This session discusses the new JMS nodes introduced in Message broker Version 6.

Agenda

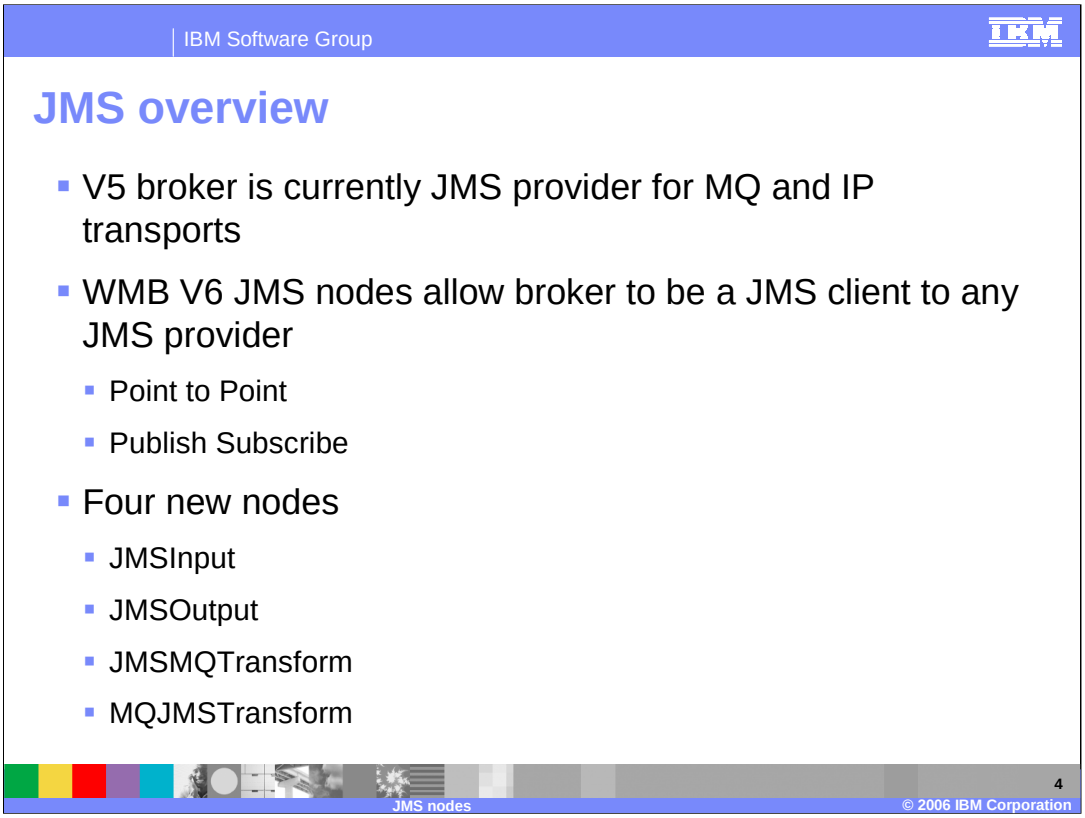
- Overview
- Message handling
- Configuration
- Debugging
- Summary and references

The agenda for this presentation covers message handling, configuration, and debugging…

Section

Overview

…starting with an overview.

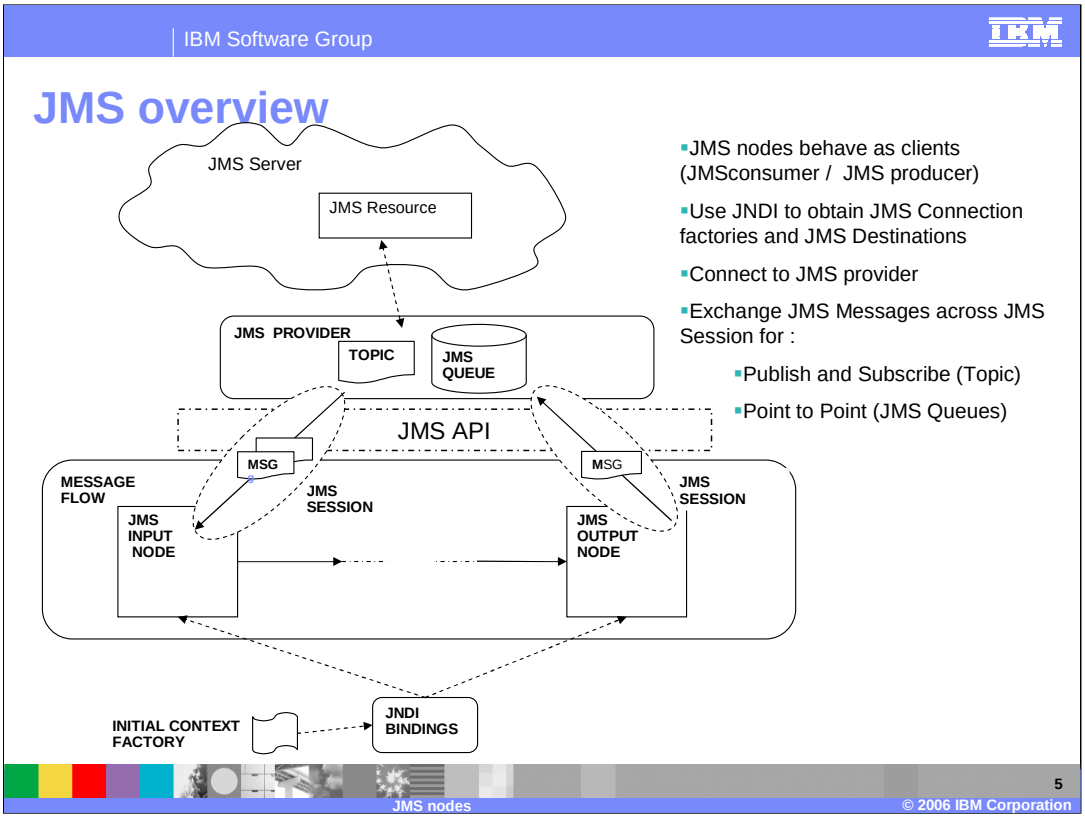


In previous versions of WebSphere Message Broker, the support for JMS messages extended only to JMS provision. In WebSphere Message Broker Version 6.0, brokering value has been added to enable the broker to behave like a JMS client. In Message Broker Version 6, the JMS clients can be embedded into a message flow. This means that an input node can be JMS message consumer, and an output node can be a message producer. The two primary functions that are available in these scenarios are ‘Point to Point’ and ‘Publish Subscribe’ JMS messaging.

In previous versions of the broker, the WebSphere MQ Real-time Transport enabled support for JMS provision; the built in nodes, Real-timeInput node, Real-timeOptimizedFlow node, and Publication node, allow JMS applications to communicate with applications that use other supported protocols and transports.

In this implementation, the Real-time node acts as a server for a JMS client, where the client can be WebSphere MQ.

WebSphere Message Broker Version 6.0 adds brokering value to a JMS network. Four new built-in nodes, JMSInput node, JMSOutput node, JMSMQTransform node, and MQJMSTransform node, provide support for the broker to act like a JMS client. JMS messages can be sent and received, and can be transformed into other message formats.

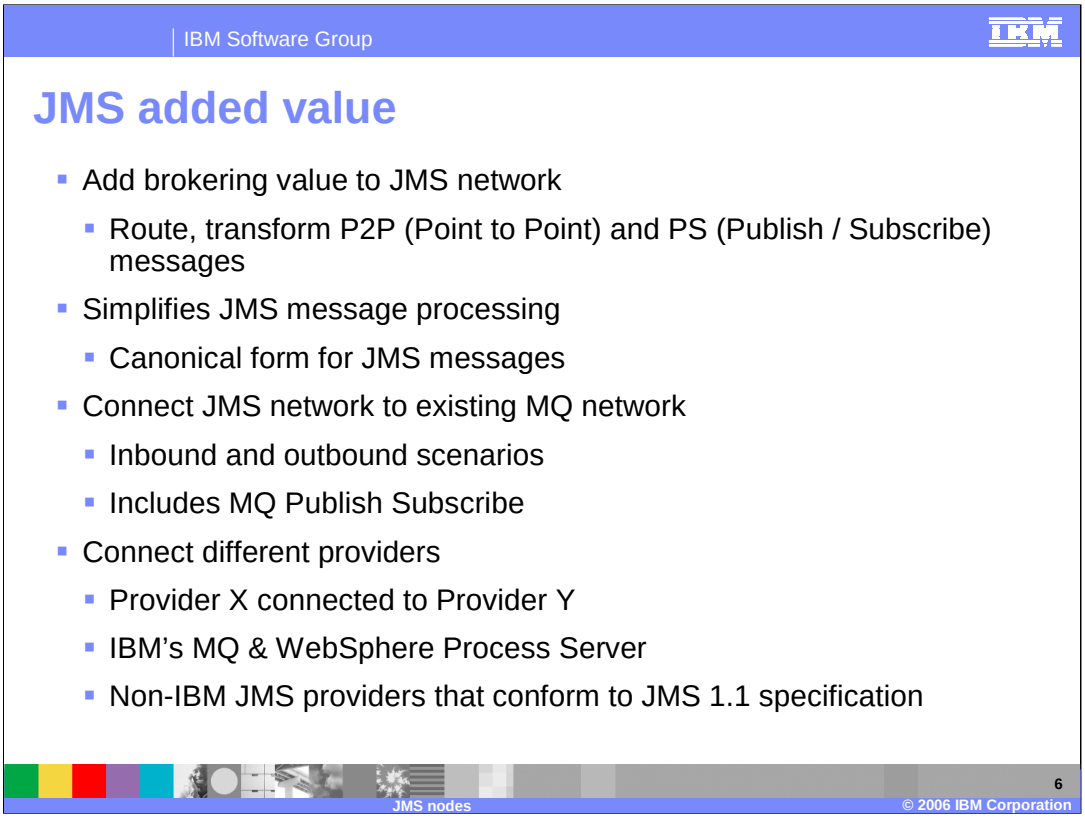


- JMS nodes behave as clients (JMSconsumer / JMS producer)
- Use JNDI to obtain JMS Connection factories and JMS Destinations
- Connect to JMS provider
- Exchange JMS Messages across JMS Session for :
  - Publish and Subscribe (Topic)
  - Point to Point (JMS Queues)

This schematic shows a high-level picture of JMS support in Message Broker Version 6. The JMS nodes work with the WebSphere MQ JMS provider, WebSphere Application Server Version 6.0, the IBM Service Integration Bus, and any JMS provider that conforms to the Java™ Message Service Specification, version 1.1.

The JMSInput and JMSOuput nodes can be regarded as stand-alone JMS clients. Just like any other JMS client, they connect to a JMS provider in order to exchange messages. They look up various artifacts called “JNDI administered objects” requiring a connection factory within JNDI (Java Naming and Directory Interface). This provides a pointer into the JMS namespace which is used to create a connection to a JMS provider. Once this connection is established, across the JMS API, a session to the JMS provider is created. All message sends and receives are done using this session.

The JMS support allows for two different types of JMS destinations; the ‘Topics’ for the ‘Publish Subscribe’ message domain, and the JMS queues for the ‘Point to Point’ message domain. This is shown in the centre of the schematic.



This new function allows you to integrate applications which connect over more protocols than just WebSphere MQ. With this new function, you can connect your message flow to an external JMS provider, you can route JMS messages to alternative destinations, and you can transform data passing through the broker. All the function available with WebSphere MQ are available when messages arrive over JMS.

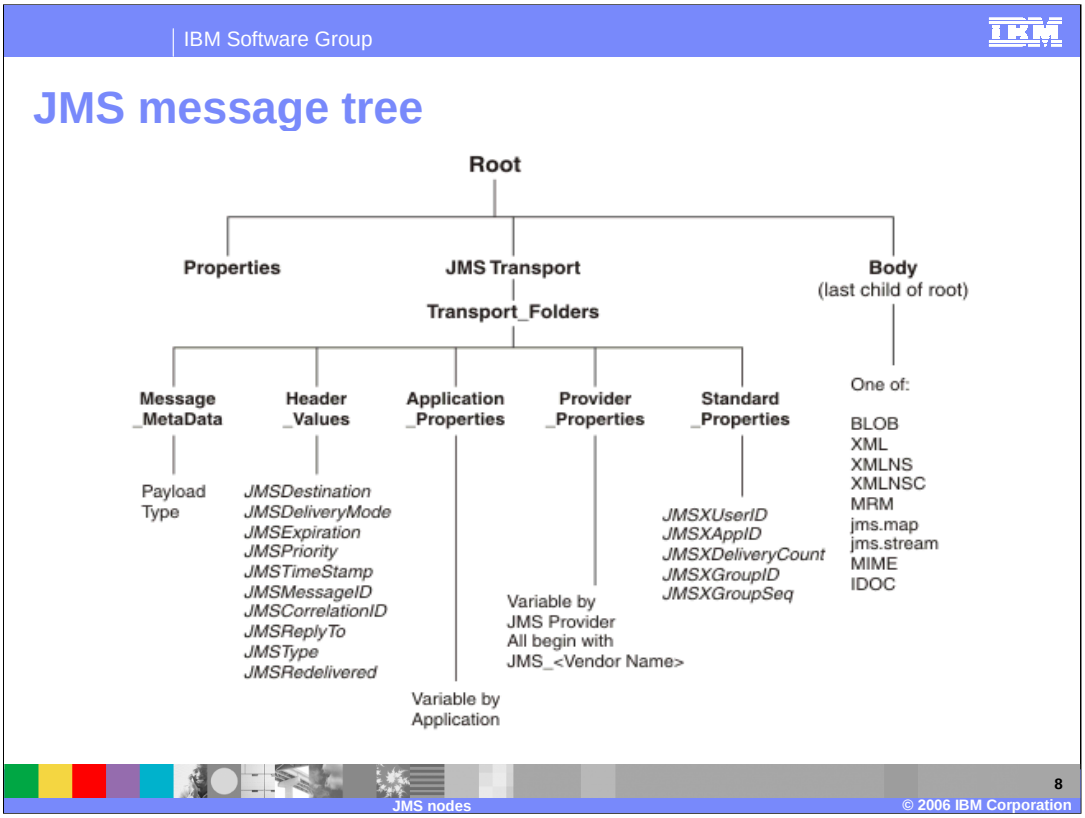
The JMS function provides a simplified view of the JMS message; this has to be transformed into a broker message tree so that any JMS message from any JMS provider can be represented within a broker message tree. This data is then independent of the source of the data, and can be operated on by a normal message flow.

This allows you to connect a JMS network to an MQ network, thus providing a bridge between two different messaging environments. You can also connect different JMS providers; the broker can be used to bridge between different JMS providers within the broker environment. This includes the JMS provider included with MQ JMS and the WebSphere messaging platform. It also includes JMS providers from other vendors, providing that they conform to the JMS 1.1 specification.

Section

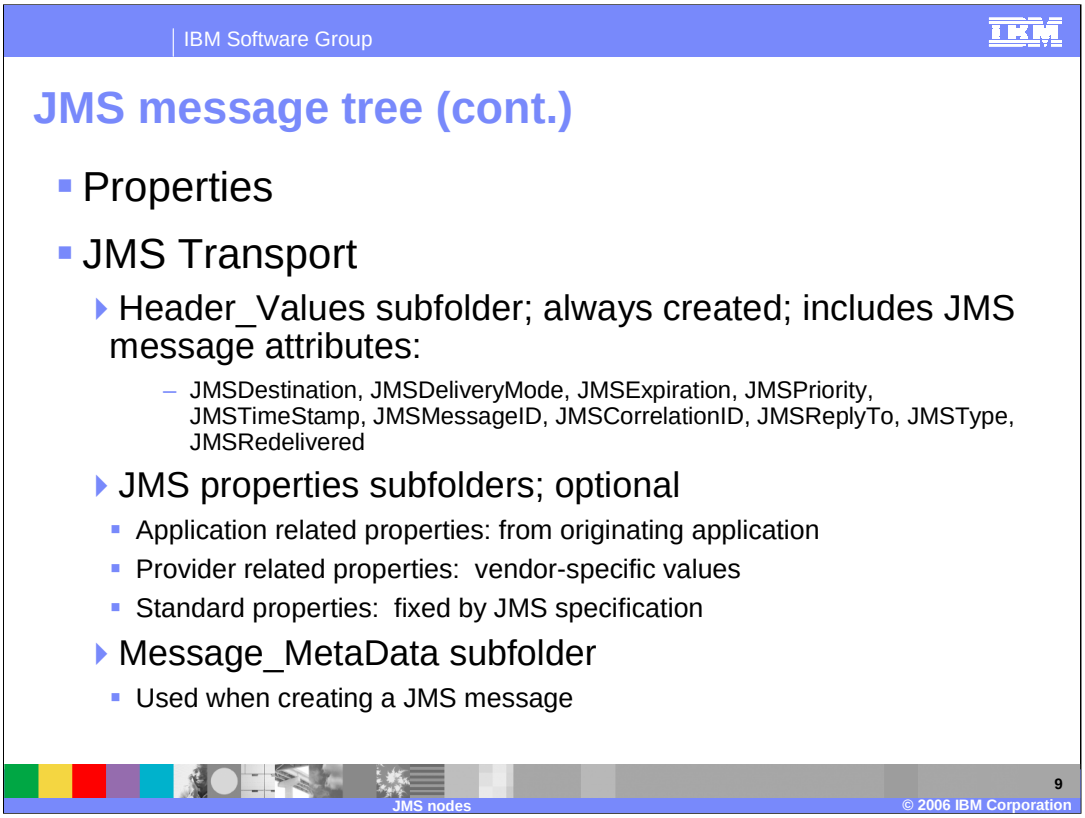
Message handling

This section covers message handling.



At the JMSInput node, a message is received as a Java object and not as a bit stream wire format (as would be the case with an MQInput node). The message does not populate an MQMD and RFH2 header, but instead populates a new message tree that represents a JMS message in a more native way.

To represent a JMS message in a message tree, a new canonical form has been created. This new message tree allows for representation of JMS message header data and message properties. The JMS message tree is in a format that is recognizable to Java programmers.



The Properties folder contains information such as message domain, message set, message type and message format.

The JMSTransport folder is comprised of several subfolders. The Header_Values subfolder is mandatory and is always created. It includes the fixed header fields shown here.

JMS message properties are optional. If they are present in input messages, they are stored in the appropriate property subfolder.

Application related properties are assigned by a Java application and are set before the message is delivered. The property names of the application are meaningful only to the sending and receiving applications.

Every JMS provider can define proprietary properties that can be set either by the client or automatically by the provider. Provider related properties are prefixed with *JMS_* followed by the vendor name and the specific property name. For example, the WebSphere MQ JMS client sets the provider property to be *JMS_IBM_MsgType*.

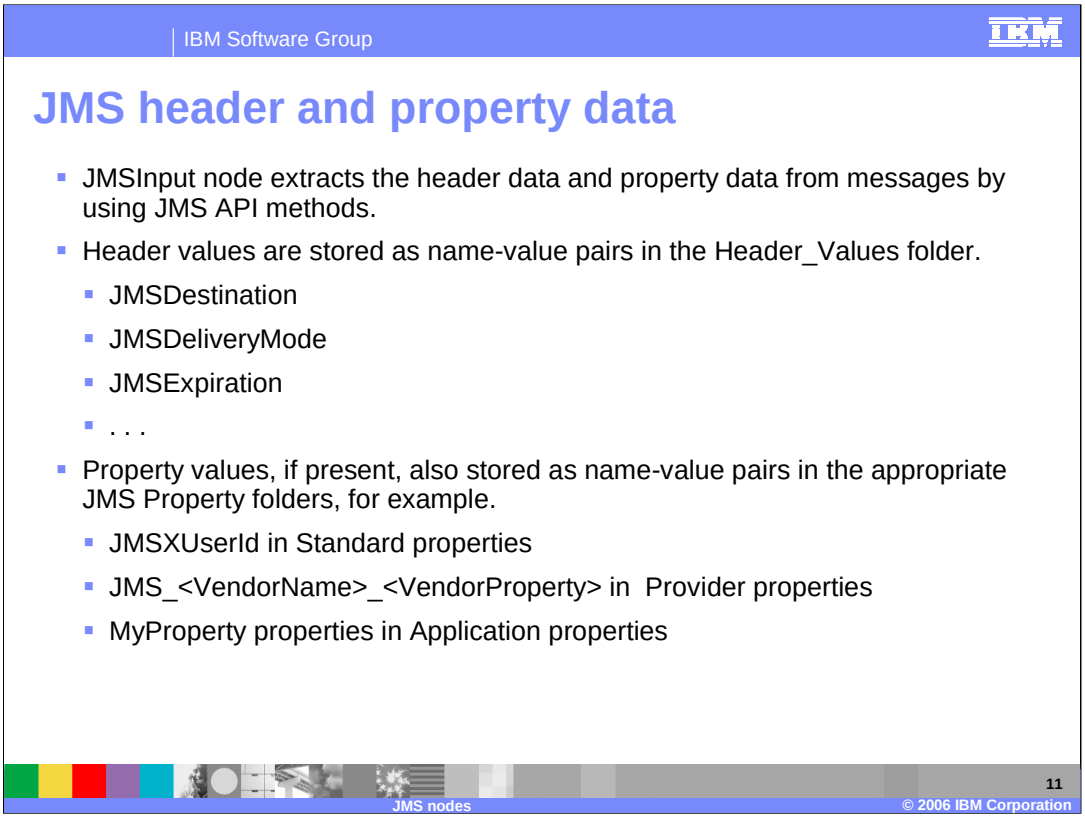
Standard properties are set by the JMS provider when a message is sent. The JMS provider vendor can choose to support none, some, or all standard properties. Standard property names start with *JMSX*; for example: *JMSXUserid* or *JMSXDeliveryCount*.

Message_MetaData subfolder is included in order to preserve the payload type of the JMS message. It is used by the JMSOutput node when creating a JMS message.

JMS message tree (cont.)

- Body
  - Payload transferred by:
    - XML
    - XMLNS
    - XMLNSC
    - BLOB
    - JMSMap
    - JMSStream
    - MRM
    - MIME
    - IDoc

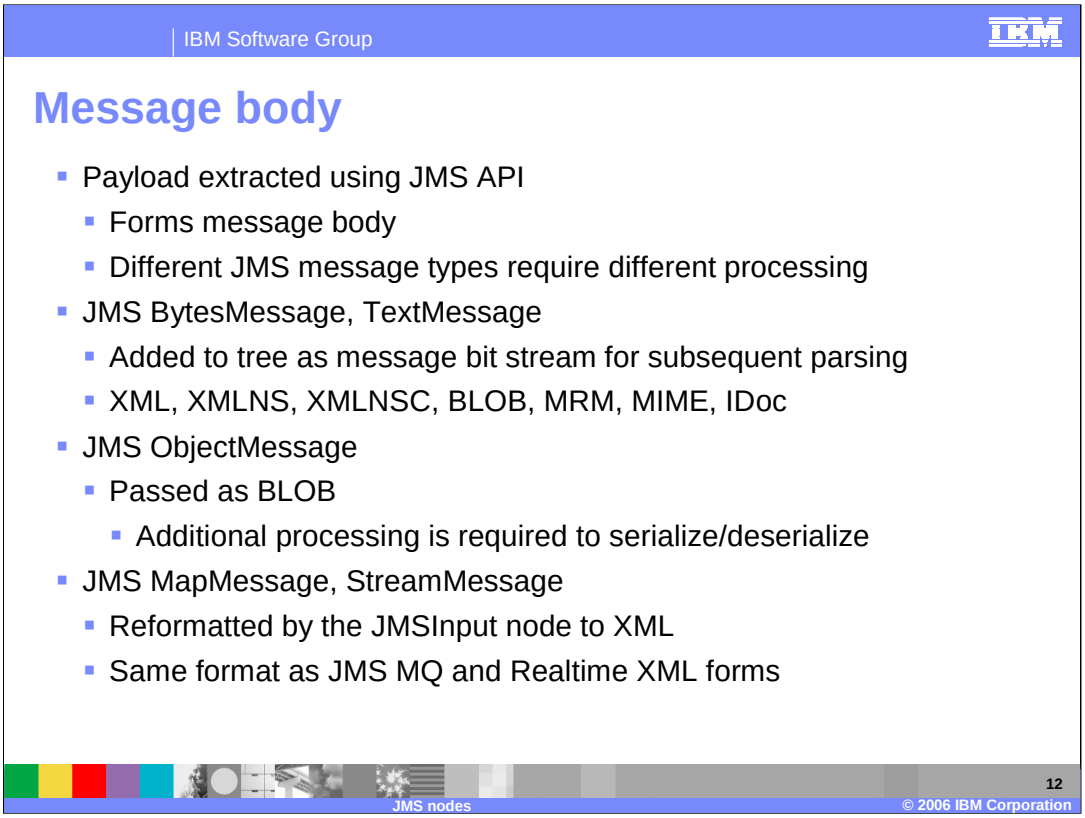
The message payload is stored in the body folder, which is the last child of Root. The payload is transferred by using one of the message domain parsers listed here.



The JMSInput node extracts header data from messages by using JMS API methods. Header data is stored as name-value pairs in the *Header_Values* folder. The API methods return the value.

In a similar way, the JMSInput node extracts property data from messages by using JMS API methods. Property data is stored as name-value pairs in the properties folders. The API method returns a value for every property name with which it is supplied.

The JMSInput node uses the header and property data to create an XML representation of the JMSTransport folders. The node passes the XML data to the JMSTransport parser as a byte array. The byte array is then used to populate or to refresh the elements in the message tree. The JMSTransport parser is a new parser type.



JMS defines six message interface types; a base message type and five subtypes. The base class is used for event notification and does not have a payload. The message types are defined according to the type of the message *payload*, where the payload is the body of a message that holds the content. JMS specifies only the interface and does not specify the implementation. This allows for vendor specific implementation and transportation of messages while using a common interface.

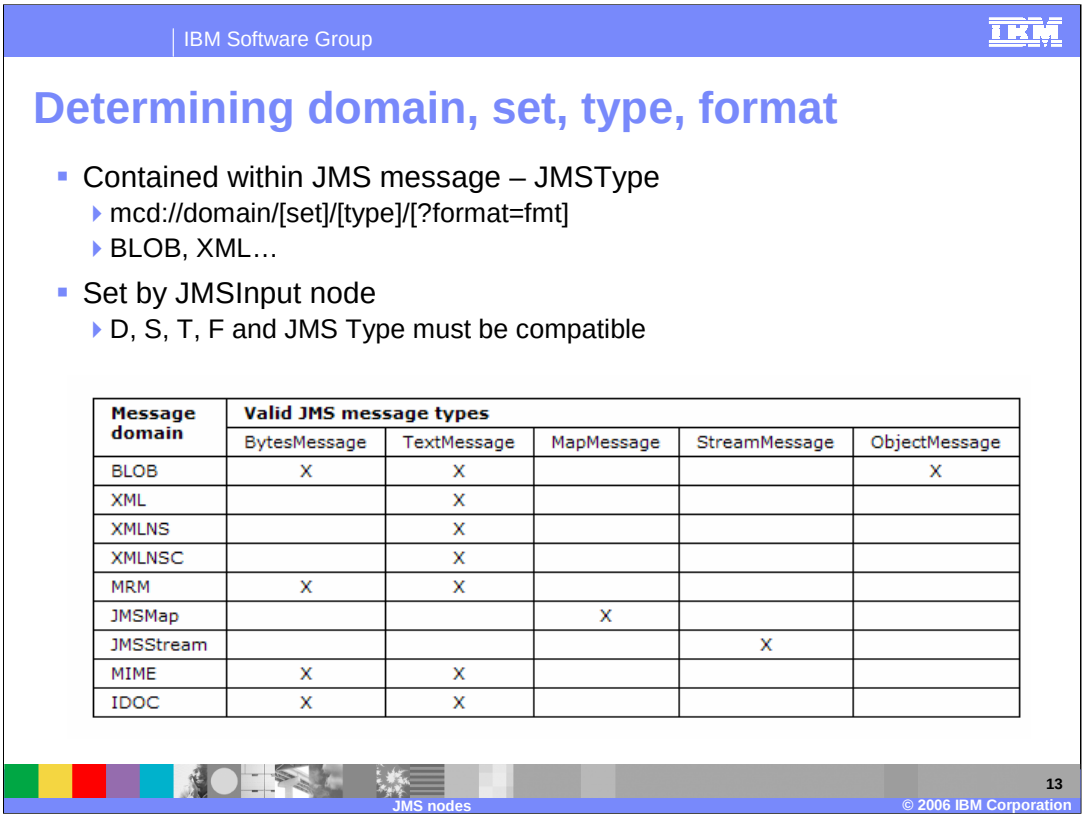
In BytesMessage the payload is stored as an array of bytes. This message type is useful for exchanging data in an application's native format and when JMS is used as a transport between two systems, where the JMS client does not know the message payload type.

TextMessage data is stored as a string. This message type is useful for exchanging simple text messages and for more complex character data, such as XML documents.

ObjectMessage carries a serializable Java Object as its payload. It is useful for exchanging Java objects.

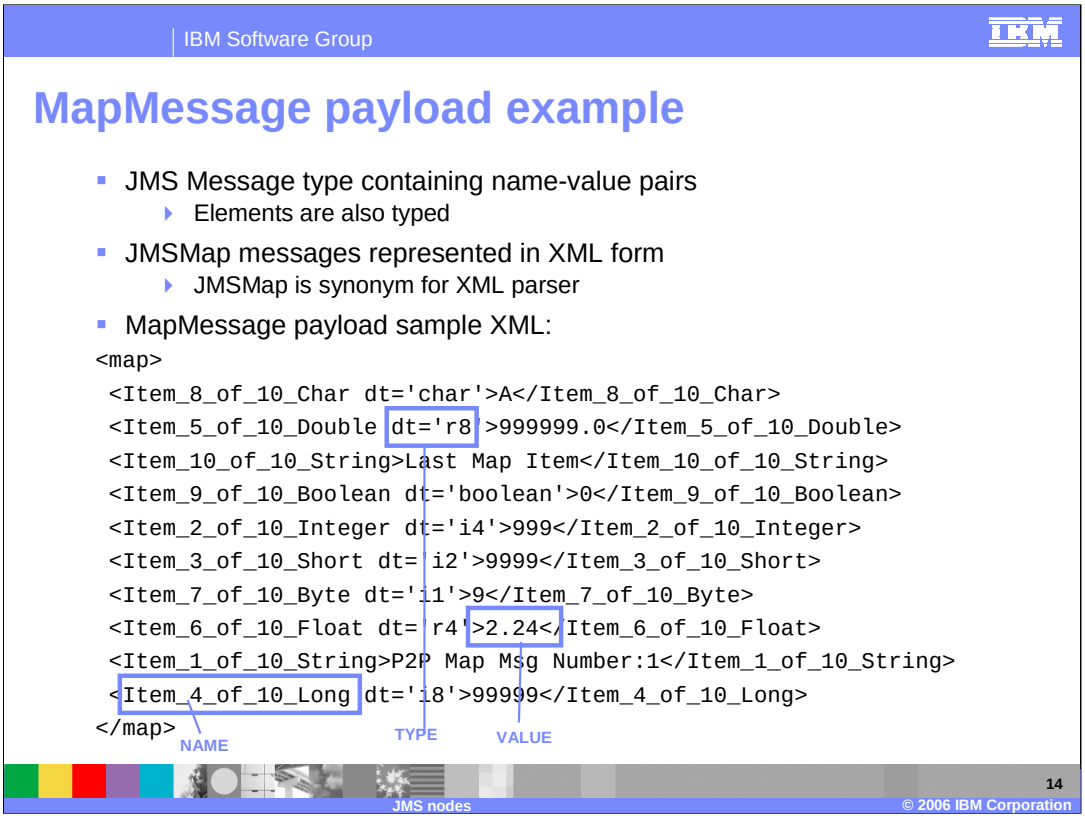
A StreamMessage is a sequence of primitive Java types. The message object keeps track of the order and the types of these primitives within the stream. Formal conversion rules apply

The payload of a MapMessage is stored as a set of name-value pairs. The name is defined as a string and the value is typed. The MapMessage is useful for delivering keyed data that can change from one message to the next.



When a JMS Message is received by the JMSInput node, the message domain is derived according to the following criteria and in the following order of precedence:

- If the Message Domain property is set to a specific domain type the node expects to receive only the JMS message types shown in this slide.
- If the Message Domain property is *blank* (default), the JMSType header value from the JMS input Message is used to determine the format
- If the Message Domain property is *blank* (default) and the JMSType header value from the JMS input message is also blank, the message domain is set according to the JMS Message Java Class as follows:
    - TextMessage XML
    - BytesMessage BLOB
    - Message JMSMap
    - StreamMessage JMSMap
    - StreamObjectMessage BLOB



The payload for MapMessage and StreamMessage can be extracted only as individual elements and must be reformatted by the JMSInput node before it can be used to create the message body. The JMSMap domain is a synonym for the broker XML parser, which expects a stream of XML data. MapMessage payload data however, is extracted as sets of name-value pairs from the message object. The JMS API is used to obtain the name-value pairs. The JMSInput node appends each name-value pair to a bit stream as an XML element and value, and preserves the type of the value by using the dt= attribute. This example shows the XML that is generated by the JMSInput node for the MapMessage payload.

The first blue box (Item 5) encompasses a type, indicating that this schema has been borrowed from the real-time processing in MQ JMS client code. It is a way of representing data within the message tree so it can be recreated in an outbound message.

In Item 6 the value is highlighted; ‘2.24’ is the value of Map element.

The last blue box shows the element name.

StreamMessage payload example

- JMS message containing sequence of UNNAMED values
  - Elements are typed
  - Default element name “elt” is used to generate the XML elements.
- JMSStream messages represented in XML form
  - JMSStream is synonym for XML parser
- StreamMessage payload sample XML

```
<stream>
        <elt>P2P Stream Message Number :7</elt>
        <elt dt='i4'>999</elt>
        <elt dt='i2'>9999</elt>
        <elt dt='i8'>99999</elt>
        <elt dt='r8'>999999.0</elt>
        <elt dt='r4'>2.24</elt>
        <elt dt='i1'>9</elt>
        <elt dt='char'>A</elt>
        <elt dt='boolean'>0</elt>
        <elt>Last Stream Item</elt>
</stream>
```


The sample XML shown here is an illustration of what a StreamMessage would look like within the message tree. A StreamMessage is a sequence of unnamed values. Each ‘elt’ is followed by a data type and value.

Both the JMS MapMessage and JMS StreamMessage domains are synonyms for the XML parser within the broker.

Section

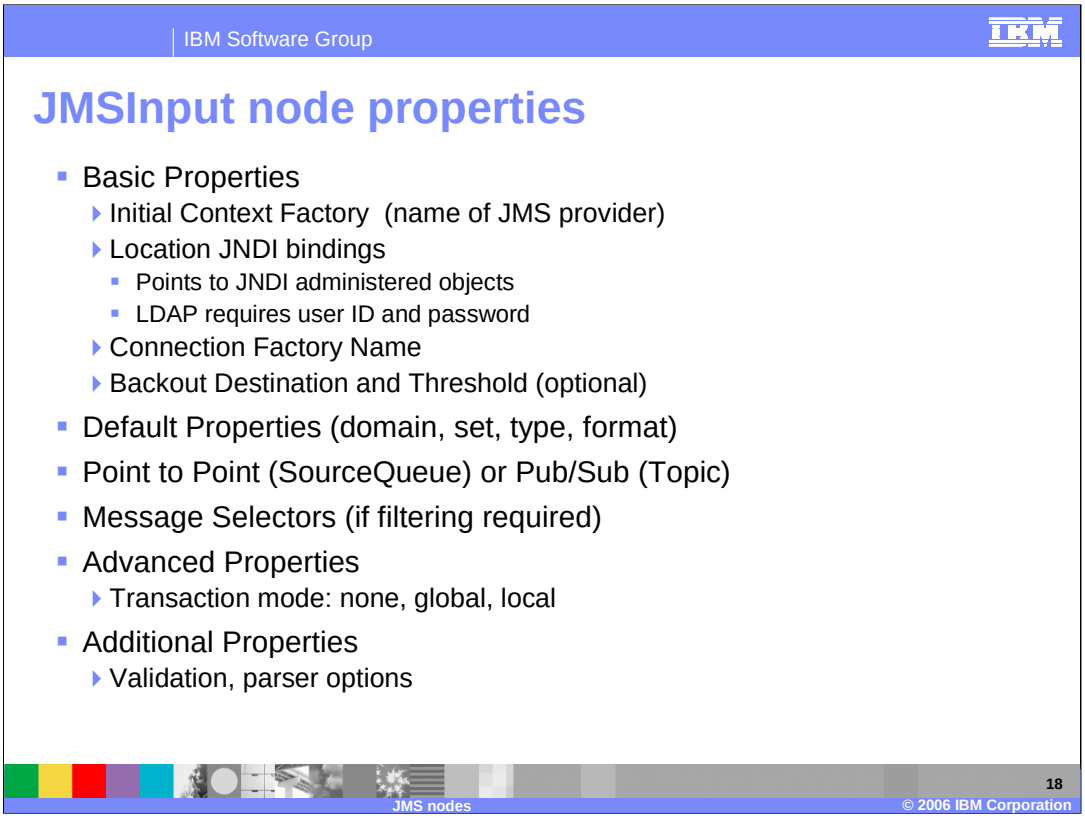
Configuration

This section covers configuration.

JMSInput node

- First node in message flow
  - Point to Point or Pub/Sub
  - Responsible for creating JMS tree from JMS input message
  - Hands payload to appropriate broker parser

Message flows which handle messages that are received from connections to JMS providers must always start with a JMSInput node. If you include an output node in a message flow that starts with an JMSInput node, it can be any of the supported output nodes (including user-defined output nodes); you do not have to include an JMSOutput node. However, if you do not include a JMSOutput node, you must include the JMSMQTransform node to transform the message to the format that is expected by the output node.



To configure the properties right-click the node in the editor view and click Properties. The basic properties of the node are displayed in the properties dialog. All mandatory properties that do not have a default value defined are marked with an asterisk on the properties dialog.

Configure the following Basic properties:

•Enter an *Initial Context Factory* value. A JMS application uses the initial context to obtain and look up the JNDI administered objects for the JMS provider. The default value is com.sun.jndi.fscontext.RefFSContextFactory, which defines the file-based initial context factory for the WebSphere MQ JMS provider. To identify the name of the Initial Context Factory for the JMS provider, refer to the JMS provider documentation.

•Enter a value for the *Location JNDI Bindings*. This value specifies either the file system path or the LDAP location for the bindings file. The bindings file contains definitions for the JNDI administered objects that are used by the JMSInput node.

•Enter a *Connection Factory Name*. The connection factory name is used by the JMSInput node to create a connection to the JMS provider. This name must already exist in the bindings file.

•Enter a *Backout Destination* name. Input messages are sent to this destination when errors prevent the message flow from processing the message, and the message must be removed from the input destination. The backout destination name must exist in the bindings file.

•Enter a value for the *Backout Threshold*. This value determines when an input message is put to the *Backout Destination*. For example, if the value is 3, the JMS provider attempts to deliver the message to the input destination three times. After the third attempted delivery, the message is removed from the input destination and is sent to the backout destination. The default value is 0.

Default in the properties provide values for the properties that describe the message domain, message set, message type, and message format.

If the JMSInput node is to be used to subscribe to a topic, select *Pub/Sub* in the properties dialog navigator. Enter the name of the *Subscription Topic*. If the node is to receive publications from a durable subscription topic, enter a *Durable Subscription ID*.

If the JMSInput node is to be used to receive point to point messages, select *Point to Point* in the properties dialog navigator.

If filtering of messages is required, select *Message Selectors* in the properties dialog navigator. Use Advanced properties to specify transaction mode.

JMSInput node configuration

- Terminals
  - Input
  - Out
  - Failure
  - Catch
  - Define a backout destination
- Coordinated transactions
  - To coordinate JMS with other transactional resources such as MQ, database
  - Transaction Mode in the Advanced node property specifies message to be received within syncpoint
  - Additional configuration steps required; once per provider

For each message that is received successfully, the JMSInput node routes the message to the out terminal. If this fails, the message is retried. If the retry threshold is reached, where the threshold is defined by the BackoutThreshold property of the node, the message is routed to the failure terminal.

You can connect nodes to the failure terminal to handle this condition. If you have not connected nodes to the failure terminal, the message is written to the backout destination. If a backout destination has not been provided, an error message is issued and the node stops processing further input. The error message is bip4669.

If the message is caught by the JMSInput node after an exception has been thrown elsewhere in the message flow, the message is routed to the catch terminal. If you have not connected nodes to the catch terminal, the node will backout message for re-delivery until the problem is resolved or the backout threshold is reached.

When you include a JMSInput node in a message flow, the value that you set for *Transaction Mode* in Advanced properties defines whether messages are received under syncpoint. When messages are to be received under external syncpoint, additional configuration steps are required. These steps need only be applied the first time that a JMSOutput or JMSInput is deployed to the Broker for a particular JMS provider.

The Transaction Mode in the Advanced properties is used to define the transactional characteristics of how the message is handled.

•Select *none* if the incoming message is to be treated as non persistent.

•Select *local* if the JMSInput node should coordinate the commit or roll back of JMS messages received by the node, along with any other resources such as DB2 or WebSphere MQ that perform work within the message flow.

•Select *global* if the JMSInput node should participate in a global message flow transaction that will be managed by the broker's external syncpoint coordinator. The syncpoint coordinator is the broker's queue manager on distributed platforms and RRS (Resource Recovery Services) on z/OS®.

JMSOutput node configuration

- Send JMS Message to destination
  - Point to Point or Pub/Sub message producer
  - Responsible for creating JMS output message from JMS tree
- Basic properties
  - Similar to JMSInput node Basic, but values can be different!
    - For example, different providers
- Point to Point (Destination Queue) or Pub/Sub (Topic)
- Request properties
  - Determines whether output message is a Datagram, Request or Reply
- Advanced properties
  - New Correlation ID
  - Transaction Mode: Global, Local, None
  - Delivery Mode
  - Message Expiration in milliseconds
  - Message Priority 0 (hi) to 9 (low)
- Validation

Use the JMSOutput node to send messages to JMS destinations. The JMSOutput node acts as a JMS message producer and can publish all six message types that are defined in the JMS v1.1 specification. Messages are published by using method calls, which are described in the JMS specification.

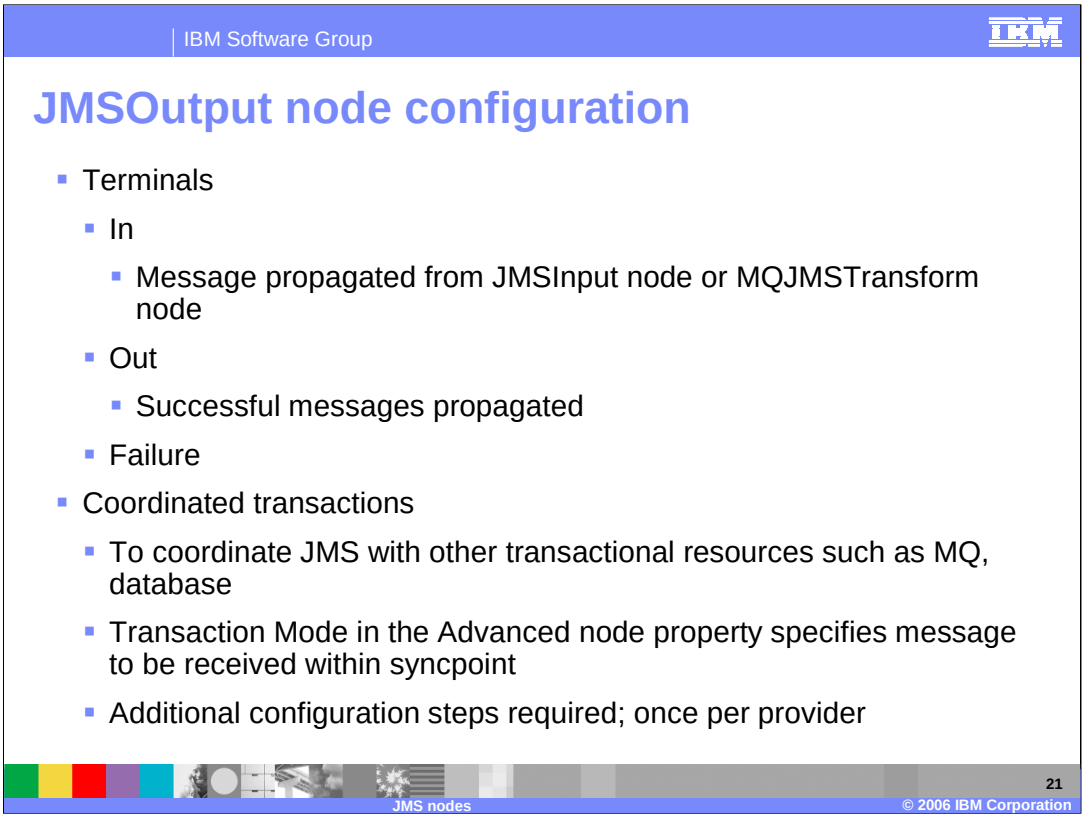
The JMSOutput node acts as a message producer and supports the following message scenarios:
Sending a datagram message
Sending a reply message
Sending a request message

The Basic properties are required and supply:
•Initial Context Factory (name of JMS provider)
•Location JNDI bindings
•Connection Factory Name

If the JMSOuput node is to be used to publish a topic, select *Pub/Sub* in the properties dialog navigator and enter the name of the *Publisher Topic*.
If the JMSOutput node is to be used to send point to point messages then select *Point to Point* in the properties dialog navigator and enter the *Destination Queue* name for the JMS queue name that is listed in the bindings file.

The Request property is used to supply a destination for a requested reply. The default value is *Destination Name*. If this is selected, the message is treated as a request of a datagram and it targets either the *Publication Topic* or the *Destination Queue* . If the message is to be treated as a reply, select *Reply Destination Name* and provide a value for *Reply To Destination*. The *Reply To Destination* is the name of the JMS destination to which the receiving application should send a reply message. The default value is blank, in which case the JMS output message can be regarded as a datagram. If the field is blank, the JMSOutput node does not expect a reply from the receiving JMS client.

The Advanced properties provide the capability to specify:
•A New Correlation ID (allows a link to an MQOutput node)
•Transaction Mode
•Delivery Mode (persistence)
•Message Expiration (in milliseconds)
•Message Priority (0 to 9)



On the JMSOutput node, there are three terminals: In, Out and Failure.

Connect the in terminal of the JMSOutput node to the node from which outbound messages are routed. Connect the out terminal of the JMSOutput node to another node in the message flow if you want to process the message further, to process errors, or to send the message to an additional destination.

As with the JMSInput node, the transaction mode values can be set to none, local or global. These can be set independently of the values specified in the JMSInput node.

JMSMQTransform node
MQJMSTransform node

- To simplify MQ interoperation with any JMS provider
  - Add JMSInput, JMSOutput nodes to existing MQ based flows
- Converts between JMS canonical and MQ specific formats
  - Transforms JMS tree to MQRFH2 format
- Bidirectional
  - MQ->JMS
  - JMS->MQ
- Terminals
  - In
    - Input JMS/MQ tree
  - Out
    - Transformed MQ/JMS tree
  - Failure
    - Transformation failed

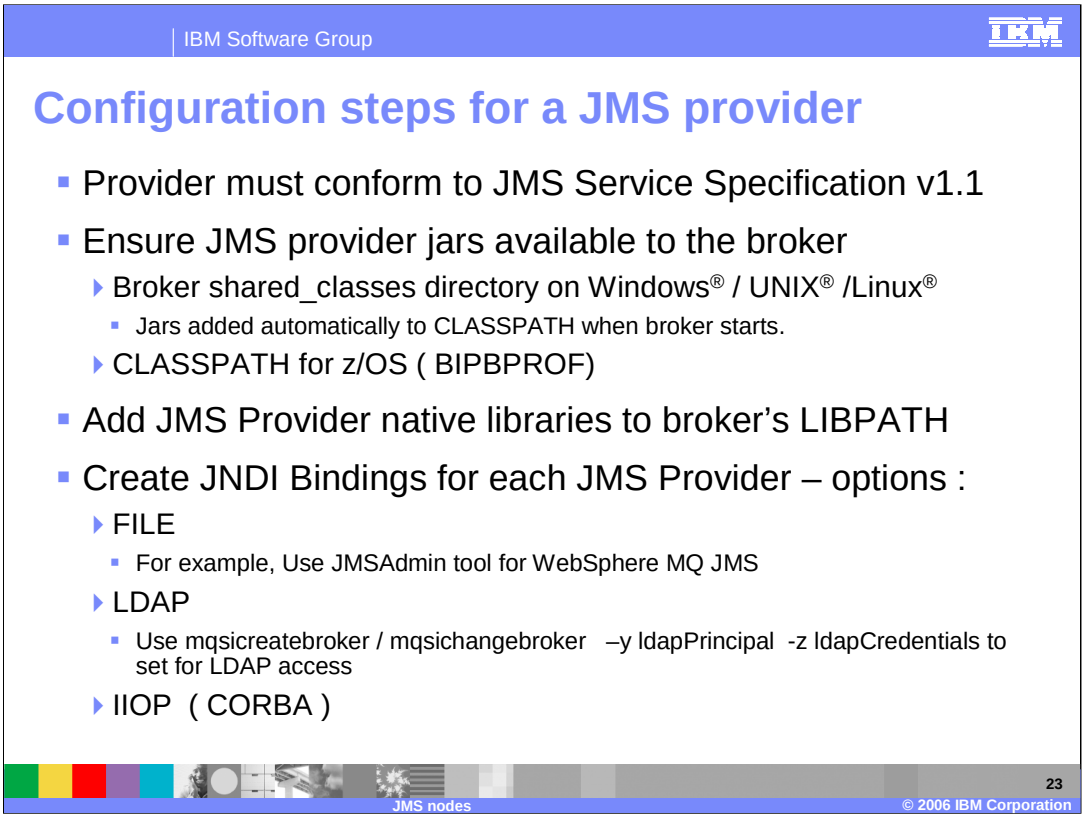
Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a message tree structure compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

The JMSMQTransform node can be used to send messages to legacy message flows and to interoperate with WebSphere MQ JMS and WebSphere Event Broker publish subscribe.

The MQJMSTransform performs this transformation in the opposite direction. Use the MQJMSTransform node to receive messages that have a WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

The MQJMSTransform node can be used to send messages to legacy message flows and to interoperate with WebSphere MQ JMS and WebSphere Event Broker publish subscribe.

These nodes require no configuration, apart from connecting the input and output terminals.

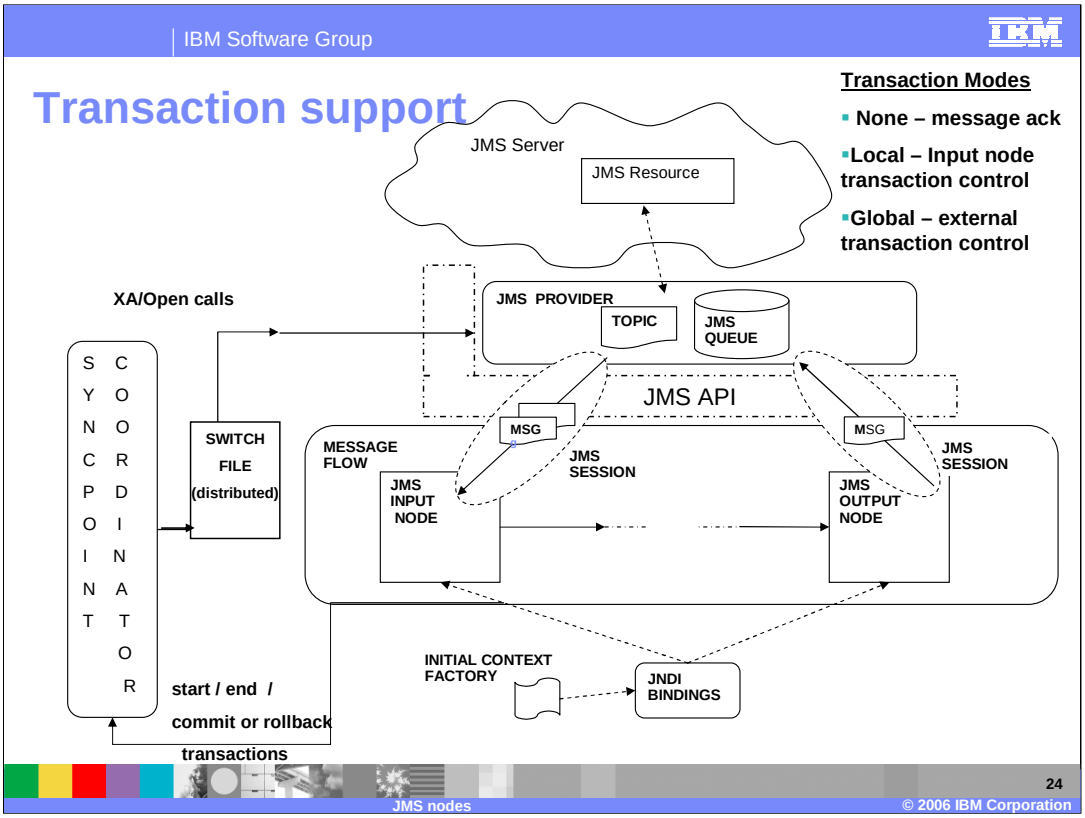


Additional configuration is required to enable global transaction support for the JMSInput and JMSOutput node. Any JMS provider that conforms to the Java Message Service Specification, version 1.1 and that supports the JMS XAResource API through the JMS session can be used if transaction coordination is required. If the message designer has specified a non XA compliant provider, the non transactional mode only is supported.

These steps must be completed for each JMS provider, but not for each flow using that JMS provider. The JMS provider can supply additional jar files that are required for transactional support. Refer to the JMS provider documentation for details. For distributed platforms, copy the java .jar files and any native libraries for the JMS provider client into a the broker shared-classes directory. For example, on Windows C:\Documents and Settings\All Users\Application Data\IBM\MQSI\shared-classes. This ensures that the java class path for the JMS nodes is set correctly.

For z/OS, there is no shared-classes directory. Instead you must specify each JMS provider java .jar file in the class path in the BIPPROF member of the broker's PDS (Partitioned Data Set). Then update the LIBPATH with any native libraries, and submit the BIPGEN JCL job to update the broker ENVFILE.

JNDI bindings must be created for each JMS provider. For example, the MQ JMS provider supplied a tool called JMS-Admin. Other providers have their own method of creating JNDI administered objects. Each has different ways of storing the JNDI data. This can either be file based, which is the simplest form, or it can be stored within an LDAP directory. If you are using LDAP you may need to specify the security credentials. This can be done by specifying the LDAP user, the LDAP principle and the password, passing these parameters as options on the "MQSICreateBroker" or "MQSIChangeBroker" commands. Alternatively, you can use CORBA, over IIOP.

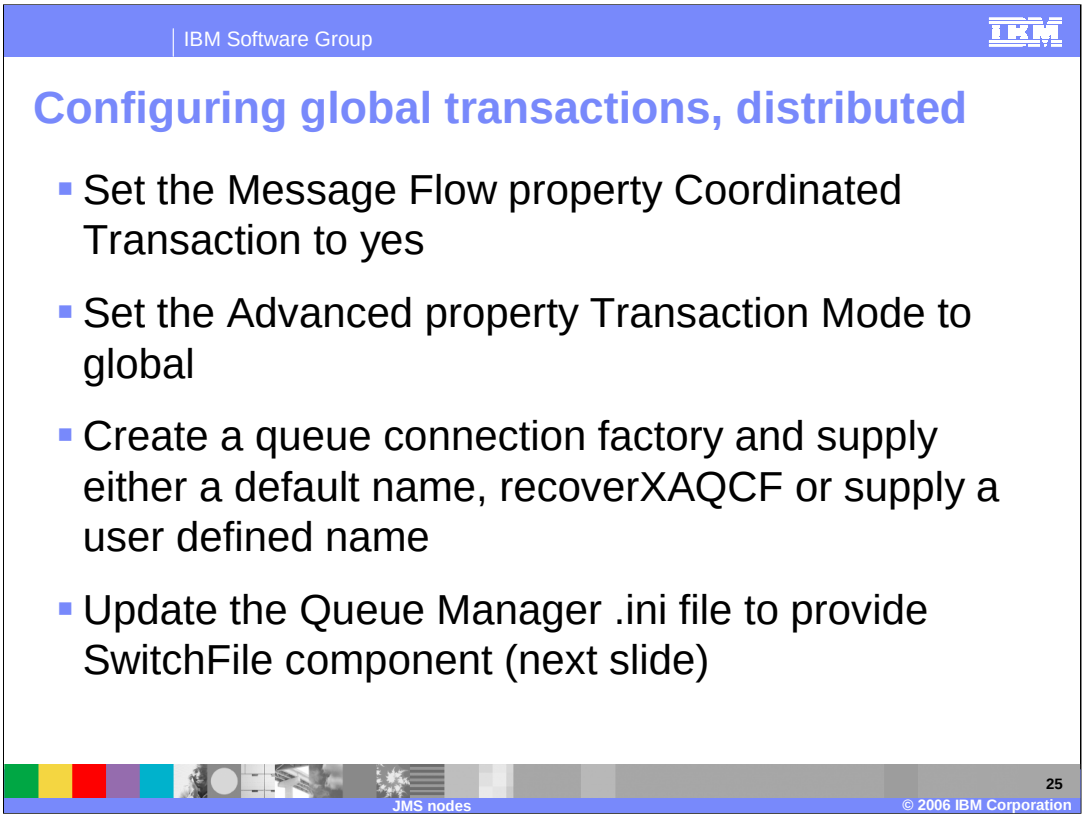


In this diagram, messages are consumed from a topic by a JMSInput node, and are produced to a JMS queue JMSOutput node. The nodes are connected with and are in session with a JMS provider. Any message flow input node can inform the external Syncpoint Coordinator when a message flow transaction starts and ends, and whether any resources that have been touched by the flow should be committed or rolled back.

The Syncpoint Coordinator sends XA/Open compliant requests to all participating Resource Managers to inform them to prepare. Any changes are then either committed or rolled back. Resource Managers such as WebSphere MQ, DB2 and any XA compliant JMS provider can participate in a global transaction. The external Syncpoint Coordinator is WebSphere MQ on distributed platforms, and RRS (Resource Recovery Services) on z/OS.

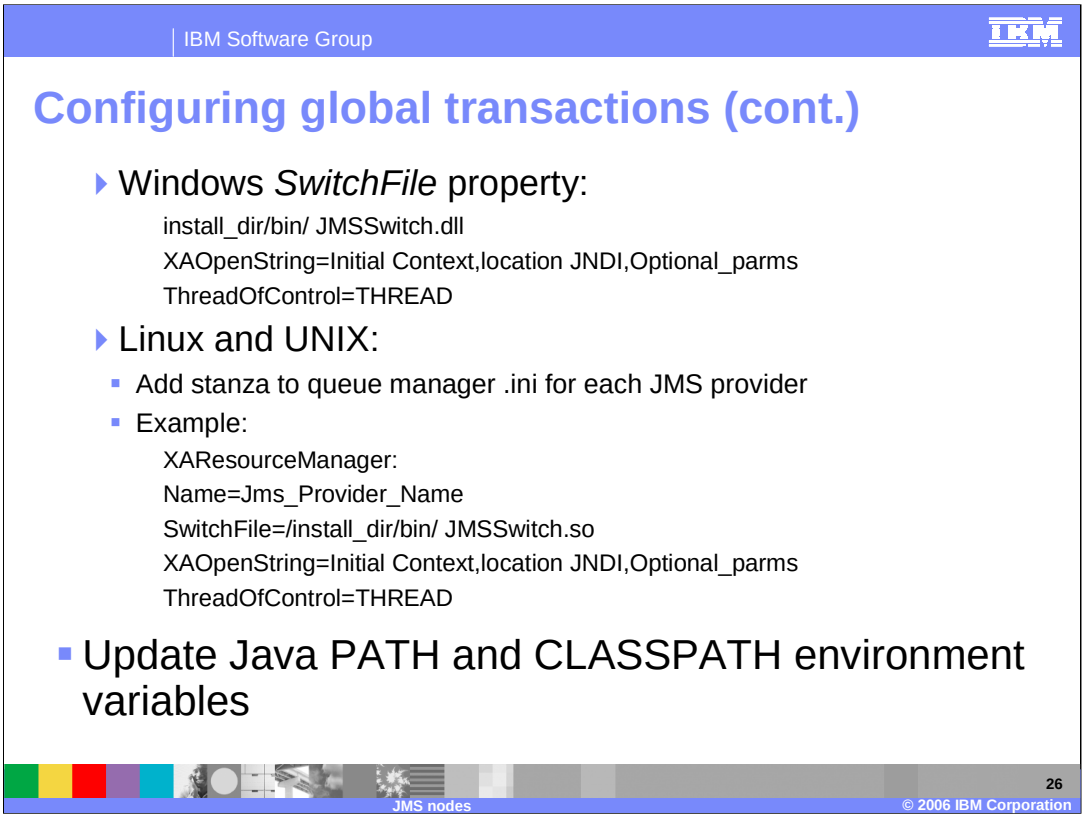
The JMSInput node and JMSOutput node can participate in a global transaction only if the JMS provider to which they connect supports the XA/Open interface through the JMS XAResource Class. An example JMS provider is the WebSphere MQ Java Client.

On distributed platforms an additional component called the ‘Switch File’ is needed for Global transactions. The Switch file is a shared library (a DLL on Windows). When the broker's WebSphere MQ queue manager starts up, it loads the Switch file. The Switch file forwards XA/Open transaction calls from the Syncpoint Coordinator to the JMS Provider. This ensures that the JMS resources that participate in the transaction can be coordinated in synchronization with other Resource managers that are involved in the same transaction.



Additional configuration is required to enable global transaction support for the JMSInput and JMSOutput node. Before deploying a message flow that contains JMS nodes complete the following steps:

1. Set the Message Flow property *Coordinated Transaction* to *yes*.
2. For each JMSInput or JMSOutput node that is required to participate in the global transaction, set the Advanced property *Transaction Mode* to *global*.
3. Create a queue connection factory and supply either a default name, *recoverXAQCF* or supply a user defined name.
4. On distributed platforms, the external syncpoint coordinator for the broker is WebSphere MQ. Before you deploy a message flow in which the *Transaction Coordination* is set to Global, modify the queue manager .ini file to include extra definitions for each JMS provider Resource Manager that participates in globally coordinated transactions.



On Windows, set the *SwitchFile* property to the value shown here.

On Linux and UNIX platforms, add a stanza to the queue manager ini file for each JMS provider as in this example.

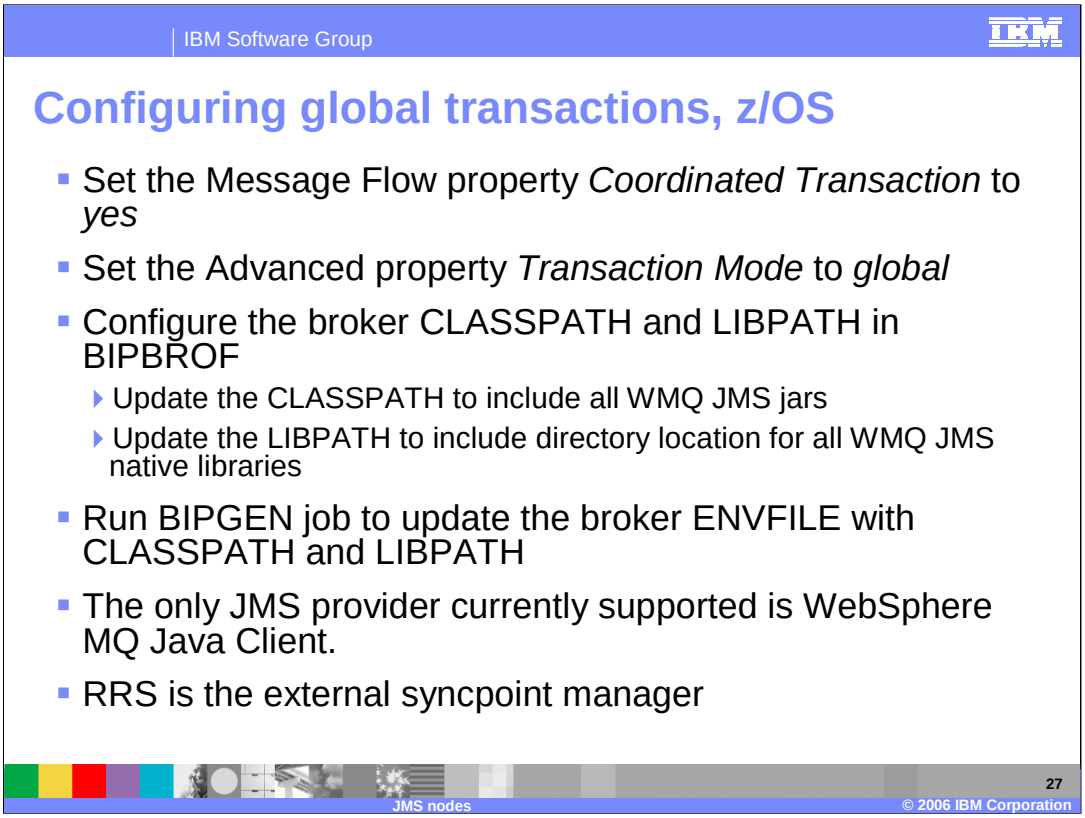
5. The JMS provider can supply additional jar files that are required for transactional support.

Update the Java CLASSPATH environment variable for the broker's queue manager to include a reference to xarecovery.jar. For example:

install_dir/classes/xarecovery.jar

Update the Java PATH environment variable for the broker's queue manager to point to the bin directory, in which the Switch File is located. For example

install_dir/bin



As on distributed platforms, the message flow properties for coordinated transactions and global transactions must be set.

Specify each JMS provider java .jar file in the class path in the BIPPROF member of the broker's PDS (Partitioned Data Set). Then update the LIBPATH with any native libraries, and submit the BIPGEN JCL job to update the broker ENVFILE.

On z/OS, the external syncpoint manager is Resource Recovery Services (RRS). The only JMS provider that is supported on z/OS is WebSphere MQ JMS. The only Transport option that is supported for WebSphere MQ JMS on z/OS is the Bind option. Syncpoint control for the JMS provider is managed with RRS syncpoint coordination of the queue manager of the broker. You do not need to modify the .ini file.

Section

Debugging

This section covers debugging.

Debugging (1/7)

- BIP2211
  - Configuration error at deployment time
  - Incompatible to configure a JMS node for both Topic and Source Queue .
  - Not valid to configure node property “Transaction Mode”=‘global’ when message flow property “Coordinated Transaction” = ‘no’.
- User Action :
  - Modify properties and redeploy
- NOTE: In the Service Trace the JMS nodes are called JMSClientInputNode and JMSClientOutputNode and not JMSInput and JMSOutput .

The primary debugging tools are ‘service trace’ and the ‘event log’. Note that in the service trace the JMS nodes are called JMSClientInputNode and JMSClientOutputNode.

If there is any configuration error at run-time, you will receive the standard BIP2211 message. To recover from this, modify the appropriate properties which should be indicated in the message and redeploy the message flow.

Debugging (2/7)

- BIP4640
  - Usually found on first deployment – but can occur whenever JNDI bindings changed
  - JNDI is unable to obtain the Initial Context, or a Connection Factory, or a JMS Destination
- User Action :
  - Check that the JNDI bindings contain definitions for the names of the Connection Factories and Destinations (Topic or Queues) specified in the JMS node properties
  - Check that the JNDI bindings are in the location specified in the node property
  - If The JNDI administered objects are stored in LDAP check that the LDAP Principal and LDAP Credentials have been configured using mqsicreatebroker or mqsichangebroker command
  - If still having problems then collect Service Trace . Search for the string “lookUpFactoryInJNDI()“ in the trace. This will show the JNDI Exception message
  - Check System Event Log for details of the JNDI Exception message
  - Refer to the vendor documentation for the JMS provider on how to create the JNDI administered objects

If there is anything wrong with the JNDI administered objects, you may receive a BIP4640 message. This slide gives some options to correct this.

Debugging (3/7)

- BIP4643
  - Error receiving a JMS message in the JMSInput node
- User Action :
  - Check the broker event log message for any associated JMS provider exception message.
  - Collect broker service trace and look for the string BIP4643 for any JMS Provider exception message and return codes.
  - Check the vendor documentation for the JMS provider to determine the cause of the provider error code.
  - Verify that the JMS message at the input Destination is correctly formed.

If there is an error receiving the JMS message in the JMSInput node, you get a BIP4643 failure. This may be as a result of a failure outside the broker environment. Refer to the vendors documentation for further problem diagnosis information.

Debugging (4/7)

- BIP4651
  - Error sending a JMS message in the JMSOutput node
- User Action :
  - Check the broker event log message for any associated JMS Provider exception message.
  - Collect broker service trace and look for the string “sendOutputMessage()“ to look for any JMS provider exception message and return codes.
  - Check the vendor documentation for the JMS provider to determine the cause of the Provider error code.
  - Verify that any node that modifies or creates the JMS message in the flow is correctly formatting elements in the JMS message tree.

If there is an error sending the JMS message in the JMSOutput node, you get a BIP4651.

Debugging (5/7)

- BIP4648
  - Unable to connect to the JMS provider at deploy time or start-up, or when a JMS provider becomes unavailable.
  - This message will be output approximately every 30 minutes while the JMS node attempts to connect (or reconnect) to the JMS provider.
- User Action :
  - Check that the JMS Provider is running and accessible to the broker.
  - Check the broker event log message for any associated JMS Provider exception message.
  - Collect broker service trace and look for the string “(((JMSPROVIDER)))” to look for more detailed JMS Provider exception message and return codes.
  - Check the vendor documentation for the JMS Provider to determine the cause of the Provider error code.

If you are unable to connect to the JMS provider, you see a BIP 4648 message telling you that the node is unable to connect to the JMS provider. Perhaps the JMS provider is out of action. No user action is required other than to try and restart the provider and the node will continue to retry connecting.

Debugging (6/7)

- BIP4655 through BIP4664
  - Problems obtaining data from a JMS message in the JMSInput node or creating a JMS message in the JMSOutput node.
- User Action :
  - Check the broker event log message for any associated JMS Provider exception message.
  - Collect broker service trace and look for the string “JMSClient” to search for more detailed JMS Provider exception message and return codes associated with the BIP message number.
  - Check the vendor documentation for the JMS Provider to determine the cause of the Provider error code.

If there are any problems with the data from a message as it is being processed through a message flow, there are a suite of messages, BIP4655 through to BIP4664.

Debugging (7/7)

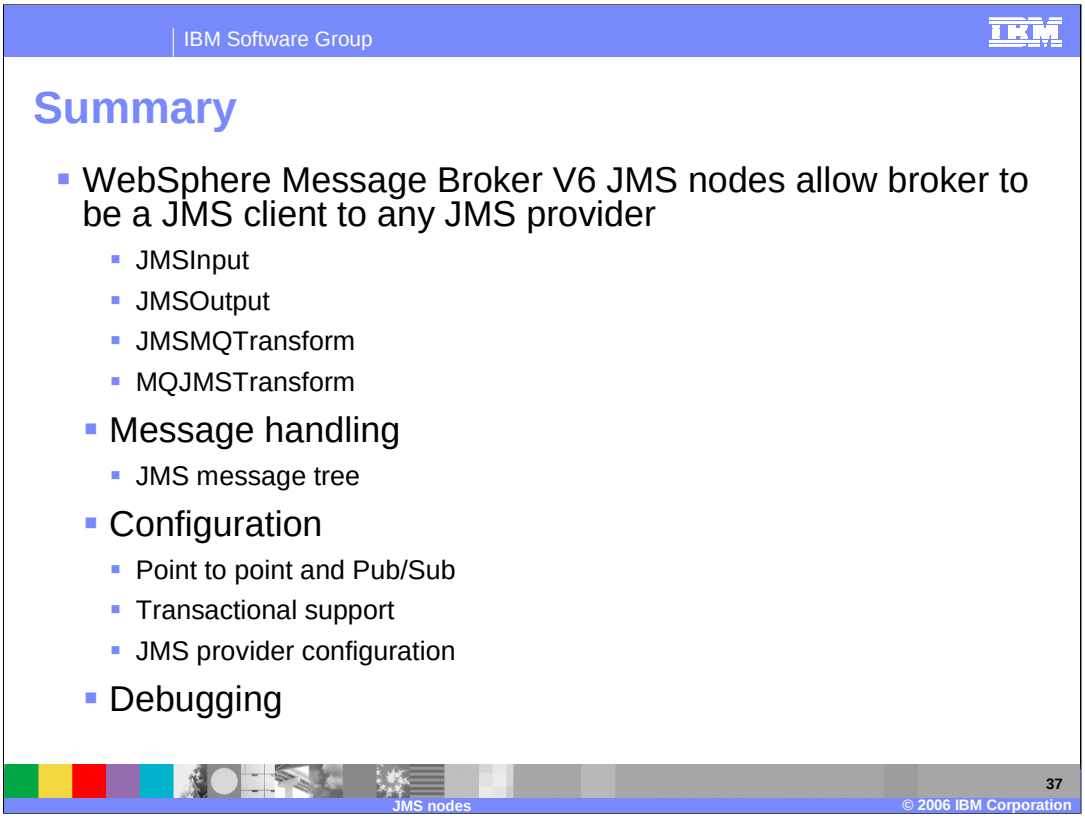
- BIP4669
  - There has been a failure in the message flow when trying to process a JMS Message received by a JMSInput node, so the input node does not commit the receipt of the message , which therefore remains on the input Destination.
  - The JMSInput node attempts to receive the message, again and finds that the Delivery count (JMSXDeliveryCount message property) exceeds the value in the “Backout Threshold” node property, and attempts to send the Message to the Destination defined in the “Backout Destination” node property.
  - A Backout JMS destination has not been specified in the JMSInput node or has not been defined in the JNDI administered objects
- User Action :
  - Confirm that a Backout Destination has been specified in the JMSInput node
  - Conform that the Backout Destination has been defined in the JNDI administered objects (JNDI bindings)
  - Collect broker service trace and look for the string “JMSClientInput” to search for more detailed JMS Provider exception message and return codes associated with the BIP message number.

If you have not provided the backout destination to handle failed messages, then you will receive a BIP4669 message, which will tell you that you need to configure that destination.

Section

Summary and references

This section contains a summary and references.



This module presented the new capabilities of JMS processing within the broker. The four new nodes allow the broker to be a JMS client to any JMS provider. The JMS message tree can be read in, transformed, and put out. Transactional support is available. JMS provider and the JMS nodes configuration details were provided.

References

- WebSphere Message Broker library:

http://www-306.ibm.com/software/integration/wbimessagebroker/library/

- WebSphere Message Broker Information Center:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere z/OS

Windows, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, JMS, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.