Considerations When Using WebSphere MQ with Java EE Application Servers

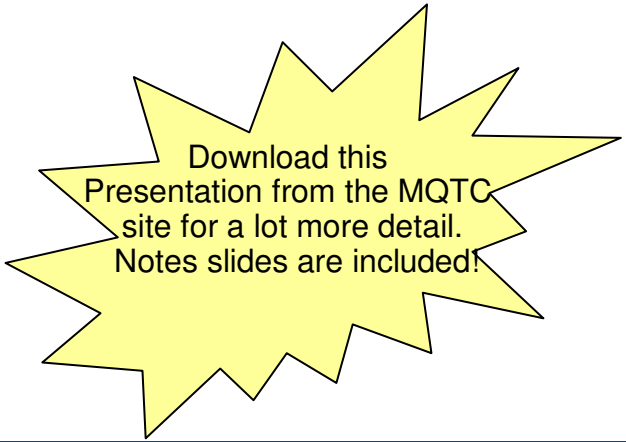
Christopher Frank

IBM WebSphere Connectivity

chrisfra@us.ibm.com

Agenda

- **Why are We Here?**
- **Java Enterprise Edition (JEE) and Messaging**
  - What is JMS?
- **Accessing WMQ from JEE Environments**
  - How to configure
  - WMQ features useful in JEE environments
- **Configuring WMQ and WAS**
  - Useful tuning parameters
  - High Availability and clusters



Why are we here?

- **Some reasons for being here...**

"We currently have applications using WebSphere MQ, including applications running in JEE Application Server environments that need to access the same data"

"We're developing applications to run in JEE environments and we're trying to choose a messaging provider, and are thinking that WebSphere MQ might fit the bill"

...

How do we achieve this?

- **We need a fully compliant Java Enterprise Edition (JEE) Application Server**
  - WebSphere Application Server
  - Others include JBoss, WebLogic, etc
  - Must provide a way to integrate support for application messaging using Java Message Service (**JMS**) providers
    - This can be as either a native or a foreign JMS provider
- **WebSphere MQ is a fully compliant JMS provider**
- **Therefore, JMS is the answer!**

Agenda

- Why are we here?
- Java Enterprise Edition (JEE) and Messaging
  - What is JMS?
- Accessing WMQ from WAS
  - How to configure
  - New WMQ V7 features useful for WAS
- Configuring WMQ and WAS
  - Useful tuning parameters
  - High Availability and clusters

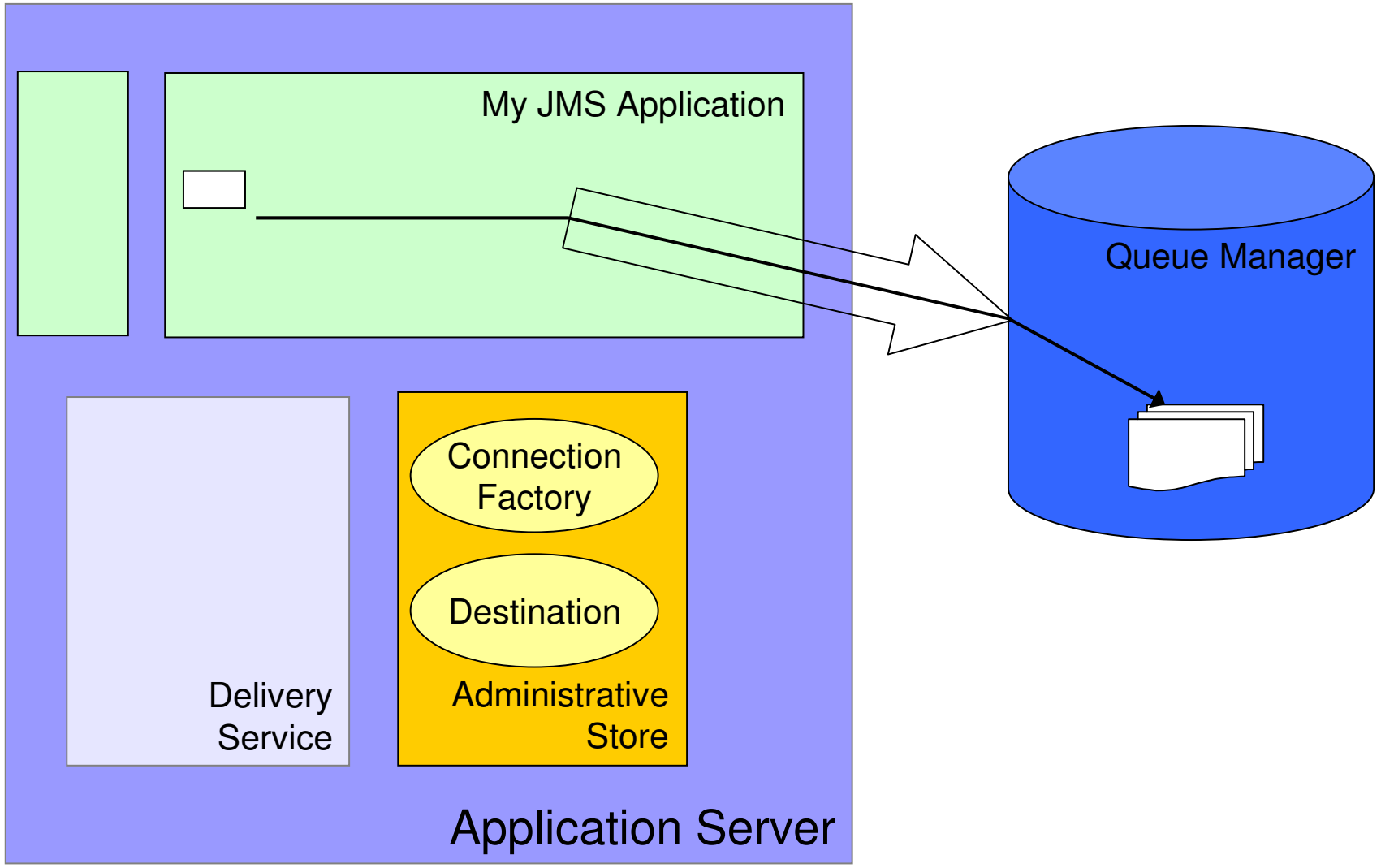
What is JMS?

- **A standardized Java API to allow applications to perform messaging**
  - Applications can be unaware of implementation
  - Not a wire protocol
- **Uses administered objects to define provider and configuration**
  - Can be changed without recompiling application
- **Supports point-to-point messaging and publish/subscribe messaging**

NOTES

- JEE application servers must provide support for JMS
  - JEE 1.4 and later also must support the Java Connector Architecture, which WMQ has shipped for some time
- JMS is a standardised programming interface
  - Allows Java application interactions to be loosely coupled
  - JMS applications can be coded to be ignorant of the underlying JMS provider
    - Can easily change the provider without altering the application

How does a JMS based application work?



JMS application types in JEE

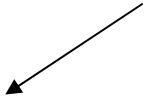
- **Message-driven Beans (MDB)**
  - Special type of EJB that has an “onMessage” Java method to receive messages
  - JEE “Container” deals with getting the message, transactions, and running the application.
  - Application only needs to process the message data
  - Recommended way of getting messages **into** JEE servers
- **Enterprise Java Bean (EJB) and Servlets/JSPs**
  - Application needs to create connection and send/receive message
  - Container provides connection pooling and transaction management

NOTES

- Provider specific configuration, such as how to connect to a messaging provider, is contained within *JMS resources* held in **JNDI**, defined at application deployment time
- JMS is not a transport protocol
  - No standardised ability to couple different JMS providers
- JMS Supports two messaging models
  - **Point-to-point** – using queues
  - **Publish/subscribe** – using topics

Message-driven beans

Code driven when a message is available



```
public void onMessage(Message message) {
    try {
     if (message instanceof TextMessage) {
           TextMessage textMsg = (TextMessage) message;
           System.out.println("Message text is " +
  textMsg.getText());
       }
    } catch (JMSException ex) {
       System.out.println("JMSException occurred : " +
  ex);
    }
 }
```

Enterprise Java Beans (EJB)

```
@Resource()
private ConnectionFactory cf;
@Resource()
private Queue d;

public void receiveMessage() {
    try  {
      Connection conn = cf.createConnection();
      conn.start();
      Session sess = conn.createSession(true,Session.AUTO_ACKNOWLEDGE);
      MessageConsumer consumer = sess.createConsumer(d);
      Message msg = consumer.receive(30000);
      if (msg instanceof TextMessage) {
        System.out.println("Message received:" + ((TextMessage)
  msg).getText());
      }
     conn.close();
    } catch (Exception ex) {
    System.out.println("Exception : " + ex);
    }
  }
```

Define EJB 3 Resource references that will be initialised at runtime

More code required to get a message compared to an MDB, and less efficient

What to use? Base Java MQI or JMS?

- **Both are Java APIs. So does it Matter? Yes!**
- **Using the Base Java API is *not recommended* in JEE servers. Why?**
- **There are significant restrictions on using the MQ Base Java API within JEE:**
  - **JTA Transactions restrictions**
    - An application under JTA control can use the MQ Base Java API, but any work performed through these calls will be *outside* the JTA transaction scope
    - Any rollback of the JTA transaction will not result in a rollback of any sent or received messages
  - **Restrictions on Thread Creation**
    - Certain JEE applications must not spawn new threads; work should be done under the main application threads only
      - The MQ Base Java API internally spawns threads for various operations.
      - These threads will cause the application to be non-compliant with the container specification
  - **Security restrictions**
    - Security policies implemented by an application server may block certain operations of the MQ Base Java API
      - e.g. creating and operating new threads of control (as described above)
    - Thus the MQ Base Java API is not supported in environments where Java 2 Security is enabled.
- **Using the JMS API will avoid all these complications!**

And if that's Not Enough…

- **A key benefit of running in a JEE environment is *Application Isolation***
  - Changes to one application should not adversely affect others in the same JEE server
- **The design and implementation of the MQ Base Java API predate JEE**
  - It is possible to use it in a manner contrary to application isolation.
    - For example, by using static (JVM process-wide) settings within the MQEnvironment class
  - Modifying MQ Environment properties for one application can also impact other applications
    - Avoid using properties configured in the process-wide MQEnvironment class
  - The MQEnvironment class also introduces a number of static methods
    - These act globally on all applications using the MQ Base Java API within the same JVM process
    - There is no way to override this behavior for particular applications
- **Application owners must be aware that invoking such methods will affect all applications running in the same JEE environment**
- **The MQ Base Java Classes do not support Auto Client Reconnection**

NOTES

- WMQ provides two Java APIs: The MQ Base Java API (a Java version of the MQI), and a JMS API implementation
- The question of which one to use in JEE environments is one that comes up frequently. The answer is that the MQ JMS API is best suited for use in JEE environments. Why?
- The JMS and JEE specifications have been developed together specifically to allow JMS-based messaging to fully interact with a JEE environment, and the MQ JMS API has been designed and implemented with this in mind.
- The MQ Base Java API was designed primarily for JSE environments, and is not well integrated with J2EE/JEE, so IBM does not recommend the use of the MQ Base Java API in JEE environments. The WebSphere MQ base Java API has restrictions when being used within a JEE environment, which must be taken into account when designing, implementing and managing an MQ Base Java application that will be run inside of a JEE environment.
- **Restrictions on using the MQ Base Java API within JEE**
  - JTA Transactions restrictions
    The only supported transaction manager for applications using the WebSphere MQ base Java API is WebSphere MQ itself. While an application under JTA control can make calls to the MQ base Java API, any work performed through these calls will not be controlled by the JTA units of work. They will instead form local units of work separate from those managed by the application server through the JTA interfaces. In particular, any rollback of the JTA transaction will not result in a rollback of any sent or received messages. This restriction applies equally to application/bean managed transactions as to container managed transactions. In order to perform messaging work directly with MQ inside application server-coordinated transactions, the MQ JMS API must be used instead.
  - Thread Creation
    The MQ Base Java API internally spawns threads for various operations. Certain JEE applications must not spawn new threads; instead all work should be performed on the main application threads managed by the application server. When applications use the MQ Base Java API the application server may not be able to distinguish between application code and the MQ Base Java API code, so these threads described above will cause the application to be non-compliant with the container specification. The MQ JMS API does not break these JEE specifications and so should be used instead.
  - Security restrictions
    Security policies implemented by an application server may prevent certain operations that are undertaken by the WebSphere MQ base Java API, such as creating and operating new threads of control (as described above). To prevent problems with thread management, the use of the MQ Base Java API is not supported in environments where Java 2 Security is enabled.

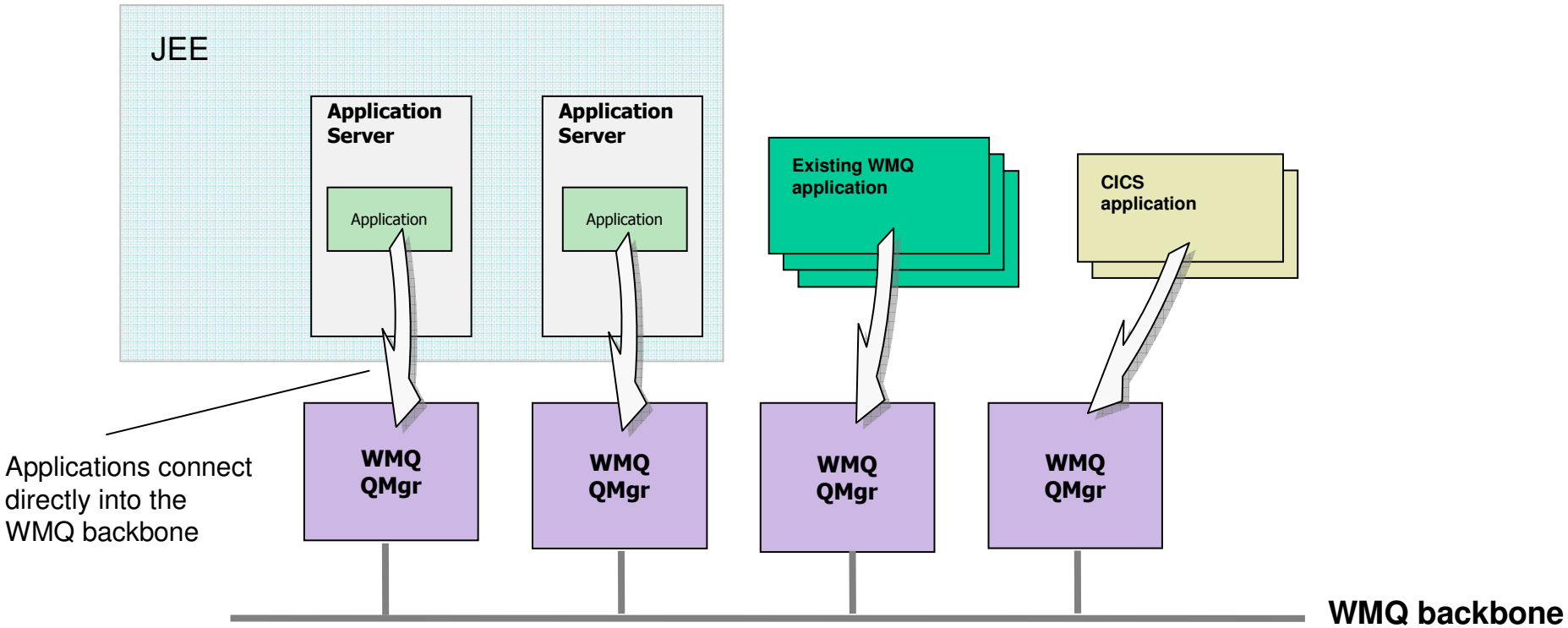
NOTES

- **Additional considerations for using the MQ Base Java API in JEE**
  - One of the intended benefits of running applications within a J2EE/JEE environment is that of application isolation. Changes to one application should not adversely affect others running in the same application server. The design and implementation of the MQ Base Java API predate the J2EE/JEE environment, and can be used in a manner which does not readily support this concept of application isolation. For example:
  - The use of static (JVM process-wide) settings within the MQEnvironment class, such as:
    - the userid and password to be used for connection identification and authentication
    - the hostname, port and channel used for client connections
    - SSL configuration for secured client connections
  - Modifying any of the MQ Environment properties for the benefit of one application will also impact other applications making use of the same properties. When running in a multi-application environment such as J2EE or JEE, it is strongly recommended that each application uses its own distinct configuration through the creation of MQQueueManager objects with a specific set of properties, rather than defaulting to the properties configured in the process-wide MQEnvironment class.
  - Similarly, the MQEnvironment class introduces a number of static methods which act globally on all applications using the WebSphere MQ base Java API within the same JVM process, and there is no way to override this behavior for particular applications. Examples include:
    - the configuration of SSL properties, such as the location of the key store
    - the configuration of client channel exits
    - enabling or disabling diagnostic tracing
    - managing the default connection pool used to optimize the use of connections to Queue Managers
  - Application owners must be aware that invoking such methods will affect all applications running in the same JEE environment.
  - Because the MQ JMS API depends to some degree on the MQ Base Java API for its internal operations, mixing MQ JMS API and MQ Base Java API applications within the same application server can lead to the same application isolation issues. This particularly applies to the SSL configuration, connection pooling and diagnostic tracing managed through the MQEnvironment class.
  - SSL configuration for applications using the MQ Base Java API is performed by way of the MQEnvironment class and MQQueueManager object properties, and is not integrated with the managed security configuration of the application server itself. Application owners must ensure that they perform the necessary MQ Base Java API configuration to provide their required level of security, and not use the application server configuration.
  - The diagnostic facilities provided by the MQ Base Java API are not integrated with those of the application server.

Agenda

- Why are we here?
- Java Enterprise Edition (JEE) and Messaging
  - What is JMS?
- **Accessing WMQ from JEE Environments**
  - **How to configure**
  - WMQ features useful in JEE Environments
- Configuring WMQ and WAS
  - Useful tuning parameters
  - High Availability and clusters

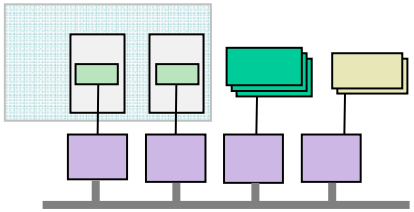
Option 1 - Using WMQ Directly as the JMS Provider



Using WMQ Directly as the JMS Provider

Pros

- The most **direct** route to WMQ queues
  - Best performing way to get messages on and off a WMQ queue
  - Least moving parts
    - Least things to configure
- Fewest places where things can go wrong
  - A single messaging system to maintain rather than multiple

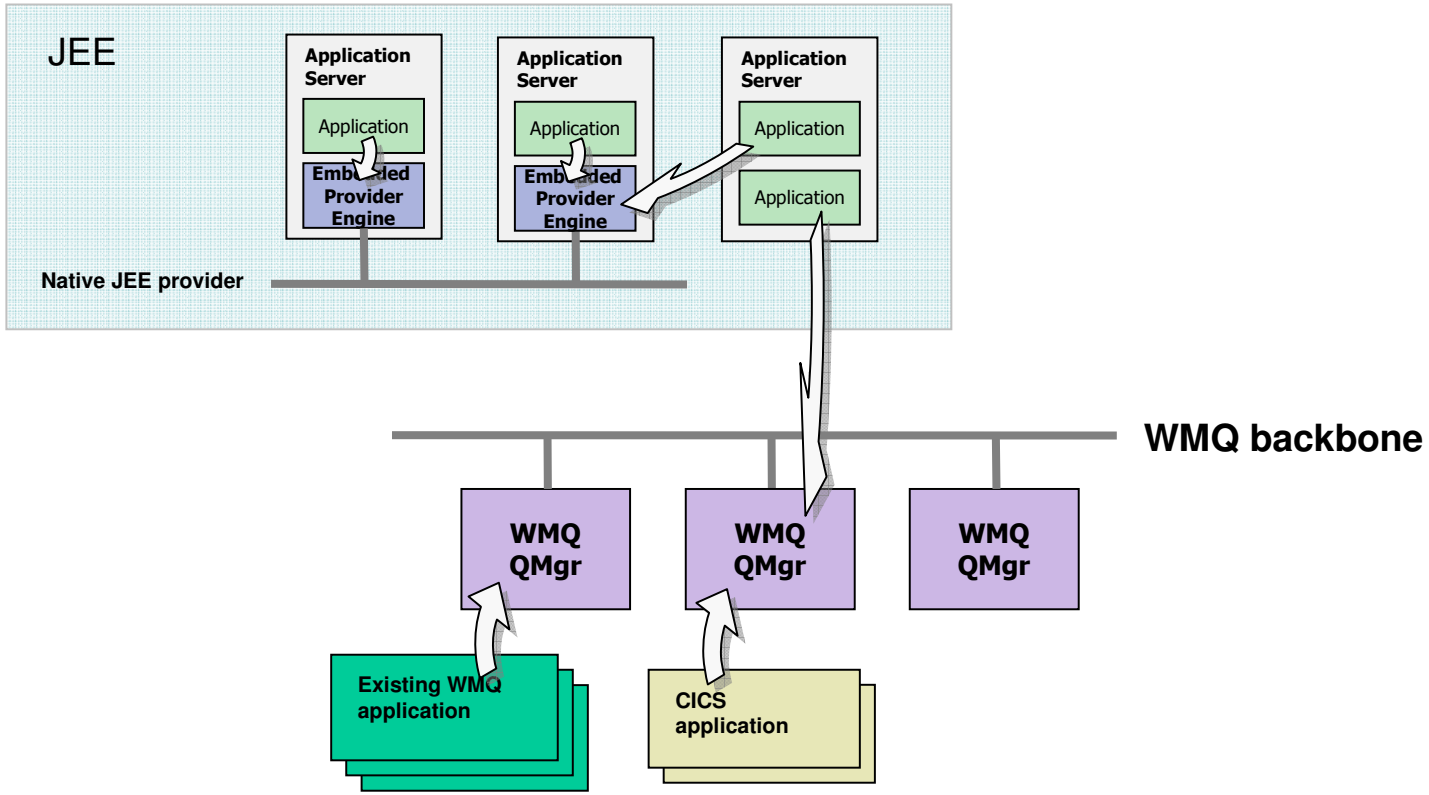


Cons

- Application deployment has some dependency on knowledge of WMQ topology
  - e.g. the WMQ queue location is required when connecting applications
  - Can be managed using Administered Objects
- Applications dependent on two systems being simultaneously available
- Different approaches to implementing HA can be complex to exploit

Option 2 - Bridging embedded JMS provider with WMQ

Native provider (e.g. SIBus in WAS) for Java-to-Java application messaging, WMQ for Java-to-'*other*' messaging



Bridging a JEE Embedded Provider to WMQ

Pros

- JEE Application deployment needs no knowledge of WMQ topology
  - Can be configured using facilities provided with the Application Server
- JMS provider may run in the same process as the application server
  - Better performing in *some* scenarios
  - Embedded provider removes dependency on two systems being simultaneously available
- Can participate in the JEE Server's HA mechanism

Cons

- **indirect** route to WMQ queues
  - Required a "bridge" to/from WebSphere MQ
    - May be provided, but not always (if not, you get to create one!)
    - Varying degrees of robustness
- Most moving parts
  - JEE Server + Bridge + MQ
    - More things to configure, more places where things can go wrong
  - Multiple messaging systems to maintain; wire protocol conversion adds overhead

Notes

N

O

T

E

S

- Using a single provider for Java/JEE and non-Java messaging has many advantages. As this is the most direct way to access MQ queues and topics, it provides the best performance. When multiple JMS providers are used, it is necessary to translate message wire formats between WMQ and the other provider(s) wire formats, which incurs overhead.
- A single provider is also easier to manage, as there are fewer moving parts than there would be if two (or more!) JMS providers were used and bridged together.
- That's not to say there are no benefits to using an embedded messaging provider. For Java-to-Java messaging, the necessity to translate between wire formats may be removed. Also, because by definition an embedded messaging provider runs in-process with the JVM, performance can be better for Java-to-Java applications. And as the provider is in-process, the likelihood of the application being available when the provider is not (as could occur when the provider is separate) is greatly reduced.
- So there are situations where a hybrid approach can be of benefit. But the advantages and disadvantages need to be weighed.

Using WMQ as the messaging provider in WAS

- **JEE application servers work differently when connecting to "foreign providers"**
- **WebSphere Application Server provides very specific benefits to using WMQ**
  - WMQ components are pre-deployed and ready to go
- **Fully integrated into WAS administrative tools**
  - Does not need to be installed as a "Foreign Provider"
  - Can even do some WMQ administration directly from WAS
- **WAS can use WMQ JCA Resource Adapter**
  - WebSphere MQ provided as a single unit "Resource Adapter"
  - WMQ Resource Adapter shipped as a component of WAS
  - Updates applied using WAS process, not WMQ
- **Access to WMQ via TCP or shared memory ("bindings")**
- **Supports connections to variety of WMQ queue manager versions**

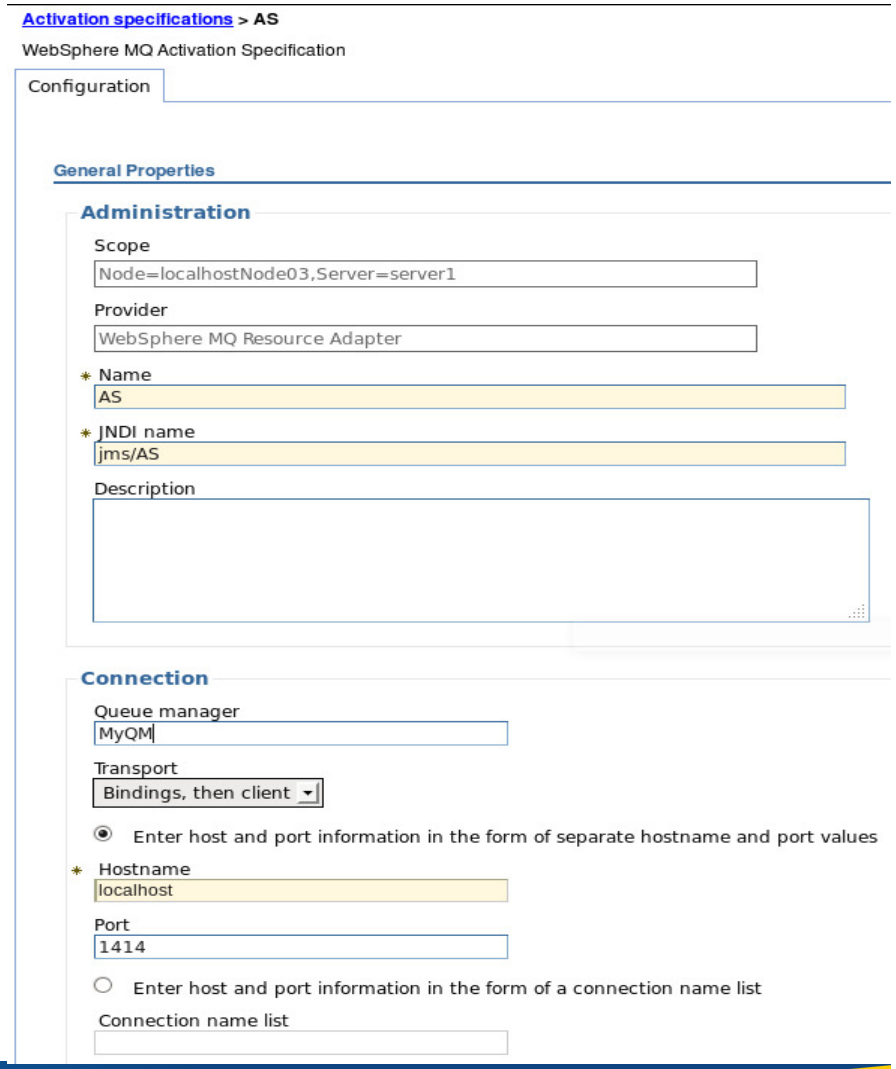
NOTES

- WMQ support is built directly into WAS
  - WMQ already deployed to WAS, no need to deploy WMQ
  - When creating a JMS resource, you are asked to choose between “default” messaging and WMQ messaging.
  - Maintenance of the WMQ Resource Adapter (RA) is through WAS service channels, such as WAS interim fixes/fix packs
- WMQ JMS code can connect to any version of WMQ queue manager
  - No tight coupling between WMQ JMS client version and QM version.
  - WMQ JMS version in use detailed here: http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg21248089
- The WMQ transport can be either TCP, or shared memory/bindings mode
  - Bindings mode available when queue manager and application server on the same machine
    - Uses JNI interface, requires native library to be loaded into the WMQ Resource Adapter.
    - Need to configure the native library using the RA configuration panel
- JCA Interface, well defined interface for application servers to access Enterprise Information Systems.
  - Same WMQ interface used in other JEE application servers
  - Same connector interface as used by SIB and other providers in WAS

Configuring for WMQ : Activation Specifications

Resources > JMS > Activation Specifications > [New]

- **Activation Specs are the standardised method for delivering messages to a Message Driven Bean (MDB)**
- **The WebSphere MQ messaging provider includes support for Activation Specs**
  - Incorporated into WAS since V7
  - Listener Port functionality is stabilized.
- **Activation Specs combine the configuration of connectivity, the JMS destination to be processed and the runtime characteristics of the MDB itself**
- **JEE product may support testing during creation to verify they work.**
- **With WAS, Activation Specs can be defined at all configuration scopes**
  - As with ConnectionFactories and Destinations.



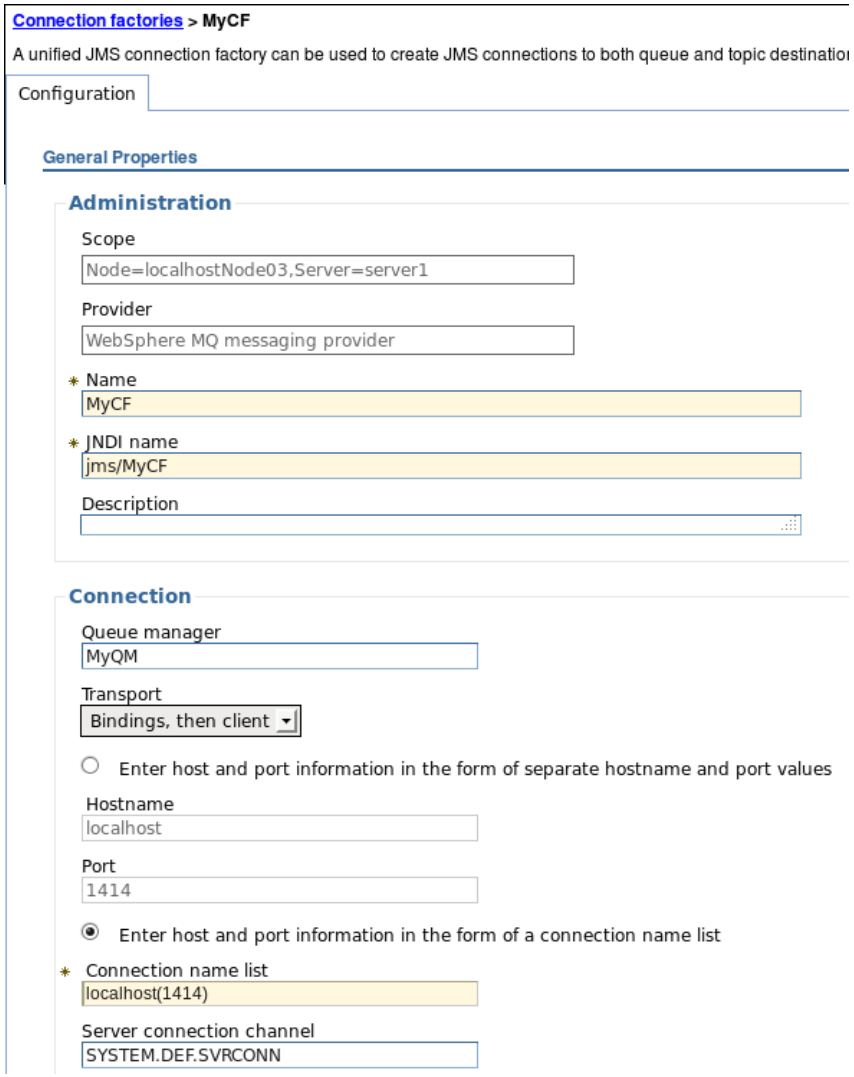
NOTES

- This screenshot shows an example of configuring Activation Specs in WebSphere Application Server. Other application servers provide their own techniques, but what can be configured is the same.
- Beginning with WAS V7, the WebSphere MQ V7 JCA 1.5 Resource Adapter was introduced, enabling MDB's to use Activation Specs to connect to queue managers rather than using listener ports.
- Activation specs are the strategic direction for delivering JMS messages to MDBs Listener Ports were a proprietary approach, and while still available, as functionally stabilised.
- Activation Specs are easier to administer as they are definable at any scope whereas Listener Ports are defined on a per server basis
- With WAS, a wizard is included which assists in the migration of existing Listener Ports to Activation Specs. Note that this only creates a new Activation Spec with the same configuration as the Listener Port, it does not modify application deployments to use the newly created Activation Spec.

Configuring for WMQ : Connection Factories

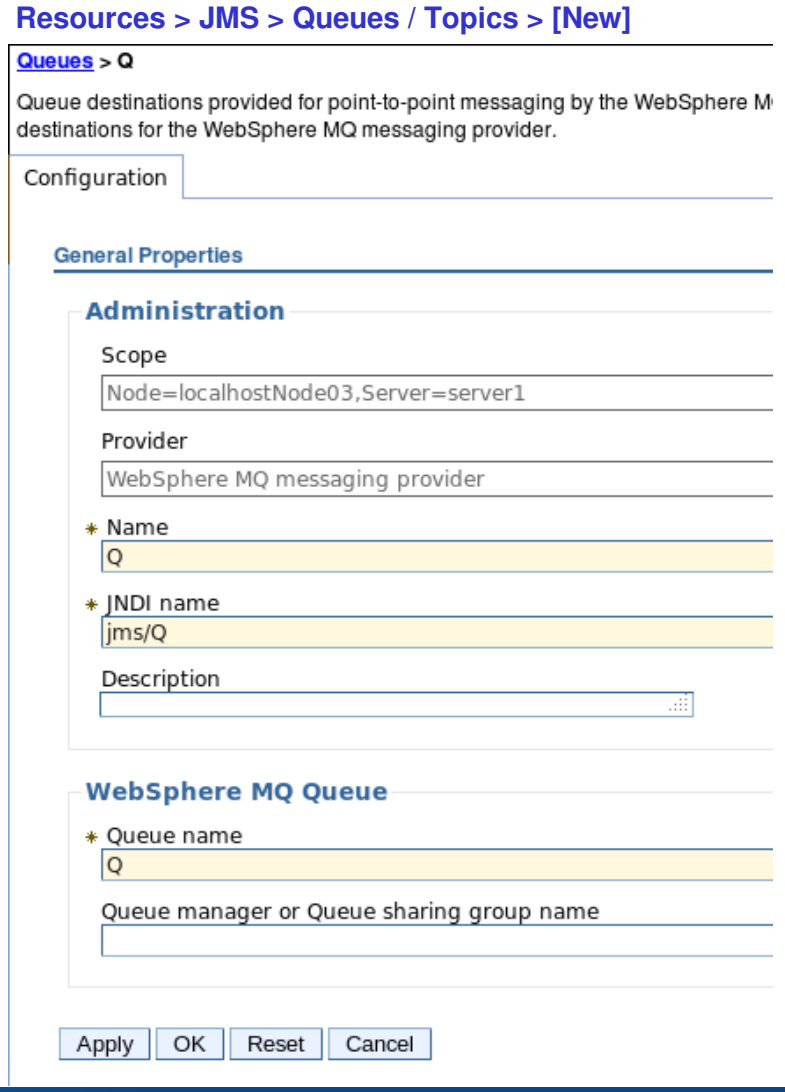
- **Specify how an application connects to a WMQ queue manager**
- **Typically requires:**
  - Queue manager name
  - Hostname and port
  - Channel name
  - Other parameters such as:
    - Transport type (client or bindings)
    - Use of SSL
- **Or, can use WMQ** client channel definition table **(CCDT) URL**

Resources > JMS > Connection factories > [New]



Configuring for WMQ : Destinations

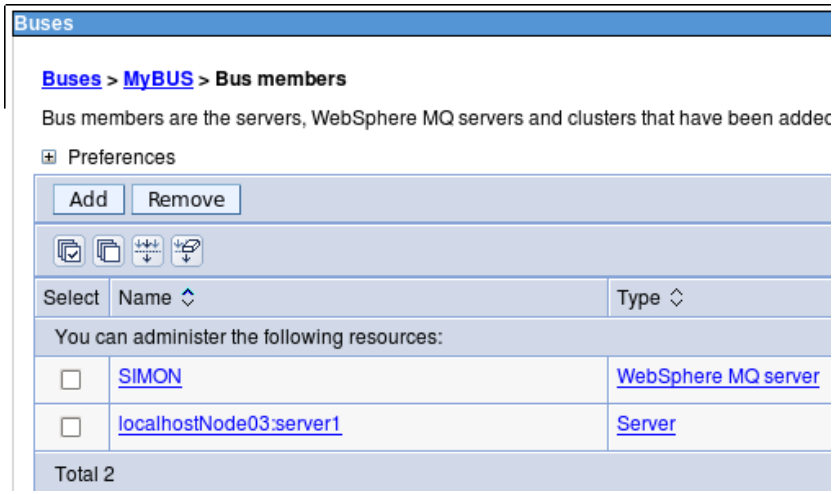
- **Defines references to the resources in WMQ that a JMS application will use**
  - The WMQ resources must be created using WMQ administration
- Queues
  - Identifies the actual queue in WMQ
    - Can be used to set properties such as persistence, priority, etc.
- Topics
  - Defines the WMQ publish/subscribe destination



Configuration for WAS SIBus : MQ Link

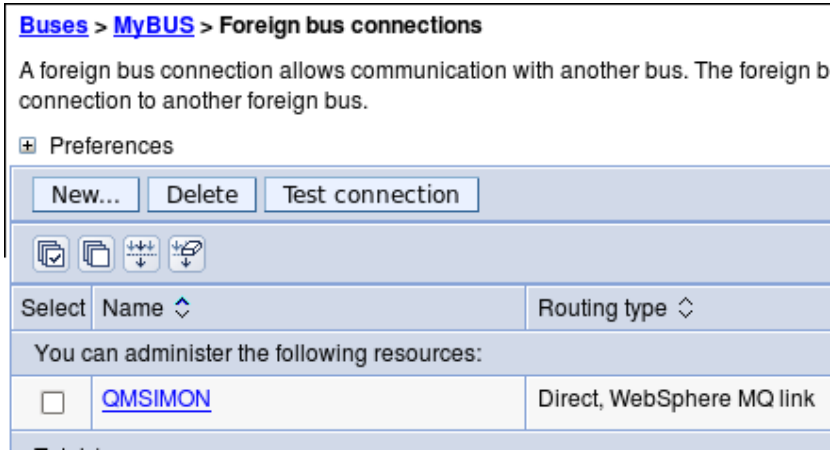
Queue manager as a bus member

- WMQ Queues linked as bus destinations
- No asynchronous delivery, if QM is unavailable, destinations cannot be used
- Can easily read and write messages to queues



Queue manager as a foreign bus

- WMQ Queues linked as foreign destinations
- Messages stored on SIBus if QM is unavailable
- Cannot read messages directly from WMQ



NOTES

- For a hybrid implementation, with WMQ and WAS, there is specific enablement for creating links between WMQ and WAS (SIBus)
- For a Queue Manager to be represented as a ***bus member***
  - First define the queue manager as a WebSphere MQ server in the “Servers” section
  - In the Service Integration → Buses → *BusName* → Bus members view, Add the queue manager
  - Add the destinations you want to make available on the bus, for example in the Service Integration → Buses → *BusName* → Destinations view, add a new queue destination.
    - Specify the appropriate WebSphere MQ bus member, and use the queue filter to locate the existing WMQ queue
  - Create JMS connection factories and JMS destinations, and then use as normal
  - Can read and write messages directly from WMQ Queues, with confidence that messages arrive at their destination.

NOTES

- For a Queue Manager to be represented as a ***foreign bus***
  - In the Service Integration → Buses → *BusName* → Foreign Bus connections view, click New, and create a Direct connection to a WebSphere MQ queue manager
    - Define unique Foreign Bus name and MQLink name
    - Specify sender/receiver channels on WMQ to use (and ensure they exist on WMQ!)
    - The WMQ port used by WAS is defined by the port value SIB_MQ_ENDPOINT_ADDRESS (e.g.5558)
  - Create JMS destinations that reference the foreign bus destination (Connection Factories still connect to the local bus)
  - Uses WMQ Sender/receiver channels, and so messages are sent to existing queues. Return path uses remote queue definitions/transmission queues. Undelivered messages sent to dead-letter queues.
- QM as a bus member is strongest for incoming messages, QM as a foreign bus is strongest for outgoing (it can queue up messages when the QM is unavailable).
  - Can use a combination of methods.

Agenda

- Why are we here?
- Java Enterprise Edition (JEE) and Messaging
  - What is JMS?
- Accessing WMQ from JEE Environments
  - How to configure
  - **WMQ features useful in JEE Environments**
- Configuring WMQ and WAS
  - Useful tuning parameters
  - High Availability and clusters

Working with Native MQ Messages

- **There are situations when working with native MQ messages is desirable**
  - JMS applications interacting with non-JMS applications are an example
  - This is the main reason customers give for wanting to use the MQ Base Java API
- **MQ JMS does provide for working with native MQ message**
  - By making use of IBM-specific Destination properties MDREAD, MDWRITE and MSGBODY
- **Allows full access to the MQMD header values as well as message payload**
  - Useful for sending or receiving messages from MQ applications that use specific header properties.
- **JMS message property names begin with**
  - "JMS_IBM_MQMD…"
- **A supported alternative to using the MQ Base Java classes**
- **Provides access to proprietary MQ features outside the JMS Specification**
- **Enables use of MQ JMS features not in the MQ Base Java classes**

N

O

T

E

S

- There are situations where it may be desirable to work with MQ messages in their native format, as opposed to the MQ JMS format.
- Three destination properties available to facilitate this:

- **MDREAD**
  This property determines whether a JMS application can extract the values of vendor-specific MQMD fields that are set on messages sent to or received messages from this destination.
  - Comments: If set to NO (the default), when sending messages the JMS_IBM_MQMD* vendor-specific properties on a sent message are not updated to reflect the updated field values in the MQMD, and when receiving messages, none of the JMS_IBM_MQMD* properties are available on a received message, even if the sender had set some or all of them. If set to YES, when sending messages all of the JMS_IBM_MQMD* vendor-specific properties on a sent message are updated to reflect the updated field values in the MQMD, including the properties that the sender did not set explicitly. When receiving messages, all of the JMS_IBM_MQMD* properties are available on a received message, including the properties that the sender did not set explicitly.

- **MDWRITE**
  This property determines whether a JMS application can set the values of vendor-specific MQMD fields sent to this destination prior to the send operation.
  - Comments: If set to NO, all JMS_IBM_MQMD* vendor-specific properties are ignored and their values are not copied into the underlying MQMD structure. If set to YES, JMS_IBM_MQMD* vendor-specific properties are accepted and their values are copied into the underlying MQMD structure.

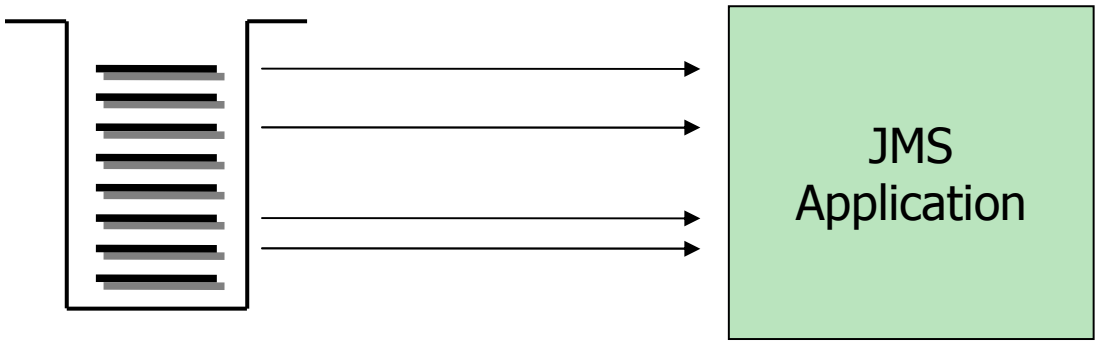
- **MSGBODY**
  Determines in what form the message payload is returned to the JMS application, and whether access is provided to underlying MQ headers, including the MQRFH2.
  - Comments: This option allows a JMS application to access the complete WMQ message data. What can be read and/or manipulated depends on the setting of the MDREAD and MDWRITE properties. To access all the data, the message must be a JMSBytesMessage. How the message is presented depends on the option specified:
    - WMQ_MESSAGE_BODY_MQ allows access to the complete WMQ message data, including the payload, any MQRFH2 header and any other WMQ headers that might be present.
    - WMQ_MESSAGE_BODY_JMS allows access to the message body alone (no MQRFH2 header) with the properties of the JMSBytesMessage extracted from properties set in the RFH2.
    - WMQ_MESSAGE_BODY_UNSPECIFIED is the same as setting WMQ_MESSAGE_BODY_JMS.

WMQ functional areas of interest to JMS

- Message Properties + Selectors
- Asynchronous Consumption of messages
- Simplified Browse + Co-operative Browse
- Asynchronous Put Response
- Read-ahead of messages
- Connection changes
- Multi-instance Queue Managers

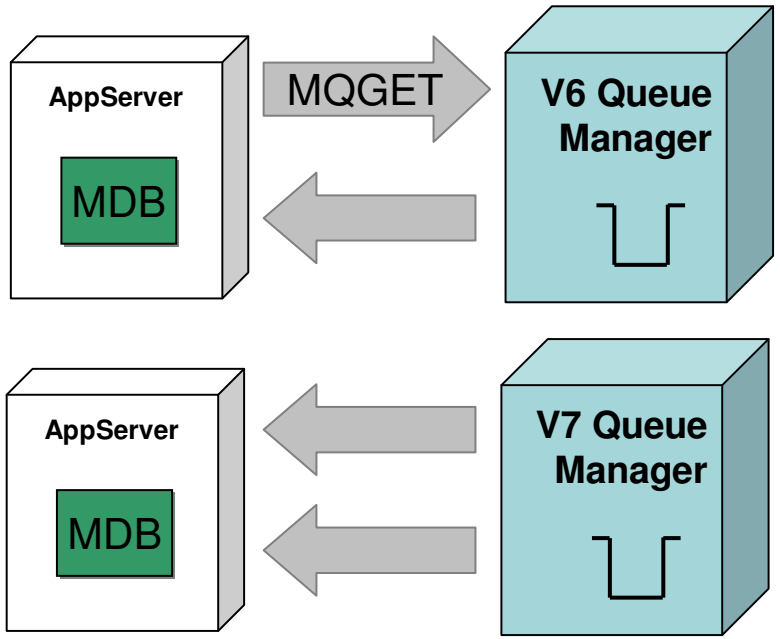
Message properties and Selectors

- **JMS Selectors allow messages from a destination to be filtered**
  - Filter on system or user message properties
- **WMQ queue managers (V7+) understand JMS message properties**
- **Queue manager evaluates JMS selector**
  - Only transmits matching messages to the client
  - Efficient delivery of messages



Asynchronous Consume

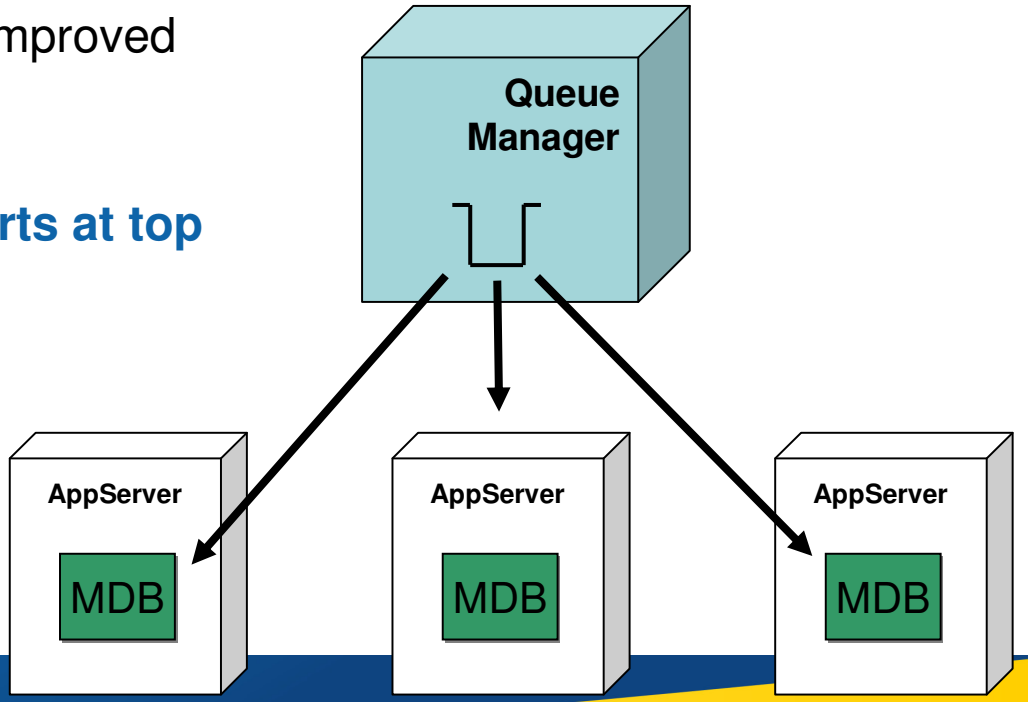
- JMS has had concept of asynchronous consumer since inception
  - MessageListener / onMessage code
- Event driven processing 'natural' for the Java Environment
- WMQ V7 introduced ability of queue manager to drive consumer when suitable message available
  - Avoids need for 'polling thread'
  - More efficient message delivery, especially with selectors.



- With proper support in WMQ – JMS implementation simplified
- Less client CPU overhead of this busy thread

Asynch Consume and Browse-with-mark

- **Multiple application servers will contend for messages when processing the same queue.**
  - Can result in failed message delivery and wasted resources
- **Browse-with-mark allows the application server to hide messages it has browsed and intends to process**
  - No contention for messages, and improved throughput.
- **The MDB browse thread always starts at top**
  - Message priority honored



N

O

T

E

S

- Prior to MQ V7, only MQ on z/OS supported the ‘Browse-Mark’ feature
  - Now all MQ V7 and V8 queue managers have browse mark
- Let a browser mark messages when it had browsed them
- Other browsers can request to not see marked messages
  - Messages marked for a period of time defined on the queue manager
  - Browsers that do not opt to hide marked messages can see all messages
    - V6 style JMS clients can still remove messages and cause message contention
- This lets asynchronous message delivery be more efficient
- Less chance of contention when scaled up to multiple JVMs
  - Contention can still occur if application servers take a long time (> message browse mark interval) time to process message.
  - For WAS, JMSCC0108 error reports this condition in SystemOut.log
    - Indication that server needs to be tuned, e.g. increase MDB sessions, number of MDB processing threads, or decrease MDB processing time.
- Reduction in failed MQGETs and rolled back transactional work in when servicing MDBs gives significant performance benefit

Agenda

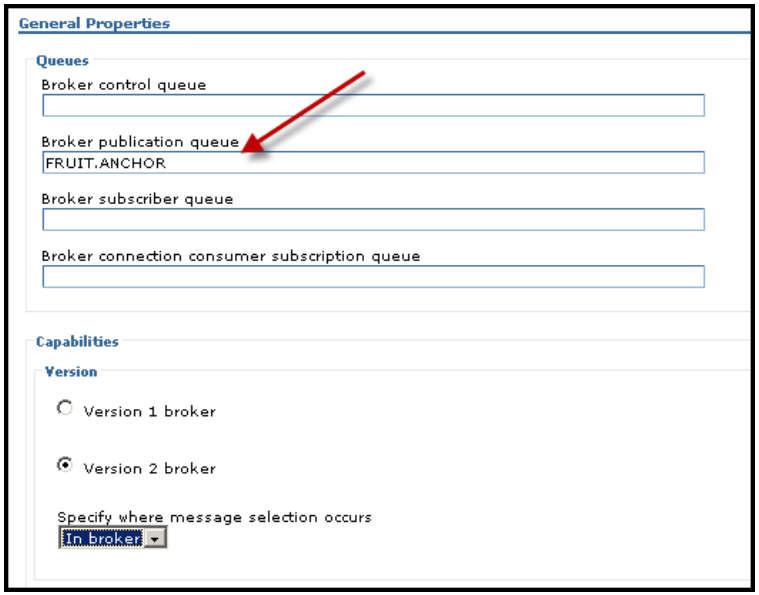
- Why are we here?
- Java Enterprise Edition (JEE) and Messaging
  - What is JMS?
- Accessing WMQ from JEE Environments
  - How to configure
  - WMQ features useful in JEE Environments
- **Configuring WMQ and WAS**
  - **Useful tuning parameters**
  - High Availability and clusters

WMQ Connection Factory Properties:
APPLICATIONNAME

- **Use to assign a name to Java applications to identify what is using this Factory**
  - Associates an application identity with this connection to the queue manager.
- **This application name is shown by –**
  - The DISPLAY CONN MQSC/PCF command (where the field is called APPLTAG)
  - The MQExplorer Application Connections display (where the field is called App name).
- **Enables Java client applications be identified with a more meaningful name than the default "WebSphere MQ Client for Java"**
  - Very useful when a number of different applications are running in a JEE server
- **There are some restrictions to be mindful of:**
  - Works only in Client mode, as there is no corresponding QM support for bindings
  - Works only with Distributed queue managers, as there is no corresponding QM support on zOS
  - The name is restricted to a maximum length of 28 bytes (as that is the queue manager limit)

WMQ Connection Factory Properties:
BROKERPUBQ

- **Used to designate a "publication point" in the pub/sub topic tree**
  - Enables JMS applications to take advantage of MQ pub/sub "topic tree isolation"
  - Allows partitioning of topic tree into multiple subtrees
- **In pre-V7 MQ, this property identified the name of the pub/sub stream queue to use**
  - This would also be the use if the connection is running in "migration mode"
- **In V7 and later, the value for this "queue" can be a Topic Object name**
  - *Not* a topic string
- **This point will serve as the root for this isolated topic subtree**
  - Any topic string used will then be relative to this position in the topic tree

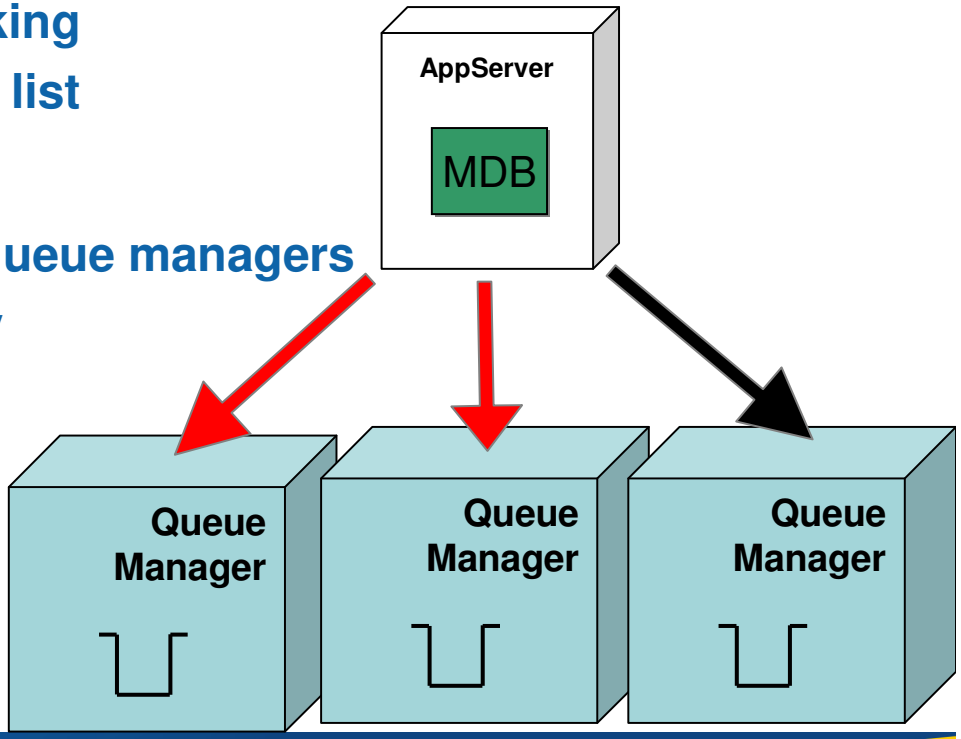


WMQ Connection Factory Properties:
PROVIDERVERSION

- **The version of WMQ that the queue manager this Factory is pointing to is running.**
  - The V7/V8 client is optimized to connect to V7 and above queue managers
  - Connecting in this mode (called “normal mode”) allows V7+ functionality, such as:
    - Asynchronous message delivery
    - Asynchronous put
    - Conversation sharing
    - Message selection performed by queue manager
    - Queue manager embedded publish/subscribe engine
    - Read ahead
    - MQCB, MQCTL and MQSUB
- **The WMQ JMS client can be used to connect to pre-V7 queue managers as well.**
  - In this case the connection runs in “migration mode”, and the above functionality is not available
  - No tight coupling between WMQ JMS client and queue manager versions
- **“Pre-warning” the WMQ JMS code it is connecting to a pre-V7 queue manager speeds up connection time**
  - Less important today since V6 is no longer supported.

WMQ Connection Factory Properties:
CONNECTIONNAMELIST

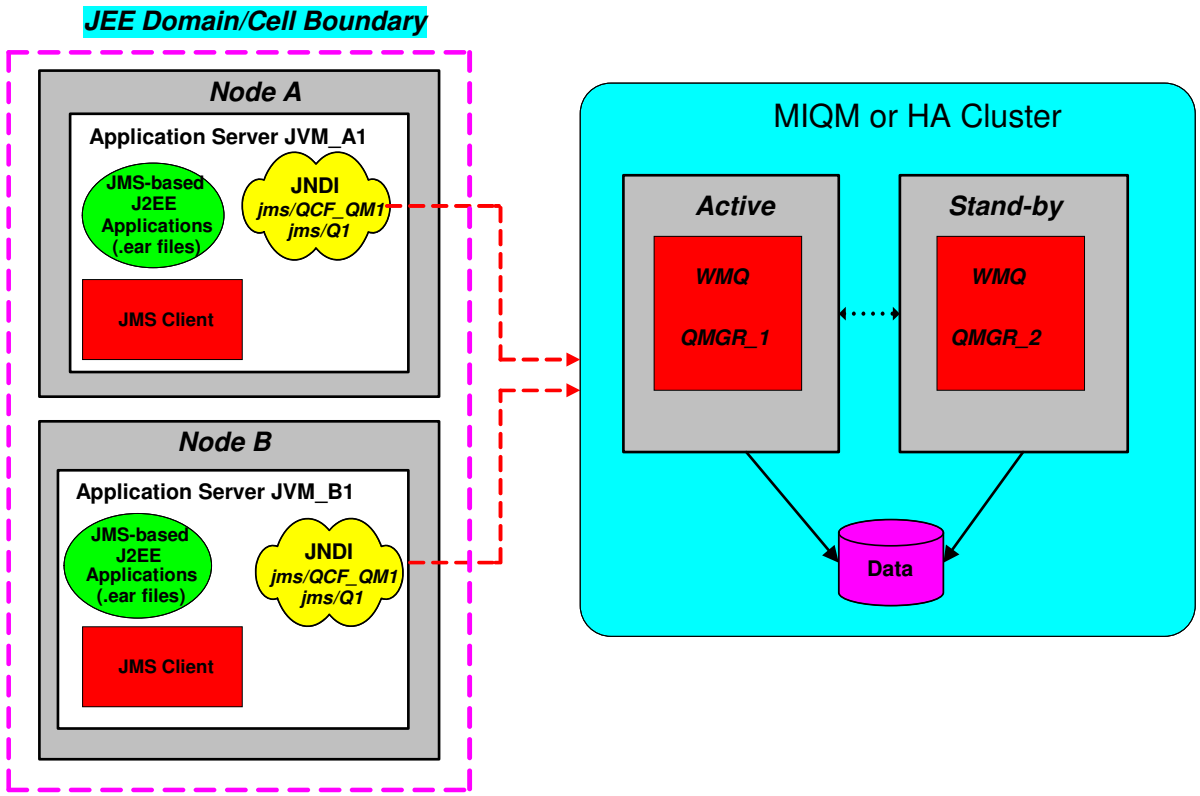
- **This property specifies a list of hostnames and ports to attempt to connect to.**
  - Comma-separated list of “hostname(port)” entries
- **Entries in the list are tried until a working connection is found, or the end of the list is reached.**
- **Intended for use with multi-instance queue managers to automatically reconnect to standby**
- **Must be same QM data if using XA**
- **Can also be used like a CCDT**
  - But note that it behaves differently



NOTES

- CONNECTIONNAMELIST is primarily intended for use with multi-instance queue managers. However, it can also be used in a manner similar to how CCDTs and queue manager groups.
  - Be aware that this approach does not do connection balancing that is possible with CCDTs
- If sessions are transacted, the entries in the list must refer to the same QM data (such as the active and standby instances of a single multi-instance queue manager)
  - The reason for this is that any external transaction manager (e.g. a JEE application server) will be unaware that the connection is to a different queue manager, and will expect the same transactions to be visible when connecting – and if different queue managers are referenced, XA transaction recovery will not work, and transactions will require manual resolution.
  - Note that "XA transaction recovery" will mean the transaction will be rolled back. But that is still considered to be transaction "recovery" because the queue manager and JEE server will be in agreement as to the state of the transaction, without the need for any manual intervention.
  - Also note that the same restriction applies to CCDTs
- Setting the custom property:
  - Connection Factories:
    - Use the JMS property called CONNECTIONNAMELIST to the list of host/port names that you wish to connect to. For example: host1(port1),host2(port2)
  - Activation Specs:
    - Use the property called connectionNameList on the activation spec with the same format as above: host1(port1),host2(port2)

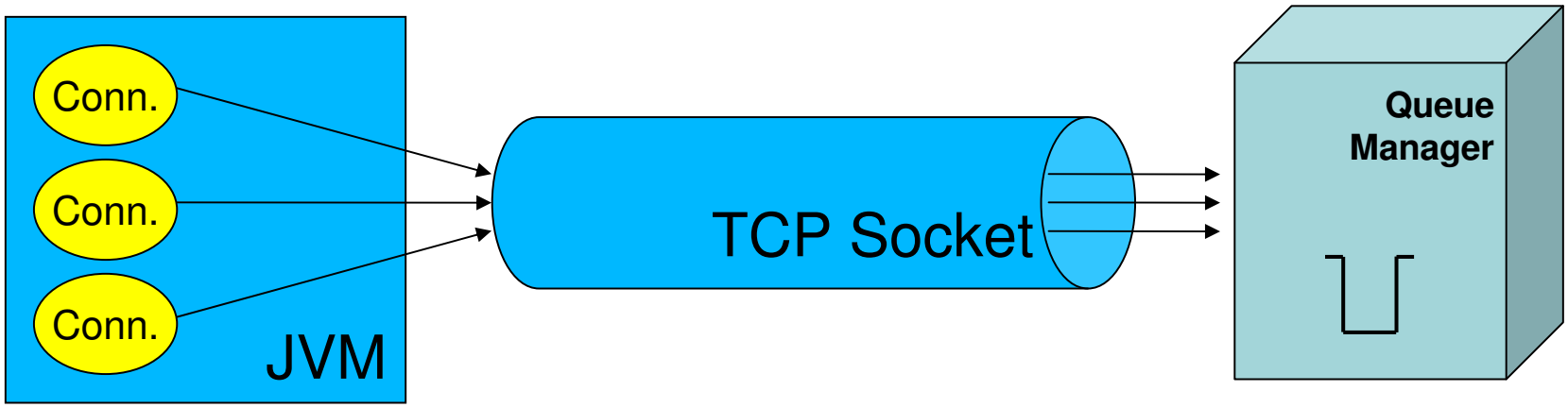
WMQ Connection Factory Properties:
CLONESUPP



- **Enables scaling durable subscribers.**
- **IBM-specific extension to the JMS specification**
  - Applications using a TopicConnectionFactory with CloneSupport enabled and an identical ClientId will be treated as 'clones' and each message will only be delivered to one of the set of clones
- **Less important with JMS 2.0**
  - Specifications now supports this capability
  - MQ V8 supports JMS 2.0

WMQ Connection Factory Properties:
SHARECONVALLOWED

- **This property specifies whether JMS applications that use this Factory can share their connection to a queue manager.**



- **Useful to reduce the number of network connections to a queue manager.**
- **Can have a performance impact.**
  - Multiple JMS applications will be sending data to a queue manager and waiting for a response over the same channel.
  - Less than ideal when applications are sending data at volume
  - Set server connection channel SHARECNV to 1 for maximum performance
    - This is even more true in MQ V8, where performance was enhanced further

NOTES

- Possible Values are **YES** (JMS applications running in the same JVM that use this Factory can share their connections to the queue manager, or **NO** (every JMS application that uses this Factory will create a new connection to the queue manager).
- If **YES**, the amount of sharing is controlled by the SHARECNV value of the SVRCONN channel they are connecting to.
- JMS applications tend to use more QM connections, as MQI connections are created for each JMS connection as well as each session under that connection.
- Historically, this has led to the QM Max Channels limit being reached more easily, and requiring the limit to be increased. Shared Conversations mitigate this.
- But Shared Conversations are less than ideal for some messaging patterns, as well as when volume is high and/or throughput delays cannot be tolerated.
  - Think of a queue sender or topic publisher putting messages at high volume. Now imagine 5 or 10 of them all trying to do so over the same shared TCP socket. They will very likely interfere with one another and performance will suffer.
  - Think of a queue or topic consumer, getting messages at high volume. Now imagine 5 or 10 of them all trying to do so over the same shared TCP socket. They will very likely interfere with one another and performance will suffer.
- For such cases, use SHARECNV(1) for the SVRCONN channel to which such applications are connecting
  - This will deliver best overall performance
  - Especially true in MQ V8, as additional performance enhancements when using SHARECNV(1) have been introduced with that release.

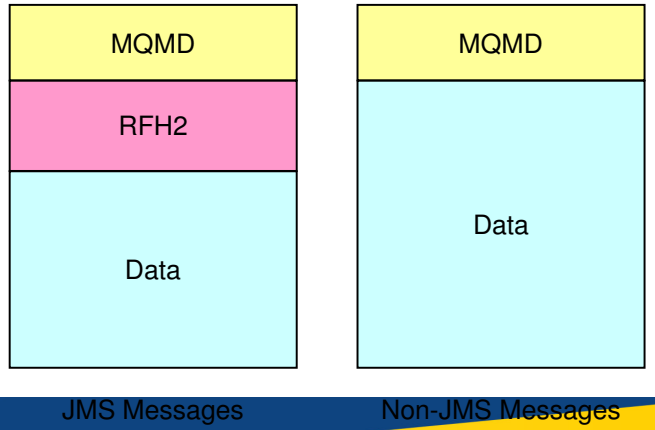
Using WMQ JMS with existing WMQ apps

MQMD Read/Write enabled

- Determines if MQMD properties can be set via JMS get/set message property methods.
- Allows full access to the MQMD header values
- Useful for sending or receiving messages from MQ applications that use specific header properties.
- JMS message property names begin with “JMS_IBM_MQMD…”

Target Client (JMS/MQ)

- Indicates what the receiving application is
- Removes the RFH2 header from **sent** messages WMQ applications may have issues with

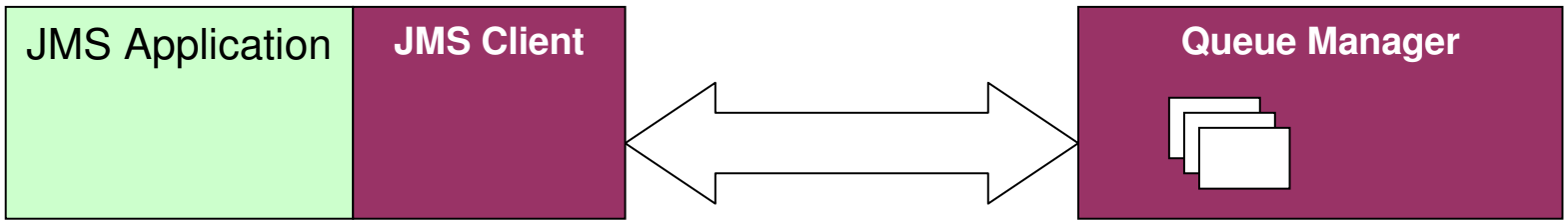


NOTES

- These are destination properties, set on the JMS Queue/Topic in use
- WMQ Manual page on MQMD Read/Write:
  - http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/topic/com.ibm.mq.csqzaw.doc/jm41030_.htm
  - Note that MQMD Message Context also needs to be set to allow some properties to be set on the message
  - Care should be taken with direct access to the MQMD, as it is possible to violate the JMS specification

Read Ahead Allowed

- **In general, messages are sent to JMS applications one at a time.**
- **The Read Ahead Allowed property tells the queue manager whether non-persistent messages can be streamed to the client application in preparation for them being consumed.**
  - Messages are stored in a buffer on the client.
  - If the client application terminates unexpectedly, all unconsumed non-persistent messages are discarded.



WMQ Connection Factory Properties:

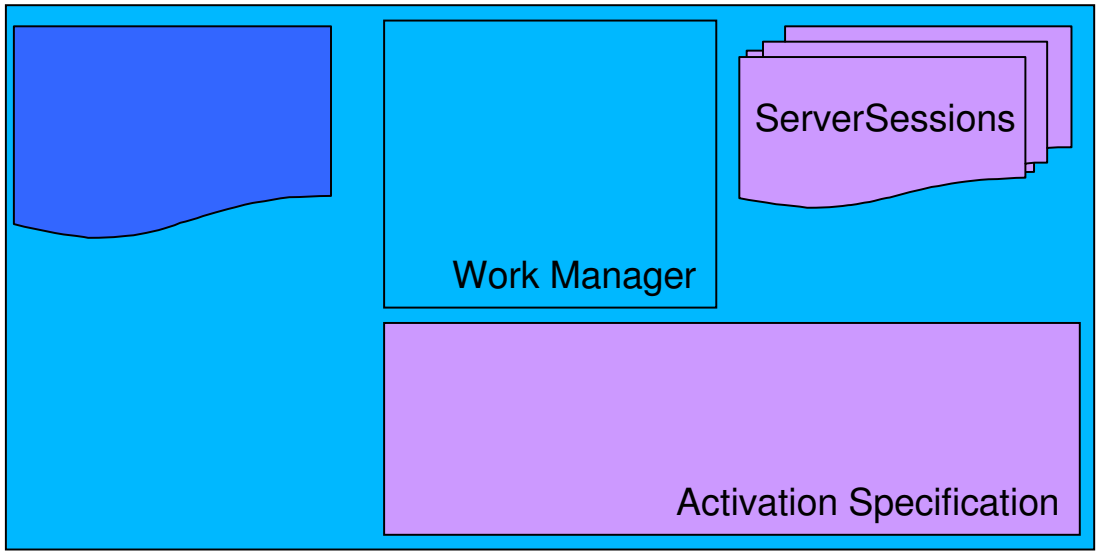
~~USECONNECTIONPOOL~~

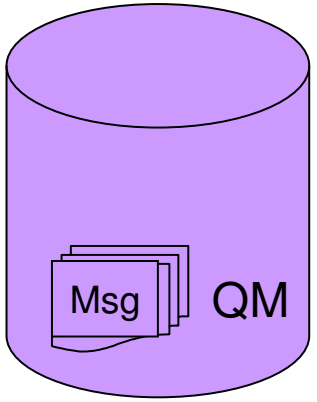
- **USECONNECTIONPOOL should not be used (because it is no longer there!)**
- **Prior to MQ V7, the MQ Java client classes did provide limited connection pooling**
  - A fixed pool of 10 connections was available
  - This was intended for use in JSE environments only
    - JEE application servers capable of doing a better job of managing connection pooling
- **With MQ V7, connection pooling was removed from the MQ Java classes**
  - Because changes to the V7 client rendered this of little or no value to JSE applications
    - Note that this is not present even when clients connect in “migration mode”
- **This has led to some customer issues**
  - Some were using this MQ-supplied connection pooling in Java EE environments
    - Removal resulted in performance problems
  - Solution is to work with Application Server team and/or Developers to properly implement pooling
- **Note that this property has been removed as of MQ V8**

JEE Configuration:

Activation Specifications

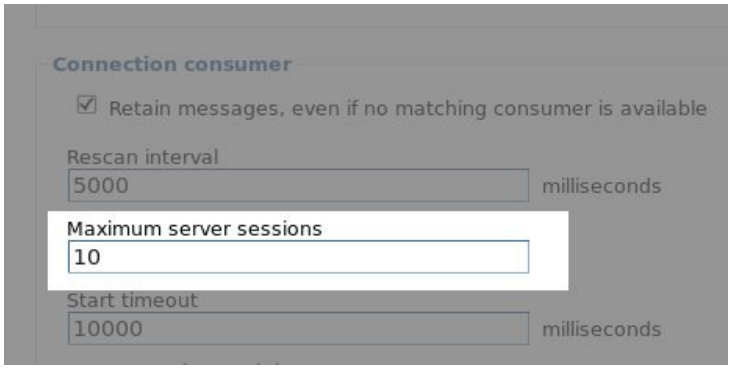
- How are messages destined for MDBs processed?

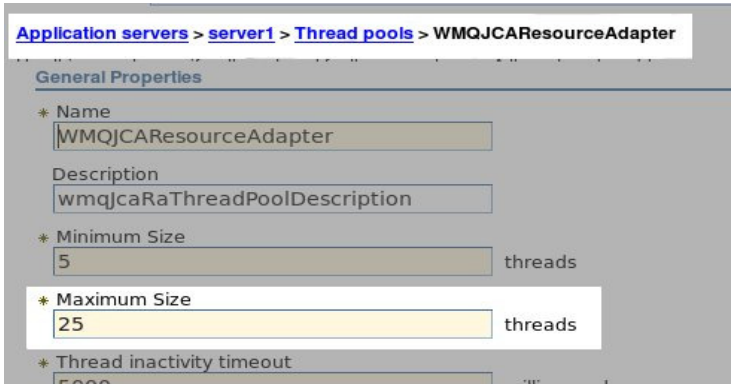




Activation Specifications - Threads

- To process a message, a ServerSession and thread are required
- Activation Specification parameter **Maximum server sessions** configures ServerSession pool, default 10
  - ServerSession pool per MDB
- Application Server Thread pool WMQJCAResourceAdapter used, default **Maximum size** 25
  - Thread pool used by all MDBs.
- So, by default, 3 MDBs could exhaust the threads
  - Will cause slower processing
  - Recommendation is that thread pool maximum accounts for all MDB maximums





Activation Specifications:

Recovery

- **What happens if the connection to a queue manager used by Activation Specifications is broken?**
- **The Activation Specification may have recovery features to allow it to attempt several reconnection attempts before stopping**
  - Not all JEE Servers have this capability
- **WebSphere Application Server can be configured to attempt recovery**
  - Default is to try 5 times in 5 minute intervals.
  - Configured using reconnectionRetryCount / reconnectionRetryInterval (ms) on the Resource Adapter, for all Activation Specifications.
- **If the Activation Specification stops, this is only reported in Application Server SystemOut.log logs**
- **Not all Java EE applications servers provide this capability**
  - Consult your product's documentation is using a different application server

N

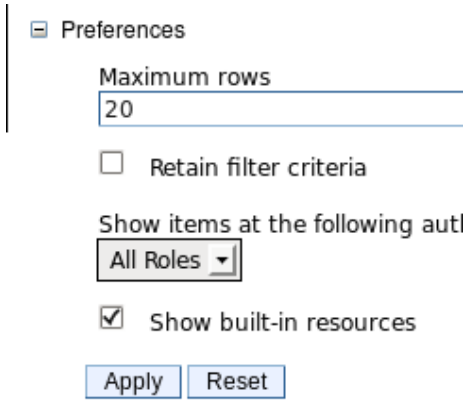
O

T

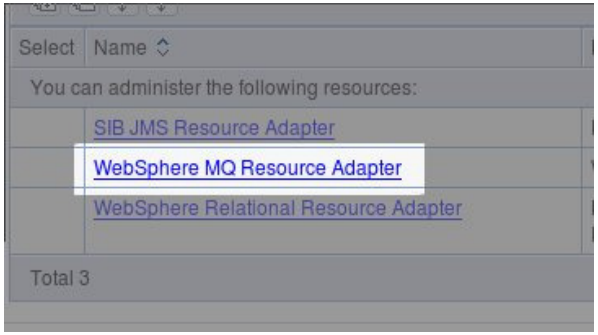
E

S

- To configure WMQ RA settings in WebSphere Application Server, go to Resources → Resource Adapters, select a scope to configure, and in the Preferences, select "Show built-in resources", and Apply.
- Select the "WebSphere MQ Resource Adapter"
- Select custom properties, and modify the "reconnectionRetryCount" and "reconnectionRetryInterval" properties



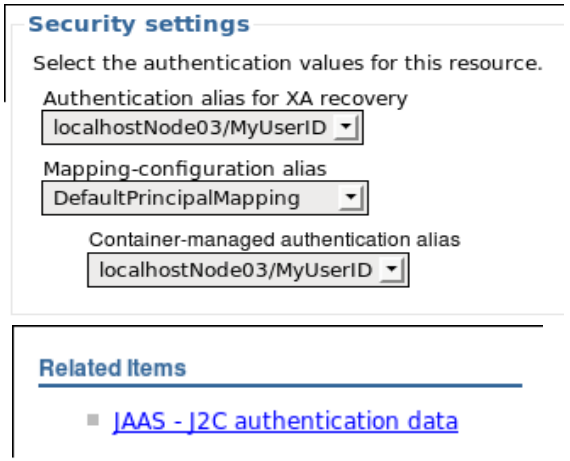
- Upon first failure, the RA will try an immediate reconnection attempt, and if that fails, retry at the configured interval.
- The properties are configured on the RA, and so affect ALL Activation Specifications using that RA



JMS User authentication

`createConnection(user,password)`

- **In Java EE, JMS application-specified user and passwords are not necessarily used.**
  - "Container-managed" authentication



- **Activation Specifications / Connection Factories can be associated with authentication data**
  - JAAS – J2C Authentication Data defines username and password details
  - The application needs to use JEE resource-references to access the connection factory
  - The *authenticationType* parameter needs to be set to container for container-managed authentication.

- **As for other WMQ clients, security exits may be required to validate passwords**
  - Prior to WMQ V8, only the userid would be checked
    - Security exits required to validate passwords
    - WMQ V8 may remove need for security exits
  - Userids more important with channel authentication (MQ V7.1 and above)

N

O

T

E

S

- WAS has built-in user credential repositories that can be defined with username and password details
  - JAAS – J2C Authentication Data
- Activation Specifications allow this to be configured in the Security parameters
- Connection Factories also allow this to be configured, although application configuration determines if it will be used.
  - The application needs to use JEE resource-references to access the connection factory
  - The res-auth parameter needs to be set to “container” for container managed authentication.
- `<res-auth>application</res-auth>`, or not using resource-references means the application is responsible for any authentication information
- Resource references have other benefits, such as decoupling application configuration names from real JNDI names.
- As for other WMQ clients, security exits may be required to validate passwords, since prior to V8 MQ would only check the userid.
  - V8 may now eliminate the need for a security exit
  - Also, userids are more important with WMQ 7.1 channel authentication
- If nothing specified, by default WMQ JMS passes a blank id
  - Custom property “com.ibm.mq.jms.ForceUserID=true” will pass the JVM process id.
  - http://www-01.ibm.com/support/docview.wss?uid=swg2147080
- ejb-jar.xml resource-ref values need a res-auth of container to use the CF security settings.
- Using annotations, will need to use the @Resource authenticationType

```
<resource-ref id="ResRef_1">
   <description></description>
   <res-ref-name>jms/CF</res-ref-name>
   <res-type>javax.jms.ConnectionFactory</res-type>
   <res-auth>Container</res-auth>
   <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

```
@Resource(authenticationType = AuthenticationType.CONTAINER)
private ConnectionFactory cf = null;
```

Agenda

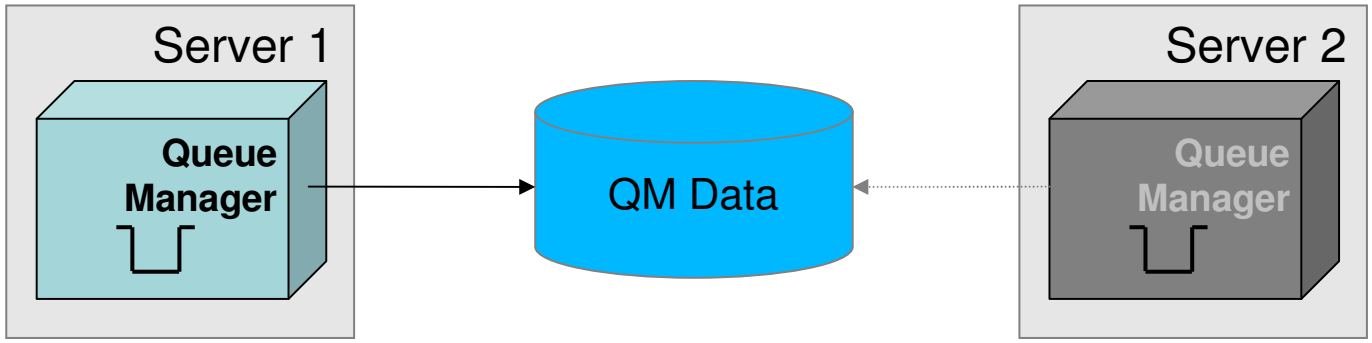
- Why are we here?
- Java Enterprise Edition (JEE) and Messaging
  - What is JMS?
- Accessing WMQ from JEE Environments
  - How to configure
  - WMQ features useful in JEE Environments
- Configuring WMQ and WAS
  - Useful tuning parameters
  - **High Availability and clusters**

High Availability

- **Treat WAS and WMQ as separate resources**
- **WMQ features make it easier for JEE applications to reconnect during failover**
  - Quicker detection of broken connections
  - Connection Name Lists
- **OS-level HA solutions (HACMP etc.)**
  - Active – Passive failover
  - Transparent as resources such as IP addresses are moved over
  - Can include WMQ as well as other resources in the same failover
  - Third party solution, introduces expense

Multi-instance Queue Managers

- **WMQ's basic implementation of Active – Passive failover**
  - Only moves WMQ resources
  - Can be faster than OS-level HA
- **Needs data to be stored in external storage with locking, such as NFS V4**
- **One active queue manager, with other standby instance**
- **Standby instance can detect failure and take over**
  - Can also trigger failover
- **Same data is shared, so it's the SAME queue manager.**



NOTES

- The Multi-Instance Queue Manager (MIQM) feature of WMQ, introduced in V7.0.1, provides basic, built-in failover support without the need for any external HA coordinator
  - Faster takeover: fewer moving parts
  - Cheaper: no specialised software or administration skills needed
  - Windows, Unix, Linux platforms
- Queue manager data is held in networked storage
  - NAS, NFS, GPFS, GFS2 etc so more than one machine sees the queue manager data
  - Improves storage management options: formal support for these even without failover config
- Multiple instances of a queue manager on different machines
  - One is “active” instance; other is “standby” instance
  - Active instance “owns” the queue manager’s files and will accept app connections
  - Standby instance does not “own” the queue manager’s files and apps cannot connect
    - If active instance fails, standby performs queue manager restart and becomes active
- Instances share data, so it’s the SAME queue manager

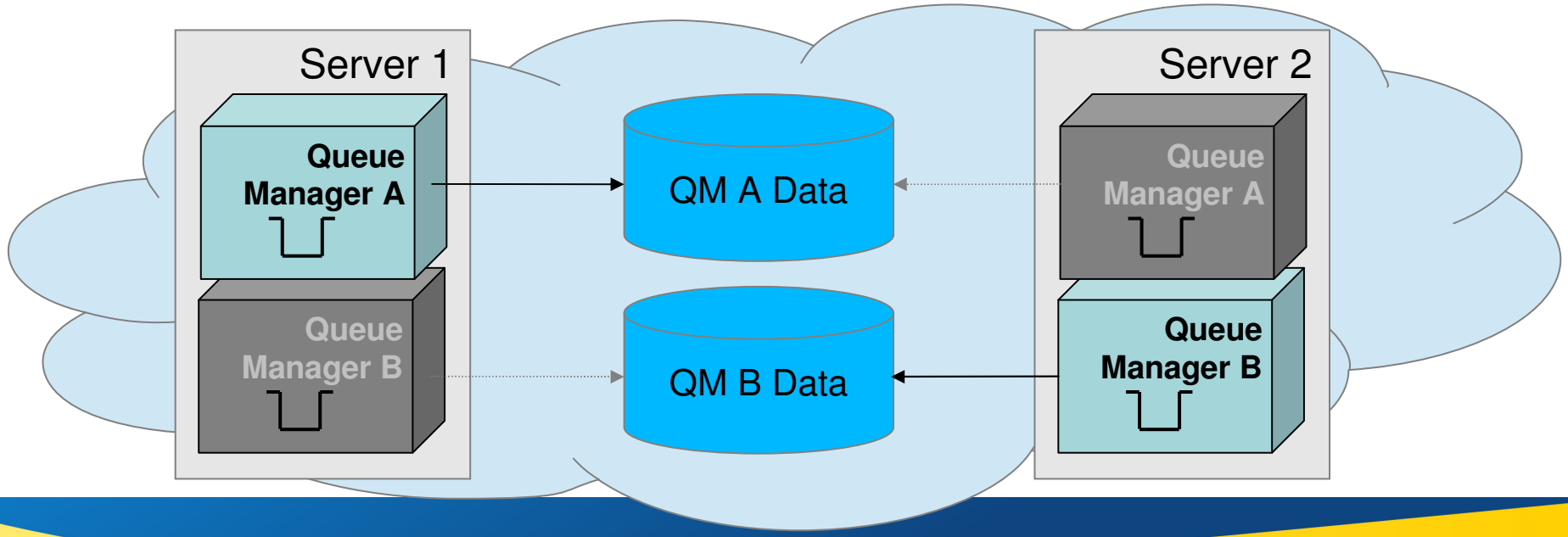
Multi-Instance MQ and Java EE

- **Connection to Multi-Instance Queue Managers is supported**
  - Minimum of WAS 7.0.0.13 required (contains WMQ 7.0.1.3)
  - Can be configured using CCDT
  - Can be configured using Connection Name Lists

- **The Automatic Client Reconnection feature of MQ *is not supported***
  - Not supported in EJB/Web container, due to XA ramifications
    - Enable it, and it will disable itself!
  - This does not mean that *reconnection* is not possible in Java EE environment
    - MDB based applications use ActivationSpecification-based reconnection
  - Automatic Client Reconnect can be used in unmanaged/client environments
  - See this document for guidance:
    - http://www.ibm.com/support/docview.wss?uid=swg21508357

High Availability continued...

- **Active – Active WMQ**
  - Multiple active running queue managers
  - Provides service resilience and load balancing
    - e.g. WMQ Clusters, z/OS Queue Sharing Groups (QSG)
  - Can have stranded messages problems if queue manager becomes unavailable.
    - Can use HACMP/Multi-instance queue managers to avoid



Useful Information:

Product Connectivity Scenarios

- **Scenario-based instructions for implementing a solution in a business context**
- **Providing a thorough, complete, and single end-to-end main path from the business need (high level) to the tasks (low level)**
- **Optional features help you learn as you progress through each task**
  - Information about *why* you are instructed to do something
  - Information about *what else* you might do, or want to learn about, related to what you are reading in the main window
- **Includes samples that you can try out and adapt**

http://publib.boulder.ibm.com/infocenter/prodconn/v1r0m0/index.jsp

Summary

- **Why were we here?**
- **Messaging in Java EE**
  - What is JMS?
- **Accessing WMQ from Java EE**
  - How to configure
  - New WMQ V7 features useful for Java EE
- **Configuring WMQ and WAS**
  - Useful tuning parameters
  - High Availability and clusters

Questions & Answers

Thank You

Download this Presentation from the MQTC site for a lot more detail. Notes slides are included!