JMS Administration in WebSphere 5.0

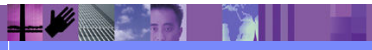
WebSphere. software

e business software

Updated 5/16/2003

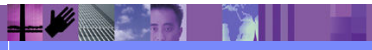
Objectives

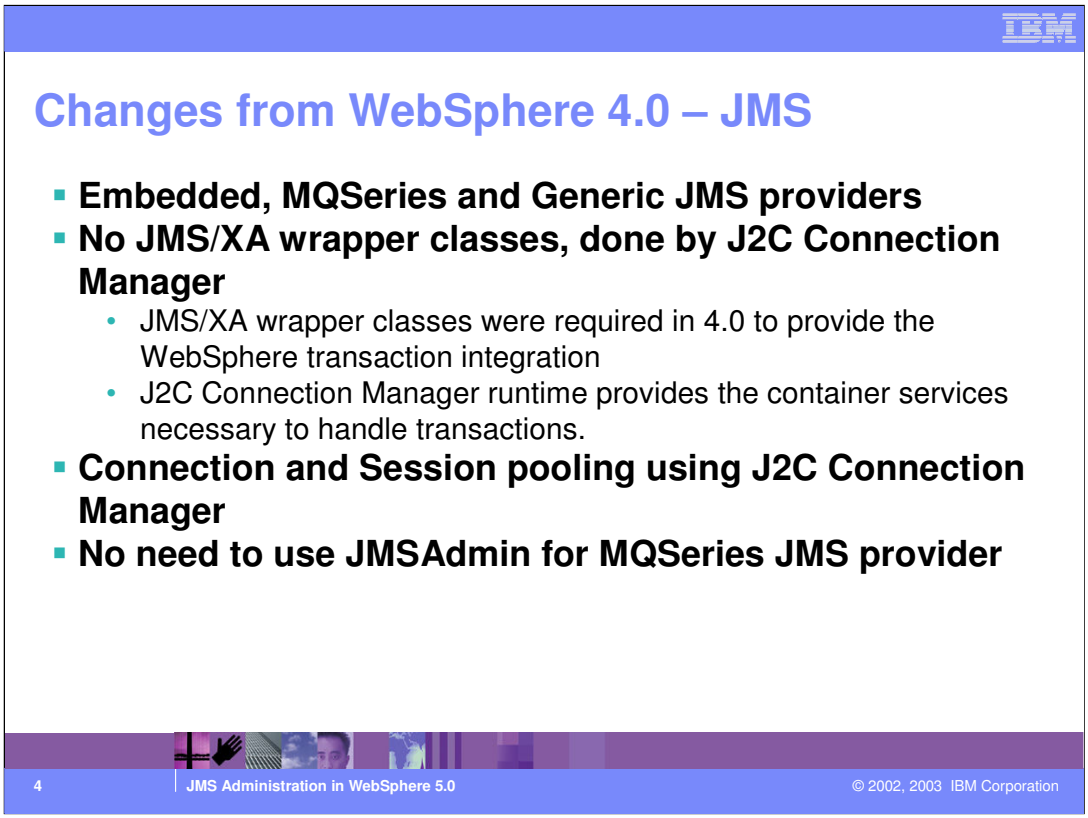
- Describe the improvements in Messaging support in WebSphere 5.0
- Administer and configure JMS resources using WebSphere Administrative console
- Discuss troubleshooting JMS related problems in a WebSphere environment



Agenda

- **Overview - Changes from WebSphere 4.0**
- **WebSphere JMS Support**
  - WebSphere JMS Server Support
  - WebSphere MQ 5.3
- **Administration**
  - Internal Provider
  - External Provider
- **Configuration**
  - Deploying MDB with JMS destinations
  - Deploying JMS Clients
- **Troubleshooting**





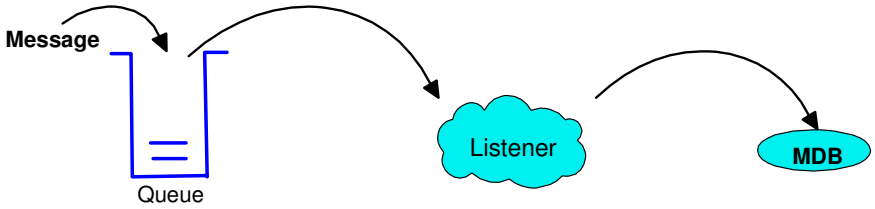
For WebSphere 4.0, JMS support requires the separate installation of a JMS product, for example MQSeries for point-to-point messaging, with the addition of WMQ Integrator or the MA0C Product Extension for publish/subscribe. WebSphere 5.0 has included an embedded JMS Provider which is integrated with the product and can be installed with WebSphere,

In order for JMS operations to be part of a global transaction, the container must be made aware that the JMS Session used for the operations is to be included in the transaction. Unfortunately, J2EE 1.3 does not define how JMS should be enlisted into the container's global transactions. WebSphere 4.0 got around the problem for MQ by wrapping the MQ ConnectionFactory, Connection, and Session objects, with "WASMQ" versions, which are stored in JNDI in place of the normal MQ objects. This has been redefined in WebSphere 5.0 using the J2C Connection Manager runtime which provides the container services necessary to handle transactions.

JMS Connection and session resources are managed by J2C Connection Manager. The advantage of using the J2C runtime component is that it enables the integration of the container services such as transaction and security.

MQSeries JMS resources can be administered using the WebSphere 5.0 Administrative console.

Changes from WebSphere 4.0 – MDBs



- **Message Driven Beans are fully supported (J2EE 1.3)**
- **WebSphere Studio Application Developer and AAT support for EJB 2.0 MDBs**
- **MDB Listener fully integrated with EJB Container**

A Message-driven Bean is a stateless component that is invoked by the J2EE container (app server) as a result of the arrival of a JMS message at a particular JMS Destination (that is, a Queue or a Topic). You think of the MDB as being “triggered” by the arrival of a message. The Listener component is the part of the Application

Server that invokes the MDB when the message arrives; you think of this as providing the equivalent function for JMS and J2EE to that provided by the CICS Bridge for MQSeries and CICS. WebSphere 4.0 Enterprise Edition supports Message beans which were similar to Message Driven Beans but Message beans are actually stateless session beans.

WebSphere Studio Application Developer can be used to develop code and deployment descriptors for Message driven beans. Application assembly tool can also be used to assemble Message driven beans.

MDB listener is fully integrated with EJB container. MDBs can be administered fully by from the WebSphere 5.0 Administrative clients. MDB Listener recovery from messaging domain failures is new in 5.0 - for example, if the external queue manager fails, in 4.0, the listener stopped and you had to restart it. In 5.0 there is recovery in such cases.

Changes from WebSphere 4.0 – MDBs

- **MDB configuration fully SM administered (Message Listener Service, Listener Ports)**
- **Migration utility for the Enterprise Edition 4.0 Message Beans to MDBs (mb2mdb)**
- **MDB Listener recovery from messaging domain failures**
- **Managed MDB durable subscriptions**
- **Migrated MDB samples**

Durable pub/sub - A subscriber can specify a client id on their JMS connection and then create a durable subscriber. This causes the JMS server to store persistently all messages matching the subscription on the server. If the client disconnects and then reconnects with the same client id then it should start receiving messages from where it left off.

Managed MDB durable subscriptions - If applications are deleted, then unwanted subscriptions are tidied out rather than taking out valuable resources.

4.0 Message bean samples have been migrated to 5.0 MDB samples.

WebSphere 5.0: JMS Server Support

- **J2EE 1.3 requires App Servers to provide a JMS Server**
- **Full function JMS Server is included with WebSphere Application Server**
  - Installed as part of the server installation
  - Fully integrated with the server's administration and runtime
  - Fully compliant with J2EE 1.3 compliance tests

Two of the “sub-specifications” of J2EE 1.3 relate to messaging: Java Message Service (JMS) and Message Driven Beans (MDBs). J2EE 1.3 specification requires any application server claiming compliance to the J2EE spec to:-

1•Provide an implementation of the Java Messages Server (JMS), version 1.0

2•Support the Message-driven Beans programming model, defined in the EJB 2.0 specification.

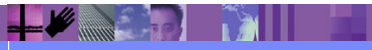
Hence, as per the first criterion, WebSphere provides an embedded JMS Server that can be installed along with a WebSphere base install.

JMS is the standard Java API for applications to use to perform messaging. It has two “domains” corresponding to two ways of using messaging: point-to-point and publish/subscribe. The specification was developed by Sun Microsystems with help from IBM, other messaging vendors, and other application server vendors.

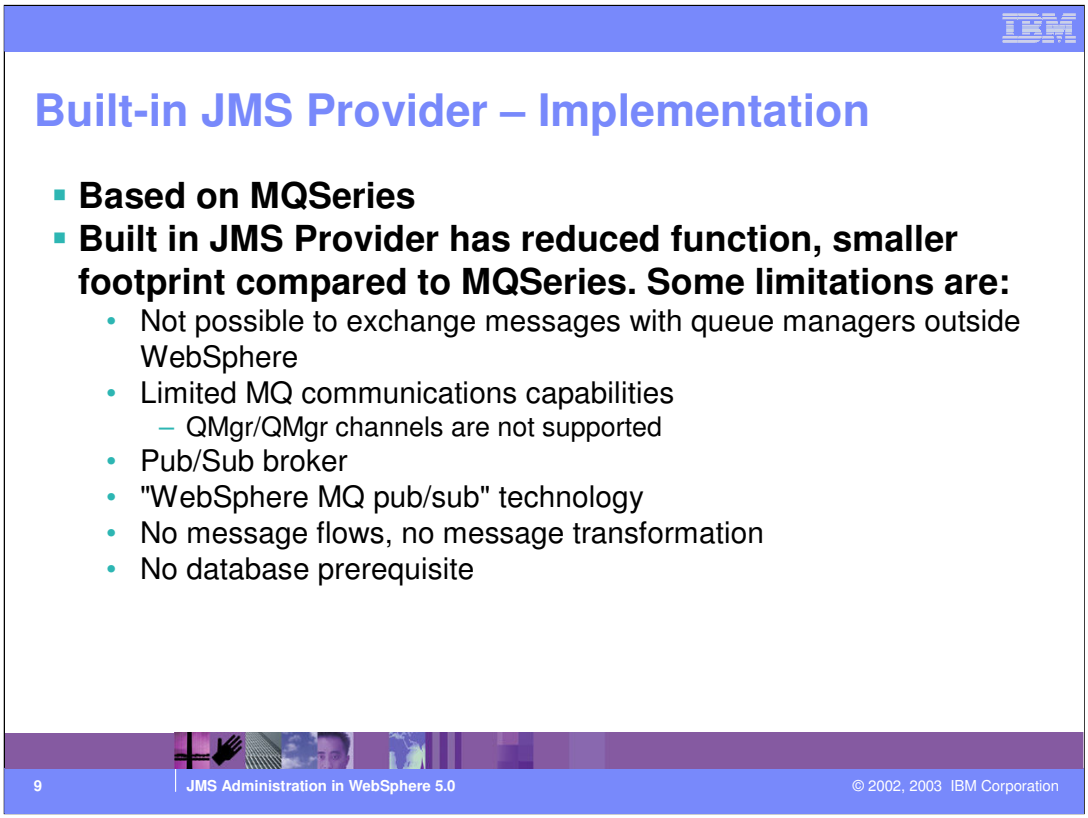
The standard defines a package of Java Interfaces, to be implemented by the messaging vendor, or “provider” to use the J2EE terminology. The internal JMS Provider is fully compliant to the specifications.

Built-in JMS Provider

- **Built-in JMS may be used by**
  - EJBs, JSPs, and Servlets running in the application server
  - Java applications running in the WebSphere Application client
- **Built-in JMS does not provide**
  - API support other than J2EE JMS API standards
  - Inter-operation with MQSeries



The internal JMS Provider is not interoperable with WebSphere MQ, and therefore is not useful for heterogeneous communication between WebSphere Application Server and other environments.



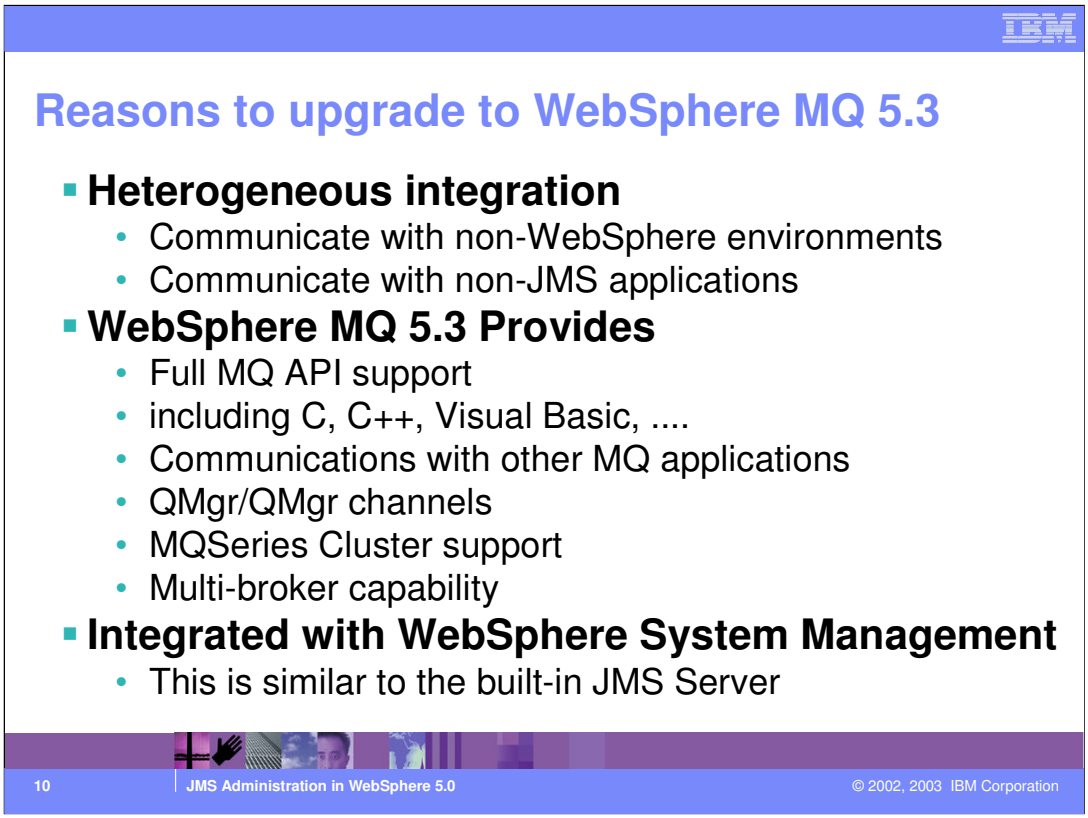
In J2EE 1.3, the Java Messaging Service becomes an integral part of the J2EE platform. In v5.0, WebSphere satisfies this requirement with the embedded WebSphere JMS Provider.

The implementation of the embedded WebSphere JMS Provider is based on the integration of IBM's WebSphere MQSeries product and the JMS Client (MA88) support pack into WebSphere Application Server. The versions of these products that are embedded are reduced foot print and reduced function versions of the independently shipped MQ products.

The internal or built-in JMS Provider provides both point-to-point as well as pub/sub messaging support. Point-to-Point messaging employs a Queue as the JMS destination. One JMS client will retrieve the message from this queue. Publish/Subscribe service employs a Topic as the JMS destination. A message is published to a topic. Multiple JMS clients may subscribe to this topic, and each subscribing client will receive the message.

Each JMS Server is responsible for managing a WebSphere MQ Queue Manager and the internal provider of the Publish/Subscribe service.

Since the built in JMS provider is a reduced footprint of MQSeries, there are some limitations to using it. Primarily it is not possible to communicate with non WebSphere environments using the internal JMS provider. Also, some advanced options such as QMgr/QMgr channels , message flows and message transformations are not supported.



Although WebSphere 5.0 provides a fully-compliant, built-in JMS provider, we expect that many customers will continue to select WebSphere MQ as their JMS Provider, for either of two reasons:

As described earlier, the WebSphere JMS Provider can only be used from within the WebSphere environment. Many WebSphere customers are using WMQ today for heterogeneous integration, allowing their J2EE applications to communicate with other MQ applications, that may be written in other languages and run on other platforms (Microsoft COM, for example).

The other reason for using WebSphere MQ is in order to take advantage of more advanced messaging topologies, including the use of:

• Queue manager clustering, for example to support higher throughput levels and continuous availability for new messages triggering MDBs.

• HA technologies, such as HACMP on AIX, Microsoft Cluster Server, or HP ServiceGuard. This technique, sometimes called “hardware availability”, uses shared disks to enable both the messaging service and individual in-flight messages to be highly available.

• Broker collectives, for example using multiple copies of WebSphere MQ Integrator.

• z/OS Shared Queues - the ultimate in high-availability messaging.

MQSeries JMS resources can also be administered and configured via WebSphere Admin Console. This is similar to the System Management support for the built-in JMS server.

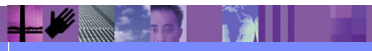
Using WebSphere MQ as external JMS Provider

- **Install WebSphere MQ 5.3**
  - Information about MQSeries 5.3 is available at http://www.ibm.com/software/ts/mqseries/messaging/v53
  - With MQ 5.3, the Java and JMS Support pack (MA88) are not needed separately.
- **Install a recommended CSD on top of MQSeries**
- **To use pub/sub**
  - Install a pub/sub broker
  - Use WebSphere MQ Integrator 2.1 or Event Broker 2.1
  - MA0C is not supported.
    - However it can be used for testing now.

WebSphere MQ, V5.3 brings new performance levels, enhanced security, and added features to enhance cross-platform consistency (harmonization).

Usage scenarios

- **Customers who do not require full function MQSeries, can use the Internal JMS Provider.**
  - This will simplify the configuration and management of messaging.
- **Customers currently using full function MQSeries products are expected to continue doing so, and can ignore the embedded WebSphere JMS Provider.**
- **Customers can also start by using the internal JMS provider.**
  - Can upgrade to full MQSeries product at a later time



The introduction of the Internal JMS Provider in no way changes the existing application usage scenarios for applications using JMS. Customers that do not required some all of the advanced features that MQSeries provides can use the embedded WebSphere JMS Provider. This will simplify the configuration and administration involved.

Customers currently using full function MQSeries products are expected to remain using the full function products and can completely ignore the embedded JMS Provider.

Customers who are introducing messaging to their application can start by using the Internal JMS provider. They can upgrade to the full blown MQSeries product at a later time if desired.

WebSphere Admin clients allow us to manage three types of JMS Resources:-

WebSphere JMS Server Resources are resources related to the built in JMS Server

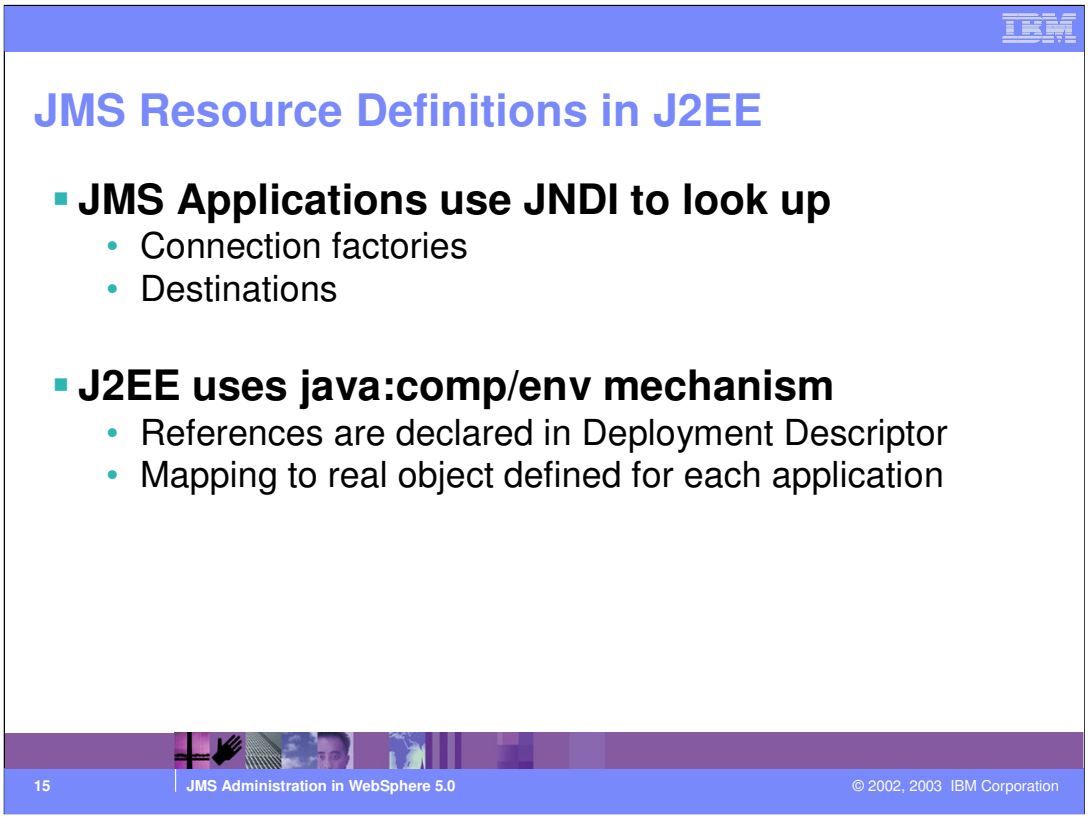
WebSphere MQSeries Resources are resource related to the full WebSphere MQ Series 5.3 product. You can have WebSphere MQ Series installed along side the internal JMS Server.

Generic JMS Server Resources relate to third party JMS servers that you may want to configure.

JMS Server – Administration (continued)

- **Integrated provider runs in a JMS Server process**
  - In the single server configuration, JMS Server runs within the App Server
  - In a Network Deployment configuration, you only have one active JMS Server per node
    - Gets created for you, you only need to configure it
    - Runs in its own JVM
    - JMS resources defined in a JMS Server are accessible from anywhere in the cell

The integrated JMS Server itself is defined differently for Base and Network Deployment. In the base or single server configuration, JMS Server runs within the Application Server's JVM. In the Network Deployment edition, the JMS Server is started in its own JVM. There can only be one JMS Server defined on a given node.



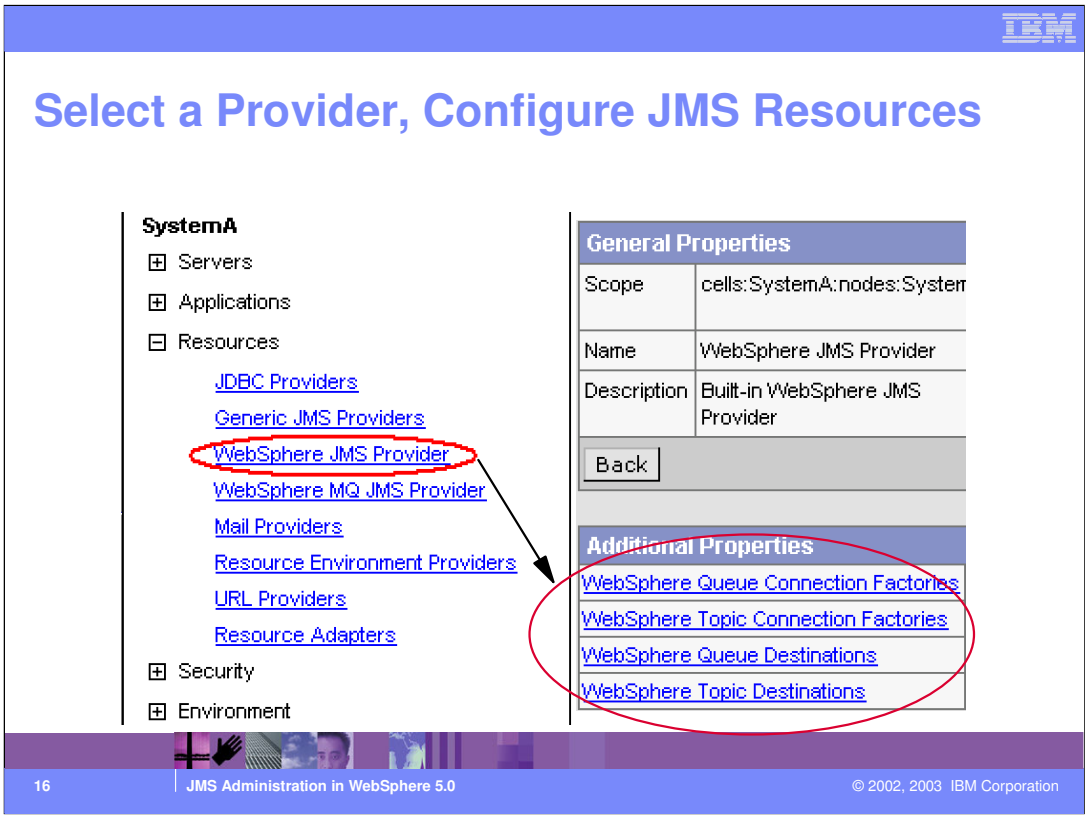
In JMS certain objects are created by an administrator and stored in a directory. Applications normally use Java Naming and Directory Interface (JNDI) to retrieve the objects from the directory, rather than creating them directly in the application code. Specifically,

•ConnectionFactory objects, which define how to connect to the messaging service, are retrieved using JNDI in order to provide vendor-independence

•Destination objects, that is, Queue and Topic references, are retrieved using JNDI so that knowledge of the underlying queuing topology or topic space is not coded into the application. Destination objects can also have vendor-specific properties.

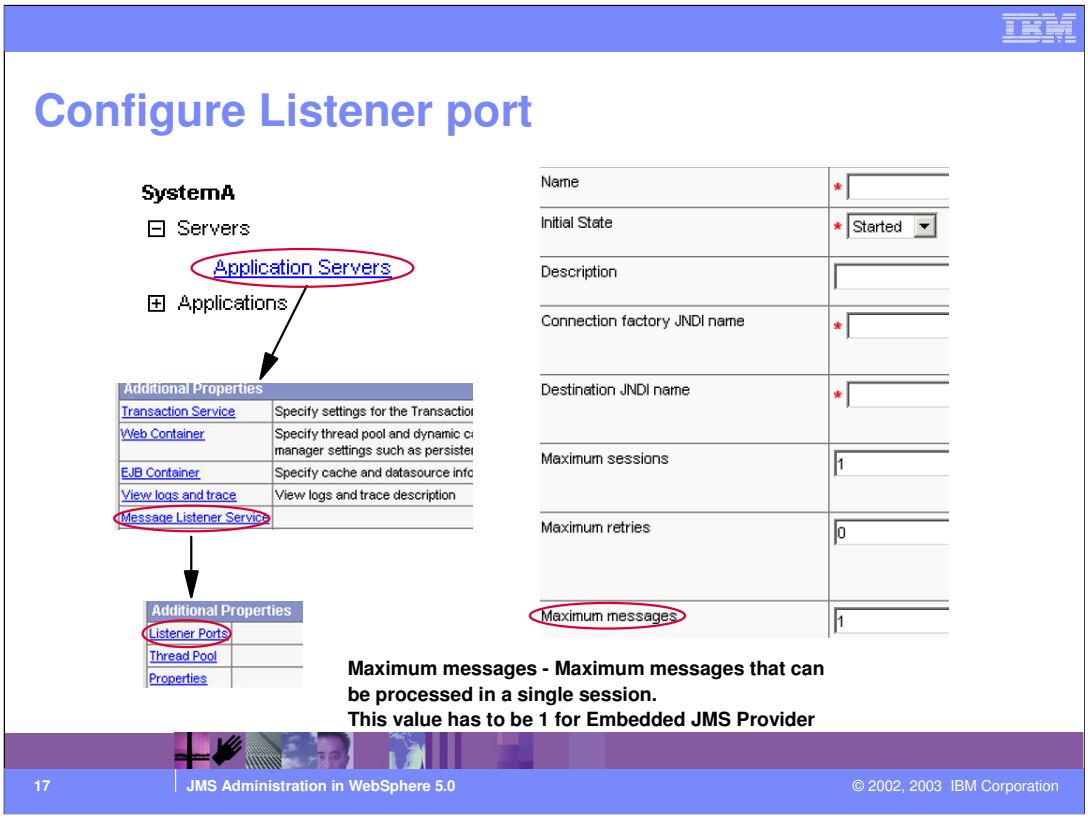
J2EE adds a further level of indirection: It is possible to deploy multiple independently-developed applications into the same application server. For example, imagine an Application Service Provider supporting multiple outsourced applications in one large mainframe. The two applications could easily select the same topic name

for some of their messaging operations, “catalogue/priceupdate”, for example. To avoid this kind of name collision, J2EE uses a mechanism with the rather cumbersome name of “java:comp/env”. Instead of looking up the Destination “priceupdate” using JNDI, each application looks up “java:comp/env/priceupdate”. The administrator can then define different resolutions of the lookup for each application, and the container is responsible for returning values to “java:comp/env” lookups based on the application from which the JNDI call was made. This mechanism is used for JMS and for access to other, such as JDBC.



You can use the WebSphere administrative console to configure the internal WebSphere JMS provider or an MQSeries JMS provider. If you install another JMS provider, you need to configure that JMS provider by using the tools and information provided with the JMS provider. For each JMS provider, you can configure the properties of JMS resources.

The foil shows a screen shot of how the Admin console can be used to manage WebSphere JMS Provider properties. You can use the WebSphere Admin console to configure a general set of JMS server properties, which add to the default values of properties configured automatically for the embedded WebSphere JMS provider. It is also possible to manage WebSphere queues, WebSphere topics and WebSphere connection factories using the Admin Console.

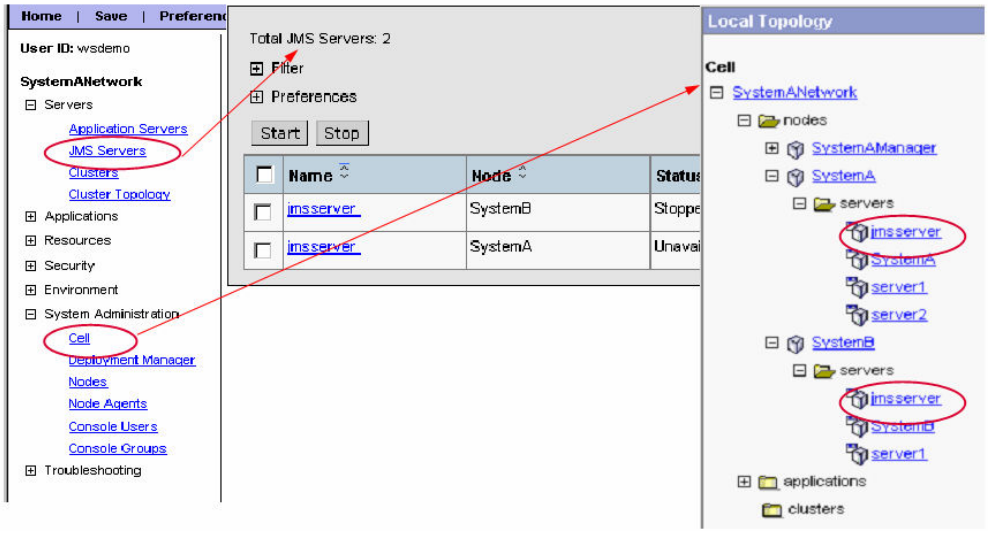


You can also use the WebSphere administrative console to configure and control other runtime components of WebSphere JMS support, including the message listener service and listener ports. The message listener service is an extension to the JMS functions of the JMS provider. It provides a listener manager that controls and monitors one or more JMS listeners, which each monitor a JMS destination on behalf of a deployed message-driven bean. Each listener port is used with a message-driven bean to automatically receive messages from an associated JMS destination. The deployer associates a listener port with a Connection Factory JNDI name and a Destination JNDI name.

The Maximum Messages listener port value relates to the maximum number of messages that can be processed during a single execution of a JMS Listener Server Session. The JMS Listener works based on the JMS ASF model of using Connection Consumers, which involves the use of a Server Session Pool. When the JMS provider receives messages it "loads" these messages into a Server Session from the pool, and then "runs" the Server Session. The Maximum Messages is actually the last parameter on the JMSConnection.createConnectionConsumer() method, and is the maximum number of messages that can be "loaded" into a Server Session by the provider for each "run".

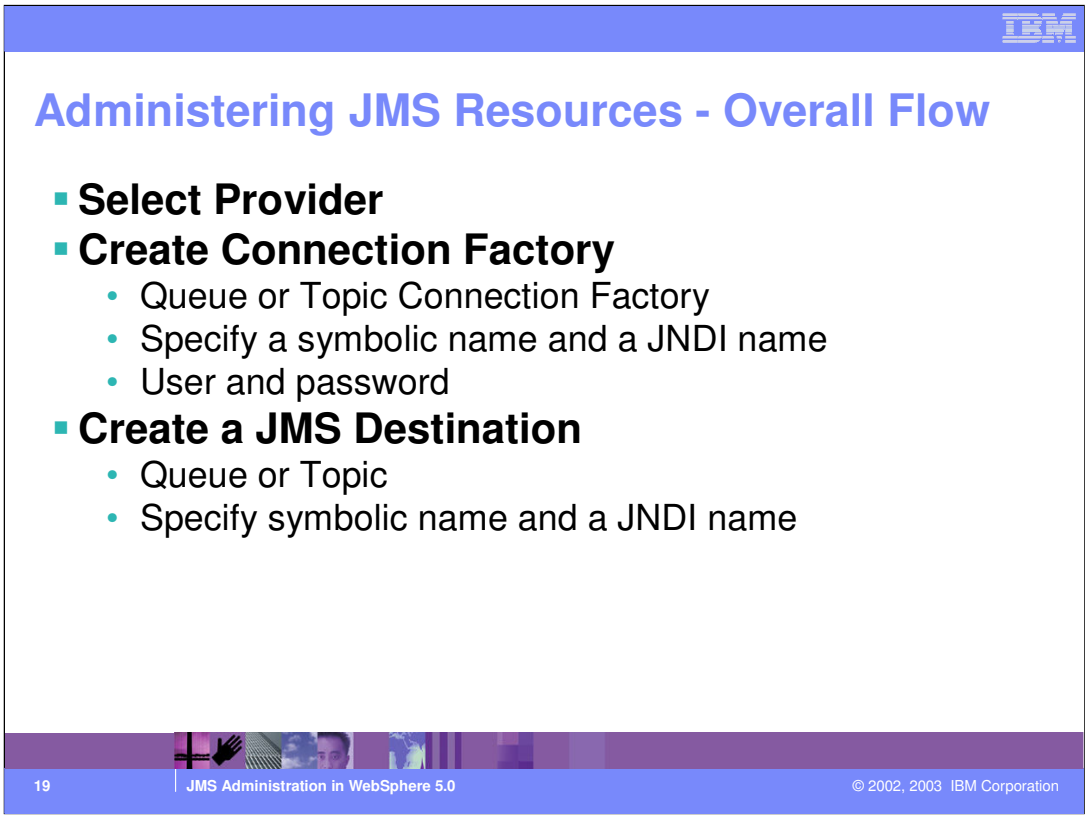
Loading more than 1 message into a Server Session actually requires a certain transaction model, which MQSeries does not support. MQSeries and hence Embedded JMS Provider, thus has a restriction that maxMessages is forced to always be 1.

Managing JMS Servers - Network Deployment



In an Network Deployment installation, the JMS Server is basically the Broker and additionally manages the QueueManager. It is a peer of the other servers in a given node, and it runs in its own JVM. There can only be one JMS Server defined on a given node.

The foil shows a screen shot of the Admin Console. The cell has two nodes:- SystemA and SystemB. Each node has a JMS server which runs in its own JVM.



Message-driven Bean support in WebSphere 5.0 introduces several new administered objects. The full set of tasks required to run a Message-driven Bean are:

1. Choose the provider. You can choose the Integrated JMS provider, MQSeries or a third party JMS provider.

2. Create the ConnectionFactory for the JMS provider you wish to use. Create a queue connection factory if you want to do point-to-point messaging. Create a topic connection factory if you want to use publish/subscribe

3. Create the Destination from wish you want the MDB to be triggered. The destination can be a queue or a topic depending on whether you are using point-to-point or pub-sub messaging.

3. Create something called a “ListenerPort”, specifying the ConnectionFactory and Destination. Associated with the ListenerPort is a configurable pool of threads on which the MDBs onMessage method will be dispatched.

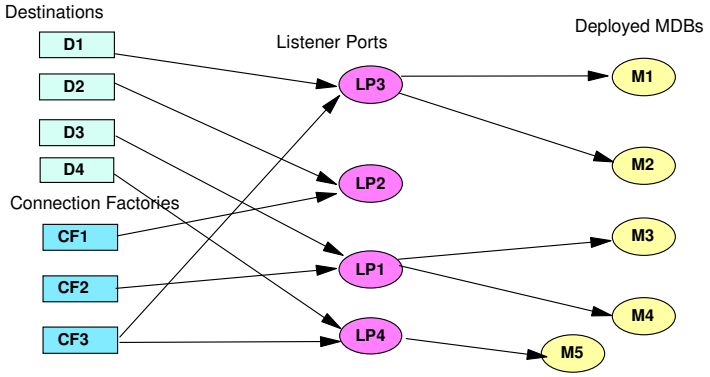
Finally, Deploy the MDB, specifying the ListenerPort (and hence Destination), with which it is to be associated.

Administering JMS Resources (continued)

- **Create a Listener Port**
  - Select an Application Server
  - Create listener port and specify:
    - Connection Factory and Destination for the port
    - Maximum Concurrent Sessions
    - Maximum Concurrent Messages
    - Maximum Retries

MDB Resource configuration

- **At application install time, specify listener port for MDBs**
  - MDBs run on App Servers
  - Listener port defines the destination an MDB listens to



Once the JMS resources have been configured, deploy the MDB, specifying the ListenerPort, with which it is to be associated. Note that the listener port is in turn associated with the connection factory and the destination JNDI name.

A listener port is used to simplify administration of the association between a connection factory, destination, and deployed message-driven bean. This enables deployed message-driven beans associated with the port to retrieve messages from the destination.

In WebSphere 5.0, all of the runtime artifacts associated with Message-driven Beans can be started and stopped:

• Just as Entity and Session Beans, the MDB itself can be started and stopped.

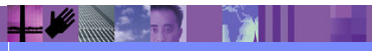
• The ListenerPort can be started and stopped.

• The whole Listener Service can be started and stopped.

In WebSphere 5.0 we have introduced support for the MQSeries JMS CloneSupport feature in the Embedded JMS Provider and External MQ Provider. This feature is for Durable Topic subscriptions, and basically enables sharing of a durable subscription across WebSphere cluster members. This means you can have the same durable subscription on separate cluster members, such that a single publication will only be delivered to one of them, hence a stream of publications get distributed across the cluster members.

Binding J2EE Application Clients

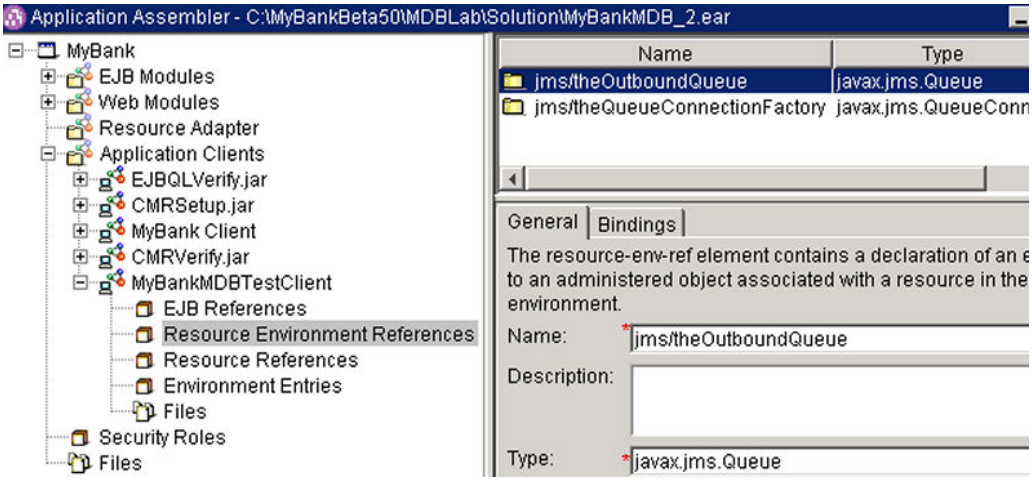
- **Using Application Assembly Tool, bind the JMS destination**
  - Define the reference to the Destination in the client code as a resource-env-ref
  - Specify the bindings for it in terms of the global JNDI name
- **Using Application Assembly Tool, bind the JMS Connection Factory**
  - Define the reference to the Connection Factory in the client code as a resource-env-ref
  - Specify the bindings in terms of the global JNDI name

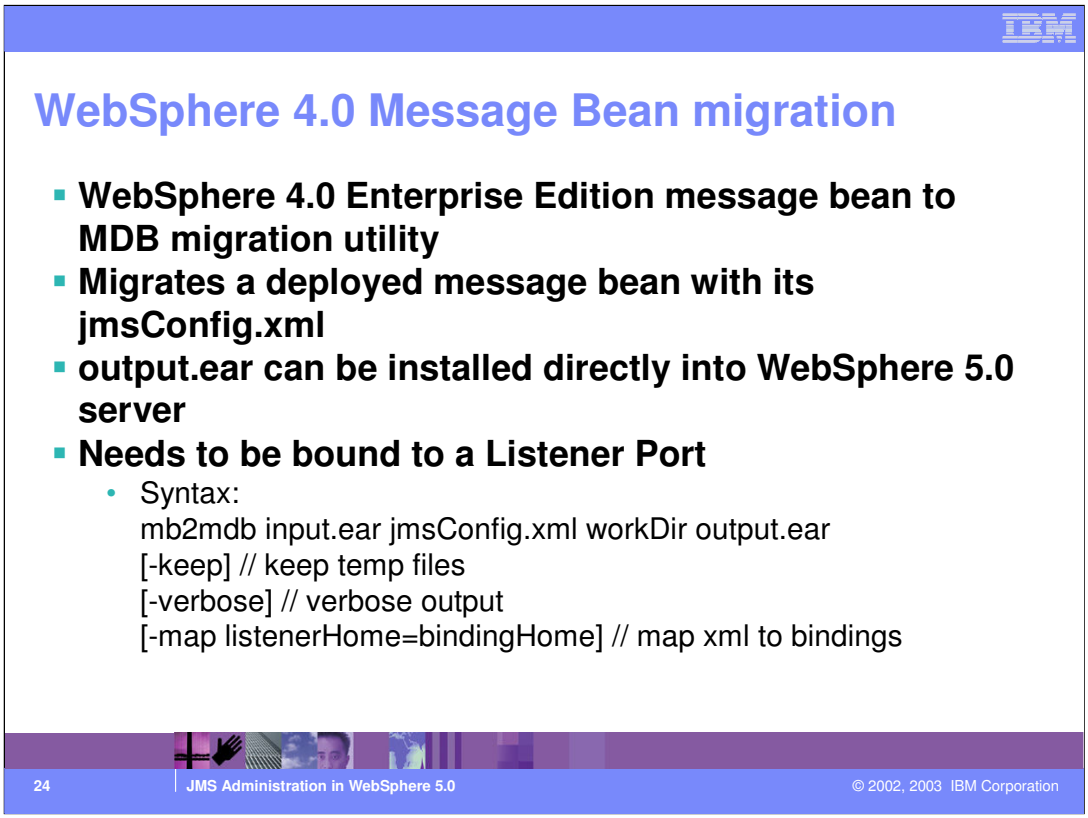


Package the WebSphere J2EE Application Client using the Application Assembly Tool (AAT). The J2EE platform allows the application client code to use nicknames or short names for JMS resources such as destinations and connection factories. These nick names are defined within the client application deployment descriptor as "Resource Environment References". At deployment time, these references are resolved to the JNDI names using bindings. This resolution eliminates changes to the client application code, when the underlying object or resource either changes or moves to a different server.

The Application Assembly Tool can be used to configure the resource-env-ref and the corresponding bindings for the J2EE Application Clients.

Binding J2EE Application Clients





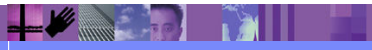
WebSphere 4.0 Enterprise Edition adds support for “Message Beans”. Programmatically, Message Beans look exactly like J2EE 1.3 Message Driven Beans – the idea was to provide support for the MDB programming model ahead of WebSphere 5.0 and full J2EE 1.3 compliance. Like MDBs, using Message Beans you can have a two-phase commit for inbound messages. WebSphere 4.0 Message Beans are configured using an XML configuration file, which is read by the JMS Listener service.

The command line utility, mb2mdb, takes as its input either a deployed MessageBean.jar module or a deployed Enterprise Application (.ear) that contains a message bean, along with the JMS listener configuration XML file that defines the WebSphere 4.0 message beans. The result is a new .jar/.ear module that can then be deployed directly into a WebSphere 5.0 application server.

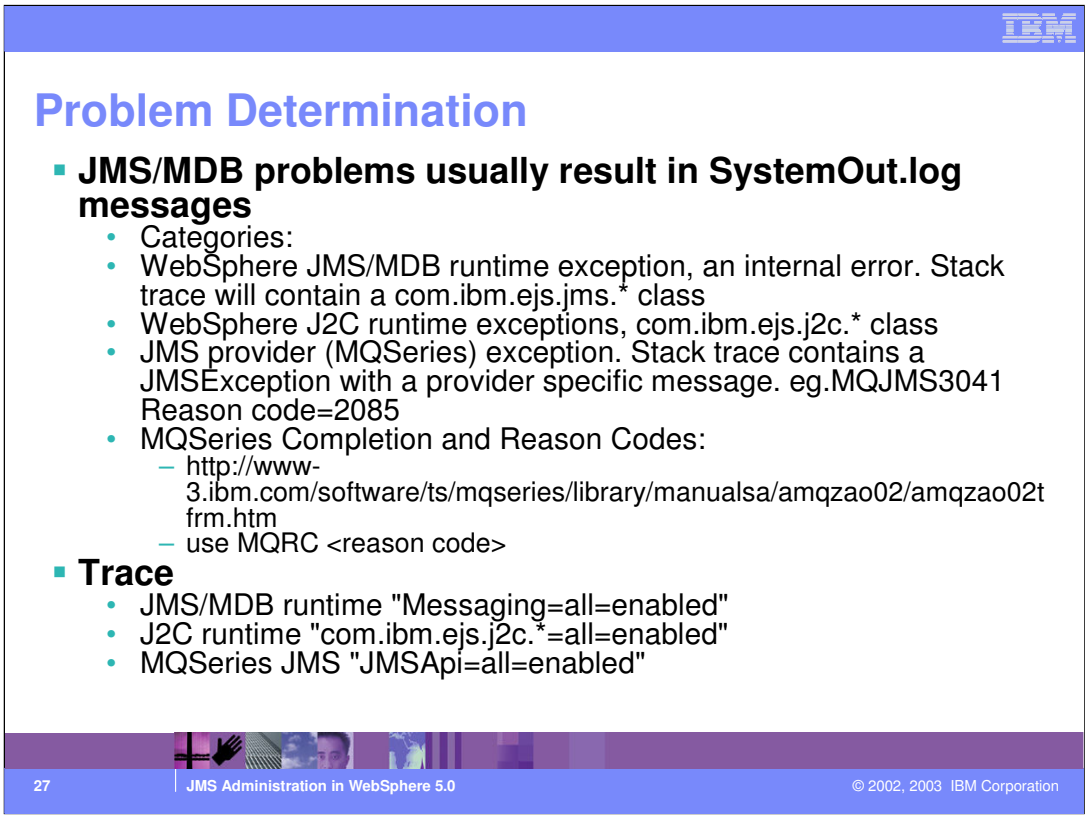
Troubleshooting

Problem with Embedded messaging function

- **Check the was_install/mq_install.log to see whether there was a problem during installation**
- **Failures during the WebSphere Messaging Prerequisite stage usually indicate a shortage of space.**



If you have a problem with the Embedded Messaging function, you will need to first check the mq_install.log to see whether there was a problem during the installation. Failures during the WebSphere Messaging Prerequisite stage usually indicate a shortage of space.

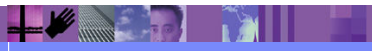


Problems related to Messaging and Message driven beans usually result in SystemOut.log messages. So, if you feel that your problems are JMS related, then inspect the SystemOut.log file. If it is a WebSphere JMS/MDB runtime exception then the stack trace will contain com.ibm.ejs.* class. If it is WebSphere J2C runtime exception then the stack trace will contain com.ibm.ejs.j2c.* class. If the exception is from the JMS Provider, then the stack trace will contain a provider specific message. For example, a message such as MQJMS3041 Reason Code = 2085 indicates that it is an MQ Series exception. For each MQSeries call, a completion code and a reason code are set, either by the queue manager or by an exit routine, to indicate the success or failure of the call. To understand more information about the failure either refer the website :- http://www-3.ibm.com/software/ts/mqseries/library/manualsa/amqzao02/amqzao02tfrm.htm or use the MQRC utility.

If the log files do not provide enough information, then turn on tracing to obtain information to aid in your problem determination. Be aware that tracing creates a large amount of data, hence try to trace the specific component where the problem is occurring.

Common Problems

- **Unable to start MDBListener, cannot open QueueManager:**
  - Reason: JMSServer is not running. Check in Admin Console whether the JMSServer status indicates started
  - In general check if Queue Manager and Broker are open for business in SystemOut.log
- **"No broker response"**
  - Embedded Broker failed to install, check mq_install.log, and createmq.log
  - Broker isn't started! e.g.. using external MA0C broker and haven't done strmqbrk

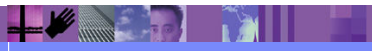


Some common problems are listed in these two foils. If you are unable to start the MDBListener, or cannot open the Queue Manager, then the JMSServer may not be started. Make sure you check that the status indicates started in the Admin Console.

When you run a pub/sub application using the Embedded JMS Provider, and you get a message " No Broker Response", then the Embedded Broker probably failed to install. Check the mq_install.log and createmq.log for errors. If you are using MQSeries' MA0C broker, then the broker may not be started.

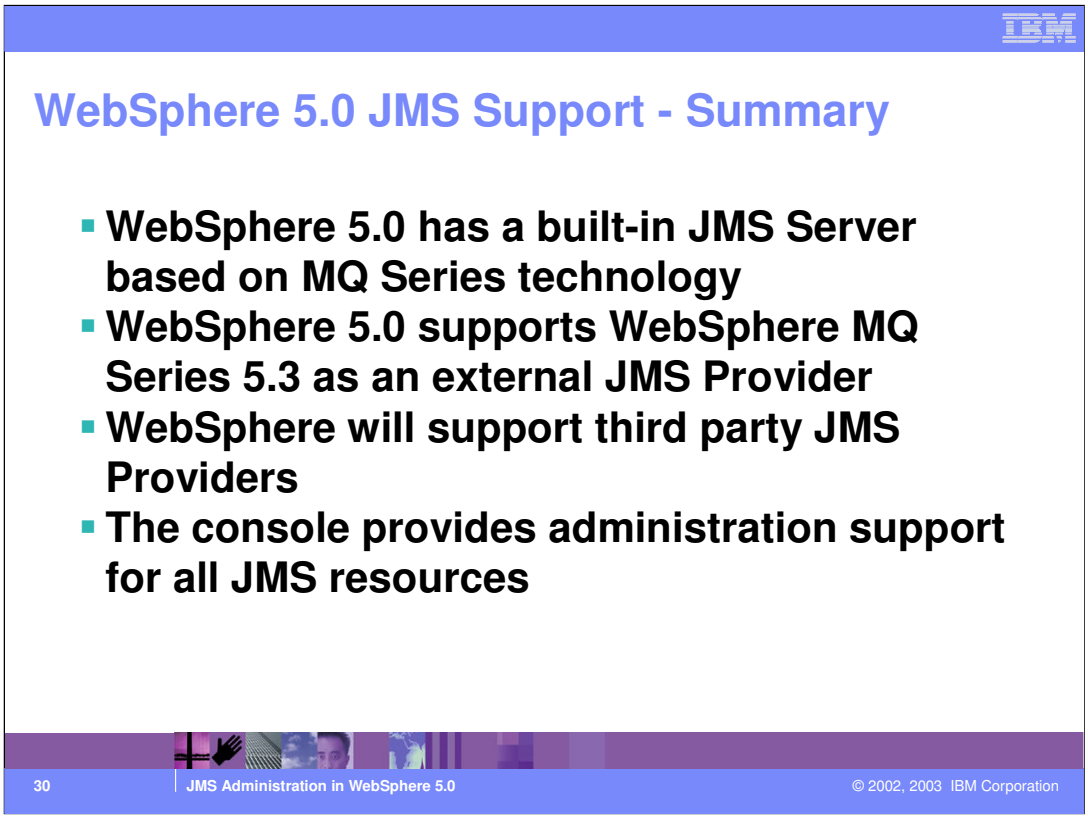
Common Problems

- **Configuration errors:**
  - MDBListener fails to lookup resources, ListenerPort has wrong JMS JNDI names
  - Using Embedded "DIRECT" Broker with transactions, must use "QUEUED" with transactions
- **Security exceptions, MQJMS Invalid security credentials**
  - WebSphere security is enabled and an invalid credential is specified in the QCF/TCF, or specified on createConnection(user,pass)



Configuration errors occur pretty frequently too. For example, if the listener port is associated with wrong connection factory or/and destination JNDI names, then the MDB listener will fail to look up the resources. Also, if you are using the Embedded JMS provider for pub/sub, and you need transactions to be supported make sure you have selected "QUEUED" for the port field. The Port field refers to which of the two ports connections use to connect to the JMS Server. The QUEUED port is for full-function JMS Pub/Sub support, the DIRECT port is for non-persistent, non-transactional, non-durable subscriptions only. Note that Message-driven beans cannot use the direct listener port for Pub/Sub support. Therefore, any topic connection factory configured with Port set to Direct cannot be used with message-driven beans.

You will get Security exceptions if security is enabled and invalid credentials are specified in the queue/topic connection factory or in the application code's createConnection method.



WebSphere 5.0 has a built-in JMS Server based on MQ Series technology:

Runs in a JMS Server

Administered through the easy to use Web Based Administration console

WebSphere 5.0 supports WebSphere MQ Series 5.3 as an external JMS Provider

Quality of service, robustness

WebSphere will support third party JMS Providers

The console provides administration support for all JMS resources

Built-in JMS Provider and external WebSphere MQ provider

Trademarks and Disclaimers