Unit 20.JMS administration

What this unit is about

This unit covers the way WebSphere MQ supports the Java Messaging System (JMS).

What you should be able to do

After completing this unit, you should be able to:

- Describe WebSphere MQ as a Java Message Service (JMS) provider
- Describe the advantages and disadvantages of using JMS versus native Java WebSphere MQ calls
- Plan a WebSphere MQ environment to support JMS
- Administer JMS resources in WebSphere MQ

How you will check your progress

- *Checkpoint quiz*
- Lab to follow

References

WebSphere MQ V7 Information Center

Sun JMS 1.1 Specification

Unit objectives

After completing this unit, you should be able to:

- Describe WebSphere MQ as a Java Message Service (JMS) provider
- Describe the advantages and disadvantages of using JMS versus native Java WebSphere MQ calls
- Plan a WebSphere MQ environment to support JMS
- Administer JMS resources in WebSphere MQ

Figure 20-1. Unit objectives WM203 / VM2032.0

Notes:

JMS overview topic objectives

After completing this topic, you should be able to:

- Describe the purpose of JMS
- State the difference between JMS and WebSphere MQ Java

Figure 20-2. JMS overview topic objectives WM203 / VM2032.0

Notes:

What JMS is

- JMS is a messaging API for Java applications
- JMS V1.1 is current standard
- Designed by Sun to be part of J2EE
- Platform neutral
- Portable
- Supports point-to-point (P2P) and pub/sub messaging styles

Figure 20-3. What JMS is

WM203 / VM2032.0

Notes:

Why use the JMS API

- Java applications (but not exclusively)
- J2EE applications
- Multiplatform applications
- Commercial applications
- Common skill set
- Integration with J2EE platforms

Figure 20-4. Why use the JMS API WM203 / VM2032.0

Notes:

The most obvious reason to use JMS is when Java applications are using messaging. But there are also non-JMS WebSphere MQ classes supplied that more closely resemble the MQI and these can be used. However, if an application is J2EE compliant, then messaging must use the standard JMS API.

If an application needs to run on various platforms, then Java is a good choice, so JMS should be considered. Similarly, commercial applications can follow the "write-once, run anywhere" philosophy of Java and JMS.

Since JMS is standard, staff JMS programming skills can be redeployed with more flexibility than those with more specialized programming skills.

J2EE application servers like WebSphere Application Server typically have significant integrated JMS administration interfaces. In the case of WebSphere Application server, potentially any JMS provider such as WebSphere MQ is supported.

WebSphere MQ classes for Java or JMS

WebSphere MQ Classes for Java	JMS
• Encapsulate MQI	• Implement Sun JMS interfaces
• Easy to implement for those familiar with MQI in other languages	• Easy for those with JMS skills
• Can exploit the full features of WebSphere MQ	• Integral part of J2EE
• Similar object model to other OO language interfaces like C++ and .NET	• Can use message-driven beans
	• Can use other JMS providers
	• Bridge applications between JMS providers

Figure 20-5. WebSphere MQ classes for Java or JMS WM203 / VM2032.0

Notes:

Here are some of the considerations for choosing between JMS and WebSphere MQ classes for Java. For the classes for Java, there is a close correlation to the underlying MQI. Those familiar with programming in the MQI should have little trouble adapting that knowledge to the classes for Java. The full functionality of WebSphere MQ can be used.

Some of the JMS considerations are already discussed. Message-driven beans (MDB) are a J2EE way to drive code when a message arrives on a destination.

If the need arises to bridge JMS providers of any heritage, this is easy with a JMS implementation.

JMS providers

- WebSphere MQ
- IBM WebSphere Application Server V5 Default Messaging Provider
- IBM WebSphere Application Server V6: Service Integration Bus
- Open Source: Apache ActiveMQ
- Just about any other J2EE provider

Figure 20-6. JMS providers WM203 / VM2032.0

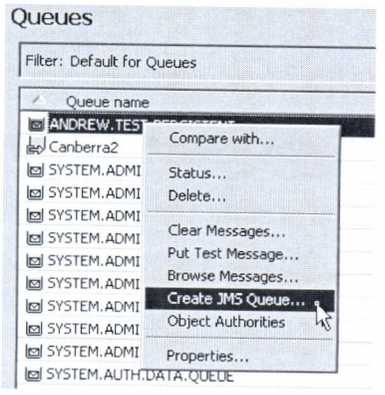
Notes:

There are many JMS providers. Since every J2EE implementation must have a JMS provider, there are as many JMS providers as there are J2EE providers.

WebSphere Application Server V6 provides Service Integration Bus (SIB) to provide JMS V1.1 support. WebSphere Application Server also provided interfaces to external JMS providers like JMS and other generic JMS providers.

JMS support in WebSphere MQ V7

- JMS and WebSphere MQ for Java are now peers in terms of implementation.
- Lots of application level improvements
- Improvements in the administration using WebSphere MQ Explorer:



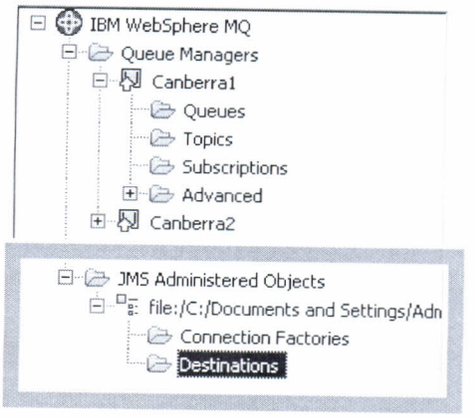


Figure 20-7. JMS support in WebSphere MQ V7

Notes:

In WebSphere MQ version 7.0, the implementation of WebSphere MQ classes for JMS is no longer dependent on WebSphere MQ classes for Java. WebSphere MQ classes for Java and WebSphere MQ classes for JMS are now peers that use a common Java interface to the MQI.

JMS overview topic summary

Having completed this topic, you should be able to:
- Describe the purpose of JMS
- State the difference between JMS and WebSphere MQ Java

Figure 20-8. JMS overview topic summary WM203 / VM2032.0

Notes:

JMS administration facilities topic objectives

After completing this topic, you should be able to:

- Describe the use if the JMSAdmin utility
- Configure the JMSAdmin utility
- Administer JMS administered objects using the WebSphere MQ Explorer

Figure 20-9. JMS administration facilities topic objectives WM203 / VM2032.0

Notes:

JMS administered objects

- Connection Factory:
  This is the object a client uses to create a connection with a provider.

- Destination:
  This is the object a client uses to specify the destination of messages it is sending and the source of messages it receives.

Figure 20-10. JMS administered objects WM203 / VM2032.0

Notes:

There are two JMS *Administered Objects.* The first *ConnectionFactory* is the object a client uses to create a connection with a JMS provider. In terms of WebSphere MQ, this object holds queue manager connection information.

The second object *Destination* is used to specify the source or destination of messages. For WebSphere MQ as a JMS provider, this object holds queue or topic information.

These objects are stored in a central repository accessed using the Java Naming and Directory Interface: JNDI.

JNDI namespace

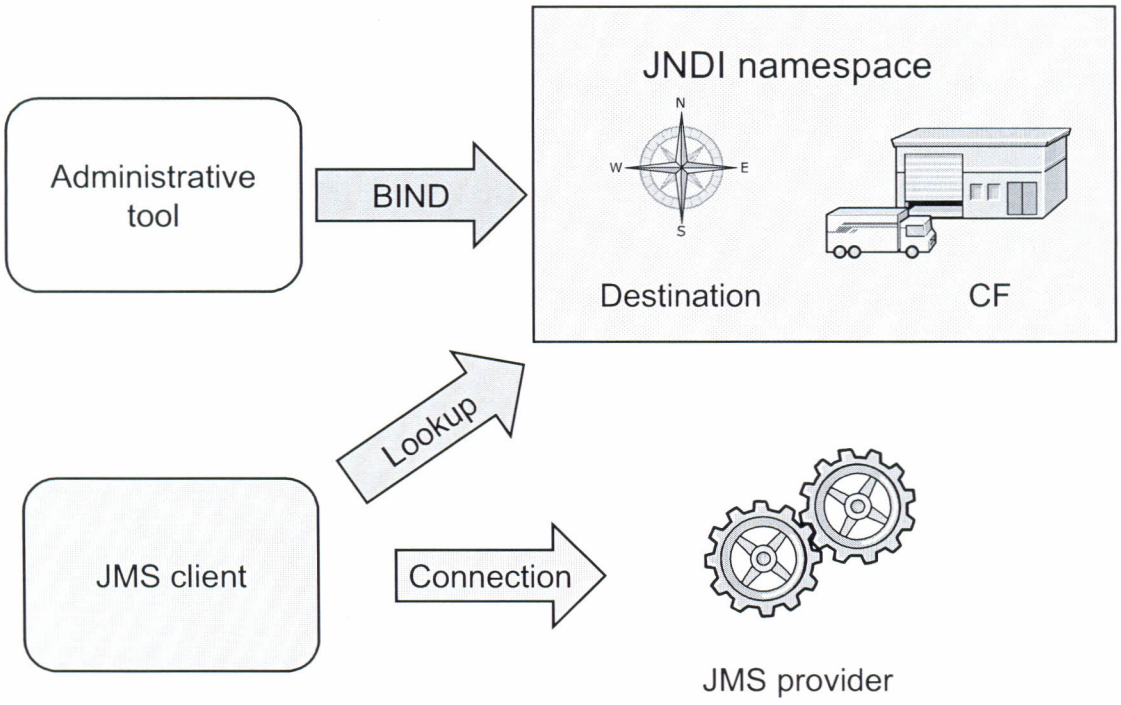


Figure 20-11. JNDI namespace

Notes:

The JNDI namespace provides a consultation facility. JMS uses it to store the destination and connection factory administrative objects.

JMS clients consult the JNDI namespace to retrieve an object that create a connection to the JMS provider. The retrieved destination object provides a connection to the queue or topic.

JMSAdmin utility

- Command line based
  - Provided with WebSphere MQ
- Use to define administered object to JNDI for
  - LDAP
  - Text file
  - Other JNDI
- First configure:
  `JMSAdmin.config`
- Run:
  `JMSAdmin.bat /JMSAdmin.sh`

Figure 20-12. JMSAdmin utility WM203 / VM2032.0

Notes:

Use the JMSAdmin utility to manager administered JMS definitions in a JNDI namespace.

The JMSAdmin.config file needs to be configured before JMSAdmin can be invoked. The two basic properties to be configured are the JNDI type and location.

JMSAdmin configuration

- Example:
  - Using a local file to hold JNDI:
    - C:\WMQ\JMSContext (directory)
  - JNDI Type: file system

- JMSAdmin.config:

```
INITIAL_CONTEXT_FACTORY=
    com.sun.jndi.fscontext.RefFSContextFactory
PROVIDER_URL=file:/C:/WMQ/JMSContext
```


Figure 20-13. JMSAdmin configuration WM203 / VM2032.0

Notes:

The JMSAdmin utility needs at least two settings:

- INITIAL_CONTEXT_FACTORY which is essentially how to access the JMDI repository
- PROVIDER_URL which indicates the location of the repository

JMSAdmin example (1 of 2)

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

InitCtx> display ctx
  Contents of InitCtx

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

InitCtx> define ctx(jms)
InitCtx> dis ctx

  Contents of InitCtx

  [D] jms                          javax.naming.Context

  1 Object(s)
    1 Context(s)
    0 Binding(s), 0 Administered

InitCtx> change ctx(jms)
InitCtx/jms>
```


Figure 20-14. JMSAdmin example (1 of 2)

Notes:

JMSAdmin example (2 of 2)

```
InitCtx/jms> dis ctx

  JMSADM4089 InitCtx/jms

      .bindings                    java.io.File
   a  QCF_AJGMQ1                   com.ibm.mq.jms.MQQueueConnectionFactory
   a  DEST_Q1                      com.ibm.mq.jms.MQQueue

  3 Object(s)
    0 Context(s)
    3 Binding(s), 2 Administered

InitCtx/jms> dis q(DEST_Q1)

    FAILIFQUIESCE(YES)
    QUEUE(Q1)
       :
    TARGCLIENT(MQ)
       :
    EXPIRY(APP)
    VERSION(7)
```


Figure 20-15. JMSAdmin example (2 of 2) WM203 / VM2032.0

Notes:

JMS messages are prepended with a header called an RFH2 header. If a receiving application is not a JMS application, the RFH2 header can be stripped by specifying TARGCLIENT(MQ). The default is TARGCLIENT(JMS).

JMSAdmin - Managing administered objects

- ALTER *TYPE*(name) [property]*
- DEFINE *TYPE*(name) [property]*
- DISPLAY *TYPE*(name)
- DELETE *TYPE*(name)
- COPY *TYPE*(nameA) *TYPE*(nameB)
- MOVE *TYPE*(nameA) *TYPE*(nameB)

Figure 20-16. JMSAdmin - Managing administered objects WM203 / VM2032.0

Notes:

JMSAdmin - Available object types

Object Type	Keyword	Description
MQConnectionFactory	CF	This represents a factory object for creating connections in the both the point-to-point and publish/subscribe domains.
MQQueueConnectionFactory	QCF	This represents a factory object for creating connections in the point-to-point domain.
MQTopicConnectionFactory	TCF	This represents a factory object for creating connections in the publish/subscribe domain.
MQQueue	Q	This represents a destination for messages in the point-to-point domain.
MQTopic	T	This represents a destination for messages in the publish/subscribe domain.

Figure 20-17. JMSAdmin - Available object types WM203 / VM2032.0

Notes:

This table shows some of the administered objects that can be managed using JMSAdmin. Each of these objects has an array of attributes. Consult the WebSphere MQ Information Center for a complete description.

WebSphere MQ Explorer facilities for JMS

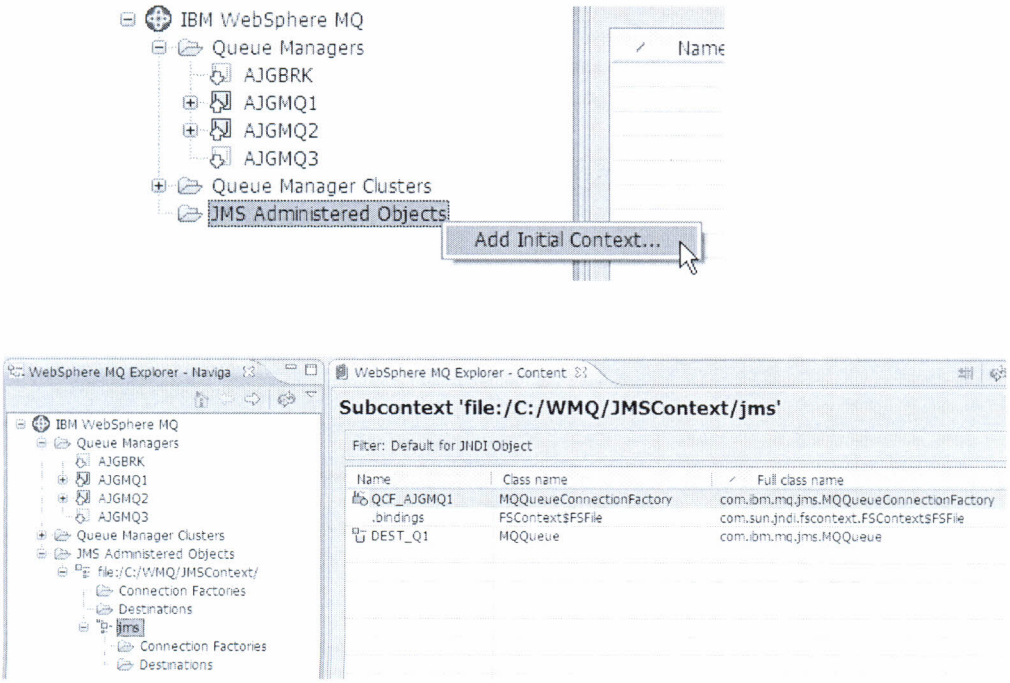


Figure 20-18. WebSphere MQ Explorer facilities for JMS WM203 / VM2032.0

Notes:

The same way JMSAdmin needs to be configured, the WebSphere MQ Explorer first needs to be told the type and location of the JNDI repository.

Once connected, the WebSphere MQ Explorer may be used to perform most JMS administration tasks.

JMS object creation wizards

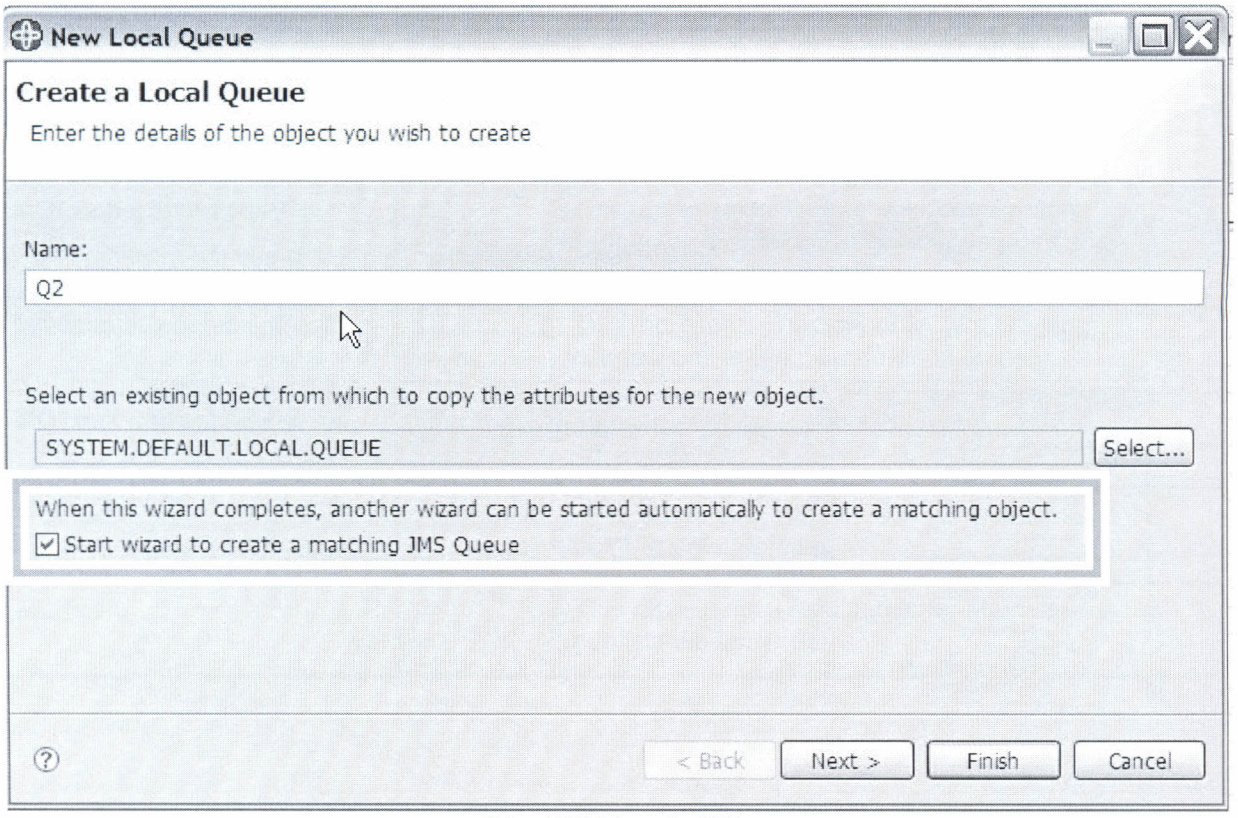


Figure 20-19. JMS object creation wizards WM203 / VM2032.0

Notes:

There are a number of integrated JMS definition wizards to assist with the creation of JMS administered objects for newly created queues and topics and existing queues and topics.

Checkpoint questions

1. True or false:
   a. JMS is an API
   b. JMS is platform neutral
   c. JMS part of J2EE
   d. JMS is programming language neutral

2. Which of the following objects are JMS Administered objects:
   a. Queue
   b. Destination
   c. Collection Factory
   d. TARGCLIENT

3. True or false: The JNDI namespace can be a database

4. True or false: If a J2EE provider (Application Server) administration tool has facilities for managing JMS administered definitions, you should use that facility.

Figure 20-20. Checkpoint questions

Notes:

JMS administration facilities topic summary

Having completed this topic, you should be able to:
- Describe the use if the JMSAdmin utility
- Configure the JMSAdmin utility
- Administer JMS administered objects using the WebSphere MQ Explorer

Figure 20-21. JMS administration facilities topic summary WM203 / VM2032.0

Notes:

Unit summary

Having completed this unit, you should be able to:

- Describe WebSphere MQ as a Java Message Service (JMS) provider
- Describe the advantages and disadvantages of using JMS versus native Java WebSphere MQ calls
- Plan a WebSphere MQ environment to support JMS
- Administer JMS resources in WebSphere MQ

Figure 20-22. Unit summary

WM203 / VM2032.0

Notes: