Using a Client Channel Definition Table (CCDT) in WebSphere MQ V7 for Queue Manager Groups

IBM Techdoc: 7020848
http://www.ibm.com/support/docview.wss?uid=swg27020848

Date last updated: 7-Feb-2017

Angel Rivera – rivera@us.ibm.com
IBM WebSphere MQ Support

+++ Objective +++

The objective of this techdoc is to show an example of how to define and use a Client Channel Definition Table (CCDT) in WebSphere MQ V7 (and later versions) for a Queue Manager Group.

The chapters in this techdoc are:

There are 7 scenarios described in this techdoc:

Scenario 1 - Simplest case, using same QMNAME for a single queue manager (QM3 in host1)
Scenario 2 - Simplest case, using same QMNAME for a single queue manager (QM3 in another host: host2)
Scenario 3 - Queue Manager Group: QM1 (no leading asterisk)
Scenario 4 - Queue Manager Group: *QM1 (with leading asterisk)
Scenario 5 - Queue Manager Group: ' ' (1 blank character)
Scenario 6 - Queue Manager Group: QMGROUP1
Scenario 7 - Multi-Instance Queue Manager

Notes:
1: It is necessary to define 2 channels with the SAME name: one server-connection channel (SVRCONN) and one client-connection channel (CLNTCONN).
2: The CCDT file is updated ONLY when the client-connection channel is created or altered.

+ References

http://www.ibm.com/support/docview.wss?uid=swg27024109
Webcast: Using a Client Channel Definition Table (CCDT) in WebSphere MQ V7 for Queue Manager Groups
The objective of this webcast is to describe in details how to exploit from a client application that is using the MQ interface in C or the WebSphere MQ Classes for JMS V7 the connection to multiple queue managers by using an MQ "Client Channel Definition Table" (CCDT).
Level of Difficulty: Intermediate
Date: 21 February 2012

Some of the scenarios are mentioned in the following section of the online manual for MQ will be used:
https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.0.0/com.ibm.mq.dev.doc/q027490_.htm
IBM MQ > IBM MQ 9.0.x > IBM MQ > Developing applications >
Developing MQI applications with IBM MQ >
Writing client procedural applications >
Running applications in the IBM MQ MQI client environment >
Connecting IBM MQ MQI client applications to queue managers >
Role of the client channel definition table >
Queue manager groups

Queue manager groups in the CCDT

The following MQ C-code samples are used to exercise the CCDT functionality:
https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.0.0/com.ibm.mq.dev.doc/q024200_.htm
IBM MQ > IBM MQ 9.0.x > IBM MQ > Developing applications > Developing MQI applications with IBM MQ >
Sample IBM MQ procedural programs > Sample procedural programs (platforms except z/OS) >
The High availability sample programs

“The amqsghac, amqsphac and amqsmhac high availability sample programs use automated client reconnection to demonstrate recovery following the failure of a queue manager.

The programs are started from the command line, and can be used in combination to demonstrate reconnection after the failure of one instance of a multi-instance queue manager.
Alternatively, you can also use the samples to demonstrate client reconnection to single instance queue managers, typically configured into a queue manager group.
“
Only the PUT and GET sample programs are used in this document. Here are more details on the samples:

PUT: amqsphac queueName [qMgrName]
 * Puts a sequence of messages to a queue with a 2-second delay.
 * Displays events sent to its event handler.
 * No sync point is used.
 * Reconnection can be made to any queue manager.

GET: amqsghac queueName [qMgrName]
 * Gets messages from a queue.
 * Displays events sent to its event handler.
 * No sync point is used.
 * Reconnection can be made to any queue manager.

Notes:
+ A "queue manager group" is NOT the same as a "queue-sharing group" (which is only available in MQ on z/OS).

+++ Note about Connection Balancing

The CCDT was created with the default values for the client-connection channels for the following attributes:

AFFINITY(PREFERRED)
CLNTWGHT(0)

The default behavior is that there is no connection balancing and that the MQ client will choose the first channel name that is active from the list of channels in alphabetical order.

If you want to have a certain level of connection balancing, you will need to alter the following attributes for the channels in the CCDT:

AFFINITY(NONE)
CLNTWGHT(1)

The testing that I did with the altered values showed that there was a rough round robin of messages received by the active queue managers.

The following links provide more information about connection balancing:

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.0.0/com.ibm.mq.dev.doc/q027490_.htm
Queue manager groups
Question: Balance client connections across queue managers, with more clients connected to some queue managers than others.
Answer: Define a queue manager group, and set the CLNTWGHT attribute on each client channel definition to distribute the connections unevenly.
.
https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.ref.con.doc/q081750_.htm
IBM MQ > IBM MQ 9.0.x > IBM MQ > Reference > Configuration reference > Channel attributes > Channel attributes in alphabetical order >
Client channel weight (CLNTWGHT)

Specifies a weighting to influence which client-connection channel definition is used. The client channel weighting attribute is used so that client channel definitions can be selected at random based on their weighting when more than one suitable definition is available.

When a client issues an MQCONN requesting connection to a queue manager group, by specifying a queue manager name starting with an asterisk, which enables client weight balancing across several queue managers, and more than one suitable channel definition is available in the client channel definition table (CCDT), the definition to use is randomly selected based on the weighting, with any applicable CLNTWGHT(0) definitions selected first in alphabetical order.

Specify a value in the range 0 – 99. The default is 0.

A value of 0 indicates that no connection balancing is performed and applicable definitions are selected in alphabetical order. To enable connection balancing choose a value in the range 1 - 99 where 1 is the lowest weighting and 99 is the highest. The distribution of connections between two or more channels with non-zero weightings is approximately proportional to the ratio of those weightings. For example, three channels with CLNTWGHT values of 2, 4, and 14 are selected approximately 10%, 20%, and 70% of the time. This distribution is not guaranteed. If the AFFINITY attribute of the connection is set to PREFERRED, the first connection chooses a channel definition according to client weightings, and then subsequent connections will continue to use the same channel definition.

++
+++ Chapter 1: Introduction to Queue Manager Groups
++

You can define a set of connections in the client channel definition table (CCDT) as a **Queue Manager Group**. The set is defined by its entries having the same value of the QMNAME attribute in their channel definitions. You can connect an application to a queue manager that is part of a queue manager group.

Some of the reasons for choosing to use a Queue Manager Group are:

* You want to connect a client to any one of a set of queue managers that is running, to improve availability.
* You want to reconnect a client to the same queue manager it connected to successfully last time, but connect to a different queue manager if the connection fails.
* You want to automatically reconnect a client connection to another queue manager if the connection fails, without writing any client code.
* You want to automatically reconnect a client connection to a different instance of a multi-instance queue manager if a standby instance takes over, without writing any client code.
* You want to balance your client connections across a number of queue managers, with more clients connecting to some queue managers than others.
* You want to spread the reconnection of many client connections over multiple queue managers and over time, in case the high volume of connections causes a failure.
* You want to be able to move your queue managers without changing any client application code.
* You want to write client application programs that do not need to know queue manager names.

+++ When to use a leading asterisk in the name of the queue manager group

You can connect an application to a queue manager that is part of a queue manager group, by prefixing the queue manager name on an MQCONN or MQCONNX call with an asterisk.

When a queue manager group has a single entry and the names of the group and the entry are the same, then it is ok to only the name of the queue manager and not using the asterisk as a prefix.

However, when the group has multiple entries or the group name is different than the single entry, it is necessary to specify the group name with a leading asterisk.

For example, in the Scenario 6, when using as the QmgrName parameter:

*QMGROUP1

… the WebSphere MQ client selects the matching queue manager group, QMGROUP1. This group contains several client connection channels, and the WebSphere MQ client tries to connect to any queue manager using each channel in turn. In this example, the WebSphere MQ client needs to make a successful connection; the name of the queue manager that it connects to does not matter. By prefixing the queue manager name with an asterisk, the client indicates that the name of the queue manager is not relevant.

+ What happens when the leading asterisk is not specified?

When the queue manager group has multiple entries and when the name of the group is NOT specified with a leading asterisk, then attempt to connect to the queue manager will fail and a typical return code is 2058 MQRC_Q_MGR_NAME_ERROR.

For example, QM7 is a queue manager group with multiple entries, thus a leading asterisk is required; if the asterisk is omitted, then the connection will fail:

```
C:\> amqsphac Q9 QM7
Sample AMQSPHAC start
MQCONNX ended with reason code 2058
Sample AMQSPHAC end

C:\> amqsphac Q9 "QM7"
Sample AMQSPHAC start
MQCONNX ended with reason code 2058
Sample AMQSPHAC end
```

++
+++ Chapter 2: Hierarchy of precedence for the connection methods
+++ Ensuring that the CCDT will be chosen at runtime
++

It is important to understand the precedence for the difference connection methods for an application that uses the MQ client code and that is trying to connect to a queue manager.

In order to ensure that the MQ client code will use the CCDT it is necessary to disable some environment variables and stanzas in a configuration file, and enable the proper environment variables.

Otherwise, you may think during your test that you are using the CCDT but in reality you might be using the MQSERVER or the mqclient.ini (which have precedence over the CCDT).

+++ Hierarchy of precedence for the connection methods

It is important to notice that the connection via a CCDT is not the highest in the hierarchy of precedence for the connection methods.

The order of precedence for the connection methods is shown below:

1: Pre-connect exit (this is an advanced feature)

2: Using options (MQCNO flags) to the MQCONN MQI call, either hardcoded values or values passed via input parameters to an application.

If the application does not use hardcoded values in the MQCONN or if no input parameters are passed to MQCONN, then the MQ client code will use the following method (MQSERVER).

3: The environment variable MQSERVER is set, such as:

Unix (single quotes are required due to the parenthesis):

```
export MQSERVER=SYSTEM.DEF.SVRCONN/TCP/'host1(1414)'
```

Windows (no single quotes are required):

```
set    MQSERVER=SYSTEM.DEF.SVRCONN/TCP/host1(1414)
```

If other environment variables are set for the methods below, they are IGNORED if MQSERVER is defined.

If MQSERVER is not defined, then the MQ client code will use the following method (mqclient.ini).

4: Client Configuration file (mqclient.ini) via the CHANNELS stanza.
The MQSERVER equivalent setting is: ServerConnectionParms
Note: See item 6 in this precedence, regarding the equivalent CCDT settings.

An example of such stanza is:

CHANNELS:
DefRecon=YES
ServerConnectionParms=SYSTEM.DEF.SVRCONN/TCP/host1(1421)

For more information see:
https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.-con.doc/q016850_.htm
IBM MQ > IBM MQ 9.0.x > IBM MQ > Configuring > Configuring connections between the server and clients >
Configuring a client using a configuration file >
Location of the client configuration file

The 1st default location is:
Unix: /var/mqm/mqclient.ini
Windows: C:\Program Files\IBM\WebSphere MQ

The location can be specified by using the environment variable MQCLNTCF:

Unix: export MQCLNTCF=/var/mqm/mqclient.ini
Windows: set MQCLNTCF="C:\Program Files\IBM\WebSphere MQ\mqclient.ini"

If the environment variables are set for the CCDT, they are ignored if the Client Configuration file is active and if there is a CHANNELS stanza.

The 2nd default location is the current directory where the application is running.

If the environment variable for the Client Configuration file is not set or if the Client Configuration file does not have a CHANNELS stanza, then the MQ client code will use this last method (CCDT).

5: Client Channel Definition Table (CCDT).
The location and name of the CCDT can be defined by using the following environment variables.
You will need to specify the proper Queue Manager name for <QMgrName>

Unix:
export MQCHLLIB=/var/mqm/qmgrs/<QMgrName>/@ipcc/

Windows:
set MQCHLLIB=C:\var\mqm\Qmgrs\<QMgrName>\@ipcc

Note: The default value for the related environment variable is:
export MQCHLTAB=AMQCLCHL.TAB
Because this name is used in this techdoc, there is no need to explicitly define this variable.

6: Client Configuration file (mqclient.ini) via the CHANNELS stanza.
The CCDT equivalent settings are:
ChannelDefinitionDirectory and ChannelDefinitionFile

An example of such stanza is:

```
CHANNELS:
  DefRecon=YES
  ChannelDefinitionDirectory=C:\temp\mq\test-mqclient.ini\
  ChannelDefinitionFile=AMQTEST0.TAB
```

+ How to unset variables in order to force the usage of another method.

Let's suppose that you define the MQSERVER environment variable in your profile, but you want to test a CCDT file.
As mentioned in the previous section, if the MQSERVER variable is defined, then the CCDT will NOT be used.
You will need to "unset" the MQSERVER variable in order for the CCDT to be used.
This is how you unset the variables:

Unix:

To unset MQSERVER:
unset MQSERVER

To unset the variable for the Client Configuration file:
unset MQCLNTCF

To unset the variables for the CCDT file:
unset MQCHLLIB

```
unset MQCHLTAB
```

Windows:

To unset a variable in Windows specify a null string to the variable via “=” and no text after the equal sign:

```
set MQSERVER=
```

To unset the Client Configuration file:

```
set MQCLNTCF=
```

To unset the variables for the CCDT file:

```
set MQCHLLIB=
set MQCHLTAB=
```

+++ Setup to ensure to work with the CCDT

Based on the above sections, in order to ensure that the CCDT will be used, it is necessary to do the following:

1) Handle environment variables: unset some and export others.

Unix:

```
        unset MQSERVER
        unset MQCLNTCF
        export MQCHLLIB=/var/mqm/qmgrs/QMgrName/@ipcc/
Note: the following is only if the name is not the default: AMQCLCHL.TAB
        export MQCHLTAB=FILENAME
```

Windows:

```
        set MQSERVER=
        set MQCLNTCF=
        set MQCHLLIB=C:\var\mqm\Qmgrs\<QMgrName>\@ipcc
Note: the following is only if the name is not the default: AMQCLCHL.TAB
        set MQCHLTAB=FILENAME
```

2) If there is an mqclient.ini file under the install root directory for MQ, then ensure that the CHANNELS stanza is commented out by using a # character in the first column, such as:

```
# CHANNELS:
# ServerConnectionParms=SYSTEM.DEF.SVRCONN/TCP/host1(1421)
```

++
+++ Chapter 3: Configuration of the queue managers
++

This is the list of queue managers to be used in the scenarios

Host: veracruz.x.com
MQ Fix Pack: 7.0.1.4
Platform: Linux Intel 32-bit, SUSE 11
QMNAME(QM1) PORT: 1431
QMNAME(QM3) PORT: 1423
QMNAME(QMMI1) PORT: 1421

Host: aemtux1.x.com
MQ Fix Pack: 7.0.1.3
Platform: Linux PowerPC 64-bit, SUSE 9
QMNAME(QM1) PORT: 1425
QMNAME(QM4) PORT: 1434

Host: aemaix1.x.com
MQ Fix Pack: 7.0.1.2
Platform: AIX 6.1
QMNAME(QM2) PORT: 1422
QMNAME(QM3) PORT: 1451

Host: cbeech.x.com
Platform: Linux Intel 32-bit, SUSE 11
MQ Fix Pack: 7.0.1.3
QMNAME(QMMI1) PORT: 1421

+ Configuration

Using the general information from the scenarios described earlier, the following tables were created in order to provide more specific information on the queue managers and channel names.

Queue Manager Group QMNAME	Queue Manager Name	Hostname	Port	Channel Name
+ Scenario 1 - Simplest case - using QMNAME (QM3) for queue manager (QM3) in host-1				
QM3	QM3	aemaix1	1451	QM3
+ Scenario 2 - Simplest case, using same QMNAME for a single queue manager (QM3 in another host: host2)				
QM3	QM3	veracruz	1423	QM3
+ Scenario 3 - Queue Manager Group: QM1 (no leading asterisk)				
QM1	QM1	veracruz	1431	QM1.A
QM1	QM1	aemtux1	1425	QM1.B
+ Scenario 4 - Queue Manager Group: *QM1 (with leading asterisk)				
QM1	QM1	veracruz	1431	QM1.A
QM1	QM1	aemtux1	1425	QM1.B
QM1	QM2	aemaix1	1422	QM1.C
+ Scenario 5 - Queue Manager Group: ' ' (1 blank character)				
' '	QM1	veracruz	1431	DEFAULT.A
' '	QM3	aemaix1	1451	DEFAULT.B
' '	QM4	aemtux1	1434	DEFAULT.C
+ Scenario 6 - Queue Manager Group: QMGROUP1				
QMGROUP1	QM2	aemaix1	1422	QMGROUP1.A
QMGROUP1	QM3	veracruz	1423	QMGROUP1.B
+ Scenario 7 - Multi-Instance Queue Manager				
QMMI1	QMMI1	veracruz	1421	QMMI1
QMMI1	QMMI1	cbeech	1421	QMMI1

++ Batch command files for "runmqsc" to create the necessary channels

Note: For your information, the following file supplied with MQ provides a good example of MQSC script:

Windows: C:\Program Files\IBM\WebSphere MQ\java\bin\MQJMS_PSQ.mqsc
Unix: /opt/mqm/java/bin/MQJMS_PSQ.mqsc

You can run the MQSC script by doing the following:

runmqsc QMgrName < filename.mqsc > filename.out

It is a good idea to capture the output in a file, because you may want to review the output just in case that there are warnings or errors.

To facilitate the setup of the CCDT, 2 batch command (script) files will be used.

a) ccdt-svrconn.mqsc

This script will be run on EACH queue manager that participates in the scenario.
This will define the Server-Connection (SVRCONN) channels.
Strictly speaking, the same script does not have to be applied to each queue manager and only the particular channels for that queue manager would need to be applied. However, for the sake of simplicity in setting up the testing environment, the script has all the channels for all the queue managers.

b) ccdt-cltconn.mqsc

This script is going to be run ONLY once to actually create the CCDT. This CCDT file will be copied to other hosts.
This will define the Client-Connection (CLTCONN) channels.

++ MQSC command batch file for SVRCONN channels: ccdt-svrconn.mqsc

< begin script >
< skipping the header >

* The following queues are needed for the test scenarios.

```
DEFINE QLOCAL(Q9)     REPLACE
DEFINE QLOCAL(SOURCE) REPLACE
DEFINE QLOCAL(TARGET) REPLACE

* Scenario 1 - Simplest case - using QMNAME (QM3) for queue manager (QM3) in host-1
DEFINE CHANNEL(QM3) CHLTYPE(SVRCONN)  TRPTYPE(TCP) REPLACE

* Scenario 2 - Simplest case - using QMNAME (QM3) for queue manager (QM3) in host-2
* Note: There is no actual change. Using this line for completeness sake.
DEFINE CHANNEL(QM3) CHLTYPE(SVRCONN)  TRPTYPE(TCP) REPLACE

* Scenario 3 - Queue Manager Group: QM1 (no leading asterisk)
DEFINE CHANNEL(QM1.A) CHLTYPE(SVRCONN)  TRPTYPE(TCP) REPLACE
DEFINE CHANNEL(QM1.B) CHLTYPE(SVRCONN)  TRPTYPE(TCP) REPLACE

* Scenario 4 - Queue Manager Group: *QM1 (with leading asterisk)
DEFINE CHANNEL(QM1.A) CHLTYPE(SVRCONN)  TRPTYPE(TCP) REPLACE
DEFINE CHANNEL(QM1.B) CHLTYPE(SVRCONN)  TRPTYPE(TCP) REPLACE
DEFINE CHANNEL(QM1.C) CHLTYPE(SVRCONN)  TRPTYPE(TCP) REPLACE

* Scenario 5 - Queue Manager Group: ' ' (1 blank character)
DEFINE CHANNEL(DEFAULT.A) CHLTYPE(SVRCONN)  TRPTYPE(TCP) REPLACE
DEFINE CHANNEL(DEFAULT.B) CHLTYPE(SVRCONN)  TRPTYPE(TCP) REPLACE
DEFINE CHANNEL(DEFAULT.C) CHLTYPE(SVRCONN)  TRPTYPE(TCP) REPLACE

* Scenario 6 - Queue Manager Group: QMGROUP1
DEFINE CHANNEL(QMGROUP1.A) CHLTYPE(SVRCONN)  TRPTYPE(TCP) REPLACE
DEFINE CHANNEL(QMGROUP1.B) CHLTYPE(SVRCONN)  TRPTYPE(TCP) REPLACE

* Scenario 7 - Multi-Instance Queue Manager
DEFINE CHANNEL(QMMI1) CHLTYPE(SVRCONN)  TRPTYPE(TCP) REPLACE
```

< end script >

```
++ MQSC command batch file for CLTCONN channels: ccdt-cltconn.mqsc

< begin script >
< skipping the header >


* Scenario 1 - Simplest case - using QMNAME (QM3) for queue manager (QM3) in host-1
DEFINE CHANNEL(QM3) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
 CONNAME('aemaix1.x.com(1451)') QMNAME(QM3) REPLACE

* Scenario 2 - Simplest case - using QMNAME (QM3) for queue manager (QM3) in host-2
* When running Scenario 1, it is necessary to comment out the following, but uncom-
ment them when running Scenario 2
* DEFINE CHANNEL(QM3) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
* CONNAME('veracruz.x.com(1423)') QMNAME(QM3) REPLACE

* Scenario 3 - Queue Manager Group: QM1 (no leading asterisk)
* The actual queue managers are called QM1
DEFINE CHANNEL(QM1.A) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('veracruz.x.com(1431)') QMNAME(QM1) REPLACE

DEFINE CHANNEL(QM1.B) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('aemtux1.x.com(1425)')   QMNAME(QM1) REPLACE

* Scenario 4 - Queue Manager Group: *QM1 (with leading asterisk)
* It is the same as Scenario 3, in which the first 2 entries the queue manager
* is called QM1 but in the 3rd entry the queue manager is QM2
DEFINE CHANNEL(QM1.A) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('veracruz.x.com(1431)') QMNAME(QM1) REPLACE

DEFINE CHANNEL(QM1.B) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('aemtux1.x.com(1425)')   QMNAME(QM1) REPLACE

DEFINE CHANNEL(QM1.C) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
 CONNAME('aemaix1.x.com(1422)')    QMNAME(QM1) REPLACE

* Scenario 5 - Queue Manager Group: ' ' (1 blank character)
DEFINE CHANNEL(DEFAULT.A) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('veracruz.x.com(1431)') QMNAME(' ') REPLACE

DEFINE CHANNEL(DEFAULT.B) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('aemaix1.x.com(1451)')   QMNAME(' ') REPLACE
```

```
DEFINE CHANNEL(DEFAULT.C) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('aemtux1.x.com(1434)')   QMNAME(' ') REPLACE

* Scenario 6 - Queue Manager Group: QMGROUP1

DEFINE CHANNEL(QMGROUP1.A) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('aemaix1.x.com(1422)') QMNAME(QMGROUP1) REPLACE

DEFINE CHANNEL(QMGROUP1.B) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
 CONNAME('veracruz.x.com(1423)') QMNAME(QMGROUP1) REPLACE

* Scenario 7 - Multi-Instance Queue Manager
* Notice that there is only one DEFINE, but the CONNAME is used as a
* Connection Name List
DEFINE CHANNEL(QMMI1) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
 CONNAME('veracruz.x.com(1421),cbeech.x.com(1421)') +
 QMNAME(QMMI1) REPLACE

< end script >
```

++ Run the scripts and browse the output files in case that there were errors.

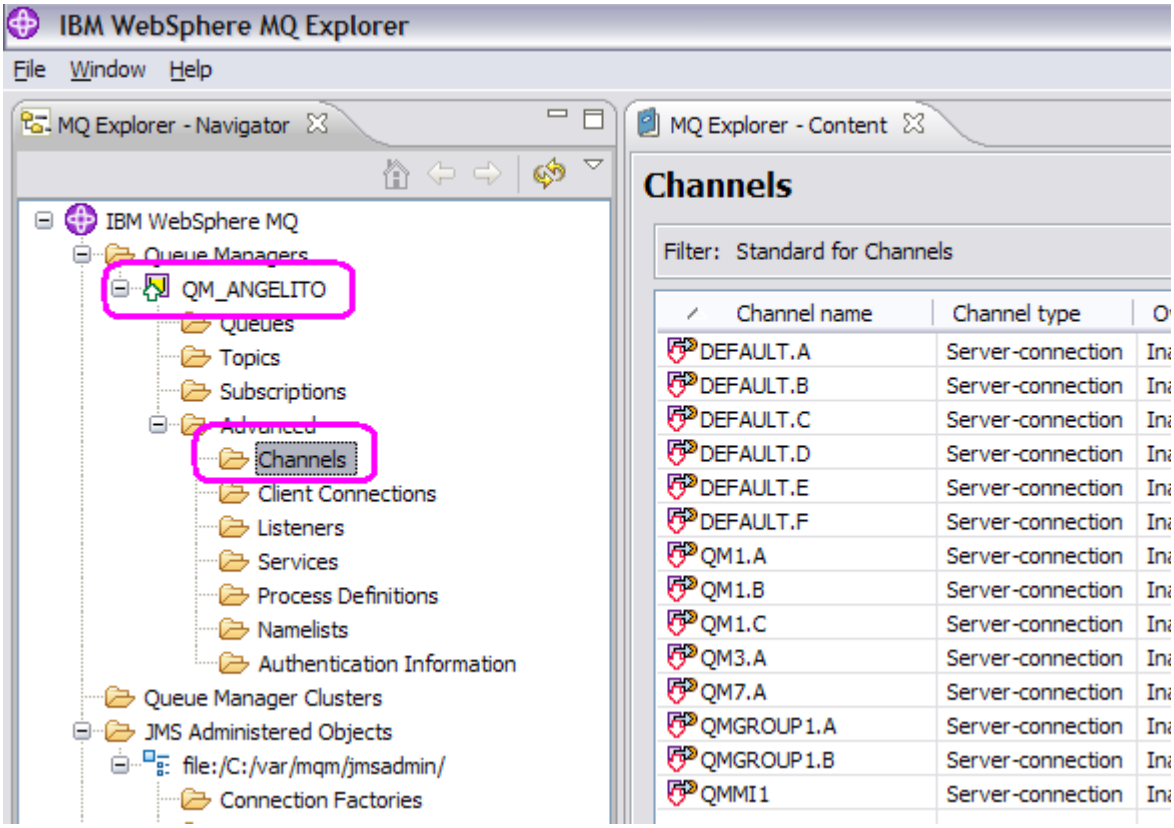
a) In each host, run the script to create the server-connection channels

runmqsc QMgr < ccdt-svrconn.mqsc > ccdt-svrconn.out
cat ccdt-svrconn.out

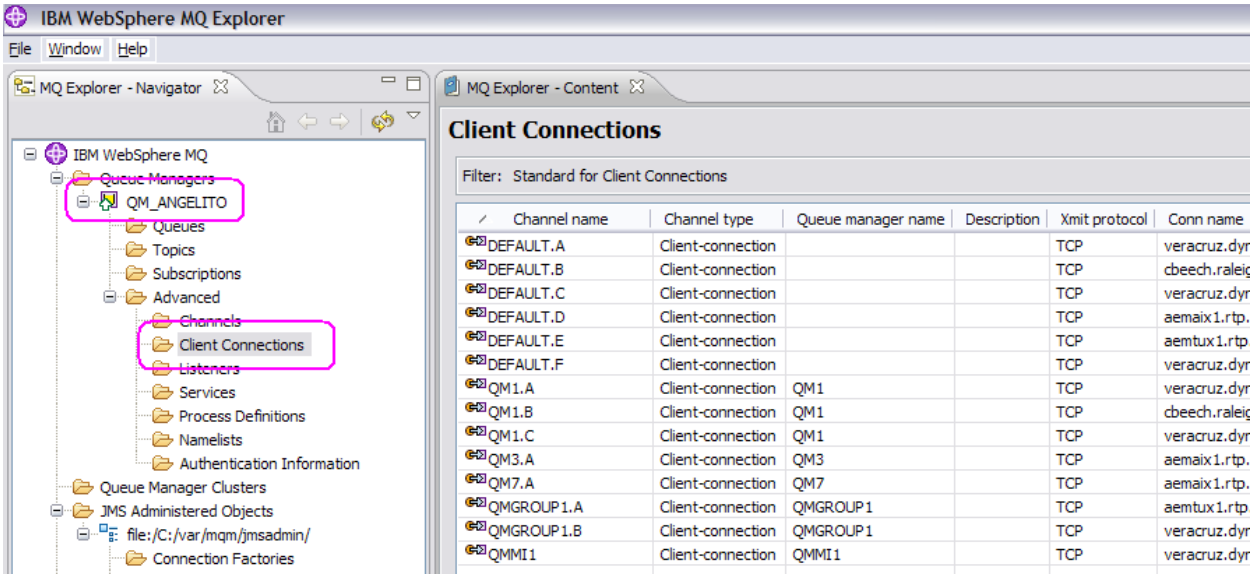
b) In the client host, run the script to create the client-connection channels

runmqsc QMgr < ccdt-cltconn.mqsc > ccdt-cltconn.out
notepad ccdt-cltconn.out

+++ Using the MQ Explorer to verify that the Server Connection channels were properly created



+++ Using the MQ Explorer to verify that the Client Connection channels were properly created



++ Using the runmqsc to verify that the Server Connection channels were properly created

Note: Showing only few of the created channels

```
display channel(*) chltype(svrconn)
    2 : display channel(*) chltype(svrconn)
AMQ8414: Display Channel details.
  CHANNEL(DEFAULT.A)                  CHLTYPE(SVRCONN)
AMQ8414: Display Channel details.
  CHANNEL(DEFAULT.B)                  CHLTYPE(SVRCONN)
AMQ8414: Display Channel details.
  CHANNEL(DEFAULT.C)                  CHLTYPE(SVRCONN)
AMQ8414: Display Channel details.
  CHANNEL(QM1.A)                     CHLTYPE(SVRCONN)
```

+++ Using the runmqsc to verify that the Client Connection channels were properly created

Note: Showing only few of the created channels

```
display channel(*) chltype(clntconn) qmname conname
    3 : display channel(*) chltype(clntconn) qmname conname
AMQ8414: Display Channel details.
  CHANNEL(DEFAULT.A)                  CHLTYPE(CLNTCONN)
  CONNAME(veracruz.x.com(1421))
  QMNAME( )
AMQ8414: Display Channel details.
  CHANNEL(DEFAULT.B)                  CHLTYPE(CLNTCONN)
  CONNAME(cbeech.x.com(1421))   QMNAME( )
AMQ8414: Display Channel details.
  CHANNEL(QM1.A)                     CHLTYPE(CLNTCONN)
  CONNAME(veracruz.x.com(1421))
  QMNAME(QM1)
AMQ8414: Display Channel details.
  CHANNEL(QM1.B)                     CHLTYPE(CLNTCONN)
  CONNAME(cbeech.x.com(1421))   QMNAME(QM1)
AMQ8414: Display Channel details.
  CHANNEL(QM3.A)                     CHLTYPE(CLNTCONN)
  CONNAME(aemaix1.x.com(1451))
  QMNAME(QM3)
```

+++ Using SupportPac to review the CCDT

The previous tasks using the MQ Explorer and "runmqsc" allow you to display the created Client-Connection channels that are known to the queue manager that was used to define them.

The Client-Connection channels are also stored in the Client Channel Definition Table (CCDT) of the queue manager, under the file:

AMQCLCHL.TAB

The location of the file is:

Unix: /var/mqm/qmgrs/QMgrName/@ipcc/ AMQCLCHL.TAB
Windows: LOCATION\ qmgrs\QMgrName\@ipcc\ AMQCLCHL.TAB

Now you can copy this file into a desired directory in each of the machines where a client application is going to use the CCDT.

The CCDT file is binary and not text. Thus, when using ftp to transfer the file from one host to another, you must ensure that you use the mode "binary", otherwise, if it is transferred as ascii, it may arrive corrupted.

In those machines, it is necessary to setup the proper environment variables to point to the CCDT file:

Unix:

Example: the file AMQCLCHL.TAB was placed in /var/mqm
export MQCHLLIB=/var/mqm/

Note: The default value for the related environment variable is:
export MQCHLTAB=AMQCLCHL.TAB

Interesting Problem!
Let's suppose that you are an MQ Administrator and you see a CCDT file that was copied from another machine: how do you look at the contents of the CCDT file to verify its contents? (Your company may have several CCDT files with different names and for different purposes)

The answer is that you cannot use runmqsc to do so.
But you can use the following SupportPac:

Note: Technical support for this SupportPac:
It is Category 2 SupportPac, which is provided in good faith and AS-IS. There is no warranty or further service implied or committed and any supplied sample code is not supported via IBM product service channels.

http://www.ibm.com/support/docview.wss?uid=swg24007769
MO72: MQSC Client for WebSphere MQ

1) Visit the above web page and download the zip file with the code and the PDF file for the documentation.

Let's assume that it is downloaded into:

Linux Intel 32-bit:
/downloads/mq/mo72-mqsc

Windows:
C:\MQ-SupportPac\MO72 MQSC Client

2) Change to the directory of the platform:

Linux Intel 32-bit:
cd /downloads/mq/mo72-mqsc/Linux Intel

Windows:
cd C:\MQ-SupportPac\MO72 MQSC Client\Windows

3) For Unix, you must give execution permission:

$ chmod u+x mqsc

4) Run the command which uses the default CCDT file name of AMQCLCHL.TAB.
You need to specify the following flags:
-n => run in client mode to access the CCDT
-t full-path-ending-with-slash => Full directory location
NOTICE that you MUST end the string with a directory separator, or slash!

INCORRECT usage:
For example, the last character of the path name is NOT a directory separator

C:\MQ-SupportPac\MO72 MQSC Client, reads CCDT\Windows>
mqsc -n -t C:\var\mqm\Qmgrs\QM_ANGELITO\@ipcc
Current channel table file not found
Can not find file 'C:\var\mqm\Qmgrs\QM_ANGELITO\@ipcc'
>

CORRECT usage:
You need to ensure that the end of the path name has a directory separator.
Notice that you get a prompt: >

Linux example:
$ **mqsc -n -t /mqexport/701/data/QMMI1/@ipcc/**
>

Windows example:
C:\MQ-SupportPac\MO72 MQSC Client, reads CCDT\Windows>
mqsc -n -t C:\var\mqm\Qmgrs\QM_ANGELITO\@ipcc
>

5) Issue the following to show the contents.
Note: only showing selected entries to keep this document shorter.

> **display channel(*)**

AMQ8414: Display Channel details.
CHANNEL(DEFAULT.A) CHLTYPE(CLNTCONN) DESCR( ) TRPTYPE(TCP)
CONNAME(veracruz.x.com(1421)) QMNAME( )
LOCLADDR( ) USERID( ) PASSWORD( ) MODENAME( )
TPNAME( ) MAXMSGL(4194304) HBINT(300) SCYEXIT( )
SCYDATA( ) SENDEXIT( ) SENDDATA( ) RCVEXIT( )
RCVDATA( ) COMPHDR(NONE) COMPMSG(NONE) SSLCIPH( )
SSLPEER( ) SHARECNV(10) CLNTWGHT(0) AFFINITY(PREFERRED)
ALTDATE(2010-05-26) ALTTIME(09.26.07)

5) To end the session (same as in normal "runmqsc") issue:

> end

+++ Copy ccdt-svrconn.mqsc and AMQCLCHL.TAB (CCDT file) into each host

For example, in Linux host "veracruz" there are several queue managers that are involved in some of the scenarios:

veracruz: QMNAME(QMMI1)	PORT: 1421
veracruz: QMNAME(QM1)	PORT: 1431
veracruz: QMNAME(QM3)	PORT: 1423

The MQSC batch file ccdt-svrconn.mqsc and AMQCLCHL.TAB are copied into /tmp

NOTICE: The CCDT file is a BINARY file and must be copied as BINARY.
If you use FTP you MUST use BINARY, otherwise, if it is transferred as ASCII TEXT then if the target machine has a different platform, then the file may arrive corrupted.

This is an example ftp session.
Using the # in the first character to indicate a comment line.

```
C:\> ftp veracruz.x.com
Connected to veracruz.x.com.
ftp> cd /tmp
250 Directory successfully changed.

# Specify ASCII Text for the MQSC file
ftp> ascii
200 Switching to ASCII mode.
ftp> put ccdt-svrconn.mqsc

# Specify BINARY for the CCDT file
ftp> binary
200 Switching to Binary mode.
ftp> put AMQCLCHL.TAB

ftp> quit
221 Goodbye.
```

+ To run the MQSC command file, issue for each queue manager

```
rivera@veracruz: /tmp
$ runmqsc QM1 < ccdt-svrconn.mqsc > qm1.out
```

+ Setup the environment variables to use the CCDT under /tmp

```
export MQCHLLIB=/tmp/
```

Note: The default value for the related environment variable is:

```
export MQCHLTAB=AMQCLCHL.TAB
```

+ Run the High-Availability sample programs that put messages and get messages

The sample amqsphac is a high-availability program that does a "put" and exploits explicitly the automatic reconnect feature introduced in MQ V7.

Note:
In Windows, do not use the single quotes around the queue manager name in amqsphac:

```
C:\MQ-doc\techdocs>amqsphac Q9 'QM3'
Sample AMQSPHAC start
MQCONNX ended with reason code 2058
Sample AMQSPHAC end
```

In Unix, it is OK to use single quotes.

++
+++ Chapter 4: Description of the Scenarios
++

There are 7 scenarios described in this techdoc.

+ Scenario 1 - Simplest case, using same QMNAME for a single queue manager (QM3 in host1)

Queue Manager Group: 'QM3'
Application uses: 'QM3'

There is ONLY 1 entry in the CCDT with QMNAME 'QM3':

1: To Queue Manager: QM3 (in Host-1) via channel:
```
DEFINE CHANNEL(QM3) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('aemaix1.x.com(1451)') QMNAME(QM3) REPLACE
```

This is the simplest case, and it is a group that has only 1 entry.

The client application passes a queue manager name, QM3, as the QmgrName parameter to its MQCONN or MQCONNX MQI call. The WebSphere MQ client code selects the matching queue manager group, QM3 in Host-1.

The group contains only 1 connection channel, and the WebSphere MQ client tries to connect to QM3 in host-1 via Channel QM3.

The MQCONN call issued by the client application succeeds when a connection is established to QM3.

- Running the scenario

Start the sample for high availability put.

In this case, the accepted ways to specify the queue manager group are shown. Note: in Windows do not use single quotes around the queue manager, to avoid the return code 2058.

Notice that because the group name is the same as the single entry queue manager, it is OK to not use the asterisk.

The rest of the scenarios will show only one way, the one with leading asterisk:
*GroupName

```
C:\> amqsphac Q9 QM1
Sample AMQSPHAC start
target queue is Q9
message <Message 1>
message <Message 2>

C:\> amqsphac Q9 "QM3"
Sample AMQSPHAC start
target queue is Q9
message <Message 1>
message <Message 2>

C:\> amqsphac Q9 *QM3
Sample AMQSPHAC start
target queue is Q9
message <Message 1>

C:\> amqsphac Q9 "*QM3"
Sample AMQSPHAC start
target queue is Q9
message <Message 1>
```

In Unix, you can use the single quotes:

```
rivera@veracruz: /home/rivera
(887) amqsphac Q9 'QM3'
Sample AMQSPHAC start
target queue is Q9
message <Message 1>
message <Message 2>
```

The messages were received at queue manager: QM3 in aemaix1

The high availability get sample is used:

```
aemaix1:/ %     amqsghac Q9 QM3
Sample AMQSGHAC start
message <Message 1>
message <Message 2>
message <Message 1>
message <Message 2>
message <Message 1>
message <Message 1>
```

+ Scenario 2 - Simplest case, using same QMNAME for a single queue manager (QM3 in another host: host2)

Queue Manager Group: 'QM3'
Application uses: 'QM3'

There is ONLY 1 entry in the CCDT with QMNAME 'QM3':

1: To Queue Manager: QM3 (in Host-1) via channel:

```
DEFINE CHANNEL(QM3) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('veracruz.x.com(1423)') QMNAME(QM3) REPLACE
```

If QM3 is moved from Host-1 to Host-2, then only the CCDT needs to be updated to reflect the new location of the queue manager:

From: CONNAME('aemaix1.x.com(1451)')
To: CONNAME('veracruz.x.com(1423)')

There is no need to modify the client application, which continues using QM3, regardless of its location.

- Running the scenario

Similar to Scenario 1.

The messages were NOT received in QM1 in aemaix1 as in Scenario 1.
Rather, the messages were received in QM1 in veracruz.

+ Scenario 3 - Queue Manager Group: QM1 (no leading asterisk)

Queue Manager Group: 'QM1'
Application uses: 'QM1'

There are 2 entries in the CCDT with QMNAME 'QM1':

```
DEFINE CHANNEL(QM1.A) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('veracruz.x.com(1431)') QMNAME(QM1) REPLACE

DEFINE CHANNEL(QM1.B) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('aemtux1.x.com(1425)')   QMNAME(QM1) REPLACE
```

In this scenario, the client application passes a queue manager name, QM1, as the QmgrName parameter to its MQCONN or MQCONNX MQI call. The WebSphere MQ client code selects the matching queue manager group, QM1.

The group contains two connection channels, and the WebSphere MQ client tries to connect to QM1 using each of these channels in turn until it finds an WebSphere MQ listener for the connection attached to a running queue manager called QM1.

The MQ V7 Information Center says:
"The order of connection attempts depends on the value of the client connection AFFINITY attribute and the client channel weightings. Within these constraints, the order of connection attempts is randomized, both over the possible connections, and over time, in order to spread out the load of making connections."

The MQCONN or MQCONNX call issued by the client application succeeds when a connection is established to a running instance of QM1.

- Running the scenario

Both queue manager QM1 in aemtux1 and in veracruz were running.

The following was issued several times, in order to get a new connection each time.

C:\> amqsphac Q9 QM1
Sample AMQSPHAC start
target queue is Q9
message <Message 1>

All the messages were received by QM1 in veracruz.

rivera@veracruz: /home/rivera
(972) amqsghac Q9 QM1
Sample AMQSGHAC start
message <Message 1>
message <Message 2>

Only when QM1 in Veracruz was stopped, then a new connection from the client resulted in messages being delivered to QM1 in aemtux1

aemtux1:/rivera 523% amqsghac Q9 QM1
Sample AMQSGHAC start
message <Message 1>

The reason for this behavior is explained in:
https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.dev.doc/q027510_.htm
IBM MQ > IBM MQ 9.0.x > IBM MQ > Developing applications > Developing MQI applications with IBM MQ > Writing client procedural applications > Running applications in the IBM MQ MQI client environment > Connecting IBM MQ MQI client applications to queue managers >
Examples of channel weighting and affinity

"If all applicable channels for a connection have a ClientChannelWeight of zero (the default) then they are selected in alphabetical order."

In this scenario, the alphabetical order of the channels is QM1.A first, followed by QM1.B. The default CLNTWGHT(0) is being used in both channels.

```
CHANNEL(QM1.A)  CONNAME('veracruz.x.com(1431)')
CHANNEL(QM1.B)  CONNAME('aemtux1.x.com(1425)')
```

When both queue managers are running the QM1.A channel is always chosen first. To alter the order, it would be necessary to modify the CLNTWGHT attribute for the channels, but this is beyond the scope of this techdoc.

* Scenario 4 - Queue Manager Group: *QM1 (with leading asterisk)

Queue Manager Group: 'QM1'
Application uses: '*QM1'

There are 3 entries in the CCDT with QMNAME 'QM1':

This queue manager is actually named QM1:
```
DEFINE CHANNEL(QM1.A) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('veracruz.x.com(1431)') QMNAME(QM1) REPLACE
```

This queue manager is actually named QM1:
```
DEFINE CHANNEL(QM1.B) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('aemtux1.x.com(1425)')   QMNAME(QM1) REPLACE
```

This queue manager is actually named QM2 (that is, it is NOT named QM1):
```
DEFINE CHANNEL(QM1.C) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
 CONNAME('aemaix1.x.com(1422)')    QMNAME(QM1) REPLACE
```

Note: The attribute QMNAME in a CLTCONN channel refers to the Queue Manager Group, and not to actually a single Queue Manager. Thus, the 3rd example shows that the actual queue manager is QM2 but QMNAME is QM1.

This scenario is similar to Scenario 3, but the QmgrName parameter is prefixed by an asterisk, *QM1. The example illustrates that you cannot determine which queue manager a client channel connection is going to connect to by inspecting the QMNAME attribute in one channel definition by itself. The fact that the QMNAME attribute of the channel definition is QM1, is not sufficient to require a connection is made to a queue

manager called QM1. If your client application prefixes its QmgrName parameter with an asterisk then any queue manager is a possible connection target.

In this case the MQCONN or MQCONNX calls issued by the client application succeed when a connection is established to a running instance of either QM1 or QM2.

The name of the queue manager that it connects to does not matter.
The rule for the order of making connection attempts is the same as before. The only difference is that by prefixing the queue manager name with an asterisk, the client indicates that the name of the queue manager is not relevant.

- Running the scenario

The 3 queue managers are running:
QM1 in veracruz
QM1 in aemtux1
QM2 in aemaix1

Two runs are done to put messages:

```
C:\> amqsphac Q9 *QM1
Sample AMQSPHAC start
target queue is Q9
message <Message 1>
message <Message 2>
message <Message 3>
^C

C:\> amqsphac Q9 *QM1
Sample AMQSPHAC start
target queue is Q9
message <Message 1>
message <Message 2>
^C
```

The messages are received ONLY at QM1 in veracruz (see explanation in Scenario 3 about choosing the channel name in alphabetical order - the channel name for veracruz is QM1.A and it is first in the list)

```
rivera@veracruz: /home/rivera
(975) amqsghac Q9 QM1
Sample AMQSGHAC start
message <Message 1>
```

message <Message 2>
message <Message 1>
message <Message 2>

Stop the QM1 in veracruz, in order to give a chance to the other queue managers to receive messages.

Two runs are done to put messages:

C:\> amqsphac Q9 *QM1
Sample AMQSPHAC start
target queue is Q9
message <Message 1>
message <Message 2>
^C

C:\> amqsphac Q9 *QM1
Sample AMQSPHAC start
target queue is Q9
message <Message 1>
message <Message 2>
^C

The messages are received ONLY at QM1 in aemtux1 - the channel name is QM1.B and it is the 2nd in the list.

aemtux1:/rivera 525% amqsghac Q9 QM1
Sample AMQSGHAC start
message <Message 1>
message <Message 2>
message <Message 1>

Stop QM1 in aemtux1.
At this point only QM2 in aemaix1 is running.

Run the sample to put messages:

C:\> amqsphac Q9 *QM1
Sample AMQSPHAC start
target queue is Q9
message <Message 1>
message <Message 2>

^C

The messages are received by QM2 in aemaix1:

aemaix1:/rivera 157% amqsghac Q9 QM2
Sample AMQSGHAC start
message <Message 1>
message <Message 2>

Note: The list shown for Scenario 3 has only 2 items, but in reality, for the testing of this techdoc, the list has 3 items, in order to perform Scenario 4.

In the following example, the queue manager name is specified without a leading asterisk.

In this particular run, QM1 in Veracruz and QM1 in aemtux1 are stopped and the only queue manager running is QM2 in aemaix1. Notice that the connection fails with RC 2058.

C:\> amqsphac Q9 **QM1**
Sample AMQSPHAC start
MQCONNX ended with reason code 2058
Sample AMQSPHAC end

In contrast, if the queue manager name is specified with a leading asterisk, then it works, because the MQ client is being told to NOT care about the actual name of the queue manager:

C:\> amqsphac Q9 ***QM1**
Sample AMQSPHAC start
target queue is Q9
message <Message 1>

+ Scenario 5 - Queue Manager Group: ' ' (1 blank character)

Queue Manager Group: ' '
Application uses: ' '

There are 3 entries in the CCDT with QMNAME ' ' (1 blank character):

```
DEFINE CHANNEL(DEFAULT.A) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('veracruz.x.com(1431)') QMNAME(' ') REPLACE

DEFINE CHANNEL(DEFAULT.B) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('aemaix1.x.com(1451)')   QMNAME(' ') REPLACE

DEFINE CHANNEL(DEFAULT.C) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('aemtux1.x.com(1434)')   QMNAME(' ') REPLACE
```

This scenario illustrates use of the default group. In this case the client application passes an asterisk, '*', or blank ' ', as the QmgrName parameter to its MQCONN or MQCONNX MQI call. By convention in the client channel definition, a blank QMNAME attribute signifies the default queue manager group and either a blank or asterisk QmgrName parameter matches a blank QMNAME attribute.

In this example the default queue manager group has client channel connections to all the queue managers. By selecting the default queue manager group the application might be connected to any queue manager in the group.

The MQCONN or MQCONNX call issued by the client application succeeds when a connection is established to a running instance of any queue manager.

Note: The default group is different to a default queue manager, although an application uses a blank QmgrName parameter to connect to either the default queue manager group or to the default queue manager. The concept of a default queue manager group is only relevant to a client application, and a default queue manager to a server application.

A variation of the Scenario 5 is shown below, in which the application uses an asterisk '*' instead of a blank.

Queue Manager Group: ' '
Application uses: '*'

- Running the scenarios

For this scenario, the following queue managers are running:

QM1 in Veracruz
QM3 in aemaix1
QM4 in aemtux1

Important caveats:
- The client is run from Windows. Notice that ONLY one variation is successful (the one that uses asterisk, the first one in the list below), and the others result in RC 2058.
- The use of ' ' (single space) is suitable for working directly with the MQCONN call, and not necessarily via samples that interact with the command prompt, such as amqsphac.

Notice that this is the only variation that is successful:

```
C:\> amqsphac Q9 *
Sample AMQSPHAC start
target queue is Q9
message <Message 1>
message <Message 2>
^C
```

These other variations fail with code 2058. The failure is due to the parsing of the command line by the samples, and not by the underlying function of queue manager groups.

```
C:\> amqsphac Q9 ' '
Sample AMQSPHAC start
MQCONNX ended with reason code 2058
Sample AMQSPHAC end

C:\> amqsphac Q9 "
Sample AMQSPHAC start
MQCONNX ended with reason code 2058
Sample AMQSPHAC end

C:\> amqsphac Q9 '*'
Sample AMQSPHAC start
MQCONNX ended with reason code 2058
Sample AMQSPHAC end
```

In this case, the messages were received by QM1 in Veracruz, because its channel name is first in the alphabetical order: DEFAULT.A

```
rivera@veracruz: /home/rivera
(984) amqsghac Q9 QM1
Sample AMQSGHAC start
message <Message 1>
message <Message 2>
```

Stop QM1 in Veracruz and resume the test from the client.

```
C:\> amqsphac Q9 *
Sample AMQSPHAC start
target queue is Q9
message <Message 1>
```

message <Message 2>

In this case, the messages are received by QM3 in aemaix1, because its channel name is next in the alphabetical list, DEFAULT.B, and the previous channel is not operational.

```
aemaix1:/rivera 164% amqsghac Q9 QM3
Sample AMQSGHAC start
message <Message 1>
message <Message 2>
```

Finally, stop QM3 in aemaix1 and resume the test:

```
C:\> amqsphac Q9 *
Sample AMQSPHAC start
target queue is Q9
message <Message 1>
message <Message 2>
```

In this case, the messages are received by QM4 in aemtux1, because the channel name DEFAULT.C is the last in the list (and the previous channels are not operational).

```
aemtux1:/rivera 532% amqsghac Q9 QM4
Sample AMQSGHAC start
message <Message 1>
message <Message 2>
```

+ Scenario 6 - Queue Manager Group: QMGROUP1

Queue Manager Group: 'QMGROUP1'
Application uses: '*QMGROUP1'

There are 2 entries in the CCDT with QMNAME 'QMGROUP1':

```
DEFINE CHANNEL(QMGROUP1.A) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
  CONNAME('aemaix1.x.com(1422)') QMNAME(QMGROUP1) REPLACE

DEFINE CHANNEL(QMGROUP1.B) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
 CONNAME('veracruz.x.com(1423)') QMNAME(QMGROUP1) REPLACE
```

In this scenario, the client application passes a queue manager name prefixed with an asterisk, *QMGROUP1 as the QmgrName parameter to its MQCONN or MQCONNX MQI call.

The WebSphere MQ client selects the matching queue manager group, QMGROUP1. This group contains two client connection channels, and the WebSphere MQ client tries to connect to *any* queue manager using each channel in turn. In this example, the WebSphere MQ client needs to make a successful connection; the name of the queue manager that it connects to does not matter.

The rule for the order of making connection attempts is the same as before. The only difference is that by prefixing the queue manager name with an asterisk, the client indicates that the name of the queue manager is not relevant.

The MQCONN or MQCONNX call issued by the client application succeeds when a connection is established to a running instance of any queue manager connected to by the channels in the QMGROUP1 queue manager group.

- Running the scenario.

Start the following queue managers:
QM1 in aemtux1
QM3 in veracruz

Run the sample to put messages:

```
C:\> amqsphac Q9 *QMGROUP1
Sample AMQSPHAC start
target queue is Q9
message <Message 1>
message <Message 2>
^C
```

The messages are received only by QM2 in aemaix2 because its channel name is first in the alphabetical order: QMGROUP1.A

```
aemaix1:/rivera 170% amqsghac Q9 QM2
Sample AMQSGHAC start
message <Message 1>
message <Message 2>
```

Stop QM2 in aemaix1 and repeat sending the messages.

```
C:\> amqsphac Q9 *QMGROUP1
Sample AMQSPHAC start
target queue is Q9
message <Message 1>
message <Message 2>
^C
```

The messages are now received by QM3 in veracruz.

```
rivera@veracruz: /home/rivera
(991) amqsghac Q9 QM3
Sample AMQSGHAC start
message <Message 1>
message <Message 2>
```

+ Scenario 7 - Multi-Instance Queue Manager

Queue Manager Group: 'QMMI1'
Application uses: 'QMMI1'

There is only 1 entry in the CCDT with QMNAME 'QMMI1':

```
DEFINE CHANNEL(QMMI1) CHLTYPE(CLNTCONN) TRPTYPE(TCP)  +
 CONNAME('veracruz.x.com(1421),cbeech.x.com(1421)') +
 QMNAME(QMMI1) REPLACE
```

Notice that because the group name is the same as the single entry queue manager, it is OK to not use the asterisk.

Do not get confused that there are 2 boxes in the figure for the same queue manager name (QMMI1), it is still a group with a single entry but for multiple-instances. That is also the reason for a single definition for the client-connection and server-connection channel: it is the same queue manager.

Setup: Both instances for QMMI1 are running in veracruz (active) and cbeech (stand-by).

Start the sample for high availability put:

```
C:\>amqsphac Q9 QMMI1
Sample AMQSPHAC start
target queue is Q9
message <Message 1>
message <Message 2>
message <Message 3>
```

The messages are received in veracruz via the sample for high availability get:

rivera@veracruz: /home/rivera/test
$ amqsghac Q9 QMMI1
Sample AMQSGHAC start
message <Message 1>
message <Message 2>

The active instance is ended and a switchover is requested:

rivera@veracruz: /home/rivera/test
$ endmqm -is QMMI1
Waiting for queue manager 'QMMI1' to end.
WebSphere MQ queue manager 'QMMI1' ending.
WebSphere MQ queue manager 'QMMI1' ended, permitting switchover to a standby instance.

The sample get is started at the host cbeech while the switchover is happening but it has not been completed yet. Notice the rc 2538

rivera@cbeech: /home/rivera/test
$ amqsghac Q9 QMMI1
Sample AMQSGHAC start
MQCONNX ended with reason code 2538
Sample AMQSGHAC end

Wait until the switchover is done:

rivera@cbeech: /home/rivera/test
$ amqsghac Q9 QMMI1
Sample AMQSGHAC start
message <Message 26>
message <Message 27>

Notice that the output of the put sample will show the pause while the reconnection is happening:

message <Message 23>
message <Message 24>
message <Message 25>
11:45:50 : EVENT : Connection Reconnecting (Delay: 109ms)
11:45:51 : EVENT : Connection Reconnecting (Delay: 0ms)
11:45:53 : EVENT : Connection Reconnecting (Delay: 0ms)
11:45:56 : EVENT : Connection Reconnecting (Delay: 1698ms)

11:46:01 : EVENT : Connection Reconnected
message <Message 26>
message <Message 27>
message <Message 28>
message <Message 29>
message <Message 30>

++
+++ Chapter 5: Troubleshooting
++

+++ Reason code rc 2058 0x0000080a MQRC_Q_MGR_NAME_ERROR

Symptom:

C:\> amqsphac Q9 'QM3'
Sample AMQSPHAC start
MQCONNX ended with reason code 2058
Sample AMQSPHAC end

Likely reasons:
a) In Windows, using single quotes around the queue manager group.
b) MQSERVER and mqclient.ini are enabled and thus, in the hierarchy of precedence, they will be block the usage of the CCDT.
c) The queue manager group has multiple entries.

Actions:

a) In Windows, do not specify the queue manager name with single quotes:
C:\> amqsphac Q9 QM3

b) Reentry the precedence of MQSERVER and mqclient.ini over the CCDT

Notice that the MQSERVER is set (which means that it will be used first!)

C:\> set mq
MQFT_JAVA_LIB_PATH=C:\Program Files\IBM\WebSphere MQ\java\lib
MQFT_JRE_BIN_PATH=C:\Program Files\IBM\WebSphere MQ\java\jre\bin
MQSERVER=SYSTEM.DEF.SVRCONN/TCP/localhost(1414)

Action:

- Need to unset MQSERVER:

C:\> set MQSERVER=

- Comment out the CHANNEL stanza in the mqclient.ini

CHANNELS:
DefRecon=YES
ServerConnectionParms=SYSTEM.DEF.SVRCONN/TCP/localhost(1414)

- Check for the locations:
C:\Program Files\IBM\WebSphere MQ

- Set the location of the AMQCLCHL.TAB file:
set MQCHLLIB=C:\var\mqm\Qmgrs\QM_ANGELITO\@ipcc

c) Queue manager group with multiple entries

When the queue manager group has multiple entries and when the name of the group is NOT specified with a leading asterisk, then the return code 2058 (or other similar return codes) will be shown:

C:\> amqsphac Q9 QM7
Sample AMQSPHAC start
MQCONNX ended with reason code 2058
Sample AMQSPHAC end

C:\> amqsphac Q9 "QM7"
Sample AMQSPHAC start
MQCONNX ended with reason code 2058
Sample AMQSPHAC end

Solution: Specify the leading asterisk:
C:\> amqsphac Q9 *QM7
C:\> amqsphac Q9 "*QM7"

+++ Reason code rc 2162 0x00000872 MQRC_Q_MGR_STOPPING

Symptom:

message <Message 25>
MQPUT ended with reason code 2162
11:19:24 : EVENT : Reason(2162)
11:19:24 : EVENT : Connection Broken
MQCLOSE ended with reason code 2009
Sample AMQSPHAC end

Likely reason:
The queue manager is stopping and it cannot accept new connections or new requests.

Action:
Ensure that the queue manager is active.

+++ Reason code rc 2538 0x000009ea MQRC_HOST_NOT_AVAILABLE

Symptom:

C:\>amqsphac Q9 *QM1
Sample AMQSPHAC start
MQCONNX ended with reason code 2538
Sample AMQSPHAC end

Likely reason:
- The port for the MQ listener is not running
- The queue manager is not running
- The wrong port is specified
- The host name including the domain name is incorrect

Actions:
- Restart the port and/or the queue manager
- Ensure that the port and the hostname are properly specified in the CCDT.

+++ Reason code rc 2540 0x000009ec MQRC_UNKNOWN_CHANNEL_NAME

C:\MQ-doc\techdocs>amqsputc Q1 QM3
Sample AMQSPUT0 start
MQCONN ended with reason code 2540

Likely reason:
- The name in the CCDT for the client-connection channel does not have a counterpart server-connection channel in the target queue manager

Action:
- Create a corresponding server-connection channel in the target queue manager.

+++ end +++