Exploiting the Automatic Client Reconnect feature in WebSphere MQ JMS 7.0.1

IBM Techdoc: 7023313

http://www.ibm.com/support/docview.wss?rs=171&uid=swg27023313

Date last updated: 19-Jun-2012

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 the automatic client reconnect feature in WebSphere MQ JMS 7.0.1.
The scope is for standalone JMS applications and not for JMS applications that run inside an application server.

There are 2 scenarios covered in detail:

1) Using a sample that utilizes the Java Naming and Directory Interface (JNDI) which reference Connection Factories and Destinations.

In UNIX, the JMSAdmin tool is used to populate the Java Naming and Directory Interface (JNDI) with the JMS Administrative objects that serve as a link between the JMS application and the physical queues and topics in the queue manager. This JNDI is in the form of a file named ".bindings".

In Windows and in Linux x86, in addition to the JMSAdmin tool, the GUI MQ Explorer tool can be used to populate the JNDI into a file named ".bindings".

2) Using a sample that does not use JNDI, and instead, specifies the characteristics of the Connection Factories and Destinations inside the code.

+++ Important technote

It is highly recommended that you review the details described in the following technote:

http://www.ibm.com/support/docview.wss?uid=swg21508357
Technote: 1508357
Using WebSphere MQ automatic client reconnection with the WebSphere MQ classes for JMS

++ Contents

+++ Software used +++

Windows:
  WebSphere MQ 7.0.1.6 queue manager and JMS client
  Java 1.6 SR 7

Linux x86 32-bit
  WebSphere MQ 7.0.1.6 queue manager and JMS client
  Java 1.6 SR 9

+++ DISCLAIMER

+++ About the Samples

There are 3 samples provided with this techdoc.
  JmsJndiProducerLoop.java => Uses JmsConnectionFactory and JNDI
  JmsProducerLoop.java = > Uses JmsConnectionFactory (no JNDI)
  JmsProducerLoopMQCF.java => Uses MQConnectionFactory (no JNDI)

a) JmsJndiProducerLoop.java

It uses JMS administrative objects that are defined in the JNDI.
An excerpt of the comments in the header is shown below:

-- begin excerpt

```
* This sample is based on the JmsJndiProducer.java sample provided with MQ V7:
 *   Windows: C:\Program Files\IBM\WebSphere MQ\tools\jms\samples\JmsJndiProducer.java
 *   UNIX:    /opt/mqm/samp/jms/samples/JmsJndiProducer.java
 *
 * A JMS producer (sender or publisher) application that sends a simple message to the named
 * destination (queue or topic) by looking up the connection factory instance and the destination
 * instance in an initial context (This sample supports file system context only).
 *
 * JmsJndiProducerLoop is an extension in which inside a forever loop, a message is sent
 * to the destination, with an interval of 5 seconds between each message.
 * At runtime, the user needs to use Ctrl-C to terminate the endless loop.
```

-- end excerpt

b) About JmsProducerLoop.java (no JNDI)

An excerpt of the comments in the header is shown below for JmsProducerLoop.java

-- begin excerpt

```
* This sample is based on the JmsProducer.java sample provided with MQ V7:
 *   Windows: C:\Program Files\IBM\WebSphere MQ\tools\jms\samples\JmsProducer.java
 *   UNIX:    /opt/mqm/samp/jms/samples/JmsProducer.java
 *
 * A JMS producer (sender or publisher) application that sends a simple message to the named
 * destination (queue or topic).
 *
 * JmsProducerLoop is an extension in which inside a forever loop, a message is sent
 * to the destination, with an interval of 5 seconds between each message.
 * At runtime, the user needs to use Ctrl-C to terminate the endless loop.
 *
 * CUSTOMIZATION: You need to customize the value for: connectionNameList
     private static String connectionNameList = "host1(1414),host2(1414)";

      // Create a connection factory
      JmsFactoryFactory ff = JmsFactoryFactory.getInstance(WMQConstants.WMQ_PROVIDER);
      JmsConnectionFactory cf = ff.createConnectionFactory();

      // Set the properties
      cf.setStringProperty(WMQConstants.WMQ_CHANNEL, channel);
      cf.setIntProperty(WMQConstants.WMQ_CONNECTION_MODE, WMQConstants.WMQ_CM_CLIENT);
      cf.setIntProperty(WMQConstants.WMQ_CLIENT_RECONNECT_OPTIONS,
WMQConstants.WMQ_CLIENT_RECONNECT);
      cf.setStringProperty(WMQConstants.WMQ_CONNECTION_NAME_LIST, connectionNameList);
```

-- end excerpt

c) JmsProducerLoopMQCF.java

An excerpt of the comments in the header is shown below for JmsProducerLoopMQCF.java

-- begin excerpt

```
 * Sample for MQConnectionFactory based on
 * the JmsProducer.java sample from MQ V7 which uses JmsConnectionFactory
 *
 * A JMS producer (sender or publisher) application that sends a simple message to the named
 * destination (queue or topic).
 *
 * JmsProducerLoopMQCF is an extension in which inside a forever loop, a message is sent
 * to the destination, with an interval of 5 seconds between each message.
 * At runtime, the user needs to use Ctrl-C to terminate the endless loop.
 * This samples uses MQConnectionFactory.
 *
 * CUSTOMIZATION: You need to customize the value for: connectionNameList
    private static String connectionNameList = "host1(1414),host2(1414)";

      if (isTopic) {
        // Create a connection factory
        MQTopicConnectionFactory cf = new MQTopicConnectionFactory();

        // Set the properties
        cf.setTransportType(WMQConstants.WMQ_CM_CLIENT);
        cf.setChannel(channel);
        cf.setConnectionNameList(connectionNameList);
        cf.setClientReconnectOptions(WMQConstants.WMQ_CLIENT_RECONNECT);

        // Create JMS objects
        connection = cf.createConnection();
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        destination = session.createTopic(destinationName);
      }
      else {
          // Create a connection factory
        MQQueueConnectionFactory cf = new MQQueueConnectionFactory();

        // Set the properties
        cf.setTransportType(WMQConstants.WMQ_CM_CLIENT);
        cf.setChannel(channel);
        cf.setConnectionNameList(connectionNameList);
        cf.setClientReconnectOptions(WMQConstants.WMQ_CLIENT_RECONNECT);

        // Create JMS objects
        connection = cf.createConnection();
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        destination = session.createQueue(destinationName);
```

-- end excerpt

++
+++ Chapter 1: Configuration in UNIX via JMSAdmin
++

It is necessary to use the JMSAdmin tool to create, view or alter the JNDI.

The JMSAdmin tool is located in /opt/mqm/java/bin (non-AIX) or /usr/mqm/java/bin (AIX)

a) Modify your profile to facilitate the running MQ tools and samples:

a.1) Add the following MQ directories into the PATH in your .profile (or .bashrc in Linux):

```
## Non-AIX:
export MQ_ROOT=/opt/mqm
export
PATH=$PATH:$MQ_ROOT/bin:$MQ_ROOT/java/bin:$MQ_ROOT/samp/bin:$MQ_ROOT/s
amp/jms/samples

## AIX:
export MQ_ROOT=/usr/mqm
export
PATH=$PATH:$MQ_ROOT/bin:$MQ_ROOT/java/bin:$MQ_ROOT/samp/bin:$MQ_ROOT/s
amp/jms/samples
```

a.2) Run the MQ script that sets the proper JMS and JAVA environment variables and CLASSPATH for MQ.

This can be accomplished by adding the following into the .profile (or .bashrc in Linux):

+ begin quote

```
## Customization for WMQ
if [ -f $MQ_ROOT/java/bin/setjmsenv64 ]
then
. setjmsenv64
else
. setjmsenv
fi
```
+ end quote

For running 32-bit applications (for example, under Linux X86 32-bit), run:

```
. setjmsenv
```

For running 64-bit applications (for other platforms, such as AIX, Solaris, HP-UX, Linux 64-bit), run the following script that is only provided if your platform supports 64-bit:

```
. setjmsenv64
```

You must type the dot, then space, then "setjmsenv" or "setjmsenv64".
An example of running it is shown below for Linux x86 32-bit:

```
rivera@veracruz: /home/rivera
$ . setjmsenv
MQ_JAVA_INSTALL_PATH is /opt/mqm/java
MQ_JAVA_DATA_PATH is /var/mqm
MQ_JAVA_LIB_PATH is /opt/mqm/java/lib
CLASSPATH is
:/opt/mqm/java/lib/com.ibm.mq.jar:/opt/mqm/java/lib/com.ibm.mqjms.jar:/opt/
mqm/samp/jms/samples:/opt/mqm/samp/wmqjava/samples
```

a.3) Expand the CLASSPATH in your .profile or .bashrc:

```
## Need to manually add the following to CLASSPATH to compile JMS programs
export CLASSPATH=$CLASSPATH:/opt/mqm/java/lib/jms.jar
```

b) Creation of physical objects (Queue and Topic) in the queue manager

The following were used in UNIX:

Queue Manager name: QM_VER
Host: veracruz.x.com
Port: 1414

```
$ runmqsc QM_VER
DEFINE QL(Q2)
DEFINE TOPIC(T2) TOPICSTR('TOPIC2')
END
```

c) Create a directory where the JMS configuration objects will be located in a file name ".bindings".
For example, in UNIX, you can create the following subdirectory in the same directory where the other MQ objects are stored:

```
   mkdir  /var/mqm/JNDI-Directory
```

d) Copy the JMSAdmin.config file from its default location, into /var/mqm

The default file in Solaris, Linux and HP-UX is:
```
       /opt/mqm/java/bin/JMSAdmin.config
```
The default file in AIX is:
```
       /usr/mqm/java/bin/JMSAdmin.config
```

In this document we follow the best practice of not writing files under /opt/mqm (/usr/mqm). Thus, you need to copy the JMSAdmin.config file:
```
   cp /opt/mqm/java/bin/JMSAdmin.config   /var/mqm/JMSAdmin.config
```

Ensure that you have write privileges for the file (the original file is read-only):
```
  chmod 644 /var/mqm/JMSAdmin.config
```

e) Modify /var/mqm/JMSAdmin.config

These 2 variables need to be properly specified in the config file.

e.1) INITIAL_CONTEXT_FACTORY

The following is common for UNIX and Windows, and it indicates that a file will be used:

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

e.2) PROVIDER_URL

For UNIX: notice the 3 forward slashes after "file:".

```
PROVIDER_URL=file:///var/mqm/JNDI-Directory
```

Note: If there are not 3 slashes, such as 2, 4, 5, etc. then the following error is displayed by JMSAdmin at runtime:
```
InitCtx> DEFINE TCF(testCF)
Unable to bind object
```

For Windows: notice only 1 forward slash after "file:" and notice the drive letter "C:/"

```
PROVIDER_URL=file:/C:/JNDI-Directory
```

Notice that ONLY one INITIAL_CONTEXT_FACTORY and PROVIDER_URL must be uncommented. If there are more uncommented lines, then the last entry is the winner, and overwrites any earlier entries of the same type.

f) Create a shell script called "myJMSAdmin.sh" that has these lines:

```
<beginning of script>
#!/usr/bin/ksh
echo "running: JMSAdmin -cfg /var/mqm/JMSAdmin.config"

if [ -f /opt/mqm/java/bin/setjmsenv64 ]
then
. setjmsenv64
else
. setjmsenv
fi

JMSAdmin -cfg /var/mqm/JMSAdmin.config
<end of script>
```

This script will invoke the JMSAdmin tool using the JMSAdmin.config that was copied and modified under /var/mqm.

Ensure that the myJMSAdmin.sh script is in a directory under $PATH.

Ensure that the script is executable:

```
   chmod 755 myJMSAdmin.sh
```

g) Start JMSAdmin by running the script - notice the prompt: InitCtx>

```
rivera@veracruz: /home/rivera
$ myJMSAdmin.sh
```

+ begin quote to show a run of the tool

```
running: JMSAdmin -cfg /var/mqm/JMSAdmin.config
MQ_JAVA_INSTALL_PATH is /opt/mqm/java
MQ_JAVA_DATA_PATH is /var/mqm
MQ_JAVA_LIB_PATH is /opt/mqm/java/lib
CLASSPATH is
:/opt/mqm/java/lib/com.ibm.mq.jar:/opt/mqm/java/lib/com.ibm.mqjms.jar:/opt/
mqm/samp/jms/samples:/opt/mqm/samp/wmqjava/samples
```

```
Starting WebSphere MQ classes for Java(tm) Message Service Administration

InitCtx>

+ end quote
```

Notice the prompt for the tool:

```
InitCtx>
```

It means: Initial Context

h) Creation of JMS Administrative objects

For a complete list of the properties for the JNDI objects, see:
http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp?topic=%2Fcom.ibm.mq.csqzaw.doc%2Fjm10910_.htm
WebSphere MQ > Using Java > WebSphere MQ classes for JMS
Properties of WebSphere MQ classes for JMS objects

```
# Define Connection Factory called CF2REC
# Notice the 2 important options:
#   CONNECTIONNAMELIST
#   CLIENTRECONNECTOPTIONS(ANY)
# Suggestion: Do not specify a value for QMGR to provide more flexibility. If you specify a name, then only when the queue manager is named the same (such as in a multi-instance queue manager) the reconnection will work.

InitCtx> DEF CF(CF2REC) TRANSPORT(CLIENT) CHANNEL(SYSTEM.DEF.SVRCONN)
CONNECTIONNAMELIST(angelito.x.com(1414),veracruz.x.com(1414))
CLIENTRECONNECTOPTIONS(ANY)

# Define Queue Q2

InitCtx> DEF Q(Q2) QUEUE(Q2)

# Define Topic T2

InitCtx> DEF T(T2) TOPIC(TOPIC2)

# Display all items in the Context:

InitCtx> DIS CTX
```

```
  JMSADM4089 InitCtx

      .bindings              java.io.File
   a  T2                     com.ibm.mq.jms.MQTopic
   a  CF2REC                 com.ibm.mq.jms.MQConnectionFactory
   a  Q2                     com.ibm.mq.jms.MQQueue

  4 Object(s)
    0 Context(s)
    4 Binding(s), 3 Administered

InitCtx> DISPLAY  CF(CF2REC)

    ASYNCEXCEPTION(ALL)
    BROKERCCSUBQ(SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE)
    BROKERCONQ(SYSTEM.BROKER.CONTROL.QUEUE)
    BROKERPUBQ(SYSTEM.BROKER.DEFAULT.STREAM)
    BROKERQMGR()
    BROKERSUBQ(SYSTEM.JMS.ND.SUBSCRIBER.QUEUE)
    BROKERVER(UNSPECIFIED)
    CCSID(819)
    CHANNEL(SYSTEM.DEF.SVRCONN)
    CLEANUP(SAFE)
    CLEANUPINT(3600000)
    CLIENTRECONNECTOPTIONS(ANY)
    CLIENTRECONNECTTIMEOUT(1800)
    CLONESUPP(DISABLED)
    COMPHDR(NONE )
    COMPMSG(NONE )
CONNECTIONNAMELIST(angelito.x.com(1414),veracruz.x.com(1414))
    CONNOPT(STANDARD)
    FAILIFQUIESCE(YES)
    HOSTNAME(angelito.x.com)
    LOCALADDRESS()
    MAPNAMESTYLE(STANDARD)
    MSGBATCHSZ(10)
    MSGRETENTION(YES)
    MSGSELECTION(CLIENT)
    OPTIMISTICPUBLICATION(NO)
    OUTCOMENOTIFICATION(YES)
    POLLINGINT(5000)
    PORT(1414)
    PROCESSDURATION(UNKNOWN)
    PROVIDERVERSION(UNSPECIFIED)
    PUBACKINT(25)
    QMANAGER()
    RECEIVEISOLATION(COMMITTED)
    RESCANINT(5000)
    SENDCHECKCOUNT(0)
    SHARECONVALLOWED(YES)
    SPARSESUBS(NO)
    SSLFIPSREQUIRED(NO)
    SSLRESETCOUNT(0)
    STATREFRESHINT(60000)
    SUBSTORE(BROKER)
    SYNCPOINTALLGETS(NO)
    TARGCLIENTMATCHING(YES)
    TEMPMODEL(SYSTEM.DEFAULT.MODEL.QUEUE)
    TEMPQPREFIX()
    TEMPTOPICPREFIX()
    TRANSPORT(CLIENT)
    USECONNPOOLING(YES)
    VERSION(7)
    WILDCARDFORMAT(TOPIC_ONLY)


InitCtx> DISPLAY Q(Q2)

    CCSID(1208)
    ENCODING(NATIVE)
    EXPIRY(APP)
    FAILIFQUIESCE(YES)
    MDMSGCTX(DEFAULT)
    MDREAD(NO)
    MDWRITE(NO)
    MSGBODY(UNSPECIFIED)
    PERSISTENCE(APP)
    PRIORITY(APP)
    PUTASYNCALLOWED(AS_DEST)
    QMANAGER()
    QUEUE(Q2)
    READAHEADALLOWED(AS_DEST)
    READAHEADCLOSEPOLICY(DELIVER_ALL)
    RECEIVECCSID(1208)
    RECEIVECONVERSION(CLIENT_MSG)
    REPLYTOSTYLE(DEFAULT)
    TARGCLIENT(JMS)
    VERSION(7)

InitCtx> DISPLAY T(T2)

    BROKERCCDURSUBQ(SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE)
    BROKERDURSUBQ(SYSTEM.JMS.D.SUBSCRIBER.QUEUE)
    BROKERPUBQ()
    BROKERPUBQMGR()
    BROKERVER(V1)
    CCSID(1208)
    ENCODING(NATIVE)
    EXPIRY(APP)
    FAILIFQUIESCE(YES)
    MDMSGCTX(DEFAULT)
    MDREAD(NO)
    MDWRITE(NO)
    MSGBODY(UNSPECIFIED)
    MULTICAST(ASCF)
    PERSISTENCE(APP)
    PRIORITY(APP)
    PUTASYNCALLOWED(AS_DEST)
    READAHEADALLOWED(AS_DEST)
    READAHEADCLOSEPOLICY(DELIVER_ALL)
    RECEIVECCSID(1208)
    RECEIVECONVERSION(CLIENT_MSG)
    REPLYTOSTYLE(DEFAULT)
    TARGCLIENT(JMS)
    TOPIC(TOPIC2)
    VERSION(7)
    WILDCARDFORMAT(TOPIC_ONLY)

# Exit JMSAdmin

InitCtx> end

Stopping Websphere MQ classes for Java(tm) Message Service Administration
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++ Chapter 2: Scenario using JNDI - UNIX
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Sample JMS code used in this chapter:
  JmsJndiProducerLoop.java

++ Compile the sample:

```
$ javac JmsJndiProducerLoop.java
Note: JmsJndiProducerLoop.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

++ Running the sample from Linux

Open 3 windows:

Window 1: To run the JMS sample
Window 2: To start and stop the queue manager QM_ANGELITO in Windows
Window 3: To start and stop the queue manager QM_VER in Linux

Window 2: Start the queue manager QM_ANGELITO in Windows

```
C:\> strmqm QM_ANGELITO
```

Window 3: Start the queue manager QM_VER in Linux

```
$ strmqm QM_VER
```

Window 1: Run the sample code specifying the Connection Factory that has a connectionNameList and a queue destination.

```
$ java JmsJndiProducerLoop -i file:///var/mqm/JNDI-Directory -c CF2REC -d Q2
Initial context found!
Sent message:

  JMSMessage class: jms_text
  JMSType:          null
  JMSDeliveryMode:  2
  JMSExpiration:    0
  JMSPriority:      4
  JMSMessageID:     ID:414d5120514d5f414e47454c49544f20b3ab9d4e20003405
  JMSTimestamp:     1318959303026
  JMSCorrelationID: null
  JMSDestination:   queue:///Q2
  JMSReplyTo:       null
  JMSRedelivered:   false
    JMSXAppID: WebSphere MQ Client for Java
    JMSXDeliveryCount: 0
    JMSXUserID: rivera
    JMS_IBM_PutApplType: 28
    JMS_IBM_PutDate: 20111018
    JMS_IBM_PutTime: 17104412
JmsJndiProducerLoop: Your lucky number today is 25
```

Window 2: Verify that Q2 in QM_ANGELITO is receiving messages:

```
C:\> amqsget Q2 QM_ANGELITO
amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
```

Window 3: Verify that Q2 in QM_VER is NOT receiving messages

```
$ amqsget Q2 QM_VER
Sample AMQSGET0 start
(no messages)
```

Window 2: End the queue manager with flags to terminate immediately and to notify the connected clients to reconnect

```
C:\> endmqm -ir QM_ANGELITO
Waiting for queue manager 'QM_ANGELITO' to end.
WebSphere MQ queue manager 'QM_ANGELITO' ending.
WebSphere MQ queue manager 'QM_ANGELITO' ended.
```

Window 1: Except for a pause for the MQ client code to detect the termination of the first queue manager and to reconnect to the second queue manager, there should not be any messages that indicate that a reconnect is happening, and the sample should continue producing messages.

```
Sent message:

  JMSMessage class: jms_text
…
    JMS_IBM_PutTime: 17362851
JmsJndiProducerLoop: Your lucky number today is 516
```

Window 3: Verify that Q2 in QM_VER in Linux is now receiving messages:

```
$ amqsget Q2 QM_VER
amqsget Q2 QM_VER
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
```

Window 2: Verify that Q2 in QM_ANGELITO in Windows is NOT receiving messages (because the Queue manager is no longer running)

```
C:\> amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
MQCONN ended with reason code 2059
```

Window 2: Restart the queue manager QM_ANGELITO

```
C:\> strmqm QM_ANGELITO
```

Window 2: Notice that the sample is still connected to QM_VER and queue Q2 in QM_ANGELITO is NOT receiving messages:

```
C:\> amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
no more messages
Sample AMQSGET0 end
```

Window 1: Notice that the sample is still in the loop to produce messages.

Window 3: Now terminate QM_VER with the flags for immediate termination and to tell the clients to reconnect.

```
$ endmqm -ir QM_VER
```

```
WebSphere MQ queue manager 'QM_VER' ending.
WebSphere MQ queue manager 'QM_VER' ended.
```

Window 1: Notice that after a brief pause, the sample should continue producing messages.

Window 2: Verify that queue Q2 in QM_ANGELITO is now receiving messages:

```
C:\> amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
message <RFH ☻>
message <RFH ☻>
```

Window 1: To terminate the sample code, enter: Ctrl-C

+++ Note about using Topic

You can repeat the above scenario using the Topic T2 that was defined:

Window 1:

```
$ java JmsJndiProducerLoop -i  file:///var/mqm/JNDI-Directory -c CF2REC -d T2
```

Window 2: Issue the following command to subscribe to the topic string TOPIC2 in QM_ANGELITO.

```
C:\> amqssub TOPIC2 QM_ANGELITO
Sample AMQSSUBA start
Calling MQGET : 30 seconds wait time
MQGET ended with reason code 2080
Sample AMQSSUBA end
```

Notice that there was a message sent to the subscriber, but it was too long for the sample to handle. The MQ utility "mqrc" 2080 displays the error name.

```
C:\> mqrc 2080

     2080  0x00000820  MQRC_TRUNCATED_MSG_FAILED
```

You could edit the amqssuba.c source code to expand the buffer in order to handle longer messages.

Or, you can use the MQ Explorer. See the corresponding Chapter for the details on using the MQ Explorer for Windows: the same technique can be used with Linux.

++
+++ Chapter 3: Scenario NOT using JNDI - UNIX
++

The following samples are used in this chapter:

```
  JmsProducerLoop.java  = > Uses JmsConnectionFactory
  JmsProducerLoopMQCF.java => Uses MQConnectionFactory
```

++ Edit the files and customize the connectionNameList

```
private static String connectionNameList = "host1(1414),host2(1414)";
```

++ Compile the samples:

```
$ javac JmsProducerLoop.java
$ javac JmsProducerLoopMQCF.java
```

++ Running the sample JmsProducerLoop (JmsConnectionFactory) from Linux

Open 3 windows:

Window 1: To run the JMS sample
Window 2: To start and stop the queue manager QM_ANGELITO in Windows
Window 3: To start and stop the queue manager QM_VER in Linux

Window 2: Start the queue manager QM_ANGELITO in Windows

```
C:\> strmqm QM_ANGELITO
```

Window 3: Start the queue manager QM_VER in Linux

```
$ strmqm QM_VER
```

Window 1: Run the sample code specifying the Connection Factory that has a connectionNameList and a queue destination.

```
$ java JmsProducerLoop -d Q2
Initial context found!
Sent message:
  JMSMessage class: jms_text
  JMSType:          null
  JMSDeliveryMode:  2
  JMSExpiration:    0
  JMSPriority:      4
  JMSMessageID:     ID:414d5120514d5f414e47454c49544f20b3ab9d4e20003405
  JMSTimestamp:     1318959303026
  JMSCorrelationID: null
  JMSDestination:   queue:///Q2
  JMSReplyTo:       null
  JMSRedelivered:   false
    JMSXAppID: WebSphere MQ Client for Java
    JMSXDeliveryCount: 0
    JMSXUserID: rivera
    JMS_IBM_PutApplType: 28
    JMS_IBM_PutDate: 20111018
    JMS_IBM_PutTime: 17104412
JmsJndiProducerLoop: Your lucky number today is 25
```

Window 2: Verify that Q2 in QM_ANGELITO is receiving messages:

```
C:\> amqsget Q2 QM_ANGELITO
amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
```

Window 3: Verify that Q2 in QM_VER is NOT receiving messages

```
$ amqsget Q2 QM_VER
Sample AMQSGET0 start
(no messages)
```

Window 2: End the queue manager with flags to terminate immediately and to notify the connected clients to reconnect

```
C:\> endmqm -ir QM_ANGELITO
Waiting for queue manager 'QM_ANGELITO' to end.
WebSphere MQ queue manager 'QM_ANGELITO' ending.
WebSphere MQ queue manager 'QM_ANGELITO' ended.
```

Window 1: Except for a pause for the MQ client code to detect the termination of the first queue manager and to reconnect to the second queue manager, there should not be any messages that indicate that a reconnect is happening, and the sample should continue producing messages.

```
Sent message:

  JMSMessage class: jms_text
…
    JMS_IBM_PutTime: 17362851
JmsJndiProducerLoop: Your lucky number today is 516
```

Window 3: Verify that Q2 in QM_VER in Linux is now receiving messages:

```
$ amqsget Q2 QM_VER
amqsget Q2 QM_VER
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
```

Window 2: Verify that Q2 in QM_ANGELITO in Windows is NOT receiving messages (because the Queue manager is no longer running)

```
C:\> amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
MQCONN ended with reason code 2059
```

Window 2: Restart the queue manager QM_ANGELITO

```
C:\> strmqm QM_ANGELITO
```

Window 2: Notice that the sample is still connected to QM_VER and queue Q2 in QM_ANGELITO is NOT receiving messages:

```
C:\> amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
no more messages
Sample AMQSGET0 end
```

Window 1: Notice that the sample is still in the loop to produce messages.

Window 3: Now terminate QM_VER with the flags for immediate termination and to tell the clients to reconnect.

```
$ endmqm -ir QM_VER
WebSphere MQ queue manager 'QM_VER' ending.
WebSphere MQ queue manager 'QM_VER' ended.
```

Window 1: Notice that after a brief pause, the sample should continue producing messages.

Window 2: Verify that queue Q2 in QM_ANGELITO is now receiving messages:

```
C:\> amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
```

Window 1: To terminate the sample code, enter: Ctrl-C

+++ Note about using Topic

You can repeat the above scenario using the Topic string "sports":

Window 1:

```
$ java JmsProducerLoop -d topic://sports
```

Window 2: Issue the following command to subscribe to the topic string "sports" in QM_ANGELITO.

```
C:\> amqssub "sports" QM_ANGELITO
Sample AMQSSUBA start
Calling MQGET : 30 seconds wait time
MQGET ended with reason code 2080
Sample AMQSSUBA end
```

Notice that there was a message sent to the subscriber, but it was too long for the sample to handle. The MQ utility "mqrc" 2080 displays the error name.

```
C:\> mqrc 2080

     2080  0x00000820  MQRC_TRUNCATED_MSG_FAILED
```

You could edit the amqssuba.c source code to expand the buffer in order to handle longer messages.

Or, you can use the MQ Explorer. See the corresponding Chapter for the details on using the MQ Explorer for Windows: the same technique can be used with Linux.

++ Running the sample JmsProduceLoopMQCF (MQConnectionFactory) from Linux

Open 3 windows:

Window 1: To run the JMS sample
Window 2: To start and stop the queue manager QM_ANGELITO in Windows
Window 3: To start and stop the queue manager QM_VER in Linux

Window 2: Start the queue manager QM_ANGELITO in Windows

```
C:\> strmqm QM_ANGELITO
```

Window 3: Start the queue manager QM_VER in Linux

```
$ strmqm QM_VER
```

Window 1: Run the sample code

```
$ java JmsProducerLoopMQCF -d Q2
Initial context found!
Sent message:

  JMSMessage class: jms_text
…
    JMS_IBM_PutDate: 20111018
    JMS_IBM_PutTime: 19242795
JmsProducerLoopMQCF: Your lucky number today is 411
```

Window 2: Verify that Q2 in QM_ANGELITO is receiving messages:

```
C:\> amqsget Q2 QM_ANGELITO
amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
```

Window 3: Verify that Q2 in QM_VER is NOT receiving messages

```
$ amqsget Q2 QM_VER
Sample AMQSGET0 start
```

```
(no messages)
```

Window 2: End the queue manager with flags to terminate immediately and to notify the connected clients to reconnect

```
C:\> endmqm -ir QM_ANGELITO
Waiting for queue manager 'QM_ANGELITO' to end.
WebSphere MQ queue manager 'QM_ANGELITO' ending.
WebSphere MQ queue manager 'QM_ANGELITO' ended.
```

Window 1: Except for a pause for the MQ client code to detect the termination of the first queue manager and to reconnect to the second queue manager, there should not be any messages that indicate that a reconnect is happening, and the sample should continue producing messages.

Window 3: Verify that Q2 in QM_VER in Linux is now receiving messages:

```
$ amqsget Q2 QM_VER
amqsget Q2 QM_VER
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
```

Window 1: To terminate the sample code, enter: Ctrl-C

+++
+++ Chapter 4: Configuration in Windows via JMSAdmin and MQ Explorer
+++

++ Section: JMSAdmin

Only the specific details for Windows are covered here.
For the common details, see the usage notes in Chapter 1.

a) Create a directory where the JMS configuration objects will be located in a file name ".bindings".
For example, in Windows, you can create the following subdirectory in the same directory where the other MQ objects are stored:

```
   mkdir  c:\var\mqm\JNDI-Directory
```

b) Copy the JMSAdmin.config file from its default location into c:\var\mqm

The default file in Windows is located at:

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

In this document we follow the best practice of not writing files under the directory structure that has the MQ runtime code. Thus, you need to copy the JMSAdmin.config file:

```
  Copy "C:\Program Files\IBM\WebSphere MQ\java\bin\JMSAdmin.config"
C:\var\mqm\JMSAdmin.config
```

c) Modify C:\var\mqm\JMSAdmin.config

These 2 variables need to be properly specified in the config file.

c.1) INITIAL_CONTEXT_FACTORY

The following is common for UNIX and Windows, and it indicates that a file will be used:

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

c.2) PROVIDER_URL

For Windows: notice only 1 forward slash after file: and notice the drive letter C:/

```
PROVIDER_URL=file:/C:/var/mqm/JNDI-Directory
```

For UNIX (notice the 3 forward slashes after file:):

```
PROVIDER_URL=file:///var/mqm/JNDI-Directory
```

Note: If there are not 3 slashes, such as 2, 4, 5, etc. then the following error is displayed by JMSAdmin at runtime:

```
InitCtx> DEFINE TCF(testCF)
Unable to bind object
```

Notice that ONLY one INITIAL_CONTEXT_FACTORY and PROVIDER_URL must be uncommented. If there are more uncommented lines, then the last entry is the winner, and overwrites any earlier entries of the same type.

d) Create a batch file called "myJMSAdmin.bat" that has this one line:

+ begin

```
"%MQ_JAVA_INSTALL_PATH%\bin\JMSAdmin" -cfg C:\var\mqm\JMSAdmin.config
```

+ end

This batch file will invoke the JMSAdmin tool using the JMSAdmin.config that was copied and modified under C:\var\mqm.

Ensure that the myJMSAdmin.bat is in a directory under PATH.

e) Start the myJMSAdmin batch file:

```
C:\> myJMSAdmin
```

You will see the following. Notice the prompt: InitCtx>

```
Licensed Materials - Property of IBM
5724-H72, 5655-R36, 5724-L26, 5655-L82
(c) Copyright IBM Corp. 2008 All Rights Reserved.
US Government Users Restricted Rights - Use, duplication or
disclosure restricted by GSA ADP Schedule Contract with
IBM Corp.
Starting Websphere MQ classes for Java(tm) Message Service Administration

InitCtx>
```

f) To create the JMS administrative objects, see the corresponding steps in Chapter 1.

g) Creation of physical objects (Queue and Topic) in the queue manager

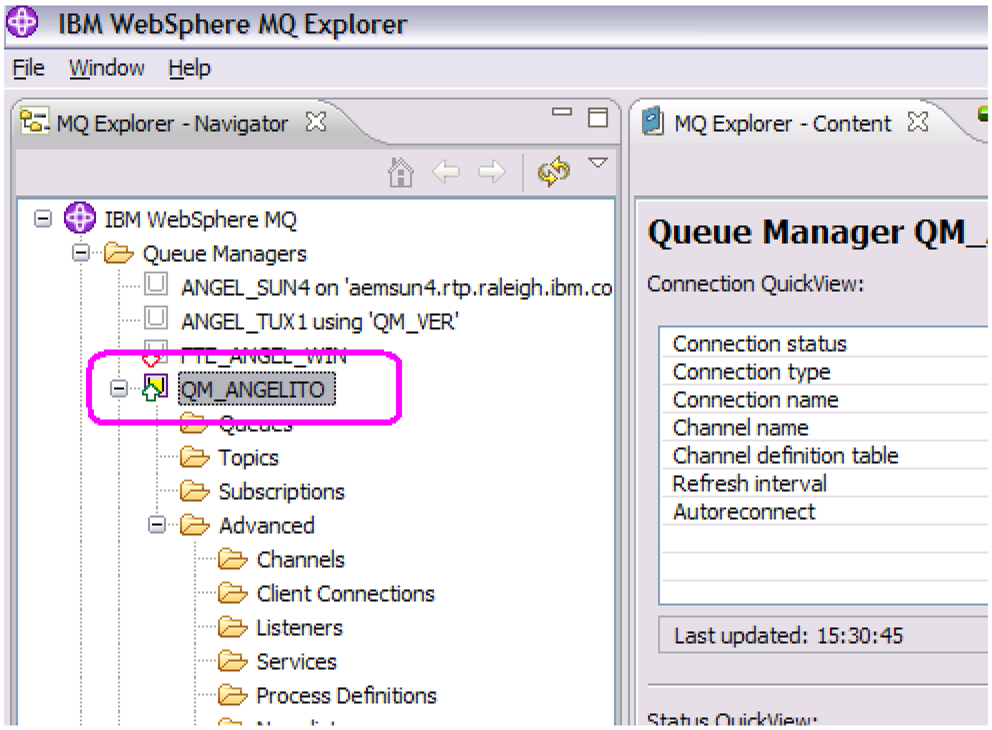
The following were used in Windows:

Queue Manager name: QM_ANGELITO
Host: angelito.x.com
Port: 1414

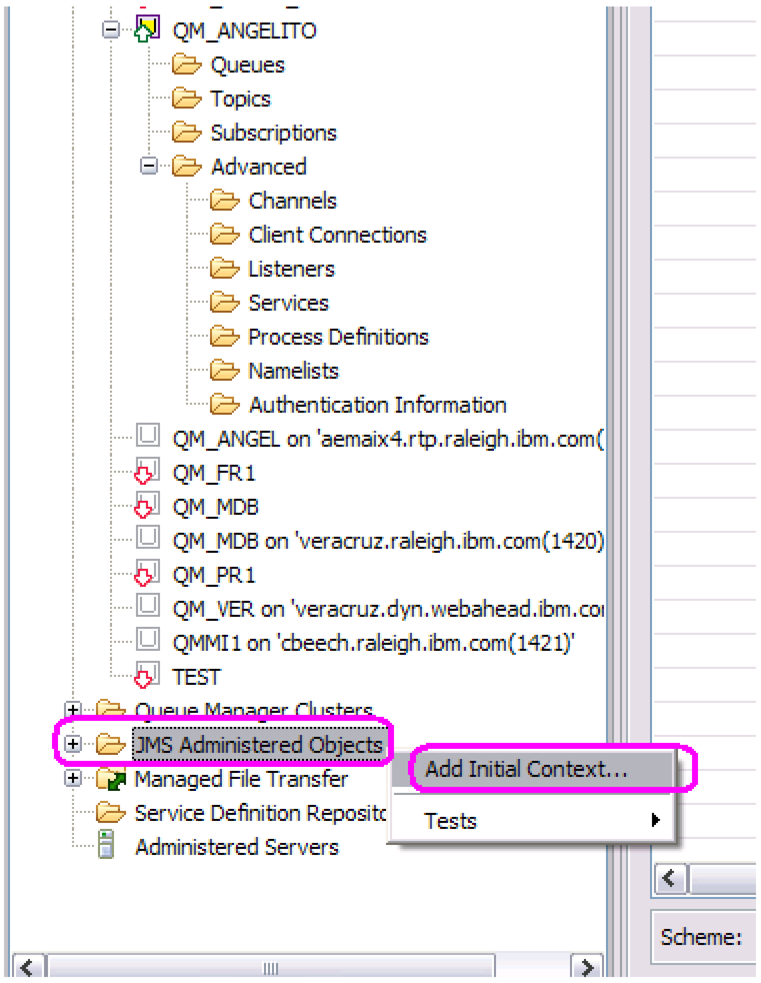
```
$ runmqsc QM_ANGELITO
define ql(Q2)
define topic(T2) topicstr('TOPIC2')
end
```

++ Section: MQ Explorer

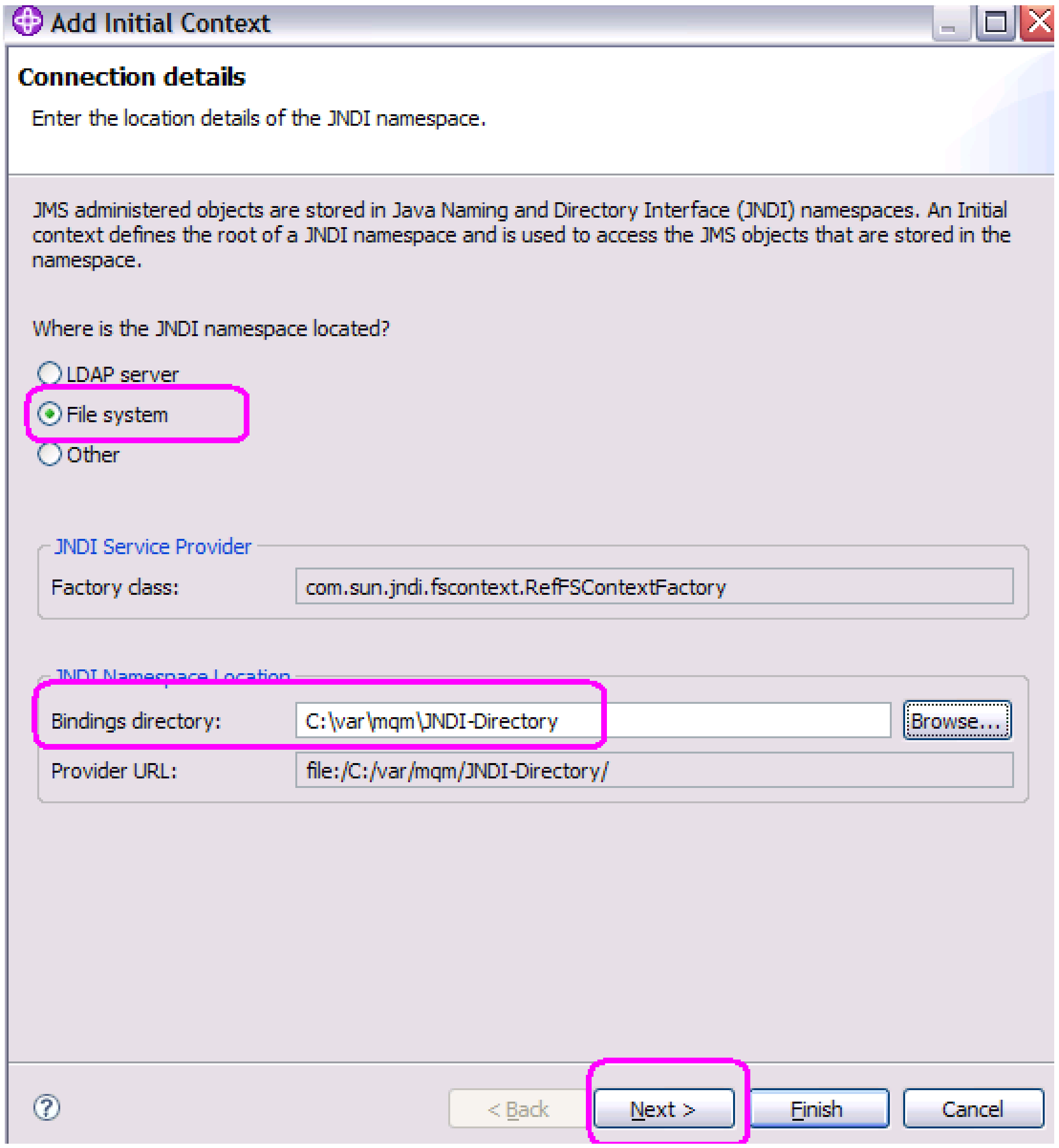
Start the MQ Explorer and the desired queue manager.
In this example, the queue manager is called: QM_ANGELITO



Scroll down to the bottom and select: JMS Administered Objects
Do right Click and select "Add Initial Context ..."



In the dialog window for "Add Initial Context", specify:
  Where is the JNDI namespace located?
    (x) File System
  JNDI Namespace location
    Bindings directory: C:\var\mqm\JNDI-Directory



Add Initial Context

Connection details

Enter the location details of the JNDI namespace.

JMS administered objects are stored in Java Naming and Directory Interface (JNDI) namespaces. An Initial context defines the root of a JNDI namespace and is used to access the JMS objects that are stored in the namespace.

Where is the JNDI namespace located?

LDAP server [✓ File system] Other

JNDI Service Provider

Factory class: com.sun.jndi.fscontext.RefFSContextFactory

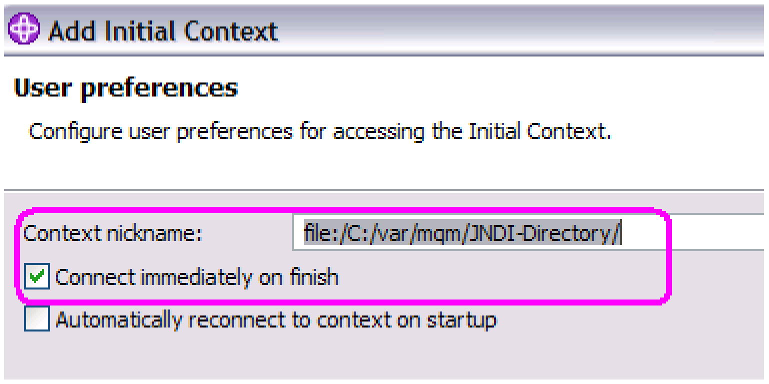
JNDI Namespace Location

Bindings directory: C:\var\mqm\JNDI-Directory

Provider URL: file:/C:/var/mqm/JNDI-Directory/

Click Next

Accept the defaults and click Finish.


Add Initial Context

User preferences

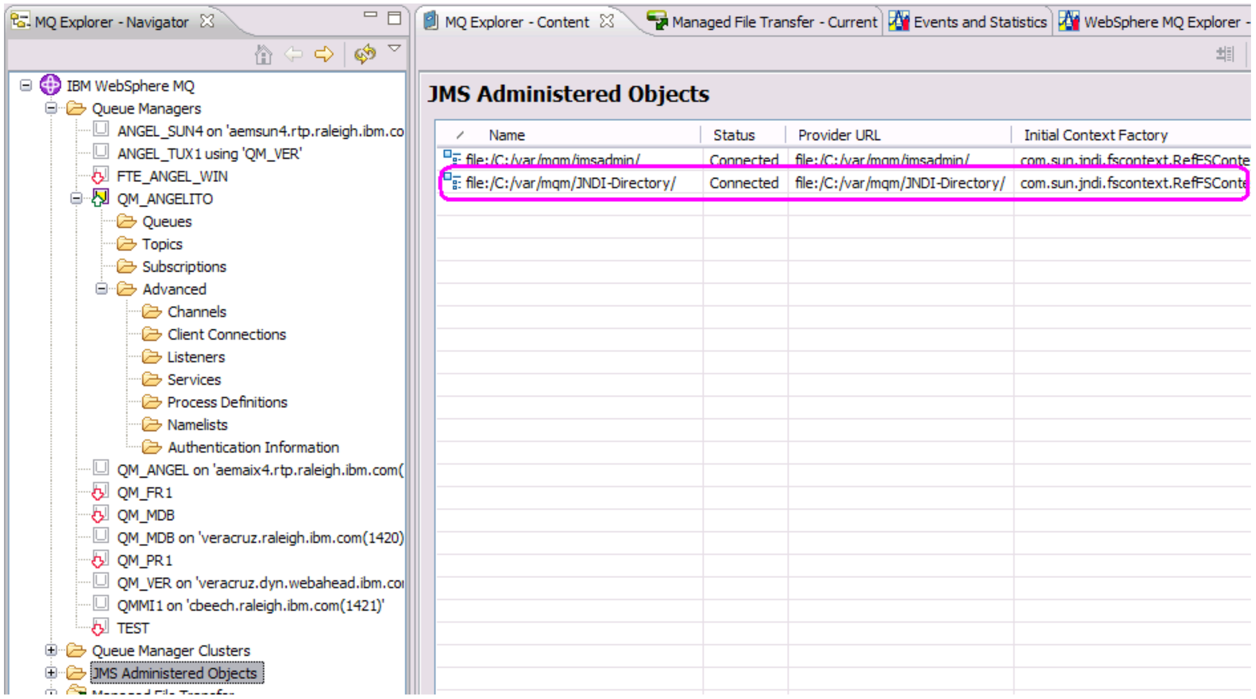
Configure user preferences for accessing the Initial Context.

Context nickname: file:/C:/var/mqm/JNDI-Directory/

[✓] Connect immediately on finish

[ ] Automatically reconnect to context on startup

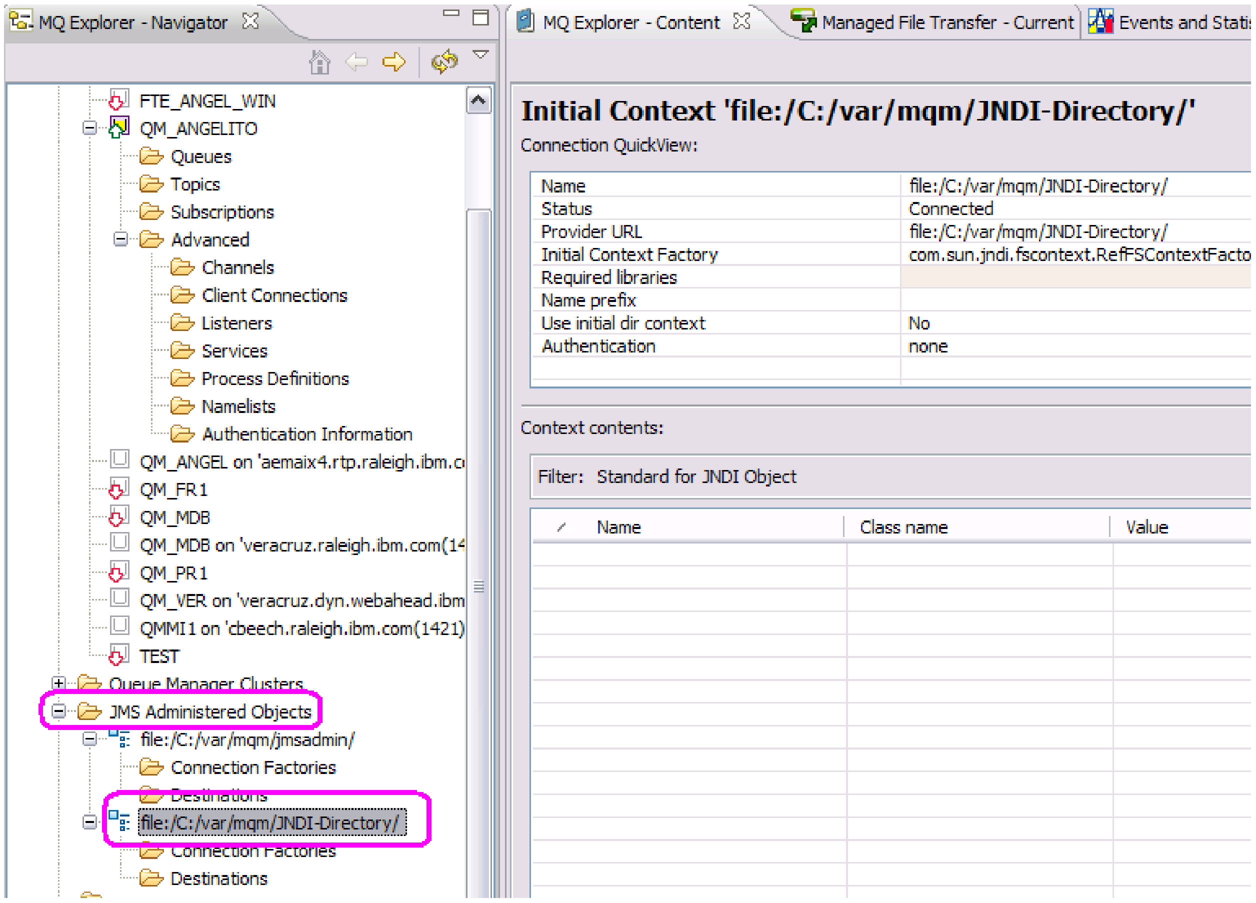
Notice that this initial context location will be shown in the right panel:



JMS Administered Objects

Name	Status	Provider URL	Initial Context Factory
file:/C:/var/mqm/jmsadmin/	Connected	file:/C:/var/mqm/jmsadmin/	com.sun.jndi.fscontext.RefFSConte
file:/C:/var/mqm/JNDI-Directory/	Connected	file:/C:/var/mqm/JNDI-Directory/	com.sun.jndi.fscontext.RefFSConte

In the left panel, expand the entry for the newly added initial context:



Initial Context 'file:/C:/var/mqm/JNDI-Directory/'

Connection QuickView:

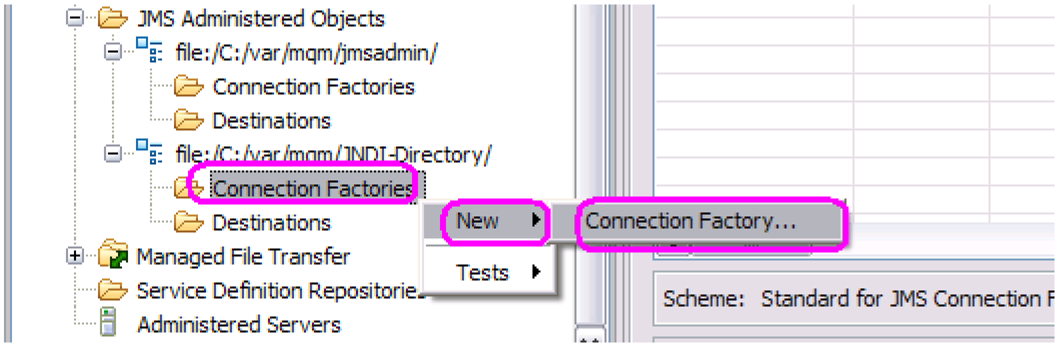
Name	file:/C:/var/mqm/JNDI-Directory/
Status	Connected
Provider URL	file:/C:/var/mqm/JNDI-Directory/
Initial Context Factory	com.sun.jndi.fscontext.RefFSContextFacto
Required libraries	
Name prefix	
Use initial dir context	No
Authentication	none

Context contents:

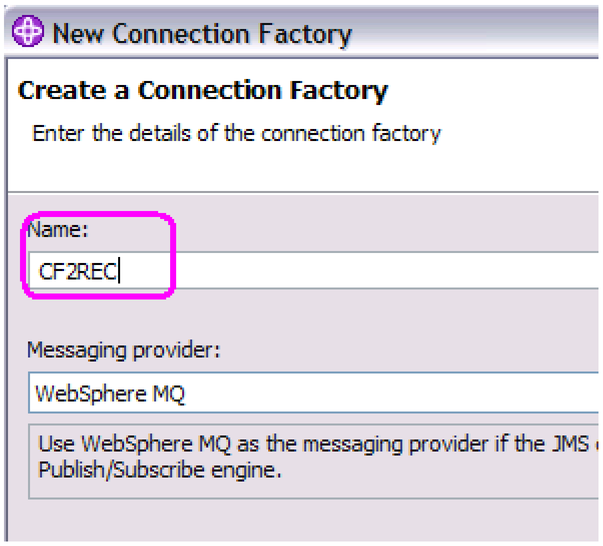
Filter: Standard for JNDI Object

Name	Class name	Value

Let's create a Connection Factory for the new initial context.
Click on Connection Factories -> New -> Connection Factory ...



Specify the name, such as: CF2REC


New Connection Factory

Create a Connection Factory

Enter the details of the connection factory

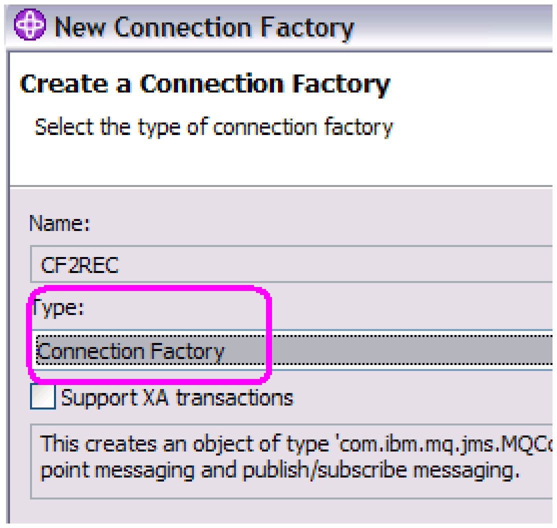
Name:
CF2REC

Messaging provider:
WebSphere MQ

Use WebSphere MQ as the messaging provider if the JMS
Publish/Subscribe engine.

Click Next.

Accept the default Type of "Connection Factory"
In case that you want "Queue Connection Factory", this is the panel where you can specify it.


New Connection Factory

Create a Connection Factory

Select the type of connection factory

Name:
CF2REC

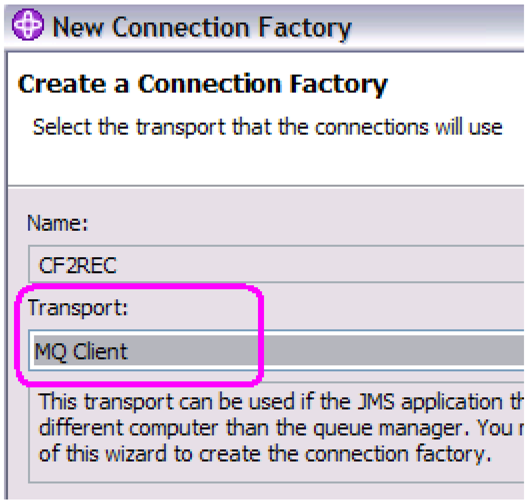
Type:
Connection Factory

Support XA transactions

This creates an object of type 'com.ibm.mq.jms.MQC
point messaging and publish/subscribe messaging.

Click Next

The default value for the Transport is Bindings.
Because this document is trying to illustrate Client mode, then change to:
  MQ Client


New Connection Factory

Create a Connection Factory

Select the transport that the connections will use

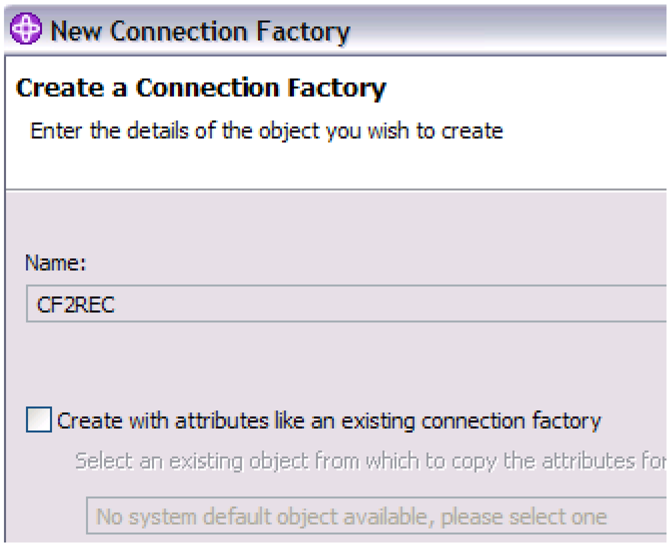
Name:
CF2REC

Transport:
MQ Client

This transport can be used if the JMS application t
different computer than the queue manager. You r
of this wizard to create the connection factory.

Click Next

If you have an existing connection factory that you want to use as a model for the rest of the fields, the following panel is the place to do it.
In this simple example, there is no such existing CF. Thus, just Click Next.


New Connection Factory

Create a Connection Factory

Enter the details of the object you wish to create

Name: CF2REC

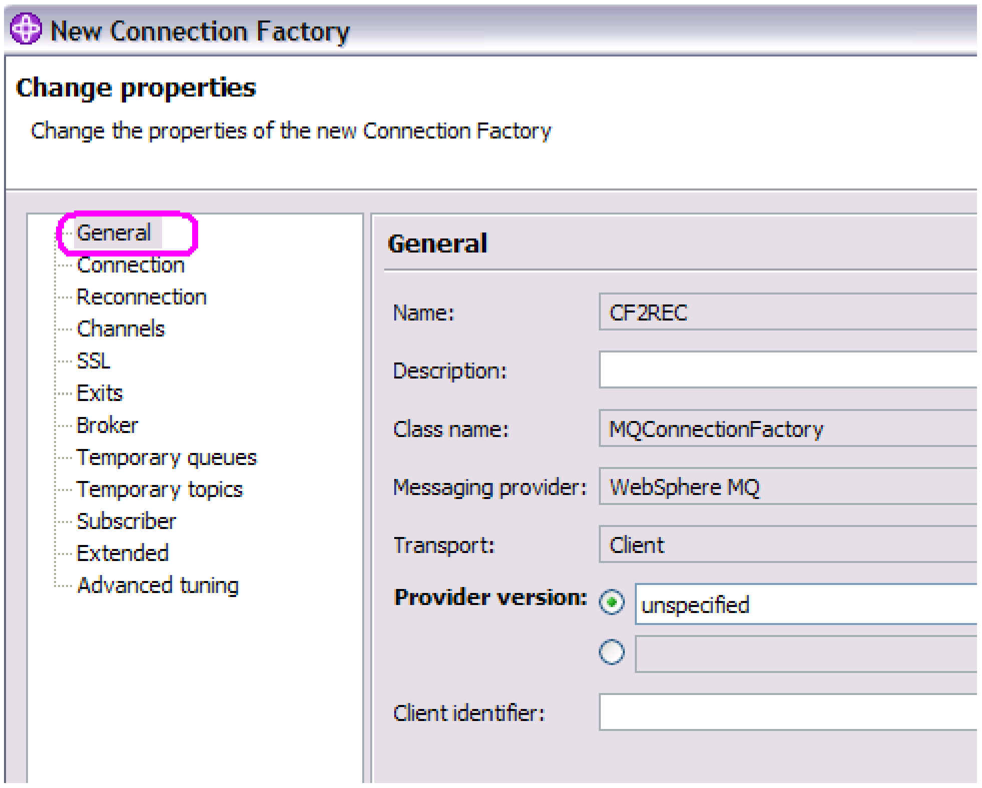
☐ Create with attributes like an existing connection factory

Select an existing object from which to copy the attributes fo

No system default object available, please select one

Click Next

You will see the following dialog with several tabs.
For this example, there are no changes in the General tab:


New Connection Factory

Change properties

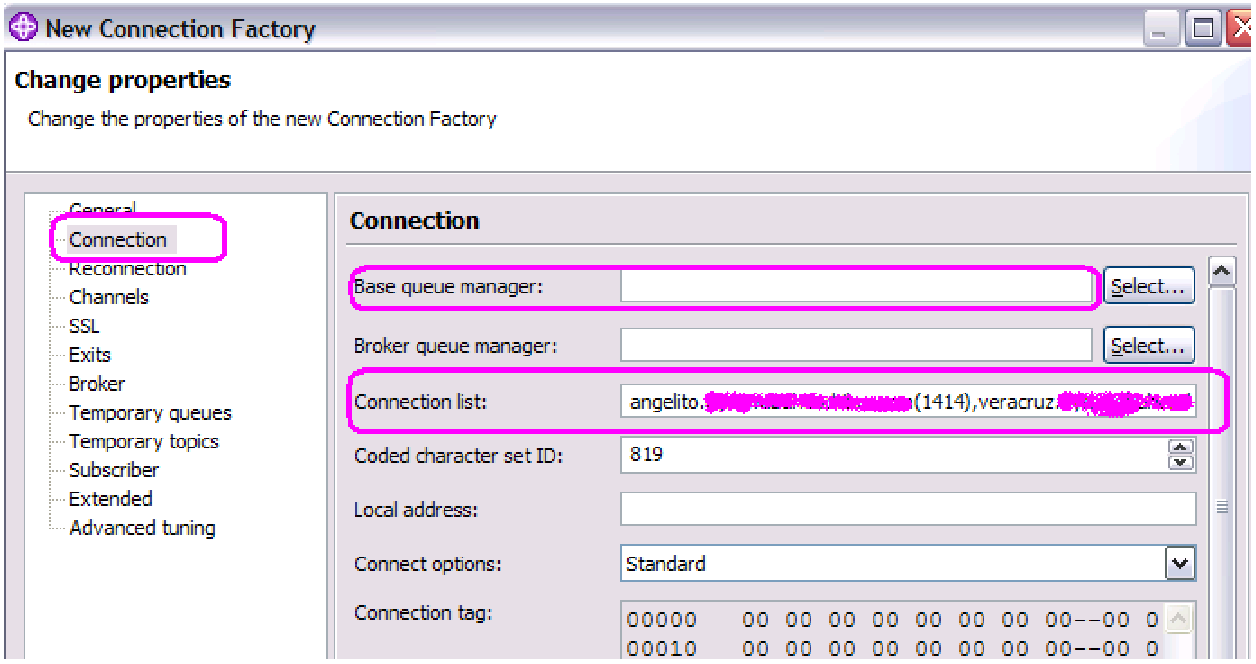
Change the properties of the new Connection Factory

- General
- Connection
- Reconnection
- Channels
- SSL
- Exits
- Broker
- Temporary queues
- Temporary topics
- Subscriber
- Extended
- Advanced tuning

General

Name:	CF2REC
Description:	
Class name:	MQConnectionFactory
Messaging provider:	WebSphere MQ
Transport:	Client
Provider version:	● unspecified
	○
Client identifier:	

In the Connection tab, enter:
  Base queue manager: (Leave it blank, to provide maximum flexibility)
  Connection list: angelito.x.com(1414),veracruz.x(1414)



Change properties

Change the properties of the new Connection Factory

Connection

Base queue manager:

Broker queue manager:

Connection list: angelito.[redacted](1414),veracruz.[redacted]

Coded character set ID: 819

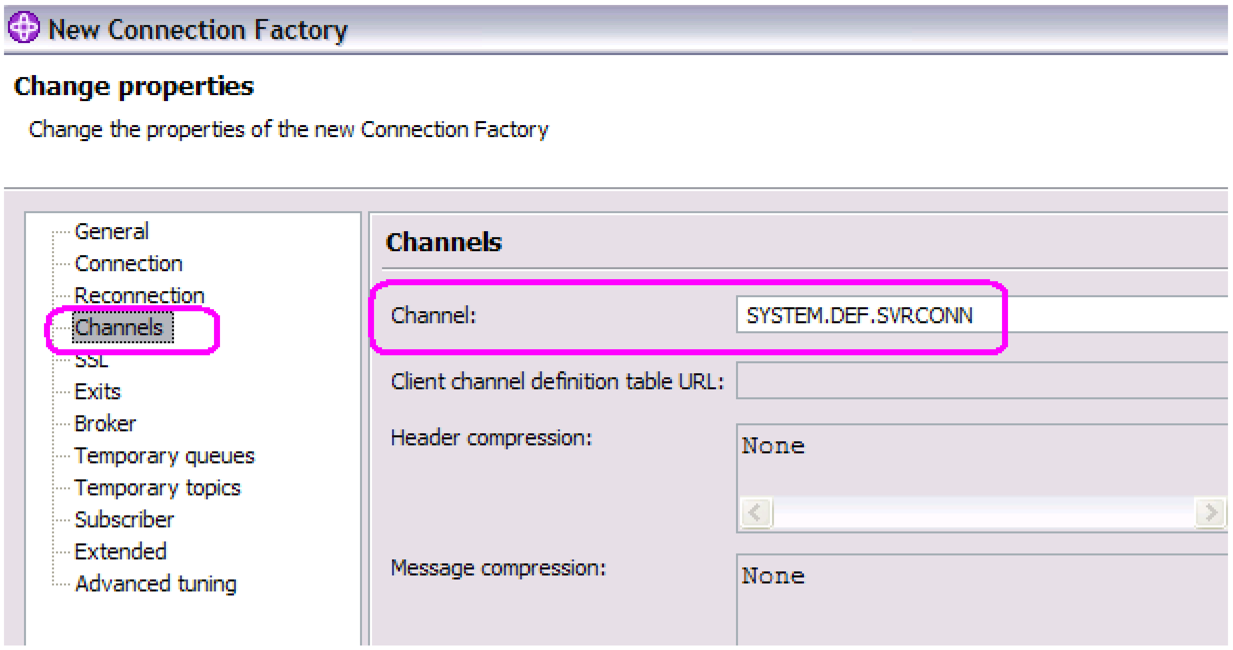
Local address:

Connect options: Standard

Connection tag:

```
00000    00 00 00 00 00 00 00 00--00 0
00010    00 00 00 00 00 00 00 00--00 0
```

On the Channels tab, notice that the channel to be used is by default:
    SYSTEM.DEF.SVRCONN



Change properties

Change the properties of the new Connection Factory

Channels

Channel: SYSTEM.DEF.SVRCONN

Client channel definition table URL:

Header compression: None

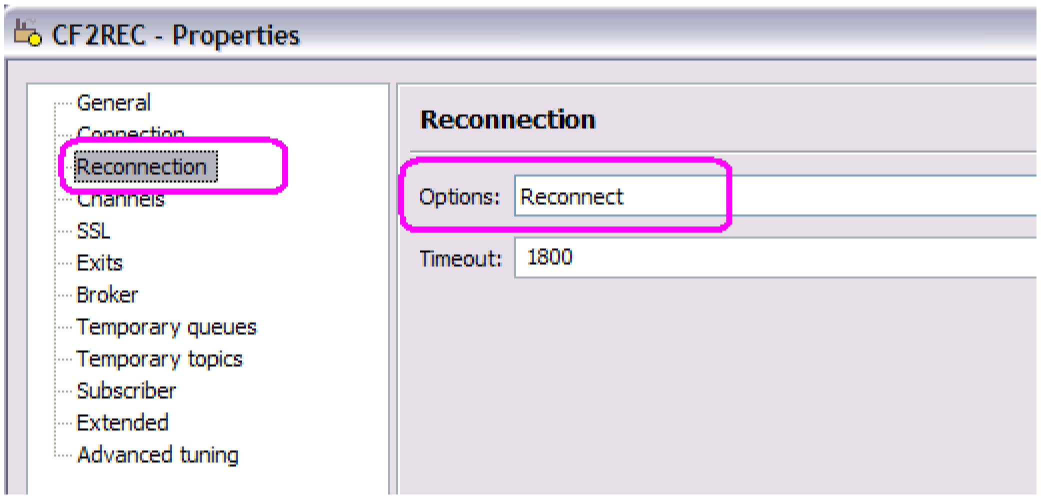
Message compression: None

Click Next

In the Reconnection tab you must change the Options!
  Options: Reconnect
  Timeout: 1800 seconds (30 minutes)

Note: If you leave the Options to the default value of "Default", you will get runtime errors during the reconnection phase:

```
com.ibm.msg.client.jms.DetailedJMSException: JMSWMQ2007: Failed to send a message to destination 'Q2'. JMS attempted to perform an MQPUT or MQPUT1; however WebSphere MQ reported an error. Use the linked exception to determine the cause of this error.
Inner exception(s):
com.ibm.mq.MQException: JMSCMQ0001: WebSphere MQ call failed with compcode '2' ('MQCC_FAILED') reason '2009' ('MQRC_CONNECTION_BROKEN').
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2009
com.ibm.mq.jmqi.JmqiException: CC=2;RC=2009
```



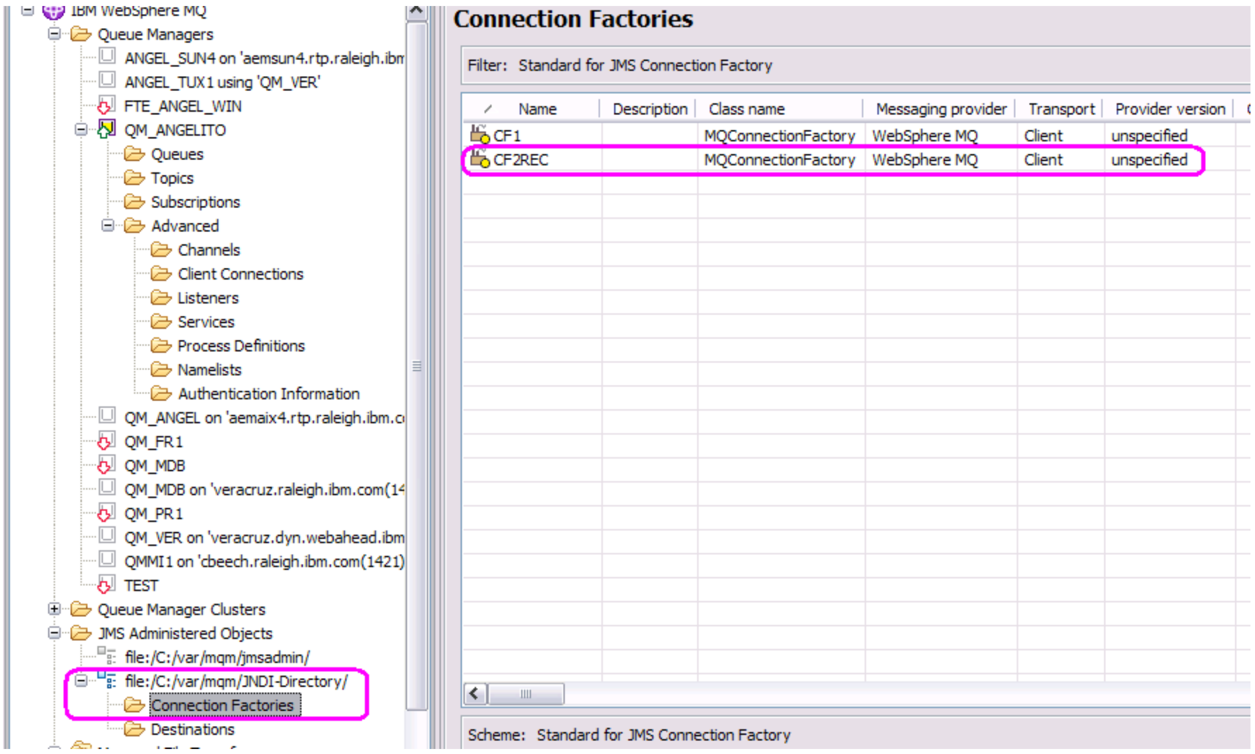
Reconnection

Options: Reconnect

Timeout: 1800

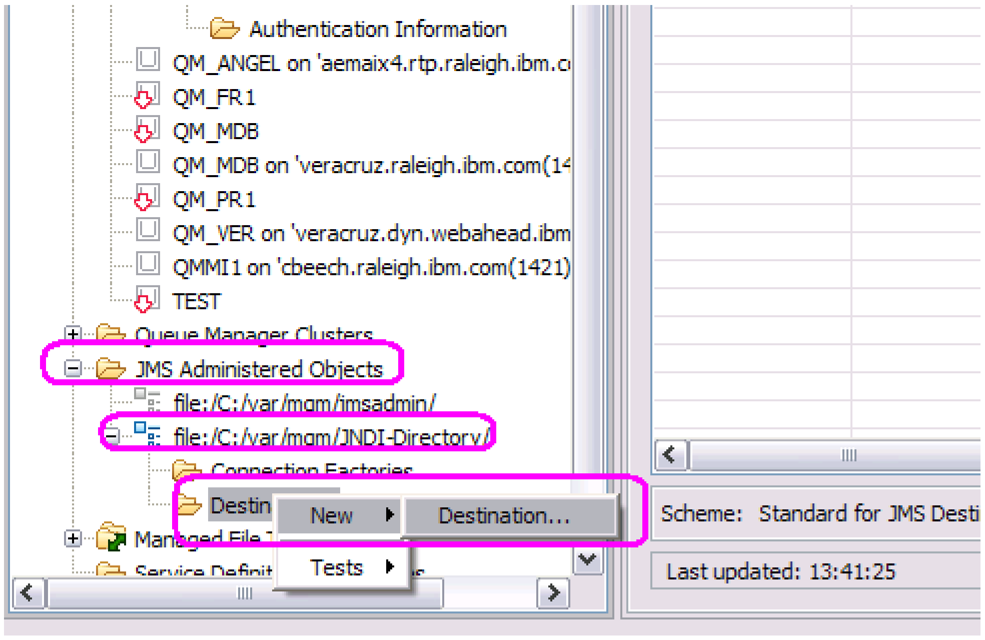
Click Finish

Notice that this new Connection Factory will be listed in the right panel.

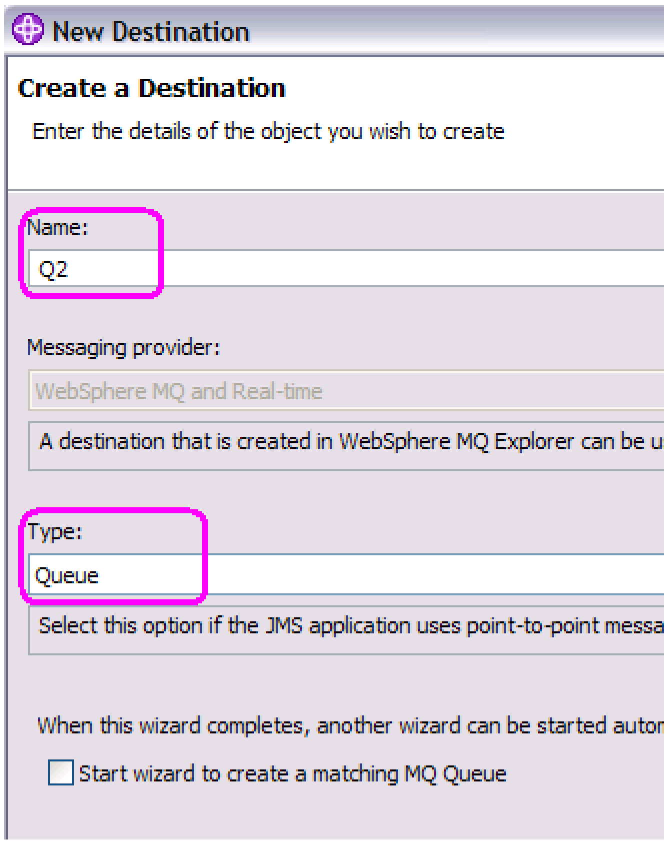

IBM WebSphere MQ
Queue Managers
ANGEL_SUN4 on 'aemsun4.rtp.raleigh.ibm
ANGEL_TUX1 using 'QM_VER'
FTE_ANGEL_WIN
QM_ANGELITO
Queues
Topics
Subscriptions
Advanced
Channels
Client Connections
Listeners
Services
Process Definitions
Namelists
Authentication Information
QM_ANGEL on 'aemaix4.rtp.raleigh.ibm.c
QM_FR1
QM_MDB
QM_MDB on 'veracruz.raleigh.ibm.com(14
QM_PR1
QM_VER on 'veracruz.dyn.webahead.ibm
QMMI1 on 'cbeech.raleigh.ibm.com(1421)
TEST
Queue Manager Clusters
JMS Administered Objects
file:/C:/var/mqm/jmsadmin/
file:/C:/var/mqm/JNDI-Directory/
Connection Factories
Destinations
Connection Factories
Filter: Standard for JMS Connection Factory
Scheme: Standard for JMS Connection Factory

Name	Description	Class name	Messaging provider	Transport	Provider version
CF1		MQConnectionFactory	WebSphere MQ	Client	unspecified
CF2REC		MQConnectionFactory	WebSphere MQ	Client	unspecified

Let's create the Destination Queue Q2:


Authentication Information
QM_ANGEL on 'aemaix4.rtp.raleigh.ibm.c
QM_FR1
QM_MDB
QM_MDB on 'veracruz.raleigh.ibm.com(14
QM_PR1
QM_VER on 'veracruz.dyn.webahead.ibm
QMMI1 on 'cbeech.raleigh.ibm.com(1421)
TEST
Queue Manager Clusters
JMS Administered Objects
file:/C:/var/mqm/jmsadmin/
file:/C:/var/mqm/JNDI-Directory/
Connection Factories
Destin
New
Destination...
Tests
Managed File
Service Definit
Scheme: Standard for JMS Desti
Last updated: 13:41:25

In the Create Destination, specify:
  Name: Q2
  Type: Queue


New Destination

Create a Destination

Enter the details of the object you wish to create

Name:
Q2

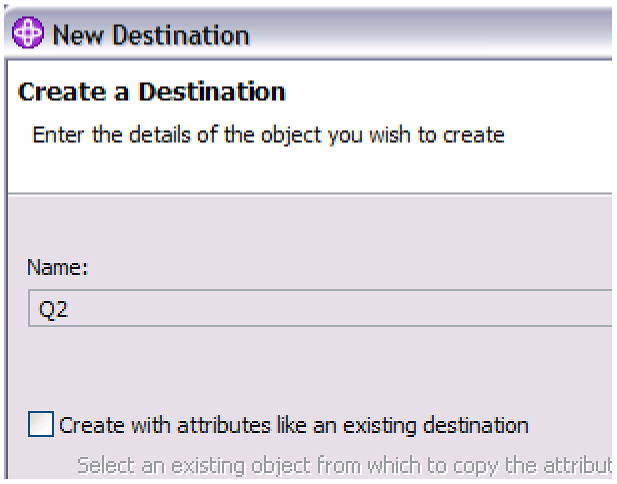
Messaging provider:
WebSphere MQ and Real-time
A destination that is created in WebSphere MQ Explorer can be u

Type:
Queue
Select this option if the JMS application uses point-to-point messa

When this wizard completes, another wizard can be started autor

☐ Start wizard to create a matching MQ Queue

Click Next.
You will see:


New Destination

Create a Destination

Enter the details of the object you wish to create

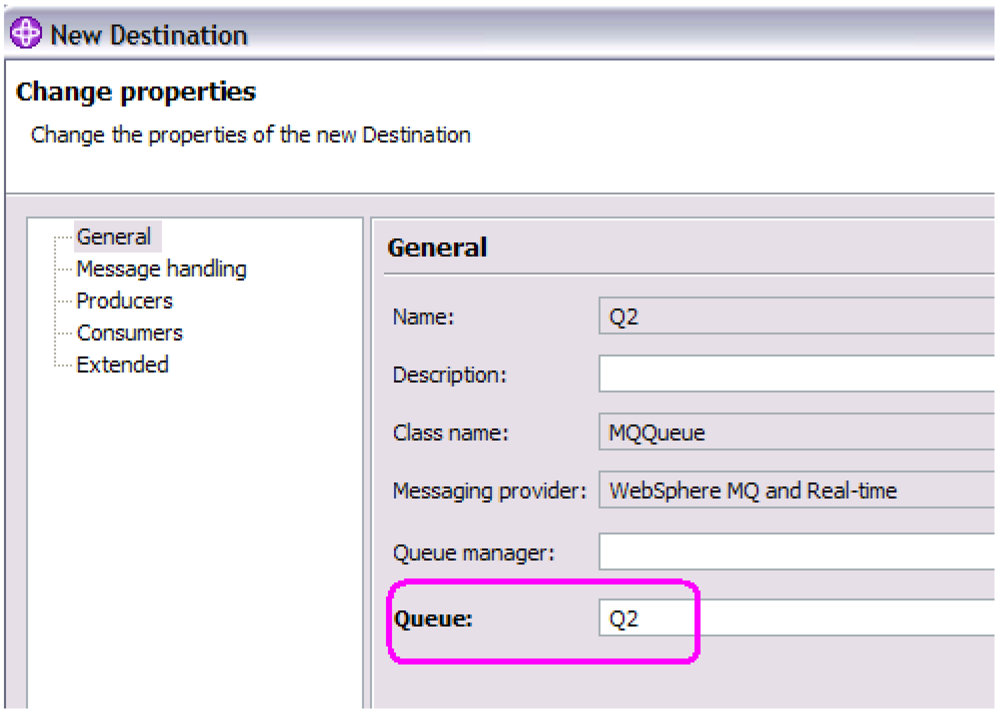
Name:
Q2

☐ Create with attributes like an existing destination

Select an existing object from which to copy the attribut

Click Next.

Then enter the name of the physical queue in the queue manager, in this case it is the same name: Q2


New Destination

Change properties

Change the properties of the new Destination

General

Name: Q2

Description:

Class name: MQQueue

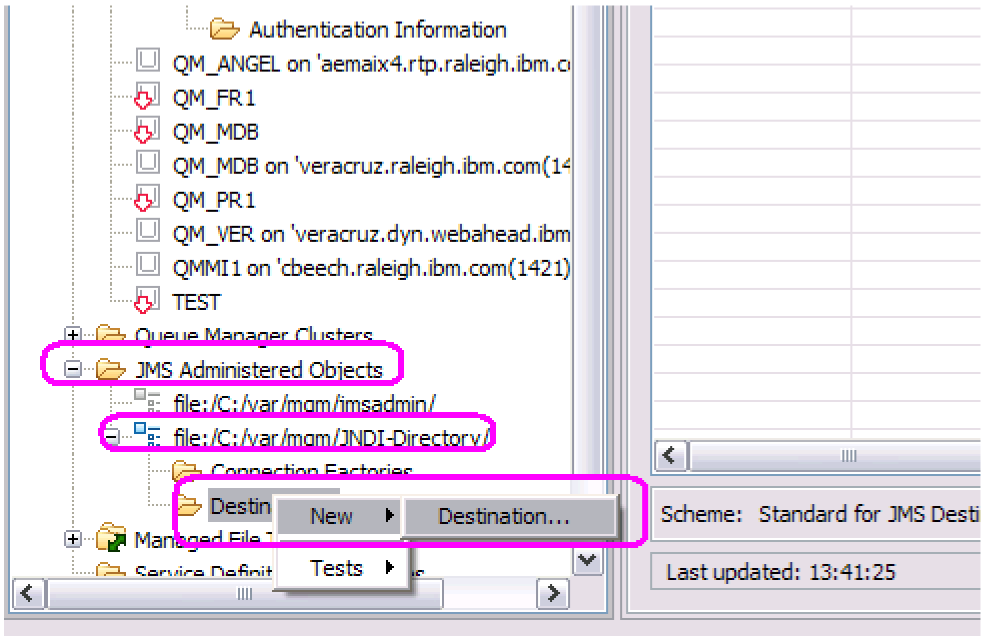
Messaging provider: WebSphere MQ and Real-time

Queue manager:

Queue: Q2

Click Finish.

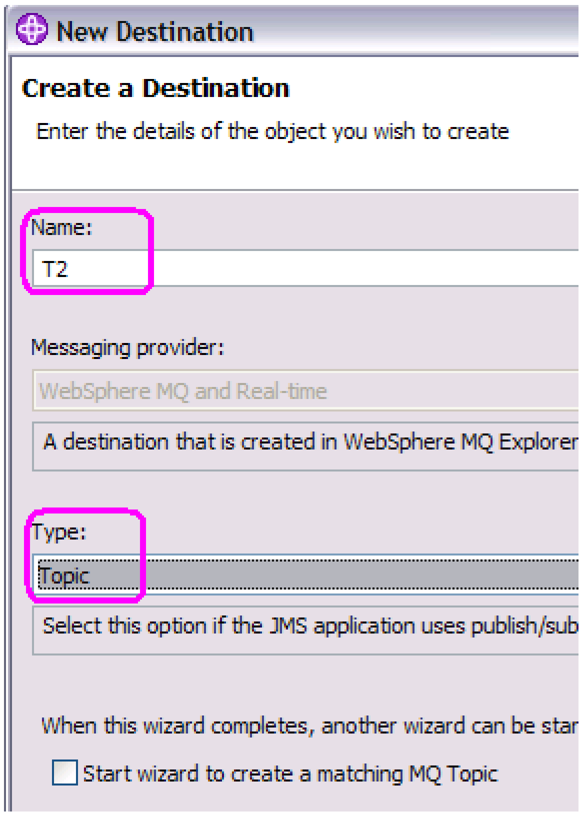
Let's create the Topic T2


Authentication Information
QM_ANGEL on 'aemaix4.rtp.raleigh.ibm.c
QM_FR1
QM_MDB
QM_MDB on 'veracruz.raleigh.ibm.com(14
QM_PR1
QM_VER on 'veracruz.dyn.webahead.ibm
QMMI1 on 'cbeech.raleigh.ibm.com(1421)
TEST
Queue Manager Clusters
JMS Administered Objects
file:/C:/var/mqm/jmsadmin/
file:/C:/var/mqm/JNDI-Directory/
Connection Factories
Destin
New
Destination...
Tests
Managed File
Service Definit

Scheme: Standard for JMS Desti

Last updated: 13:41:25

In the Create Destination, specify:
Name: T2
Type: Topic


New Destination

Create a Destination

Enter the details of the object you wish to create

Name:
T2

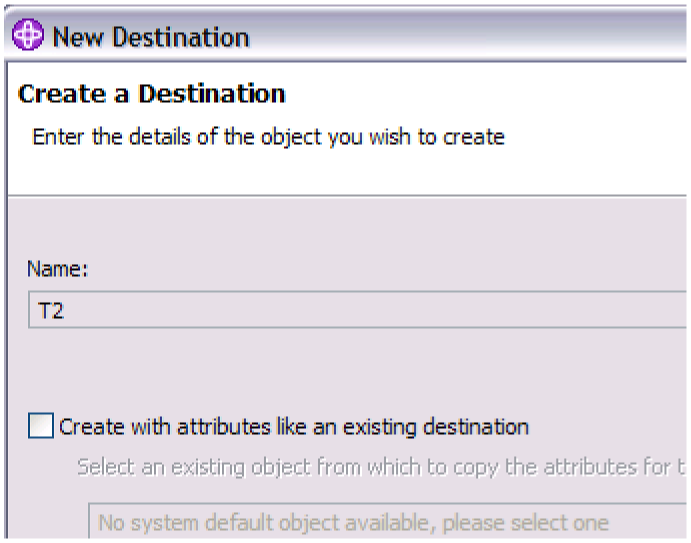
Messaging provider:
WebSphere MQ and Real-time
A destination that is created in WebSphere MQ Explorer

Type:
Topic
Select this option if the JMS application uses publish/sub

When this wizard completes, another wizard can be star

☐ Start wizard to create a matching MQ Topic

Click Next


New Destination

Create a Destination

Enter the details of the object you wish to create

Name:
T2

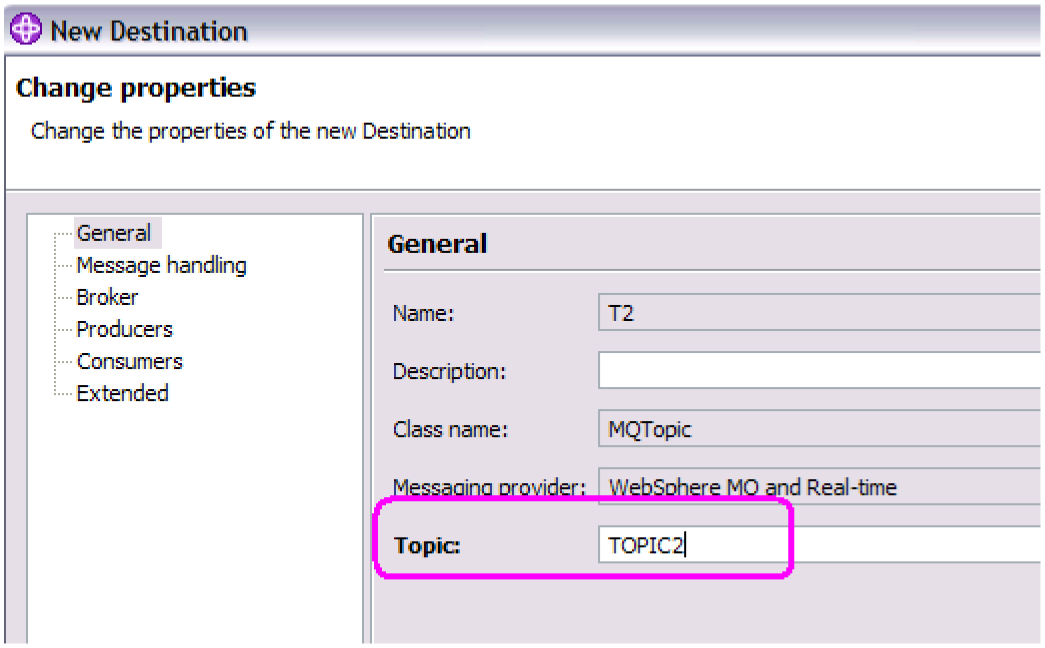
☐ Create with attributes like an existing destination

Select an existing object from which to copy the attributes for t

No system default object available, please select one

Click Next

Enter the Topic String: TOPIC2


New Destination

Change properties

Change the properties of the new Destination

General

Name:	T2
Description:	
Class name:	MQTopic
Messaging provider:	WebSphere MQ and Real-time
Topic:	TOPIC2

Click Finish

Now you can proceed to the next chapter.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++ Chapter 5: Scenario using JNDI - Windows
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Download from the techdoc, the sample JMS code:
  JmsJndiProducerLoop.java

An excerpt of the comments in the header is shown below:

-- begin excerpt

```
* This sample is based on the JmsJndiProducer.java sample provided with MQ V7:
 *   Windows: C:\Program Files\IBM\WebSphere MQ\tools\jms\samples\JmsJndiProducer.java
 *   Unix:    /opt/mqm/samp/jms/samples/JmsJndiProducer.java
 *
 * A JMS producer (sender or publisher) application that sends a simple message to the named
 * destination (queue or topic) by looking up the connection factory instance and the destination
 * instance in an initial context (This sample supports file system context only).
 *
 * JmsJndiProducerLoop is an extension in which inside a forever loop, a message is sent
 * to the destination, with an interval of 5 seconds between each message.
 * At runtime, the user needs to use Ctrl-C to terminate the endless loop.
```

-- end excerpt

++ Compile the sample:

```
C:\mqv7-jndi> javac JmsJndiProducerLoop.java
Note: JmsJndiProducerLoop.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

++ Running the sample from Windows

Open 3 windows:

Window 1: To run the JMS sample
Window 2: To start and stop the queue manager QM_ANGELITO in Windows
Window 3: To start and stop the queue manager QM_VER in Linux

Window 2: Start the queue manager QM_ANGELITO in Windows

```
C:\> strmqm QM_ANGELITO
```

Window 3: Start the queue manager QM_VER in Linux

```
$ strmqm QM_VER
```

Window 1: Run the sample code specifying the Connection Factory that has a connectionNameList and a queue destination.

```
C:\> java JmsJndiProducerLoop -i file:/C:/var/mqm/JNDI-Directory -c CF2REC -d Q2
Initial context found!
Sent message:

  JMSMessage class: jms_text
  JMSType:          null
  JMSDeliveryMode:  2
  JMSExpiration:    0
  JMSPriority:      4
  JMSMessageID:     ID:414d5120514d5f414e47454c49544f201b7b9c4e20001e11
  JMSTimestamp:     1318878069843
  JMSCorrelationID: null
  JMSDestination:   queue:///Q2
  JMSReplyTo:       null
  JMSRedelivered:   false
    JMSXAppID: WebSphere MQ Client for Java
    JMSXDeliveryCount: 0
    JMSXUserID: rivera
    JMS_IBM_PutApplType: 28
    JMS_IBM_PutDate: 20111017
    JMS_IBM_PutTime: 19010984
JmsJndiProducerLoop: Your lucky number today is 843
```

Window 2: Verify that Q2 in QM_ANGELITO is receiving messages:

```
C:\> amqsget Q2 QM_ANGELITO
amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
```

Window 3: Verify that Q2 in QM_VER is NOT receiving messages

```
$ amqsget Q2 QM_VER
Sample AMQSGET0 start
(no messages)
```

Window 2: End the queue manager with flags to terminate immediately and to notify the connected clients to reconnect

```
C:\> endmqm -ir QM_ANGELITO
Waiting for queue manager 'QM_ANGELITO' to end.
WebSphere MQ queue manager 'QM_ANGELITO' ending.
WebSphere MQ queue manager 'QM_ANGELITO' ended.
```

Window 1: Except for a pause for the MQ client code to detect the termination of the first queue manager and to reconnect to the second queue manager, there should not be any messages that indicate that a reconnect is happening, and the sample should continue producing messages.

```
Sent message:

  JMSMessage class: jms_text
  JMSType:          null
  JMSDeliveryMode:  2
  JMSExpiration:    0
  JMSPriority:      4
  JMSMessageID:     ID:414d5120514d5f414e47454c49544f20ba7d9c4e20001d05
  JMSTimestamp:     1318878672796
  JMSCorrelationID: null
  JMSDestination:   queue:///Q2
  JMSReplyTo:       null
  JMSRedelivered:   false
    JMSXAppID: WebSphere MQ Client for Java
    JMSXDeliveryCount: 0
    JMSXUserID: rivera
    JMS_IBM_PutApplType: 28
    JMS_IBM_PutDate: 20111017
    JMS_IBM_PutTime: 19111279
JmsJndiProducerLoop: Your lucky number today is 796
```

Window 3: Verify that Q2 in QM_VER in Linux is now receiving messages:

```
$ amqsget Q2 QM_VER
amqsget Q2 QM_VER
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
```

Window 2: Verify that Q2 in QM_ANGELITO in Windows is NOT receiving messages (because the Queue manager is no longer running)

```
C:\> amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
MQCONN ended with reason code 2059
```

Window 2: Restart the queue manager QM_ANGELITO

```
C:\> strmqm QM_ANGELITO
```

Window 2: Notice that the sample is still connected to QM_VER and queue Q2 in QM_ANGELITO is NOT receiving messages:

```
C:\> amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
no more messages
Sample AMQSGET0 end
```

Window 1: Notice that the sample is still in the loop to produce messages.

Window 3: Now terminate QM_VER with the flags for immediate termination and to tell the clients to reconnect.

```
$ endmqm -ir QM_VER
WebSphere MQ queue manager 'QM_VER' ending.
WebSphere MQ queue manager 'QM_VER' ended.
```

Window 1: Notice that after a brief pause, the sample should continue producing messages.

Window 2: Verify that queue Q2 in QM_ANGELITO is now receiving messages:

```
C:\> amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
```

Window 1: To terminate the sample code, enter: Ctrl-C

+++ Note about using Topic

You can repeat the above scenario using the Topic T2 that was defined:

Window 1:

```
C:\> java JmsJndiProducerLoop -i file:/C:/var/mqm/JNDI-Directory -c CF2REC -d T2
```

Window 2: Issue the following command to subscribe to the topic string TOPIC2 in QM_ANGELITO.

```
C:\> amqssub TOPIC2 QM_ANGELITO
Sample AMQSSUBA start
Calling MQGET : 30 seconds wait time
MQGET ended with reason code 2080
Sample AMQSSUBA end
```

Notice that there was a message sent to the subscriber, but it was too long for the sample to handle. The MQ utility "mqrc" 2080 displays the error name.

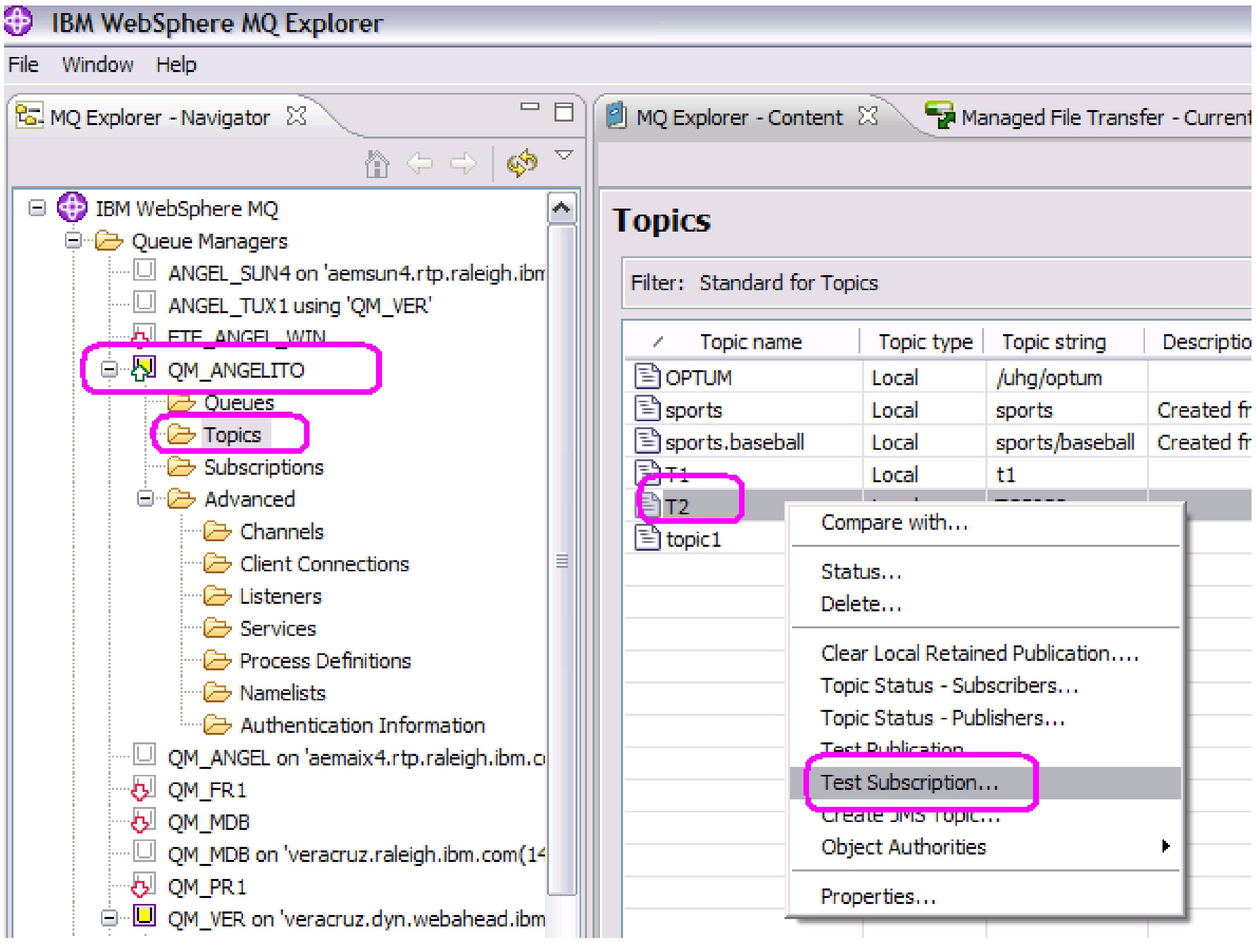
```
C:\> mqrc 2080

    2080  0x00000820  MQRC_TRUNCATED_MSG_FAILED
```

You could edit the amqssuba.c source code to expand the buffer in order to handle longer messages.

Or, you can use the MQ Explorer.

In this case, the MQ Explorer is running in Windows:

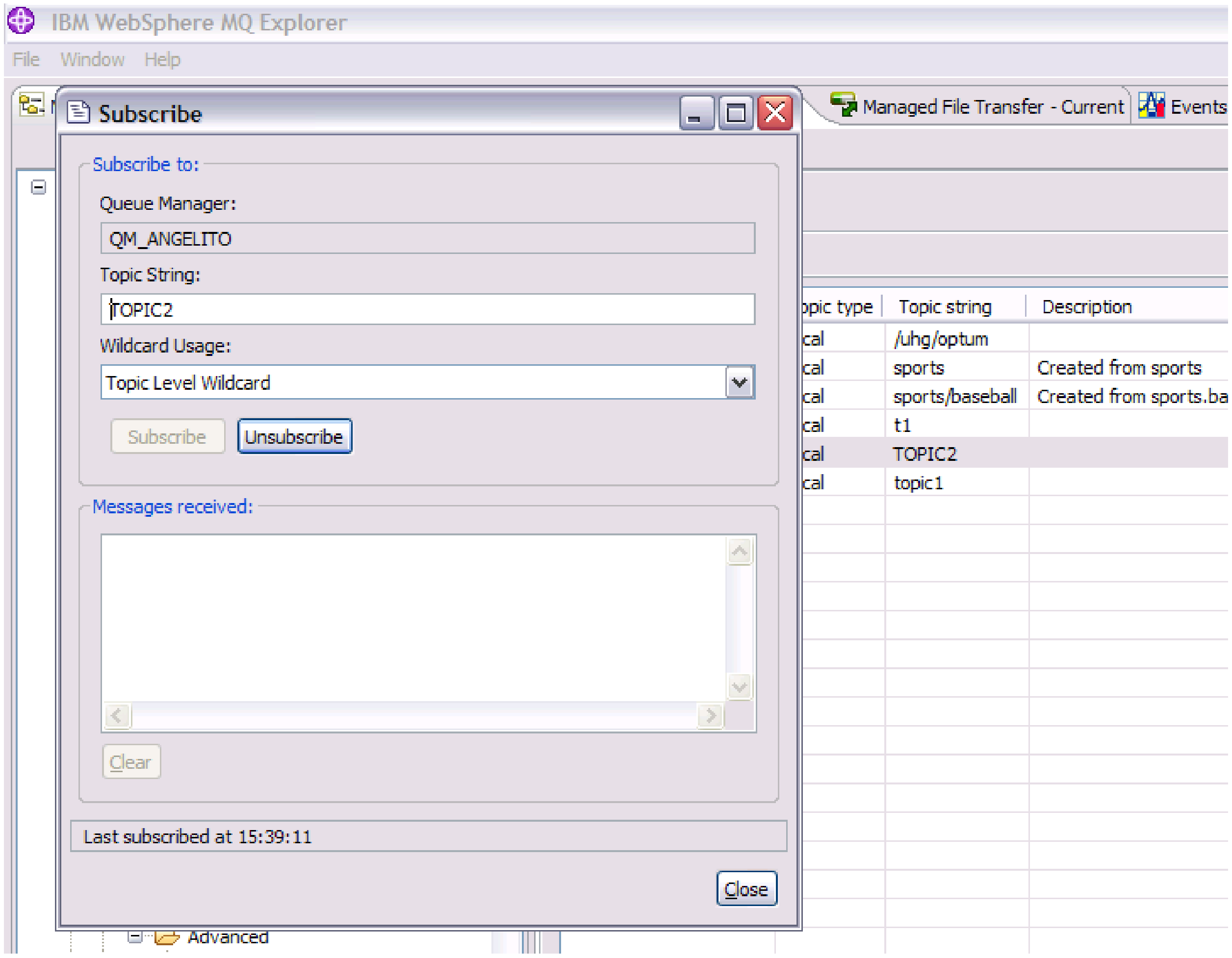
Select the Queue Manager "QM_ANGELITO" and expand the Topics.
Then select the desired topic T2 in the right panel, and do right click to bring the context menu and do a "Test Subscription".



Filter: Standard for Topics

Topic name	Topic type	Topic string	Descriptio
OPTUM	Local	/uhg/optum	
sports	Local	sports	Created fr
sports.baseball	Local	sports/baseball	Created fr
T1	Local	t1	
T2			
topic1			

You will see the Test Subscription window.
Notice that you can move this window around and still being able to interact with the rest of the MQ Explorer GUI.



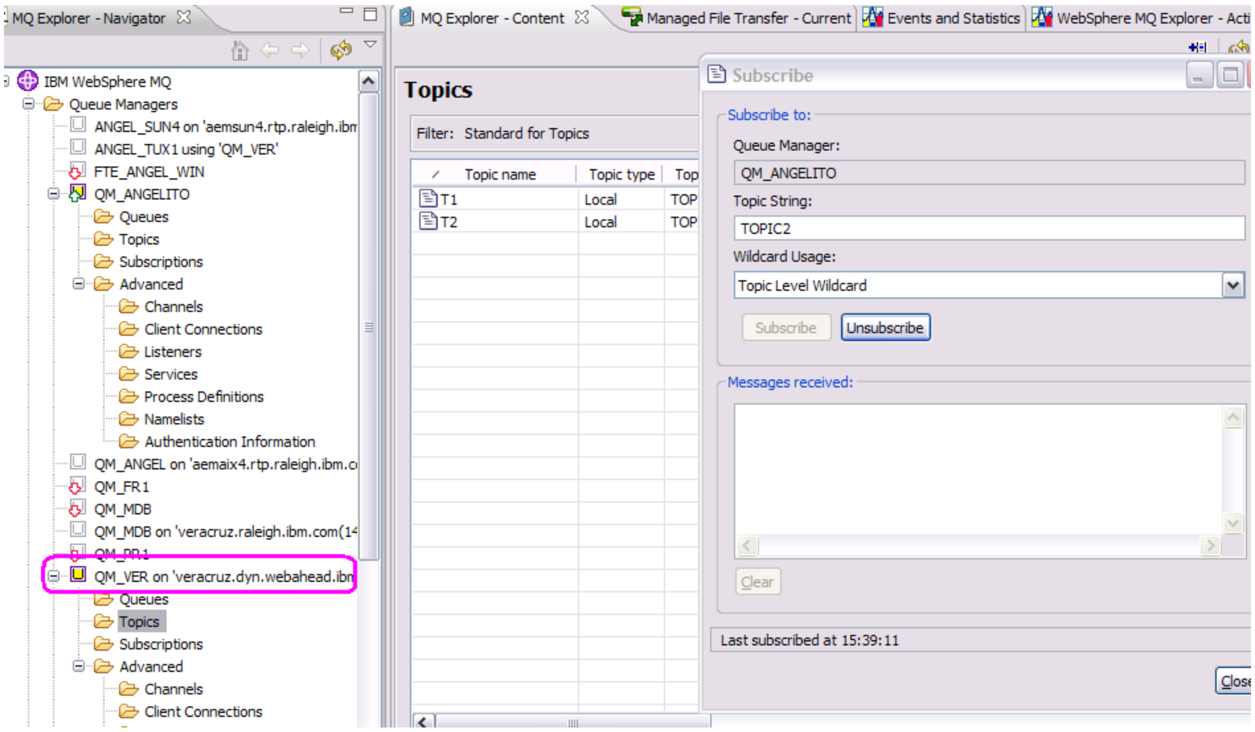
Queue Manager: QM_ANGELITO
Topic String: TOPIC2
Wildcard Usage: Topic Level Wildcard

Last subscribed at 15:39:11

opic type	Topic string	Description
cal	/uhg/optum	
cal	sports	Created from sports
cal	sports/baseball	Created from sports.ba
cal	t1	
cal	TOPIC2	
cal	topic1	

Move the window "Subscribe" to the right.
Notice that the Queue Manager is "QM_ANGELITO" and the Topic String is "TOPIC2"

Now click on the 2nd queue manager (QM_VER).
Notice that we have NOT closed the subscription for TOPIC2 in QM_ANGELITO.



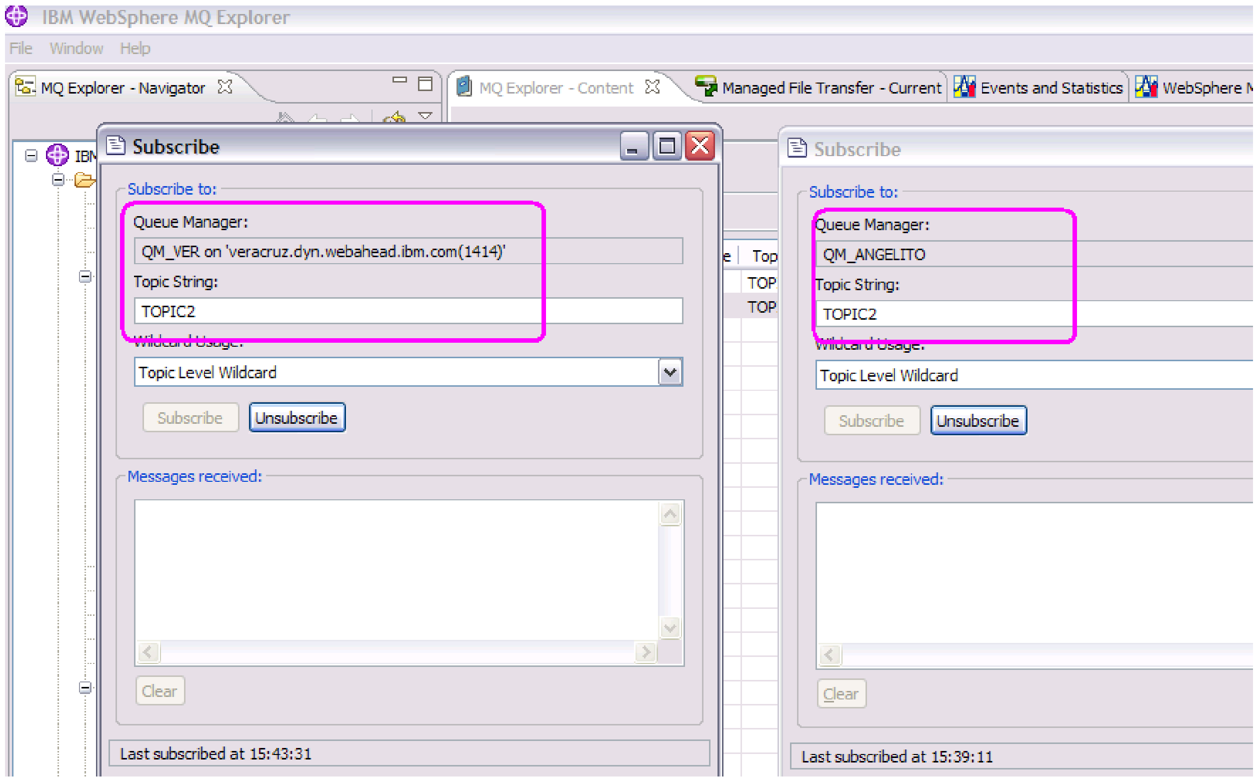
Topic name	Topic type	Top
T1	Local	TOP
T2	Local	TOP

Queue Manager: QM_ANGELITO
Topic String: TOPIC2
Wildcard Usage: Topic Level Wildcard

Last subscribed at 15:39:11

Repeat the process but this time with QM_VER: select the Topic T2 and Test Subscription.

Notice that we now have 2 test subscriptions:
On the left is for Queue Manager: QM_VER
On the right is for Queue Manager: QM_ANGELITO



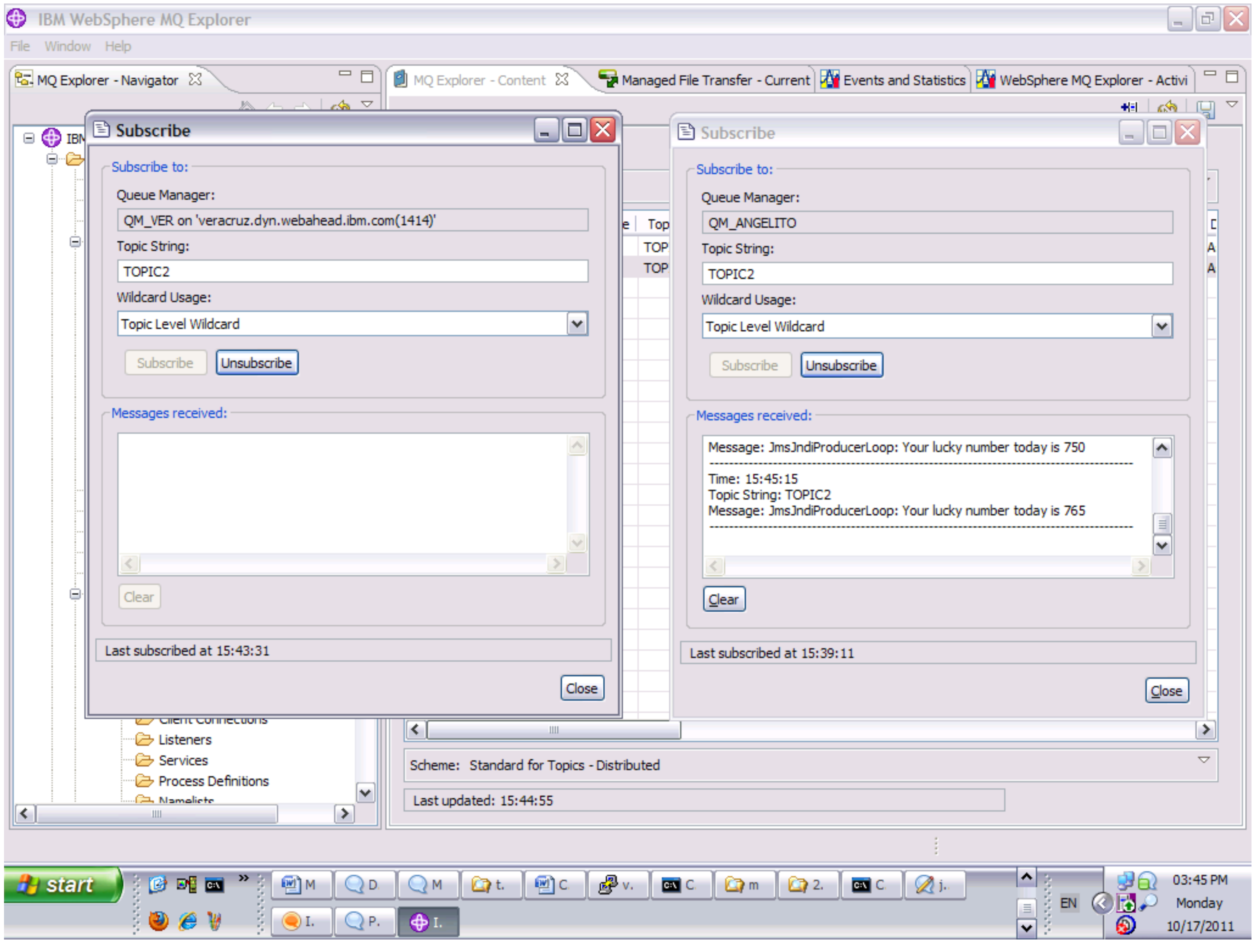
Queue Manager: QM_VER on 'veracruz.dyn.webahead.ibm.com(1414)'
Topic String: TOPIC2
Wildcard Usage: Topic Level Wildcard
Last subscribed at 15:43:31

Queue Manager: QM_ANGELITO
Topic String: TOPIC2
Wildcard Usage: Topic Level Wildcard
Last subscribed at 15:39:11

Window 1: Run the sample and specify a Topic:

```
C:\> java JmsJndiProducerLoop -i file:/C:/var/mqm/JNDI-Directory -c CF2REC -d T2
Initial context found!
Sent message:

  JMSMessage class: jms_text
  JMSType:          null
  JMSDeliveryMode:  2
  JMSExpiration:    0
  JMSPriority:      4
  JMSMessageID:     ID:414d5120514d5f414e47454c49544f20df839c4e20002903
  JMSTimestamp:     1318880689734
  JMSCorrelationID: null
  JMSDestination:   topic://TOPIC2
  JMSReplyTo:       null
  JMSRedelivered:   false
    JMSXAppID: WebSphere MQ Client for Java
    JMSXDeliveryCount: 0
    JMSXUserID: rivera
    JMS_IBM_ConnectionID: 414D5143514D5F414E47454C49544F20DF839C4E20002802
    JMS_IBM_PutApplType: 28
    JMS_IBM_PutDate: 20111017
    JMS_IBM_PutTime: 19444975
JmsJndiProducerLoop: Your lucky number today is 703
```

Notice that only the subscription on the left (QM_ANGELITO) is receiving messages:



Subscribe to:
Queue Manager: QM_VER on 'veracruz.dyn.webahead.ibm.com(1414)'
Topic String: TOPIC2
Wildcard Usage: Topic Level Wildcard
Messages received:
Last subscribed at 15:43:31

Subscribe to:
Queue Manager: QM_ANGELITO
Topic String: TOPIC2
Wildcard Usage: Topic Level Wildcard
Messages received:

```
Message: JmsJndiProducerLoop: Your lucky number today is 750
--------------------------------------------------------------
Time: 15:45:15
Topic String: TOPIC2
Message: JmsJndiProducerLoop: Your lucky number today is 765
--------------------------------------------------------------
```

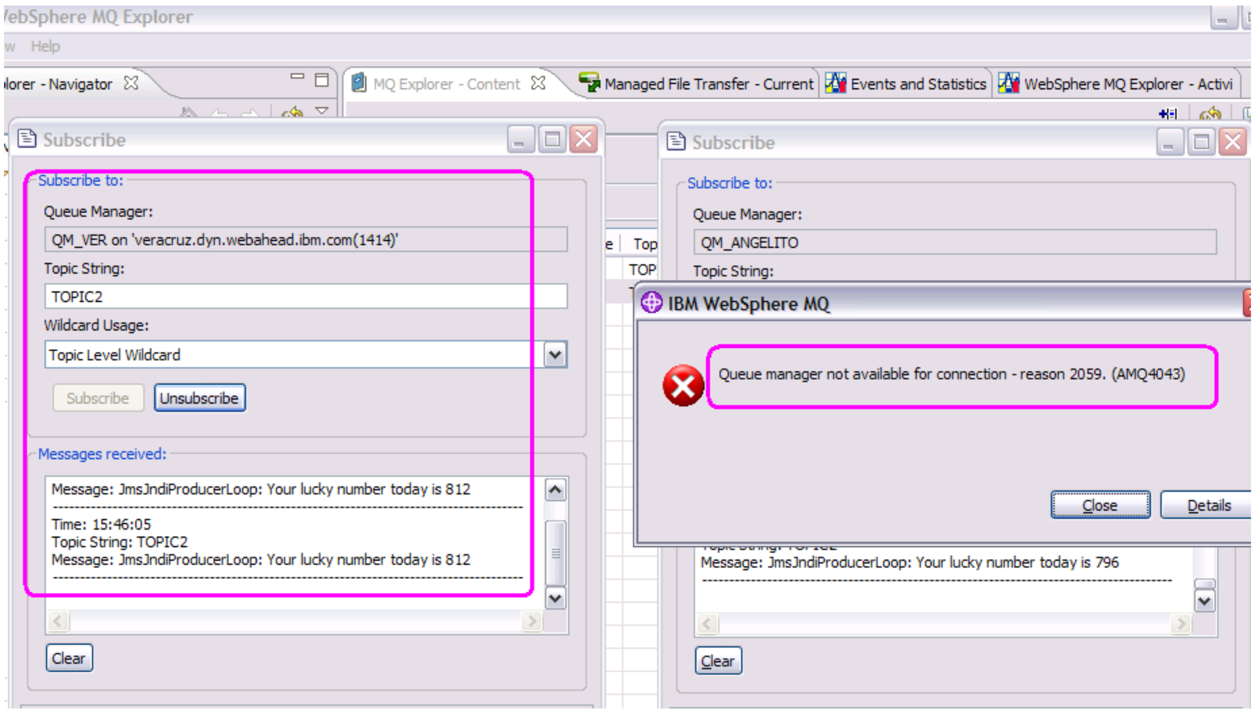
Last subscribed at 15:39:11

Scheme: Standard for Topics - Distributed
Last updated: 15:44:55

03:45 PM Monday 10/17/2011

Window 2: Terminate the queue manager QM_ANGELITO

MQ Explorer: Notice that the test subscription for QM_ANGELITO terminated (left one), the sample reconnects to the other queue manager and now the subscription for QM_VER is receiving messages:



Queue Manager: QM_VER on 'veracruz.dyn.webahead.ibm.com(1414)'
Topic String: TOPIC2
Wildcard Usage: Topic Level Wildcard

```
Message: JmsJndiProducerLoop: Your lucky number today is 812
------------------------------------------------------------
Time: 15:46:05
Topic String: TOPIC2
Message: JmsJndiProducerLoop: Your lucky number today is 812
------------------------------------------------------------
```

Queue Manager: QM_ANGELITO
Topic String:

Queue manager not available for connection - reason 2059. (AMQ4043)

```
Message: JmsJndiProducerLoop: Your lucky number today is 796
------------------------------------------------------------
```

Now you can terminate the sample that produces messages, and the test Subscription.

+++
+++ Chapter 6: Scenario NOT using JNDI - Windows
+++

The following samples are used in this chapter:
  JmsProducerLoop.java = > Uses JmsConnectionFactory
  JmsProducerLoopMQCF.java => Uses MQConnectionFactory

++ Edit the files and customize the connectionNameList

```
private static String connectionNameList = "host1(1414),host2(1414)";
```

++ Compile the samples:

```
C:\> javac JmsProducerLoop.java
C:\> javac JmsProducerLoopMQCF.java
```

++ Running the sample JmsProduceLoop (JmsConnectionFactory) from Linux

Open 3 windows:

Window 1: To run the JMS sample
Window 2: To start and stop the queue manager QM_ANGELITO in Windows
Window 3: To start and stop the queue manager QM_VER in Linux

Window 2: Start the queue manager QM_ANGELITO in Windows

```
C:\> strmqm QM_ANGELITO
```

Window 3: Start the queue manager QM_VER in Linux

```
$ strmqm QM_VER
```

Window 1: Run the sample code specifying the Connection Factory that has a connectionNameList and a queue destination.

```
C:\> java JmsProducerLoop -d Q2
Initial context found!
Sent message:
  JMSMessage class: jms_text
 …
JmsProducerLoop: Your lucky number today is 234
```

Window 2: Verify that Q2 in QM_ANGELITO is receiving messages:

```
C:\> amqsget Q2 QM_ANGELITO
amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
```

Window 3: Verify that Q2 in QM_VER is NOT receiving messages

```
$ amqsget Q2 QM_VER
Sample AMQSGET0 start
(no messages)
```

Window 2: End the queue manager with flags to terminate immediately and to notify the connected clients to reconnect

```
C:\> endmqm -ir QM_ANGELITO
Waiting for queue manager 'QM_ANGELITO' to end.
WebSphere MQ queue manager 'QM_ANGELITO' ending.
WebSphere MQ queue manager 'QM_ANGELITO' ended.
```

Window 1: Except for a pause for the MQ client code to detect the termination of the first queue manager and to reconnect to the second queue manager, there should not be any messages that indicate that a reconnect is happening, and the sample should continue producing messages.

```
Sent message:

  JMSMessage class: jms_text
…
JmsJndiProducerLoop: Your lucky number today is 516
```

Window 3: Verify that Q2 in QM_VER in Linux is now receiving messages:

```
$ amqsget Q2 QM_VER
amqsget Q2 QM_VER
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
```

Window 2: Verify that Q2 in QM_ANGELITO in Windows is NOT receiving messages (because the Queue manager is no longer running)

```
C:\> amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
MQCONN ended with reason code 2059
```

Window 2: Restart the queue manager QM_ANGELITO

```
C:\> strmqm QM_ANGELITO
```

Window 2: Notice that the sample is still connected to QM_VER and queue Q2 in QM_ANGELITO is NOT receiving messages:

```
C:\> amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
no more messages
Sample AMQSGET0 end
```

Window 1: Notice that the sample is still in the loop to produce messages.

Window 3: Now terminate QM_VER with the flags for immediate termination and to tell the clients to reconnect.

```
$ endmqm -ir QM_VER
WebSphere MQ queue manager 'QM_VER' ending.
WebSphere MQ queue manager 'QM_VER' ended.
```

Window 1: Notice that after a brief pause, the sample should continue producing messages.

Window 2: Verify that queue Q2 in QM_ANGELITO is now receiving messages:

```
C:\> amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
```

Window 1: To terminate the sample code, enter: Ctrl-C

+++ Note about using Topic

You can repeat the above scenario using the Topic string "sports":

Window 1:

```
C:\> java JmsProducerLoop -d topic://sports
```

Window 2: Issue the following command to subscribe to the topic string "sports" in QM_ANGELITO.

```
C:\> amqssub "sports" QM_ANGELITO
Sample AMQSSUBA start
Calling MQGET : 30 seconds wait time
MQGET ended with reason code 2080
Sample AMQSSUBA end
```

Notice that there was a message sent to the subscriber, but it was too long for the sample to handle. The MQ utility "mqrc" 2080 displays the error name.

```
C:\> mqrc 2080

     2080  0x00000820  MQRC_TRUNCATED_MSG_FAILED
```

You could edit the amqssuba.c source code to expand the buffer in order to handle longer messages.

Or, you can use the MQ Explorer. See the corresponding Chapter for the details on using the MQ Explorer for Windows: the same technique can be used with Linux.

++ Running the sample JmsProduceLoopMQCF (MQConnectionFactory) from Linux

Open 3 windows:

Window 1: To run the JMS sample
Window 2: To start and stop the queue manager QM_ANGELITO in Windows
Window 3: To start and stop the queue manager QM_VER in Linux

Window 2: Start the queue manager QM_ANGELITO in Windows

```
C:\> strmqm QM_ANGELITO
```

Window 3: Start the queue manager QM_VER in Linux

```
$ strmqm QM_VER
```

Window 1: Run the sample code

```
C:> java JmsProducerLoopMQCF -d Q2
Initial context found!
Sent message:

  JMSMessage class: jms_text
…
    JMS_IBM_PutDate: 20111018
    JMS_IBM_PutTime: 19242795
JmsProducerLoopMQCF: Your lucky number today is 411
```

Window 2: Verify that Q2 in QM_ANGELITO is receiving messages:

```
C:\> amqsget Q2 QM_ANGELITO
amqsget Q2 QM_ANGELITO
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
```

Window 3: Verify that Q2 in QM_VER is NOT receiving messages

```
$ amqsget Q2 QM_VER
Sample AMQSGET0 start
```

```
(no messages)
```

Window 2: End the queue manager with flags to terminate immediately and to notify the connected clients to reconnect

```
C:\> endmqm -ir QM_ANGELITO
Waiting for queue manager 'QM_ANGELITO' to end.
WebSphere MQ queue manager 'QM_ANGELITO' ending.
WebSphere MQ queue manager 'QM_ANGELITO' ended.
```

Window 1: Except for a pause for the MQ client code to detect the termination of the first queue manager and to reconnect to the second queue manager, there should not be any messages that indicate that a reconnect is happening, and the sample should continue producing messages.

Window 3: Verify that Q2 in QM_VER in Linux is now receiving messages:

```
$ amqsget Q2 QM_VER
amqsget Q2 QM_VER
Sample AMQSGET0 start
message <RFH ☻>
message <RFH ☻>
```

Window 1: To terminate the sample code, enter: Ctrl-C

+++ end +++