Using a CCDT file to connect to multiple WebSphere MQ queue managers using JMS

IBM Techdoc: 7020862


Date last updated: 16-Mar-2011

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

+++ Objective

The objective of this technical document is to describe in detail how to exploit from a client application that is using the WebSphere MQ classes for JMS Version 7 the connection to multiple queue managers by using an MQ "Client Channel Definition Table" (CCDT). These queue managers are specified in a "queue manager group".

The QMNAME attribute in the CCDT corresponds to the QMANAGER attribute in the Connection Factory.

The JMS testing of the following scenarios will be described from the JMS perspective:

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

The creation of the CCDT and the above scenarios are described in detail for C-based programs, in the following techdoc.
Using a Client Channel Definition Table (CCDT) in WebSphere MQ V7 for Queue Manager Groups

The reference techdoc provides more information on what is a queue manager group, how to set the appropriate server-connection channels and client-connection channels, and how to perform the scenarios using samples that are written in C-code.

It is very important to review the following section of the MQ V7 Information Center to understand how a CCDT can be used with JMS applications and what are the limitations:


Using a client channel definition table with WebSphere MQ classes for JMS

The samples that are used to test the scenarios are sample java (JMS) code that is shipped with MQ V7:

- JmsJndiProducer.java
- JmsJndiConsumer.java

Their location is:

Windows:  C:\Program Files\IBM\WebSphere MQ\tools\jms\samples
AIX:      /usr/mqm/samp/jms/samples
Unix:     /opt/mqm/samp/jms/samples

The chapters of this document are:

Chapter 1: Prerequisites (Sample MDB, userid with proper authority, CCDT file)
Chapter 2: Defining a Connection Factory that uses the CCDT
Chapter 3: Testing the Scenarios
Chapter 4: Troubleshooting
+++ Related articles +++

For the details on how to exploit connection to multiple queue managers from an application that runs under the WebSphere Application Server, see the following related techdocs:

http://www.ibm.com/support/docview.wss?rs=171&uid=swg27020700
Using custom property connectionNameList to connect to WebSphere MQ multi-instance queue managers from WebSphere Application Server V7

http://www.ibm.com/support/docview.wss?rs=171&uid=swg27020701
Using a CCDT file to connect to WebSphere MQ multi-instance queue managers from WebSphere Application Server V7

+++ Note about Load 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 load 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 load balancing, you will need to alter the following attributes for the channels in the CCDT:

    AFFINITY(NONE)
    CLNTWGHT(1)

The following links provide more information about load balancing:

Queue manager groups in the CCDT
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.
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 load balancing is performed and applicable definitions are selected in alphabetical order. To enable load 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: Prerequisites

+++ Version of MQ

The version of the MQ JMS Client must be 7.0.1.0 or later.

[link to MQ documentation]

Automatic client reconnection

++ Starting the active and standby instance of a queue manager

The Scenario 7 is about multi-instance queue managers. Thus, it is necessary to properly start the active and standby instances of the queue manager.

Window-1 (MQ): host-1 veracruz:

Login to host-1 ("veracruz" in this example) and start the queue manager to allow a standby instance and display the status:

```bash
$ strmqm -x QMMI1
WebSphere MQ queue manager 'QMMI1' starting.
5 log records accessed on queue manager 'QMMI1' during the log replay phase.
Log replay for queue manager 'QMMI1' complete.
Transaction manager state recovered for queue manager 'QMMI1'.
WebSphere MQ queue manager 'QMMI1' started.

$ dspmq -x -m QMMI1
QMNAME(QMMI1)                                             STATUS(Running)
INSTANCE(veracruz) MODE(Active)
```

Notice that only the active instance is running.

Ensure that the local queue (Q_MDB) is defined. This is the queue which is going to be monitored by the Listener Port in WAS.

```bash
$ runmqsc QMMI1
define qlocal(Q_MDB) replace
  1 : define qlocal(Q_MDB) replace
  AMQ8006: WebSphere MQ queue created.
end
  2 : end
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
```
Window-2 (MQ): host-2 cbeech:

Login to host-2 ("cbeech" in this example) and start the queue manager to allow a standby instance and display the status:

$ strmqm -x QMMI1
WebSphere MQ queue manager 'QMMI1' starting.
A standby instance of queue manager 'QMMI1' has been started. The active instance is running elsewhere.

$ dspmq -x -m QMMI1
QMNAME(QMMI1) STATUS(Running as standby)
  INSTANCE(veracruz) MODE(Active)
  INSTANCE(cbeech) MODE(Standby)

At this point we have 2 instances of the queue manager:
- The active one is running in host-1 "veracruz"
- The standby is running in host-2 "cbeech".

++ Creating a Client Channel Definition Table (CCDT)

For complete instructions for creating a CCDT that includes queue manager groups, see the following techdoc (mentioned earlier):

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

++ Environment variables for the Client Channel Definition Table (CCDT)

The location of the CCDT file can be defined by using the following environment variable.

Unix (default):
  export MQCHLLIB=/var/mqm/qmgrs/QMgrName/@ipcc/

Windows (default):
  set MQCHLLIB=LOCATION\Qmgrs\QMgrName\@ipcc

The name of the CCDT file can be defined by:

Unix (default):
  export MQCHLTAB=AMQCLCHL.TAB
Windows (default):
   set MQCHLTAB=AMQCLCHL.TAB

++ Transfer a copy of the CCDT file to the hosts where the MQ client will be used

Transfer in binary mode, a copy of the CCDT file (AMQCLCHL.TAB) that was created from the instructions of the mentioned techdoc 7020848 into the hosts where the MQ client will run and which will use the CCDT.

Note: The AMQCLCHL.TAB file is a BINARY file, thus, if you use ftp, you will need to ensure that the transfer is in binary mode. The following shows a sample transfer from Windows to a Unix host:

C:\var\mqm> ftp veracruz.x.ibm.com
ftp> cd /var/mqm
ftp> binary
200 Switching to Binary mode.
ftp> put AMQCLCHL.TAB
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 File receive OK.
ftp: 30338 bytes sent in 0.00Seconds 30338000.00Kbytes/sec.
ftp> quit

In this example, the following directories will contain the CCDT file:

Windows: C:\var\mqm\AMQCLCHL.TAB
Unix: /var/mqm/AMQCLCHL.TAB

Thus, the following variables are set:

Windows: set MQCHLLIB= C:\var\mqm
Unix: export MQCHLLIB=/var/mqm

++ Configuration of necessary server-connection and client-connection channels

From the mentioned techdoc 7020848, see:
   Chapter 3: Configuration of the queue managers
This chapter describes the steps in the MQ Explorer to create a Connection Factory that uses a CCDT file.

You could use the JMSAdmin tool to do so. However, this document only shows how to use the JMSAdmin to display the Connection Factory.

++ About the attribute CCDTURL

A very important field that needs to be specified in the Connection Factory is called "CCDTURL" and it provides the URL of the full path where the CCDT file is located.

The format of the CCDT URL depends on the operating system where the CCDT is located.
The "file" protocol can use either three slashes ("///") or one slash ("/").
Note: do NOT use 2 slashes!!

Windows:
  file:///C:/directoryPath/AMQCLCHL.TAB
  or
  file:/C:/directoryPath/AMQCLCHL.TAB

Unix or Linux:
  file:///directoryPath/AMQCLCHL.TAB
  or
  file:/directoryPath/AMQCLCHL.TAB

As mentioned in the previous chapter, the locations for the CCDT files used in this document are:

Windows:   C:\var\mqm\AMQCLCHL.TAB
Unix:        /var/mqm/AMQCLCHL.TAB

Therefore, the corresponding CCDTURL values will be:

Windows:
  file:/C:/var/mqm/AMQCLCHL.TAB

Unix or Linux:
  file:///var/mqm/AMQCLCHL.TAB
++ MQ Explorer: Create a Connection Factory via MQ Explorer with a CCDTURL

From the left navigation panel of the MQ Explorer V7, select "JMS Administered Objects" and expand the desired initial context (such as "file:/C:/var/mqm/jmsadmin/")

Right click on "Connection Factories" and select New > Connection Factory

From the dialogs for "New Connection Factory":

Enter the value for "Name". In this document, it is called: CF-CCDT

Under "Transport", select "MQ Client" from the pull down menu.

Click Next until you see the "Change Properties" page that shows different tabs.

In the "Connection" tab, enter the name of the queue manager group without double quotes, such as: *QM1*

Note about "Connection List":
The CF needs to have something in the field "Connection List", the default is "localhost(1414)". In the case of using a CCDT, this field should include the hostname and port of one of the queue managers in the queue manager group.

If you try to blank out this field, MQ Explorer will issue an error.
### New Connection Factory

#### Change properties

Change the properties of the new Connection Factory

<table>
<thead>
<tr>
<th>General</th>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td></td>
</tr>
<tr>
<td>Reconnection</td>
<td></td>
</tr>
<tr>
<td>Channels</td>
<td></td>
</tr>
<tr>
<td>SSL</td>
<td></td>
</tr>
<tr>
<td>Exits</td>
<td></td>
</tr>
<tr>
<td>Broker</td>
<td></td>
</tr>
<tr>
<td>Temporary queues</td>
<td></td>
</tr>
<tr>
<td>Temporary topics</td>
<td></td>
</tr>
<tr>
<td>Subscriber</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base queue manager:</td>
</tr>
<tr>
<td>Broker queue manager:</td>
</tr>
<tr>
<td>Connection list:</td>
</tr>
<tr>
<td>Coded character set ID:</td>
</tr>
</tbody>
</table>
To specify the CCDTURL, you need to be in the "Channels" page. Notice that you cannot type directly the value for the field. Here are the steps to enter a value:
1: Click on the “Edit:” button
2: Click on "File".
3: Click on "Select..." and use a File Browser to select the desired file. In this case the file is located at:
   C:\var\mqm\AMQCLCHL.TAB

Continue with the process until you finish creating the Connection Factory.
The completed Connection Factory will be shown:
++ Create a Destination queue that does not have a queue manager name

Right click on "Destinations" and select New > Destination …
Enter the value for "Name". In this document it is:
Q1-NoQMgrName
Under the "General" page for the properties:
- Do NOT enter a value for the field "Queue manager:"
- Enter the value for the "Queue". In this document is:
Q1

++ Using JMSAdmin to display the values of the Connection Factory and the Destination

There are several ways to use the JMSAdmin. I found that the easiest way is to modify the following file (which is shipped with the code):

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

Notes:
- MQ Support recommends to do not modify files under the installation directory.
- This file is a configuration file and affects only the JMSAdmin tool, thus, the risk is minimum if you make a typo when editing this file. 
- The drawback is that if you install a Fix Pack this file will be overwritten.

There are 2 modifications to the file:

a) Indicate that you want to use a File System context. This means that all the JNDI objects will be stored in the file ".bindings".

Ensure that there is ONLY one choice that is not commented out:

```
#INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
#INITIAL_CONTEXT_FACTORY=com.ibm.websphere.naming.WsnInitialContextFactory
```

b) Indicate where you want the .bindings file to be stored.

Example for Windows:

```
#PROVIDER_URL=ldap://localhost/o=ibm,c=us
PROVIDER_URL=file://C:/var/mqm/jmsadmin
#PROVIDER_URL=iiop://localhost/
```

Example for Unix:

```
PROVIDER_URL=file:///var/adm/jmsadmin
```

++ Invoking JMSAdmin

After you have done the above customization, now you can run the tool:

Windows:
```
C:\Program Files\IBM\WebSphere MQ\java\bin\JMSAdmin
```

Unix:
```
/opt/mqm/java/bin/JMSAdmin
```

Issue the following to display the properties for the Connection Factory CF-CCDT
```
    display CF(CF-CCDT)
```

Notice the following 3 values:
```
    CCDTURL(file:C:/var/mqm/AMQCLCHL.TAB)
```
CONNECTIONNAMELIST(localhost(1414))
QMANAGER(QM1)

The value in the CONNECTIONNAMELIST will be ignored because the CCDTURL is being specified.

The value of QMANAGER is going to indicate which Queue Manager Group to use.

To keep the listings short, only a subset of the output lines is shown.

InitCtx> display CF(CF-CCDT)

ASYNCEXCEPTION(ALL)
...
BROKERVER(UNSPECIFIED)
  CCDTURL(file:/var/mqm/AMQCLCHL.TAB)
  CCSID(819)
...
CONNECTIONNAMELIST(localhost(1414))
...
QMANAGER(QM1)
...

Issue the following to display the properties for the Destination Queue Q1

display q(Q1-NoQMgrName)

Notice the following 2 values:

  QMANAGER()
  QUEUE(Q1)

The value for QMANAGER needs to be blank (null). The MQ client will select the proper queue manager from the CCDT at runtime.

InitCtx> display q(Q1-NoQMgrName)

  CCSID(1208)
  ...
For each scenario, you will need to modify the attribute QMNAME of the CF-CCDT connection factory.

An excerpt of the output of the JMS samples will be shown here, to reduce the length of this techdoc.

Notes:
1: The QMNAME attribute in the CCDT corresponds to the QMANAGER attribute in the Connection Factory.
2: The MQ Explorer does not allow the entering a single quote or a double quote in the field for QMNAME (Base Queue Manager in the Connections page). However, it is possible to do so by using the JMSAdmin tool, as follows:

   InitCtx> alter cf(CF-CCDT) QMANAGER("QM3")

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

Queue Manager Group: 'QM3'
Connection Factory: InitCtx> dis cf(CF-CCDT)
   QMANAGER(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 member.

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

C:\> java JmsJndiProducer -i file:/C:/var/mqm/jmsadmin -c CF-CCDT -d Q1-NoQMgrName

Initial context found!
Sent message:
   JMSMessage class: jms_text
   JMSType: null
   JMSMessageID: ID:414d5120514d332020202020204d543a5a20001e02
   JMSTimestamp: 1297897439000
   JMSCorrelationID: null
   JMSDestination: queue:///Q1
   JMSReplyTo: null
   JMSRedelivered: false
   JMSXAppID: WebSphere MQ Client for Java
   JMS_IBM_PutApplType: 28
   JMS_IBM_PutDate: 20110216
   JMS_IBM_PutTime: 23035945
JmsJndiProducer: Your lucky number today is 937
SUCCESS

In the host aemaix1 that has the QM3 queue manager, the following is done to verify that the message arrived to the queue Q1. Notice that the text "Your lucky number today is 937" is in the output.

$ amqsbcg Q1 QM3

****Message descriptor****

StrucId : 'MD ' Version : 2
Report : 0  MsgType : 8  
Expiry : -1  Feedback : 0  
Encoding : 273  CodedCharSetId : 1208  
Format : 'MQHRF2'  
MsgId : X'414D5120514D332020202020202020204D543A5A20001E02'  
CorrelId : X'000000000000000000000000000000000000000000000000000'  
ReplyToQ : ''  
ReplyToQMgr : 'QM3'  
** Identity Context  
** Origin Context  
PutApplType : '28'  
PutApplName : 'WebSphere MQ Client for Java'  
PutDate : '20110216'  
PutTime : '23035945'  
ApplOriginData : ''  
**** Message ****  
length - 195 bytes  
00000000: 5246 4820 0000 0002 0000 0094 0000 0111 'RFH ............'  
00000010: 0000 04B8 4D51 5354 5220 2020 0000 0000 '....MQSTR ....'  
00000020: 0000 04B8 0000 0020 3C 6D 6364 3E3C 4D73 '...¸...<mcd><Ms'  
00000030: 643E 6A6D 73F7 747874 3C2F 4D73 643E 3C2F 6D63 'd>jms_text</Msd>'  
00000040: 3C2F 6D63 3E 20 20 4A6D 734A64 6950 726F 6475 '><jmsJndiProdu'  
00000050: 6365 3A 20 7072 6f64 6963 756c 6c20 6c6f 6c6f 'cer: Your lucky '  
00000060: 7461 6765 7320 6973 20 3933 37 00 00 00 00 00 00 'number today is '  
00000070: 937 '  
000000C0: 3933 37
+ **Scenario 2** - Simplest case, using same QMNAME for a single queue manager (QM3 in another host: host2)

Queue Manager Group: ‘QM3’
Connection Factory: InitCtx> dis cf(CF-CCDT)
   QMANAGER(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 Connection Factory, 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'
Connection Factory: InitCtx> dis cf(CF-CCDT)
    QMANAGER(QM1)

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

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

DEFINE CHANNEL(QM1.B) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
    CONNAME('aemtux1.*.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:\\angel\\coding\\MQ\\jms\\mqv7-jndi>java JmsJndiProducer -i
file:/C:/var/mqm/jmsadm
in -c CF-CCDT -d Q1-NoQMgrName
Initial context found!
Sent message:
...
JmsJndiProducer: Your lucky number today is 78
SUCCESS
```

All the messages were received by QM1 in veracruz.

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

The reason for this behavior is explained in:

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.*.com(1431)')
CHANNEL(QM1.B)  CONNAME('aemtux1.*.com(1425)')
```

When both queue managers are running the QM1.A channel is always chosen first.

+++ Note about load balancing

When the channels for this scenario in CCDT were modified as follows, then there was a rough round robin distribution of the message among the active queue managers.

```
   AFFINITY(NONE)  (default is PREFERRED)
   CLNTWGHT(1)     (default is 0)
```
**Scenario 4 - Queue Manager Group: *QM1 (with leading asterisk)**

Queue Manager Group: 'QM1'  
Connection Factory: InitCtx> dis cf(CF-CCDT)  
QMANAGER(*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.*.com(1431)') QMNAME(QM1) REPLACE

This queue manager is actually named QM1:  
DEFINE CHANNEL(QM1.B) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +  
CONNAME('aemtux1.*.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.*.com(1422)') QMNAME(QM1) REPLACE

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.

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)

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.

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

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

Run the sample to put messages.

The messages are received by QM2 in aemaix1.

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

Queue Manager Group: ''
Connection Factory: InitCtx> dis cf(CF-CCDT)
QMANAGER()

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

DEFINE CHANNEL(DEFAULT.A) CHLTYP(CLNTCONN) TRPTYPE(TCP) +
CONNAME('veracruz.*.com(1431)') QMNAME(' ') REPLACE

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

DEFINE CHANNEL(DEFAULT.C) CHLTYP(CLNTCONN) TRPTYPE(TCP) +
CONNAME('aemtux1.*.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 QmrgName 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 QmrgName 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

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

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

Finally, stop QM3 in aemaix1 and resume the test:

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).
**Scenario 6 - Queue Manager Group: QMGROUP1**

Queue Manager Group: 'QMGROUP1'
Connection Factory: InitCtx> dis cf(CF-CCDT) QMANAGER('*QMGROUP1')

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

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

```
DEFINE CHANNEL(QMGROUP1.B) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
  CONNAME('veracruz.*.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:
- QM2 in aemaix1
- QM3 in veracruz

Run the sample to put messages:

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

Stop QM2 in aemaix1 and repeat sending the messages.

The messages are now received by QM3 in veracruz.
+ Scenario 7 - Multi-Instance Queue Manager

Queue Manager Group: 'QMMI1'
Connection Factory: InitCtx> dis cf(CF-CCDT)
QMANAGER(QMMI1)

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

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

Notice that because the group name is the same as the single member 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 member 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 (standby).

Send messages.

The messages are received in Veracruz (because it is the Active).
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.

    Send messages again.

The messages are received by the instance running in cbeech (it was the standby, but after the failover is now the active).
+++ Chapter 4: Troubleshooting

+ reason '2059' ('MQRC_Q_MGR_NOT_AVAILABLE').

C:\>
java JmsJndiProducer -i file:/C:/var/mqm/jmsadmin -c CF-CCDT -d Q1-NoQMgrName

Initial context found!
com.ibm.msg.client.jms.DetailedIllegalStateException: JMSMQ0018: Failed to connect to queue manager 'QM_ANGELITO' with connection mode 'Client' and host name 'localhost(1414)'. Check the queue manager is started and if running in client mode, check there is a listener running. Please see the linked exception for more information.
Inner exception(s):
com.ibm.mq.MQException: JMSCMQ0001: WebSphere MQ call failed with compcode '2' ('MQCC_FAILED') reason '2059' ('MQRC_Q_MGR_NOT_AVAILAIBLE').
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2059;AMQ9204: Connection to host 'localhost(1414)' rejected.
[1=com.ibm.mq.jmqi.JmqiException[CC=2;RC=2059;AMQ9213: A communications error for occurred. [1=java.net.ConnectException[Connection refused : connect],3=localhost]],3=localhost(1414),5=RemoteTCPConnection.connnectUsingLocalAddress]
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2059;AMQ9213: A communications error for occurred. [1=java.net.ConnectException[Connection refused: connect],3=localhost ] java.net.ConnectException: Connection refused: connect

FAILURE

Possible reasons:

a) The desired queue manager is not running.
b) The MQSERVER variable was set and it takes precedence over CCDT. Thus, the CCDT was NOT used and MQSERVER is pointing to a queue manager that is not running.

Possible actions:

a) Start the queue manager.
b) Unset MQSERVER
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)

Need to unset MQSERVER:
set MQSERVER=

+ reason '2278' ('MQRC_CLIENT_CONN_ERROR').

Setup: JMSAdmin was used to set a QMANAGER value with single quotes or double quotes:

C:\angel\coding\MQ\jms\mqv7-jndi>java JmsJndiProducer -i
file:/C:/var/mqm/jmsadm
in -c CF-CCDT -d Q1-NoQMgrName
Initial context found!
com.ibm.msg.client.jms.DetailedJMSException: JMSWMQ2020: Failed to connect to queue manager "QM3" with connection mode 'Client' and supplied CCDT URL 'file:C:/var/mqm/AMQCLCHL.TAB', see linked exception for more information. Check the queue manager is started and if running in client mode, check there is a listener running. Please see the linked exception for more information.
Inner exception(s):
com.ibm.mq.MQException: JMSCMQ0001: WebSphere MQ call failed with compcode '2' ('MQCC_FAILED') reason '2278' ('MQRC_CLIENT_CONN_ERROR').
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2278
FAILURE

Action:
Modify the QMANAGER and do not include single or double quotes.

+++ end +++