a queue of type `QLOCAL`.

**Section 7: Use IBM MQ Explorer to check your subscription.**

- \_\_\_ 32. Wait until `ex8sub` ends, and start a new subscription by typing  
`ex8sub 'Course/IBMMQ' MQ01` and pressing the Enter key.



- \_\_ 33. Open IBM MQ Explorer. Select as follows:
- \_\_ a. Under the Queue Managers heading, select **MQ01**.
  - \_\_ b. Under the MQ01 heading, select **Subscriptions**.

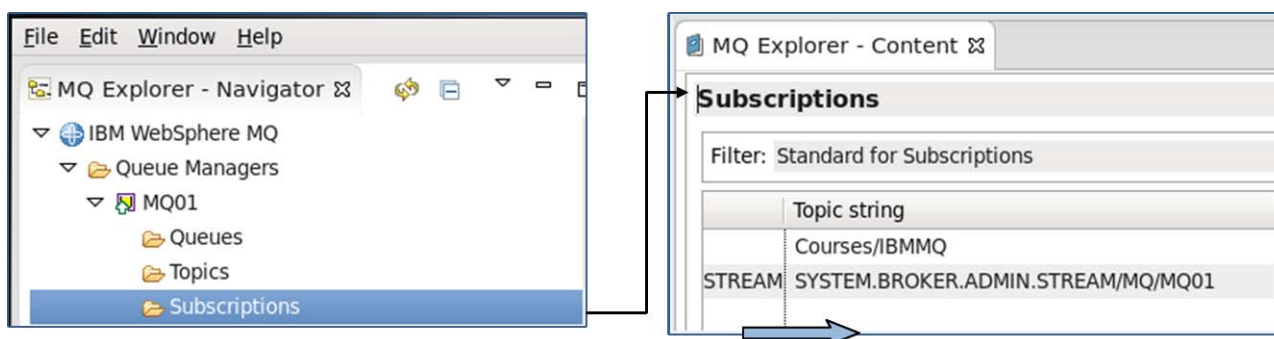


Figure 8-2. IBM MQ Explorer Subscriptions view

- \_\_ 34. Under the subscription pane, find the scroll bar. You might need to scroll to the right until you can view the “Topic string” column. You should see “Courses/IBMMQ” in the Topic string column.
- \_\_ 35. Scroll to the right until you see columns Destination class, Destination name, and Durable. An example of the view is shown in the display.

| Destination class | Destination name                         | Durable |
|-------------------|------------------------------------------|---------|
| Managed           | SYSTEM.MANAGED.NDURABLE.571E2687203E3303 | No      |

Figure 8-3. View of additional subscription detail fields

- \_\_ 36. After you scroll to the right in the subscription pane, you see more details of your subscription:
- Under Destination class, the queue manager manages the storage of the message, and thus the destination is called *Managed*.
  - The name of the dynamic queue in the destination name. If you ran these tests with two different iterations of the `ex8sub` application, the alphanumeric portion of the dynamic queue name is different from the name that the `runmqsc` command displays.
  - The lifetime property of the subscription is non-durable.

**You completed Exercise 8.**

## End of exercise

## Exercise review and wrap-up

In this exercise, you:

- Coded the `MQSUB` function to create a managed, non-durable subscription
- Learned what handle to pass to subscriber `MQGET` calls to receive messages in the managed queue that the `MQSUB` created
- Converted a point-to-point application to a publish/subscribe application
- Used the `runmqsc` utility to show information about the managed queue
- Used IBM MQ Explorer to look at details of the subscription

## Exercise 9. Connecting IBM MQ Light applications to IBM MQ applications

### What this exercise is about

This lab gives you hands-on experience with connecting IBM MQ Light applications to IBM MQ applications. You review the preconfigured AMQP-IBM MQ Light environment, and start the AMQP service and channel. You then use the queue manager as an IBM MQ Light messaging provider, and exchange messages between IBM MQ and IBM MQ Light.

### What you should be able to do

After completing this exercise, you should be able to:

- Examine the IBM MQ Light components configured in the queue manager
- Start and check the status of the AMQP service
- Start and check the status of an AMQP channel
- Use an IBM MQ V8.0.0.4 IBM MQ Light application and the sample node.js IBM MQ Light client application to subscribe to a topic of interest
- Use an IBM MQ V8.0.0.4 IBM MQ Light application and the sample node.js IBM MQ Light client application to publish messages to interested subscribers
- Use a sample IBM MQ publish/subscribe application to publish messages of interest to subscribed IBM MQ and IBM MQ Light node.js applications
- Use a sample IBM MQ publish/subscribe application to subscribe to messages of interest that might proceed from IBM MQ applications or IBM MQ Light publish applications
- Examine the IBM MQ objects that are necessary for a queue-based application to receive messages of interest from an IBM MQ Light message producer application
- Use a sample IBM MQ application to get messages from a queue that an IBM MQ Light application publishes
- Examine the IBM MQ object that is necessary for a queue-based application to put messages on a queue that can go to an interested IBM MQ Light client application
- Use a sample IBM MQ application to put messages to a queue destined to interested IBM MQ Light subscriber applications

## Introduction

This exercise does not require any programming; the programming concepts used in this exercise were completed in earlier labs with MQPUT, MQGET, and publish/subscribe. In this exercise, you use sample IBM MQ Light clients, and sample IBM MQ programs. You also use the IBM MQ Light samples with an extra parameter to use a specific topic string.

You check and start the AMQP components in the queue manager. Next, you test with IBM MQ Light to IBM MQ Light clients, and then you exchange messages between IBM MQ and IBM MQ Light.

## Requirements

- IBM MQ queue manager at V8.0.0.4 with command level converted to 8.0.1 level
- AMQP channel MQ15.AMQP defined
- npm and node.js at version 0.12.9 must be installed (for RHEL 6)
- node.js option of the IBM MQ Light clients
- Other IBM MQ publish/subscribe objects that are used with queue-based applications defined as displayed in the exercise

The applications that are used in this exercise are all sample IBM MQ Light applications and IBM MQ applications.

## Exercise instructions



### Important

If you want to reproduce this work in your environment, read the following material carefully:

Assumption: You have an IBM MQ installation at the IBM MQ V8.0.0.4 manufacturing refresh.

To mitigate dependency on infrastructure services, you can create a new queue manager by using script `SampleMQM.sh` on Linux, or `SampleMQM.bat` on Windows. After the queue manager is created, you need to address the security considerations that are detailed in the next section, **Details of preconfiguration for queue manager MQ15**. Each of the scripts referenced:

- Creates a queue manager that is called `AMQP_SAMPLE_QM`, already converted to command level 801
- Starts service `SYSTEM.AMQP.SERVICE`
- Creates channel `SAMPLE.AMQP.CHANNEL` with port 5672
- Starts channel `SAMPLE.AMQP.CHANNEL`

The script is at the `<mqinstall>/amqp/samples` directory.

All programs that are used are available as IBM MQ publish/subscribe samples, or IBM MQ Light sample `node.js` clients. You can use IBM MQ Light Java, Ruby, or other sample clients and `node.js` by downloading them from the sites noted in the lecture material for this unit.

**For this exercise, a different new queue manager called `MQ15` and channel `MQ15.AMQP` were created to show an alternative to the default configuration.**

You can have more than one AMQP channel, but they must use different ports. In this exercise, you use one channel.



### Warning

**This warning applies to the Linux environment.** If you plan to work with the `node.js` IBM MQ Light client in the RHEL 6 environment, specific requirements must be observed for the `GLIBC` library. For the update level of the RHEL 6 class environment, the `node.js` level had to be lowered to 0.12.9, rather than the most recent version of `node.js`. Failure to use `node.js` at v0.12.9 with RHEL 6 results in the error that is shown in the display:

```
Error: /lib64/libc.so.6: version 'GLIBC_2.14' not found (required by
/opt/mqlight/node_modules/mqlight/lib/node-v46-linux-x64/proton.node)
... ..
```

If you encounter this error with `node.js` and RHEL 6, replace your `node.js` version with `node.js` V0.12.9. The blog entry that follows was posted to assist with the selection of what `node.js` version to use: <https://developer.ibm.com/messaging/2015/12/23/which-version-of-node-js-to-use/>

## Details of preconfiguration for queue manager MQ15



### Important

You use queue manager MQ15 for this exercise. MQ15 is **preconfigured** for this exercise as described in this section.



### Stop

**Do not type these commands! This section is for review only to show the configuration that was done before this exercise.**

Your work starts after the **Section 1** subheading.

- The queue manager is created to use port 1625.

```
crtmqm -p 1625 MQ15 -u SYSTEM.DEAD.LETTER.QUEUE
```

- The command level for MQ15 is converted to 801.

```
strmqm -e CMDLEVEL=801 MQ15
```

WebSphere MQ queue manager 'MQ15' starting.

The queue manager is associated with installation 'Installation1'.

5 log records accessed on queue manager 'MQ15' during the log replay phase.

Log replay for queue manager 'MQ15' complete.

Transaction manager state recovered for queue manager 'MQ15'.

Migrating objects for queue manager 'MQ15'.

Default objects statistics : 3 created. 0 replaced. 0 failed.

New functions up to command level 801 enabled. **<== -e option converted CMD**



### Note

The security-related steps in the next three bullets are for purposes of the class environment. The class environment security settings **are not, and should not be, construed as optimal**. Authentication information and channel authentication should be set according to the security standards set by the organization. Contrary to the values used for this course, the channel MCAUSER value should be set to a low privilege user.

- For class purposes, the operating system (IDPWOS) authentication information object was set so that client credential checks (CHCKCLNT) are omitted:

**BEFORE:**

```
dis authinfo(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) chcklocl chckclnt
3: dis authinfo(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) chcklocl chckclnt
AMQ8566: Display authentication information details.
    AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS)
    AUTHTYPE(IDPWOS)                CHCKCLNT(REQDADM)
    CHCKLOCL(OPTIONAL)
```

**AFTER:**

```
4: ALTER authinfo(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) authtype(IDPWOS)
    chckclnt(NONE) chcklocl(NONE)
AMQ8567: WebSphere MQ authentication information changed.
```

- Disable channel authentication in the queue manager:
 

```
alter qmgr chlauth(disabled)
5: ALTER qmgr chlauth(disabled)
AMQ8005: WebSphere MQ queue manager changed.
```
- Refresh queue manager security
 

```
5: refresh security(*)
AMQ8560: WebSphere MQ security cache refreshed.
```
- A channel of type AMQP is defined with port 5675.

**Warning**

**You must use a valid MCAUSER when you define an AMQP channel.** Failure to specify a valid user might result in FFDC dumps for V8.0.0.4.

**Linux**

```
def chl(MQ15.AMQP) chlttype(AMQP) port(5675) mcauser('mqadmin8')
6: def chl(MQ15.AMQP) chlttype(AMQP) port(5675) mcauser('mqadmin8')
AMQ8014: WebSphere MQ channel created.
```

**Windows**

```
def chl(MQ15.AMQP) chlttype(AMQP) port(5675) mcauser('MUSR_MQADMIN')
6: def chl(MQ15.AMQP) chlttype(AMQP) port(5675) mcauser('MUSR_MQADMIN')
AMQ8014: WebSphere MQ channel created.
```



**Note**

This note marks the place where you start your work in this exercise.

**Section 1: Start queue manager MQ15**

- \_\_ 1. Open a terminal command window. If you are working with Linux, remember to change to the mqadmin8 user.
- \_\_ 2. Type `dsqm` and press the Enter key to display the queue manager status.
- \_\_ 3. If queue managers MQ01 and MQ03 are running, stop them by typing the command for each queue manager:  

```
endmqm MQ01  
endmqm MQ03
```
- \_\_ 4. Type the command to start queue manager MQ15, followed by the Enter key, and wait a few seconds until it starts:

```
strmqm MQ15
```

**Section 2: Confirm that the AMQP capability is enabled in the queue manager**



**Information**

Unless otherwise noted, it is assumed that you press the Enter key after you complete to type each command in this exercise.

The steps that are not enclosed in Linux or Windows boxes apply to **both** Linux and Windows environments.

When you encounter steps that are enclosed in a Linux or Windows box, for this course select the instructions in the Linux box.





## Linux

This box has instructions to **open a terminal window** for Linux. The instructions are not repeated. When you work with the Linux image, **further instructions to open a terminal window assume that you change over to the mqadmin8 user**, or your IBM MQ command path is not set in the new window.

- Open a terminal command window by selecting the terminal icon from your Linux desktop. When you first open the terminal command window, you are the `root` user.
- Change to user `mqadmin8` by typing: `su - mqadmin8`
- The expected results are:  

```
[root@rhel6base ~]# su - mqadmin8
PATH set at
/usr/lib64/qt-3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/s
bin:/sbin:/home/mqadmin8/bin:/opt/mqm/samp/bin
```

You are in your home directory at `/home/mqadmin8`



## Windows

Open a command terminal window by selecting the command prompt icon from the taskbar at the bottom of your VMware screen.

5. If you skipped reading section **Details of preconfiguration for queue manager MQ15** located directly above these steps, review that section now. Your work picks up with those items that are completed.
6. Open an MQSC command window for queue manager MQ15 by typing:  

```
runmqsc MQ15
```

The expected response is:  
5724-H72 (C) Copyright IBM Corp. 1994, 2015.  
Starting MQSC for queue manager MQ15.
7. **Keep the `runmqsc` session open until you come to the step that explicitly instructs to end it.**
8. Type the command:  

```
dis qmgr amqpcap
```

Expected output is `QMNAME(MQ15) AMQPCAP(YES)`

- \_\_ 9. Ensure that the AMQP service is present by typing:

```
dis service(SYSTEM.AMQ*)
```

The expected result is:

```
2 : dis service(SYSTEM.AMQ*)
AMQ8629: Display service information details.
SERVICE(SYSTEM.AMQP.SERVICE)
```

- \_\_ 10. Check that the AMQP channels are defined in the queue manager by typing the command that is shown. This command should display the default AMQP channel, the MQ15.AMQP channel that is defined for this class, and the port assigned to each channel:

```
dis chl(*) chltype(AMQP) port
```

The expected results are:

```
3 : dis chl(*) where(CHLTYPE EQ AMQP) port
AMQ8414: Display Channel details.
CHANNEL(MQ15.AMQP)                CHLTYPE(AMQP)
PORT(5675)
AMQ8414: Display Channel details.
CHANNEL(SYSTEM.DEF.AMQP)          CHLTYPE(AMQP)
PORT(5672)
```

Notice how the default AMQP channel is defined for port 5672, although it can be changed.

- \_\_ 11. Keep the `runmqsc` session open.



### Note

It is worthwhile to go through the tasks to confirm that **all** AMQP components are present as expected in the queue manager. However, the ability to define a channel of type AMQP is one indication that AMQP capabilities are at least partially available.

## Section 3: Start and check the AMQP service and channel MQ15.AMQP

- \_\_ 12. Start the AMQP service by typing:

```
start service(SYSTEM.AMQP.SERVICE)
```

The expected response is:

```
4 : start service(SYSTEM.AMQP.SERVICE)
AMQ8733: Request to start Service accepted.
```

- \_\_ 13. Display the `SYSTEM.AMQP.SERVICE` status by typing:

```
dis svstatus(SYSTEM.AMQP.SERVICE)
```

The expected response should denote that the **status** is **running**. Part of the information that is displayed contains platform-specific details such as the location of the AMQP command and log files:



## Linux

The result in Linux is:

```
6 : dis svstatus(SYSTEM.AMQP.SERVICE)
AMQ8632: Display service status details.
SERVICE(SYSTEM.AMQP.SERVICE)           STATUS (RUNNING)
PID(309579)                               SERVTYPE (SERVER)
STARTDA(2016-01-14)                       STARTTI (17.43.28)
CONTROL(MANUAL)                           STARTCMD(/opt/mqm/bin/amqp.sh)
STARTARG(start -m MQ15 -d "/var/mqm/qmgrs/MQ15/." -g "/var/mqm/." )
STOPCMD(/opt/mqm/bin/amqp.sh)
STOPARG(stop -m MQ15 -d "/var/mqm/qmgrs/MQ15/." -g "/var/mqm/." )
DESCR(Manages clients that use the AMQP protocol)
STDOUT(/var/mqm/qmgrs/MQ15/amqp.stdout)
STDERR(/var/mqm/qmgrs/MQ15/amqp.stderr)
```



## Windows

The result in Windows is:

```
5 : dis svstatus(SYSTEM.AMQP.SERVICE)
AMQ8632: Display service status details.
SERVICE(SYSTEM.AMQP.SERVICE)           STATUS (RUNNING)
PID(2204)                               SERVTYPE (SERVER)
STARTDA(2016-01-21)                       STARTTI (13.51.17)
CONTROL(MANUAL)
STARTCMD(C:\Program Files\IBM\WebSphere MQ\bin\amqp.bat)
STARTARG(start -m MQ15 -d "C:\ProgramData\IBM\MQ\qmgrs\MQ15\." -g
"C:\ProgramData\IBM\MQ\." )
STOPCMD(C:\Program Files\IBM\WebSphere MQ\bin\amqp.bat)
STOPARG(stop -m MQ15 -d "C:\ProgramData\IBM\MQ\qmgrs\MQ15\." -g
"C:\ProgramData\IBM\MQ\." )
DESCR(Manages clients that use the AMQP protocol)
STDOUT(C:\ProgramData\IBM\MQ\qmgrs\MQ15\amqp.stdout)
STDERR(C:\ProgramData\IBM\MQ\qmgrs\MQ15\amqp.stderr)
```

14. Start channel MQ15.AMQP by typing:

```
start chl(MQ15.AMQP)
```

Expected results are:

```
2 : start chl(MQ15.AMQP)
```

```
AMQ8018: Start WebSphere MQ channel accepted.
```



## Troubleshooting

If you received the response that is shown in this box:

```
2 : start chl(MQ15.AMQP)
AMQ8494: AMQP commands are not available.
```

The `SYSTEM.AMQP.SERVICE` is not available. Perhaps you restarted the queue manager, and overlooked restarting the AMQP service.

To be able to start the AMQP channel, start the service as shown earlier in this exercise, by typing `start service(SYSTEM.AMQP.SERVICE)`. After the service starts, you should be able to start the channel.

Display the channel status by typing: `dis chs(MQ15.AMQP)`

**Do not be concerned with the response in this step.** You should see:

```
dis chs(MQ15.AMQP)
1 : dis chs(MQ15.AMQP)
AMQ8420: Channel Status not found
```



## Note

If you omit the `chltype` attribute when you display the status of an AMQP channel, the information is not found. You must include `chltype(AMQP)` to get the status of an AMQP channel.

\_\_\_ 15. Display the channel status by typing: `dis chs(MQ15.AMQP) chltype(AMQP)`

The response should now display the channel in **running** status, as shown:

```
dis chs(MQ15.AMQP) chltype(AMQP)
2 : dis chs(MQ15.AMQP) chltype(AMQP)
AMQ8417: Display Channel Status details.
CHANNEL(MQ15.AMQP)                CHLTYPE(AMQP)
CONNECTIONS(0)                    STATUS(RUNNING)
```



## Important

Exit `runmqsc`, **but leave your terminal session open.**

\_\_\_ 16. Exit `runmqsc` by typing `end` and pressing the Enter key. You can run some other commands to check your environment.

\_\_\_ 17. Another way to check that the service is running is to check Java processes that are active in the system. The next two steps vary; select the box for the operating system that you are working with:



## Linux

From the terminal window command line, type:

```
ps -ef | grep java
```

Expected results (partial):

```
mqm      50466  50442  0 Jan18 ?          00:00:42
//opt/mqm/amqpbin/../../../../java/jre64/jre/bin/java
-Dcom.ibm.mq.mqxr.service.type=amqp
-Dcom.ibm.msg.client.config.location=file:///var/mqm/qmgrs/MQ15/./amqp/
amqp_trace.config
-Xoptionsfile=/opt/mqm/amqp/bin/./config/amqp_java.properties
-Dcom.ibm.msg.client.commonservices.wmq.logdir=/var/mqm/qmgrs/MQ15/./err
ors
... ..
```

**Leave your terminal session open.**



## Windows

- Right-click the taskbar and select **Start Task Manager**. After the Windows Task Manager pane surfaces, select the **Processes** tab. Find the Java process that runs under the IBM MQ user for Windows (you see `USER_MQADMIN` under the User Name column).
- Right-click the Java process, and you see that this process is running from the IBM MQ libraries.
- Close the Windows Task Manager.



## Note

The work that is done in the first set of sections uses **IBM MQ** as the messaging provider for the **IBM MQ Light** clients.

## Section 4: Set up your windows session to work with the IBM MQ Light clients



### Linux

You use several Linux terminal command windows in the rest of this exercise. To prevent confusion, name each session.

- \_\_ a. If you just opened the new window, remember to change over to the `mqadmin8` user, that is: `su - mqadmin8`
- \_\_ b. On your terminal window, access the Terminal menu.
- \_\_ c. Select **Set Title** and press the Enter key.
- \_\_ d. Type an appropriate title such as `amqp subscriber` and press the Enter key. Your terminal session should display the name that you typed across the top.
  - Change to the directory that holds the IBM MQ Light node.js samples by typing:
 

```
cd /opt/mqlight/node_modules/mqlight/samples
```
  - Confirm that you are in the correct directory by typing:
 

```
pwd
```

Expected result is display of directory name:

```
/opt/mqlight/node_modules/mqlight/samples
```



### Windows

**The instructions that are provided in this text box are not repeated in later steps. If you need a reminder, refer to this section.**

- You use several Windows terminal command screens in the rest of this exercise. To open extra command prompt windows:
  - Place the cursor over the command prompt icon in the bottom taskbar.
  - Right-click and select **Command Prompt**.
- To mitigate confusion, change the screen background color, and change the text to match so that you can clearly see your work. You can leave your first command prompt window with a black screen background and white screen text.
- You might, for example, have your two IBM MQ Light windows with a dark background color and white text. Then, have the two IBM MQ windows with a light background color and dark text.
- For the other command prompt screens:
  - Right-click across the top of the command prompt, and select **Properties**.

- Select the **Colors** tab, the **Screen Background** button. Select a different color for each screen.
- Check the contrast between the screen text and screen background. If you have background colors that blend with text, change the screen text color so it is clearly visible.
- **For the terminal screens that are used for the IBM MQ Light sample clients:**
  - Change to the directory that holds the IBM MQ Light node.js samples by typing:
 

```
cd C:\mq\light\cli\t\node_modules\mq\light\samples
```
  - Check the new path name in your windows command screen to ensure that the directory change proceeded correctly. Your command prompt should look as shown:
 

```
C:\mq\light\cli\t\node_modules\mq\light\samples
```
- If you have a Developer Command Prompt for VS2015 window open, close it. You do not use the compiler in this exercise.

**These instructions to set the IBM MQ Light client directory are not repeated.** If you need a reminder, return to this section.

## Section 5: Use the `recv.js` IBM MQ Light sample to subscribe to the default topic

- \_\_ 18. Start your `recv.js` subscriber. Ensure that you specify the port that the `amqp` channel is listening for. Type the command as shown.



### Warning

After you type the command that follows, the terminal window for the `amqp` subscriber session has a running process. **Do not type enter or try to exit the process; it is supposed to remain active.** You use a new terminal window for the other steps.

```
node recv.js -s amqp://localhost:5675
```

Expected result is:

```
Connected to amqp://localhost:5675 using client-id recv_98276eb
Subscribed to pattern: public
```

- \_\_ 19. Review these two points in the previous output:
- \_\_ a. Notice that you have a client ID. **Your resulting client ID is expected to be different.** For the example shown in this step, the client ID is `recv_98276eb`.
  - \_\_ b. Write the client ID from your subscription here: \_\_\_\_\_
  - \_\_ c. The topic that you are subscribed to is `public`, which is an initial default value in the sample `recv.js` client.



## Troubleshooting

If you received the error that is shown in this box, check for an error in the value that is provided in the `amqp` property, and the subscriber does not start. **Do not use this example, it contains the error!**

```
node recv.js -s amqp://localhost:5673
*** error ***
message: NetworkError: CONNECTION ERROR (localhost:5673): Connect failure:
The remote computer refused the network connection.
Exiting.
```

The result that is shown is an error that was made when testing this lab. The port number is incorrect. It should be 5675 for the definition in this exercise. No accompanying messages show in the `amqp` logs for this error. If you get this error, carefully check what you typed following the `-s` flag.

## Section 6: Open a second terminal window and check the subscription

\_\_ 20. Open a new terminal window by using the same process that is described in the previous step. If you are using Linux, you can name this terminal: `amqp publisher`

If you are using Windows, set the window background to a different color as described previously. You use this terminal later to send messages.



## Linux

Remember to change over to the `mqadmin8` user by typing: `su - mqadmin8`

\_\_ 21. Start a `runmqsc` session by typing:

```
runmqsc MQ15
```

\_\_ 22. After you are in the `runmqsc` session, display the existing subscriptions in your queue manager by typing the command that is shown:

```
dis sub(*) topicstr sub
```

\_\_ 23. More than one subscription might be displayed, depending on the work done previously. **You need to find the subscription for the “public” topic:**

```
AMQ8096: WebSphere MQ subscription inquired.
SUBID(414D51204D5131352020202020202020F16CA15620002804)
SUB(:private:recv_98276eb:public)          TOPICSTR(public) <===
AMQ8096: WebSphere MQ subscription inquired.
SUBID(414D51204D51313520202020202020205B4BA15620002204)
SUB(SOCCER.INFO)                          TOPICSTR(Sports/soccer)T
```



- \_\_ 24. Compare the SUB client that is shown in the display with the client ID in your `amqp subscriber` window. The subscription identifier that is displayed in the `DIS SUB` command should match the client ID in your subscriber terminal window. In the example that is used in this lab, the client ID is `recv_98276eb`.
- \_\_ 25. Exit the `runmqsc` session by typing `end` and pressing the Enter key.

## Section 7: Use the second terminal window to publish the `send.js` IBM MQ Light client sample default message



### Linux

- Change to the directory that holds the IBM MQ Light node.js samples by typing:  
`cd /opt/mqlight/node_modules/mqlight/samples`



### Windows

- Change to the directory that holds the IBM MQ Light node.js samples by typing:  
`cd C:\mqlightclt\node_modules\mqlight\samples`

- \_\_ 26. Publish a message by using the `send.js` sample. Type the command as shown.

```
node send.js -s amqp://localhost:5675
```

The expected result is:

```
Connected to amqp://localhost:5675 using client-id send_879eb11
Sending to: public
Hello World!
stopping client
```

“Stopping client” refers to the `send.js` client. This client uses a different client ID.

- \_\_ 27. Return to the subscriber screen named `amqp subscriber`. It should now show that it received the “Hello World” message.
- \_\_ 28. End the process in your `amqp subscriber` terminal window by holding the CTRL key down, then pressing the C (CTRL-C). **Keep the `amqp subscriber` terminal window open.**

**Note**

The lecture that preceded this exercise showed a few different IBM MQ Light - AMQP scenarios:

- IBM MQ used as the messaging provider for IBM MQ Light and AMQP applications
- IBM MQ applications and IBM MQ Light exchanging messages
- IBM MQ Light exchanging information with IBM MQ queue-based applications

In the previous step, you tested the first scenario by using IBM MQ as the messaging provider for IBM MQ Light applications that exchange messages (recv.js and send.js).

You continue with the second scenario and use IBM MQ and IBM MQ Light sample programs.

The IBM MQ sample program that is used to subscribe, `amqssub`, runs for 30 seconds. You need to have your terminal windows ready and publish or send within the 30 seconds. You use four terminals: the two you have now and two new terminal windows to run IBM MQ samples.

When you test with the `amqssub` sample program, type the commands in all publisher/sender terminal windows as needed, but **do not press** the Enter key until **right after** you start the `amqssub` subscriber. Waiting to press the Enter key mitigates the possibility that `amqssub` might end before your publications are sent.

**Hint**

In the next set of instructions, when you type the topic string, take care to type the exact same string, including the double quotation marks, or the subscriptions might not match the publications.

## **Section 8: Set up to exchange messages between IBM MQ applications and IBM MQ Light applications**

- \_\_\_ 29. Follow either the Linux or Windows processes previously outlined to set up **two** terminal windows: the IBM MQ publisher window, and the IBM MQ subscriber window.
- \_\_\_ a. Ensure that your path is set up to type IBM MQ requests. You can try typing `amqssub` by itself and pressing the Enter key. **If the response is a usage type error that states that you need to supply a topic string, the path is set correctly.**
  - \_\_\_ b. You **do not** need to change over to the IBM MQ Light libraries.
  - \_\_\_ c. Differentiate each terminal window by either providing a different name for Linux terminals, or a different color for Windows terminals. For example, you might want to select one color for the terminals with the path to the IBM MQ Light `node.js` samples, and leave the terminals used for IBM MQ with the original color.

**Information**

The IBM MQ Light node.js applications take several parameters and use initial default values for the remaining properties. You already used the `-s` identifier to pass the host and port number to the application. In the next set of testing, you use the `-t` identifier to provide a topic string.

To see the usage information for the `recv.js` or `send.js` samples, use your `amqp publisher` window and start either sample followed by `-h`. No spaces are between the “-” and the “h”. The usage for both samples is similar. For example, to see the usage information for the `send.js` sample, type the command as shown:

```
node send.js -h
```

**Note**

The work that is done in this second set of sections exchanges messages between IBM MQ applications and IBM MQ Light applications.

**Warning**

Be careful when you type to ensure that you use the correct syntax, and double check what you typed before you press the Enter key. By taking due time in your commands, you save time that is wasted to resolve problems.

## Section 9: Create an IBM MQ Light subscription to topic “Sports/soccer”

- \_\_ 30. Proceed to the terminal window used for your `amqp subscriber`.
- \_\_ 31. If the subscriber process used earlier is still running, hold down the Ctrl key and type a `c` (Ctrl-C) to end the first `recv.js` process.
- \_\_ 32. Use the `recv.js` sample to register your interest for topic string "Sports/soccer". Type the command exactly as shown, and **ensure that you leave the application in running status**. Keep all typing in a single line, even if it wraps:

```
node recv.js -s amqp://localhost:5675 -t "Sports/soccer"
```

Expected results are:

```
Connected to amqp://localhost:5675 using client-id recv_17dc80b
Subscribed to pattern: Sports/soccer
```

- \_\_ 33. Write down the client ID of your IBM MQ Light client: \_\_\_\_\_



## Troubleshooting

If you get any errors, carefully review how the request is typed. Any minor deviation, such as an extra space between the dash and the option parameter, causes an error and might cause the sample application to display either Usage information or another error, depending on the problem.



## Information

When you start the IBM MQ Light `recv.js` sample, a subscription is created in IBM MQ. This subscription is in addition to the explicitly defined objects that you use later in this exercise.

## Section 10: Use IBM MQ to publish to the IBM MQ Light “Sports/soccer” subscription

- \_\_ 34. Proceed to your terminal window for the IBM MQ publisher.
- \_\_ 35. When you started the `recv.js` client, a subscription was created in IBM MQ. Before you publish any messages, look at the subscription that was created:
- \_\_ a. Open a `runmqsc` session by typing `runmqsc MQ15` and pressing the Enter key.
  - \_\_ b. Type `dis sub(*)` and press the Enter key. Your results should resemble the results in the display.
  - \_\_ c. Compare the client ID of the subscription that is displayed with the client ID that you wrote in the step where you started the `recv.js` subscription. The client ID values should be the same. In the example that is shown in this exercise, the client ID is `recv_17dc80b`.
- ```
AMQ8096: WebSphere MQ subscription inquired.
      SUBID(414D51204D5131352020202020202020200257A65620002749)
      SUB(:private:recv_17dc80b:Sports/soccer)
```
- \_\_ d. Type `end` and press the Enter key to exit `runmqsc`.
- \_\_ 36. Use the `amqspub` sample to publish to topic “Sports/soccer”. Type the command as shown. Ensure that you include both the topic string and queue manager name as input parameters.

```
amqspub "Sports/soccer" MQ15
```

Expected results: `amqspub` starts, confirms the topic that it is publishing to, and waits for input. If you see the results as shown, proceed to the next step to provide input.

```
Sample AMQSPUBA start
target topic is Sports/soccer
```

- \_\_ 37. The subscriber for this application expects the score for the matches played. The IBM MQ publisher is going to provide scores for teams from the United States MLS league.
- \_\_ a. Type `Orlando 4, Miami 0` and press the Enter key one time.
  - \_\_ b. Press the Enter key one more time to end `amqspub`.
- \_\_ 38. Return to the `amqp subscriber` to check whether it received any messages. The expected result is the score that you published with the `amqssub` program: `Orlando 4, Miami 0`.  
**Leave the subscriber application in running status.**

## Section 11: Publish messages to both IBM MQ applications and IBM MQ Light subscribers at the same time



### Important

For this section, you type the IBM MQ and IBM MQ Light publications, but **do not press the Enter key** until after the `amqssub` command starts subscribing to the scores.

IBM MQ is going to publish scores from teams in the MLS. IBM MQ Light is going to publish scores for teams in the Premier League.

The person who types the commands can change the scores, but they should be kept as shown to make it easier to compare the combined results. If each subscriber has scores from both leagues, you know that publications from both IBM MQ and IBM MQ Light reached the subscribers.

You have a total of four windows open:

- The `amqp subscriber`, which is running.
- A window for the `amqp publisher`.
- A window for the IBM MQ publisher, which uses `amqspub`.
- A window for the IBM MQ subscriber. This subscriber runs for 30 seconds after started.

- \_\_ 39. Remember, your `amqp subscriber` is already running.
- \_\_ a. Proceed to the `amqp publisher` terminal window. Type the command that is shown in a continuous line.
  - \_\_ b. To send in one message, use the quotation marks `" "` around the actual data as shown. Otherwise, the `send.js` client publishes four messages.
  - \_\_ c. Use double quotation marks to state the topic name.
  - \_\_ d. **Do not press the Enter key** until instructed:

```
node send.js -s amqp://localhost:5675 -t "Sports/soccer" "Aston Villa 1,
Leicester 1"
```

- \_\_ 40. Proceed to the `IBM MQ publisher` terminal window. Start the publisher by typing the command that is shown and pressing the Enter key **one time**. The publisher should be running, waiting for input. You return to type the score when instructed; leave the publisher started for now:

```
amqspub "Sports/soccer" MQ15
```

Results:

```
Sample AMQSPUBA start  
target topic is Sports/soccer
```



### Important

You must complete the next three steps expeditiously to ensure all the publications reach the **IBM MQ subscriber** before it ends in 30 seconds.

- \_\_ 41. Proceed to your **IBM MQ subscriber** window. Start the IBM MQ subscriber, which runs for 30 seconds by typing the command that is shown, and **leave the subscriber program in running status**:

```
amqssub "Sports/soccer" MQ15
```

Expected results:

```
Sample AMQSSUBA start  
Calling MQGET : 30 seconds wait time
```

- \_\_ 42. Return to the `amqp publisher` window and press the Enter key. If you used the suggested text, you sent the scores from the Premier league.

- \_\_ 43. Return to the `IBM MQ publisher` window, and proceed as follows:

- \_\_ a. Type: `New York City FC 3, Orlando City SC 5`
- \_\_ b. Press the Enter key two times to send the messages, and end `amqspub`. If you used the scores as shown, this step sent the messages from the MLS league.



### Note

You subscribed, and published to topic “Sports/soccer” by using both IBM MQ Light and IBM MQ. You should see scores from the MLS and scores from the Premier League in each subscriber.

- \_\_ 44. Return to the `amqp subscriber` terminal window. You should see the Premier League scores published by IBM MQ Light, and the MLS scores published by IBM MQ. If you used the same scores, your results should look as shown. **Leave the IBM MQ Light subscriber process active after you review the results; do not end the subscriber.**

```
Aston Vila 1, Leicester 1
New York City FC 3, Orlando City SC 5
```

- \_\_ 45. Return to the `IBM MQ subscriber` terminal window. You should see the Premier League scores published by IBM MQ Light, and the MLS scores published by IBM MQ. If you used the same scores, your results should look as shown:

```
Calling MQGET : 30 seconds wait time
message <Aston Vila 1, Leicester 1>
Calling MQGET : 30 seconds wait time
message <New York City FC 3, Orlando City SC 5>
Calling MQGET : 30 seconds wait time
no more messages
Sample AMQSSUBA end
```



### Note

The work in this next set of sections also involves an exchange of messages between IBM MQ Light applications and IBM MQ applications, but for IBM MQ applications that are queue-based. In this sense, queue-based means applications that either put or get messages to and from queues, but do not use publish/subscribe topics. The applications use MQPUT and MQGET.

The queue-based scenario depends on the configuration of certain IBM MQ objects. You use sample `amqspout` and `amqsget` programs, and the IBM MQ Light AMQP clients. You **do not use** the IBM MQ publish/subscribe sample programs.

## Section 12: Use the IBM MQ Light sample client `recv.js` to receive messages that a queue-based application puts



### Information

Messages are put to a queue. The underlying IBM MQ definitions enable the movement from queue to the “Sports/soccer” topic string.

The components that are needed in this scenario are:

- The `amqp subscriber`, `recv.js`
- An IBM MQ alias queue definition that sends messages to an IBM MQ defined topic
- An IBM MQ TOPIC definition
- `amqspout` sample program to put messages to a queue

- \_\_ 46. Check that the `recv.js` subscriber is active in terminal window. **If you need to restart the subscriber:**
- \_\_ a. Ensure that you are at a window at the IBM MQ Light samples directory as done earlier in this exercise
  - \_\_ b. Type `node recv.js -s amqp://localhost:5675 -t "Sports/soccer"` and press the Enter key.
- \_\_ 47. The IBM MQ objects in the next steps **are already predefined**. Proceed to display the objects as indicated. Display the QALIAS object.
- \_\_ a. From a separate terminal window, **other than** the `amqp` subscriber window, start a `runmqsc` session by typing:
 

```
runmqsc MQ15
```
  - \_\_ b. Display the definition of alias queue `TO.SOCCER.TOPIC` by typing the `MQSC` command:
 

```
dis q(TO.SOCCER.TOPIC) targtype target
```

Expected results are:

```
AMQ8409: Display Queue details.
        QUEUE(TO.SOCCER.TOPIC)                TYPE(QALIAS)
        TARGET(SOCCER)                        TARGTYPE(TOPIC)
```
  - \_\_ c. In this definition, the type of target is a topic object, `TARGTYPE(TOPIC)`. The name of the topic object is `SOCCER`, `TARGET(SOCCER)`.
- \_\_ 48. To complete the association of the alias queue to the subscribed client, you need to review the topic object.
- \_\_ a. Display the topic by typing:
 

```
dis topic(SOCCER) topicstr
```

Expected results are:

```
AMQ8633: Display topic details.
        TOPIC(SOCCER)                            TYPE(LOCAL)
        TOPICSTR(Sports/soccer)
```
- \_\_ 49. Type `end` and press the Enter key to exit the `runmqsc` session. **Remain on the same terminal window for the next step.**
- \_\_ 50. Put messages to alias queue `TO.SOCCER.TOPIC` by using sample put program, `amqsput`.
- \_\_ a. Type the command as shown:
 

```
amqsput TO.SOCCER.TOPIC MQ15
```

The program starts and waits for further input as shown:

```
Sample AMQSPUT0 start
target queue is TO.SOCCER.TOPIC
```



- \_\_ b. Type some soccer scores, for example, the scores shown in this step. You can optionally send other text, such as “from amqspu0”.

```
Real Salt Lake 3, Sporting KC 0
```

- \_\_ c. Press the Enter key two times to end the `amqspu0` program. The expected result is:

```
Sample AMQSPU0 end
```

- \_\_ 51. Return to the `amqp subscriber` terminal window. You should see that the IBM MQ client received the published scores. If you used the suggested scores, the received score should be:

```
Real Salt Lake 3, Sporting KC 0
```

### Section 13: Use a queue-based IBM MQ application to receive messages that the IBM MQ Light `send.js` client publishes



#### Information

Messages are published to a topic by IBM MQ Light. The IBM MQ queue-based subscriber gets messages from a queue. The underlying IBM MQ definitions enable the movement from the topic string “Sports/soccer” to the queue.

The components that are needed in this scenario are:

- The `amqp publisher send.js`
- An IBM MQ local queue definition to receive the messages published
- An IBM MQ `SUB` (subscription) object definition to bridge the topic string to a queue
- `amqssub` sample program to get messages from a queue

- \_\_ 52. The IBM MQ objects in the next steps **are already predefined**. Proceed to display the objects as indicated. Display the local queue (QLOCAL) object.

- \_\_ a. From a separate non-subscriber terminal window, start a `runmqsc` session by typing:

```
runmqsc MQ15
```

- \_\_ b. Display queue `SOCCER.SCORES` by typing:

```
dis q(SOCCER.SCORES)
```

Expected results are partially shown; all initial default values were used:

```
AMQ8409: Display Queue details.
```

```
QUEUE(SOCCER.SCORES)
```

```
TYPE(QLOCAL)
```



- \_\_ 56. Get the messages from the SOCCER.SCORES local queue by using the IBM MQ sample `amqsget` program. Type the command as shown. ***It is expected that you get more scores than in the display:***

```
amqsget SOCCER.SCORES MQ15
```

Expected results are:

```
Sample AMQSGET0 start
message <Liverpool 0, Man U 1>
message <Real Salt Lake 3, Sporting KC 0>
... ..
no more messages
Sample AMQSGET0 end
```

- \_\_ 57. Review the outcome. You see that you have more than one score in queue SOCCER.SCORES because the IBM MQ objects were defined before IBM MQ put messages to the alias queue in the previous section.

The message to the alias queue was `Real Salt Lake 3, Sporting KC 0`. The last message that was published to topic “Sports/soccer” contained the score for `Liverpool 0, Man U 1`. The queue for subscription `SOCCER.INFO` also received the messages.



### Warning

If you see more messages than shown in the “Expected results” when you type the `amqsget`, ***do not be concerned***. The “Expected results” shown in the `amqsget` step shows two scores. However, since the objects were defined earlier, you see other messages from previous publications to the same topic.

***You completed Exercise 11.***

## End of exercise

## Exercise review and wrap-up

In this exercise, you exchanged messages between IBM MQ applications and IBM MQ Light applications for three different cases:

- IBM MQ is used as the messaging provider for IBM MQ Light applications and AMQP applications.
- IBM MQ applications and IBM MQ Light applications exchange messages.
- IBM MQ Light applications exchange messages with IBM MQ queue-based applications.

To accomplish the message exchanges, you:

- Examined the IBM MQ Light components configured in the queue manager
- Started and checked the status of the AMQP service
- Started and checked the status of an AMQP channel
- Used IBM MQ V8.0.0.4 IBM MQ Light server and the sample node.js IBM MQ Light client application `recv.js` to subscribe to a topic of interest
- Used IBM MQ V8.0.0.4 IBM MQ Light server and the sample node.js IBM MQ Light client application `send.js` to send information to interested subscribers
- Used the sample IBM MQ publish/subscribe application `amqspub` to publish messages of interest to subscribed IBM MQ applications and IBM MQ Light application clients
- Used the sample IBM MQ publish/subscribe application `amqssub` to subscribe to messages of interest that might proceed from either IBM MQ applications or IBM MQ Light application publications
- Examined the IBM MQ `QLOCAL` and `SUB` objects necessary for a queue-based application to receive messages of interest from an IBM MQ Light application message producer
- Used sample IBM MQ program `amqsget` to get messages, which an IBM MQ Light producer publishes, from a queue
- Examine the IBM MQ `QALIAS` and `TOPIC` objects necessary for a queue-based application to put messages on a queue that can be sent to an interested IBM MQ Light client
- Use a sample IBM MQ application `amqsput` to put messages to a queue destined to interested IBM MQ Light subscriber applications



### Warning

Before any attempt to edit the node.js samples in the **“If you want to learn more”** steps that follow, ensure that you make a copy of the `recv.js` and `send.js` files. If you edit and corrupt the node.js samples without a backup, you might jeopardize the ability to complete this exercise.



## Information

*If you want to learn more* about the node.js program code that is used in this lab exercise, you can edit the `recv.js` and `send.js` samples, and review the code.

If you are interested in using one of other languages other than node.js, you can see the IBM MQ Light developerWorks site at: <https://developer.ibm.com/messaging/mq-light/>

- To view the node.js examples in the **Linux platform**, you can use:
  - The `vi` editor from a terminal window.
  - The `gedit` utility from the desktop. The icon can be found to the far right of the Terminal icon. The `gedit` icon shows as a notepad with a pencil
- To view the node.js examples in the **Windows platform**:
  - Use the notepad++ icon to the left of your desktop.
  - Close any notices to update the notepad++ utility.
  - If you attempt to right-click and use the Open option, sometimes the code attempts to run and shows garbled text. For this reason, the option to use notepad++ is suggested.





