

## MQ 4.1 Protocol - proposed

Version 4.1.0

@(#)protocol.html 1.4 05/01/18

The following is a summary of the packets exchanged between an JSMQ client and server. The first column specifies the direction the packet is sent (client to server or server to client) and the third is the packet that is sent. The second column gives the numeric value for this packets type (used in the packet type field). Click on the packets to get the full specification for that packets.

All packets are based on the S1MQ packet format .

This document does not address administrative messages. These are addressed in a seperate document.

This document does not address server to server protocol.

### Versions

Starting with 3.5 we version the protocol to be the same as the first S1MQ release the protocol was supported in. In earlier releases the protocol was versioned sperately.

MQ Version	Packet Version	Protocol Version
<b>4.1 (Harrier)</b>	3.0.1	<b>4.1.0</b>
<b>4.0 (Hawk)</b>	3.0.1	<b>4.0.0</b>
3.6 (Shrike)	3.0.1	3.6.0
3.5 (Raptor)	3.0.1	3.5.0
3.0.1	3.0.1	2.0.0
3.0 (Falcon)	2.0.0	2.0.0
2.0 (Swift)	1.0.3	1.0.0

The last column in the table below lists the version of S1MQ that the packet was first introduced. Versions in parenthesis are releases that the packet content was modified (e.g. a new property added). For more information about exactly what changed see the specific dscription for a packet.

If at any place in this specification, a "Since" field is missing, assume "Since MQ2.0".

Packets, properties, etc. that are deprecated and are supported only for backwards compatibility are noted with **Deprecated**.

Some packets have changes made in 4.0. Some packets have proposed changes for 4.1.

All \*\_REPLY packets have a `JMQStatus` integer property and an optional `JMQReason` string property. The possible values of `JMQStatus` is specified in each \*\_REPLY packet description. If `JMQReason` is set on a \*\_REPLY packet then it contains text that gives more informtion about the error that occured.

Often it matches the message that is logged to the server's log file when the error (or condition) occurs.

<b>Client sends JMS message</b>			
C -> S	1	TEXT_MESSAGE	MQ 2.0 (4.0)
C -> S	2	BYTES_MESSAGE	MQ 2.0 (4.0)
C -> S	3	MAP_MESSAGE	MQ 2.0 (4.0)
C -> S	4	STREAM_MESSAGE	MQ 2.0 (4.0)
C -> S	5	OBJECT_MESSAGE	MQ 2.0 (4.0)
C -> S	6	MESSAGE	MQ 2.0 (4.0)

<b>Broker acknowledges receipt of message</b>			
S -> C	9	SEND_REPLY	MQ 2.0 (4.0)

<b>broker delivers JMS message to client</b>			
S -> C	1	TEXT_MESSAGE	MQ 2.0
S -> C	2	BYTES_MESSAGE	MQ 2.0
S -> C	3	MAP_MESSAGE	MQ 2.0
S -> C	4	STREAM_MESSAGE	MQ 2.0
S -> C	5	OBJECT_MESSAGE	MQ 2.0
S -> C	6	MESSAGE	MQ 2.0

<b>Acknowledge receipt of message</b>			
C -> S	24	ACKNOWLEDGE	MQ 2.0, (3.0) (4.0)
S -> C	25	ACKNOWLEDGE_REPLY	MQ 2.0 (4.0),(4.1)

<b>Register consumer</b>			
C -> S	14	ADD_CONSUMER	MQ 2.0, (3.0, 3.5, 4.0)
S -> C	15	ADD_CONSUMER_REPLY	MQ 2.0, (3.0, 3.5)

<b>Add a message producer to a destination</b>			
C -> S	18	ADD_PRODUCER	MQ 2.0 (3.5, 4.0)
S -> C	19	ADD_PRODUCER_REPLY	MQ 2.0 (3.5)

<b>Authenticate if user can establish connection</b>			
S -> C	38	AUTHENTICATE_REQUEST	MQ 2.0
C -> S	12	AUTHENTICATE	MQ 2.0 (4.0)
S -> C	13	AUTHENTICATE_REPLY	MQ 2.0

<b>Browse queue or get metrics</b>			
C -> S	26	BROWSE	MQ 2.0 (3.5)
S -> C	27	BROWSE_REPLY	MQ 2.0 (3.0)

<b>Client requests a destination be created</b>			
C -> S	34	CREATE_DESTINATION	MQ 2.0 (4.0)
S -> C	35	CREATE_DESTINATION_REPLY	MQ 2.0 (3.0)

<b>Client requests a session be created</b>			
C -> S	68	CREATE_SESSION	MQ 3.5 (4.0)
S -> C	69	CREATE_SESSION_REPLY	MQ 3.5

<b>Recreate consumer</b>			
S -> C	80	RECREATE_CONSUMER	MQ 4.1
S -> C	17	DELETE_CONSUMER_REPLY	MQ 2.0

<b>Remove a message producer from a destination</b>			
C -> S	66	DELETE_PRODUCER	MQ 3.5 (4.0)
S -> C	67	DELETE_PRODUCER_REPLY	MQ 3.5

<b>Request delivery of a list of messages (typically from a BROWSE_REPLY)</b>			
C -> S	42	DELIVER	MQ 2.0 (3.0)
S -> C	43	DELIVER_REPLY	MQ 2.0

<b>Client requests a destination be destroyed</b>			
C -> S	36	DESTROY_DESTINATION	MQ 2.0 (4.0)
S -> C	37	DESTROY_DESTINATION_REPLY	MQ 2.0

<b>Client requests a session be destroyed</b>			
C -> S	70	DESTROY_SESSION	MQ 3.5 (4.0)
S -> C	71	DESTROY_SESSION_REPLY	MQ 3.5

**Broker sends asynchronous error notification to client**

S -> C	30	ERROR	MQ 2.0
--------	----	-------	--------

**Request/Send status information**

C -> S	78	INFO_REQUEST	MQ 4.0
S -> C	79	INFO	MQ 4.0

**Client requests license information from the server**

C -> S	76	GET_LICENSE	MQ 3.6 (4.0)
S -> C	77	GET_LICENSE_REPLY	MQ 3.6

**Client requests a Unique ID from the server**

C -> S	62	GENERATE_UID	MQ 3.0
S -> C	63	GENERATE_UID_REPLY	MQ 3.0

**Client notifies broker it is going away**

C -> S	28	GOODBYE	MQ 2.0 (4.0)
S -> C	29	GOODBYE_REPLY	MQ 2.0 (3.5, 4.0)

**Broker notifies client it is going away**

S -> C	28	GOODBYE	MQ 2.0 (4.0)
--------	----	---------	--------------

**Establish a connection to broker**

C -> S	10	HELLO	MQ 2.0 (3.0, 3.5, 4.0)
S -> C	11	HELLO_REPLY	MQ 2.0 (3.0) (4.0)

**Check connection**

S -> C	54	PING	MQ 3.0
C -> S	54	PING	MQ 3.5
S -> C	55	PING_REPLY	MQ 3.5

**Client requests messages to be redelivered**

C -> S	32	REDELIVER	MQ 2.0 (3.0) (4.1)
--------	----	-----------	--------------------

<b>Flow Control</b>			
C -> S	52	RESUME_FLOW	MQ 2.0 (3.0, 3.5, <b>4.0</b> )
S -> C	52	RESUME_FLOW	MQ 3.0 (3.5, <b>4.0</b> )
C -> S	64	FLOW_PAUSED	MQ 3.0 (3.5, <b>4.0</b> )

<b>Verifying ClientID</b>			
C -> S	50	SET_CLIENTID	MQ 2.0 (3.5, <b>4.0</b> )
S -> C	51	SET_CLIENTID_REPLY	MQ 2.0 (3.5)

<b>Request broker to start message delivery</b>			
C -> S	20	START	MQ 2.0 (3.5, <b>4.0</b> )

<b>Request broker to suspend message delivery</b>			
C -> S	22	STOP	MQ 2.0 (3.5, <b>4.0</b> )
S -> C	23	STOP_REPLY	MQ 2.0

<b>Verify a destination</b>			
C -> S	40	VERIFY_DESTINATION	MQ 2.0
S -> C	41	VERIFY_DESTINATION_REPLY	MQ 2.0 (3.0)

<b>Transactions</b>			
C -> S	44	START_TRANSACTION	MQ 2.0 (3.0, 3.5) ( <b>4.0</b> )
S -> C	45	START_TRANSACTION_REPLY	MQ 2.0 (3.0) ( <b>4.0</b> )
C -> S	46	COMMIT_TRANSACTION	MQ 2.0 (3.0) ( <b>4.0</b> )
S -> C	47	COMMIT_TRANSACTION_REPLY	MQ 2.0 (3.0) ( <b>4.0</b> ), ( <b>4.1</b> )
C -> S	48	ROLLBACK_TRANSACTION	MQ 2.0 (3.0) ( <b>4.0</b> )
S -> C	49	ROLLBACK_TRANSACTION_REPLY	MQ 3.0 ( <b>4.0</b> )
C -> S	58	END_TRANSACTION	MQ 3.0 ( <b>4.0</b> )
S -> C	59	END_TRANSACTION_REPLY	MQ 3.0 ( <b>4.0</b> )
C -> S	56	PREPARE_TRANSACTION	MQ 3.0 ( <b>4.0</b> )
S -> C	57	PREPARE_TRANSACTION_REPLY	MQ 3.0 ( <b>4.0</b> ), ( <b>4.1</b> )
C -> S	60	RECOVER_TRANSACTION	MQ 3.0 ( <b>4.0</b> )
S -> C	61	RECOVER_TRANSACTION_REPLY	MQ 3.0 ( <b>4.0</b> )
C -> S	80	VERIFY_TRANSACTION	<b>MQ 4.0</b>
S -> C	81	VERIFY_TRANSACTION_REPLY	<b>MQ 4.0</b>

## Examples

Client starts, sends two non-persistent TextMessages to a Topic, and exits:

- C -> S HELLO
- S -> C HELLO\_REPLY
- S -> C AUTHENTICATE\_REQUEST
- C -> S AUTHENTICATE
- S -> C AUTHENTICATE\_REPLY
- C -> S CREATE\_DESTINATION
- S -> C CREATE\_DESTINATION\_REPLY
- C -> S ADD\_PRODUCER
- S -> C ADD\_PRODUCER\_REPLY
- C -> S TEXT\_MESSAGE
- C -> S TEXT\_MESSAGE
- C -> S GOODBYE

Same as above but client sends two persistent TextMessages (which require acknowledgement from server):

- ...
- C -> S TEXT\_MESSAGE
- S -> C SEND\_REPLY
- C -> S TEXT\_MESSAGE
- S -> C SEND\_REPLY
- ...

Client subscribes to a topic. Server delivers two persistent text message to the client. Client is using JMS auto acknowledge mode so it acknowledges each message. Client has requested server acknowledgements (of the ACKNOWLEDGE messages) so server sends a reply for each acknowledgement. Note that the two TEXT\_MESSAGES are sent to the client before the first message is acknowledged. This may or may not be the case depending on timing.

- ...
- C -> S ADD\_CONSUMER
- S -> C ADD\_CONSUMER\_REPLY
- C -> S START
- S -> C TEXT\_MESSAGE
- S -> C TEXT\_MESSAGE
- C -> S ACKNOWLEDGE
- S -> C ACKNOWLEDGE\_REPLY
- C -> S ACKNOWLEDGE
- S -> C ACKNOWLEDGE\_REPLY
- C -> S GOODBYE

Client browses a Queue that has two MapMessages:

- ...

- C -> S VERIFY\_DESTINATION
- S -> C VERIFY\_DESTINATION\_REPLY
- C -> S BROWSE
- S -> C BROWSE\_REPLY
- C -> S DELIVER
- S -> C DELIVER\_REPLY
- S -> C MAP\_MESSAGE
- S -> C MAP\_MESSAGE (with L bit set)
- ...

---

```

TEXT_MESSAGE: C -> S
BYTES_MESSAGE: C -> S
MAP_MESSAGE: C -> S
STREAM_MESSAGE: C -> S
OBJECT_MESSAGE: C -> S
MESSAGE: C -> S

```

Send a JMS message (TextMessage, BytesMessage, MapMessage, StreamMessage, ObjectMessage, or Message) to the server.

**Since:** MQ2.0

**Reply:** SEND\_REPLY (optional)

Fixed Header Fields	
Field	Value
packet type	1 (TEXT_MESSAGE), 2 (BYTES_MESSAGE), 3 (MAP_MESSAGE), 4 (STREAM_MESSAGE), 5 (OBJECT_MESSAGE), 6 (MESSAGE)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
P Bit	1 = persistent message 0 = non persistent message
Q Bit	1 = destination is a queue 0 = destination is not a queue
A Bit	1 = Have server send an SEND_REPLY acknowledgement for this message 0 = Server does not return an acknowledgement
F Bit	1 = Connection Flow paused 0 = Connection Flow not paused The F Bit is set on the last JMS message written to the connection by the client before the flow is paused. This bit informs the server that no more JMS messages will be sent to the server until the server sends a RESUME_FLOW packet to the client.

C Bit	<p>1 = Client Producer Flow paused  0 = Client Producer Flow not paused</p> <p>The C Bit is set on the last JMS message written to the connection by the client before the flow is paused due to producer based flow control limits. This bit informs the server that no more JMS messages will be sent to the server from a particular producer until the server sends a RESUME_FLOW packet to the client. If this bit is set, then the ProducerID field must also be set.</p>
R Bit	<p>1 = Client is overriding Redeliver bit to true  0 = Redeliver flag is controled by the broker</p> <p>In general, the R bit will not be set by the client. However, during failover, the client may override the R bit flag if a message was sent on a non-transacted session and the SEND_REPLY was not received before broker failure.</p>
I Bit	<p>Optional.</p> <p>1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed .  0 = this packet is new.</p>
All other bit flags	Not Applicable
priority	0-9 as defined in JMS specification
consumer ID	If A bit is set this should be a unique ID that the sender can use to help correlate the SEND_REPLY response packet. Whatever value is specified here is copied into the SEND_REPLY's consumer ID field by the server.

<b>Variable Header Fields</b>	
<b>String</b>	<b>Value</b>
transaction ID	If the packet is in a transaction, this ID is set to the transaction ID that was returned in the corresponding START_TRANSACTION packet.
Destination	Destination to send message to. Syntax defined by the JMQ URI specification for destinations.
MessageID	Optional JMSMessageID header.
CorrelationID	Optional JMSCorrelationID header.
ReplyTo	Optional JMSReplyTo header.
Type	Optional JMSType header.
ProducerID	Optional ProducerID. This is the ID of the producer that is sending the message. May be set on all messages. Must be set if the 'C' bit is set. This allows the broker to accurately respond with a RESUME_FLOW to the correct producer.

<b>Properties:</b>
MessageProperties specified by JMS client

<b>Packet Body</b>	
<b>Message Type</b>	<b>Description</b>
TEXT_MESSAGE	A UTF-8 encoded String. Specifically the byte stream generated by the String.getBytes(encoding) method. Not null terminated. Not padded.
BYTES_MESSAGE	A series of bytes in network byte order (this is the same byte order supported by Java).
MAP_MESSAGE	A serialized java.util.Hashtable
STREAM_MESSAGE	A series of bytes in network byte order (this is the same byte order supported by Java).
OBJECT_MESSAGE	A serialized Java object.
MESSAGE	No message body

**SEND\_REPLY: S -> C**

Server's reply to sending a JMS message.

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	9 (SEND_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

<b>Header Strings</b>
None

Properties:			
Property	Type	Value	Since
JMQStatus	Int	OK(200)	Send succeeded. Message has been processed by server.
		FORBIDDEN(403)	Client is unauthorized to send to destination
		NOT_FOUND(404)	The destination this message was sent to could not be found. <b>Since:</b> MQ3.6
		ENTITY_TOO_LARGE(423)	Message exceeds the single message size limit for the server or destination. <b>Since:</b> MQ3.6
		RESOURCE_FULL(414)	Destination is full and is rejecting new messages. <b>Since:</b> MQ3.6
		ERROR(500)	Send failed. Server had an internal error
		NOT_MODIFIED(304)	Message already exists on the broker. <b>Since:</b> MQ4.0.
		GONE(410)	Transaction associated with the message has been rolled back. <b>Since:</b> MQ4.0.
		TIMEOUT(408)	Transaction associated with the message has timed out. <b>Since:</b> MQ4.0.
		PRECONDITION_FAILED(412)	Transaction associated with the message is not in a STARTED state (and can not accept the message). <b>Since:</b> MQ4.0.
			MQ2.0 Modified MQ3.6 Modified MQ4.0

[Back to Summary](#)

---

```

TEXT_MESSAGE: S -> C
BYTES_MESSAGE: S -> C
MAP_MESSAGE: S -> C
STREAM_MESSAGE: S -> C
OBJECT_MESSAGE: S -> C
MESSAGE: S -> C

```

Server delivers a JMS message (TextMessage, BytesMessage, MapMessage, StreamMessage, or ObjectMessage) to a client.

**Since:** MQ2.0

**Reply:** ACKNOWLEDGE

Fixed Header Fields	
Field	Value
packet type	1 (TEXT_MESSAGE), 2 (BYTES_MESSAGE), 3 (MAP_MESSAGE), 4 (STREAM_MESSAGE), 5 (OBJECT_MESSAGE), 6 (MESSAGE)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
S bit	1 = Selectors were processed by server 0 = Selectors were not processed by server
R bit	1 = This message is being redelivered 0 = This message is being delivered for the first time
P Bit	1 = persistent message 0 = non persistent message
Q Bit	1 = destination is a queue 0 = destination is not a queue
F Bit	1 = Flow paused due to connection flow control 0 = Flow not paused The F Bit is set on the last JMS message written to the connection by the server before the flow is paused at the connectino level. This bit nforms the client that no more JMS messages will be sent to the connection until the client sends a RESUME_FLOW packet to the server.

C Bit	1 = Flow paused due to consumer flow control. 0 = Flow not paused. In MQ3.5 we introduced the concept of consumer flow control (instead of just connection flow control). If the C bit is set then the flow to the consumer has been paused (but not to the entire connection). Note that if both the C and F bit are set then the flow is paused at both the connection and consumer level.
A Bit	Not applicable. The client should always follow the JMS acknowledgement model
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
priority	0-9 as defined in JMS specification
consumer ID	The consumer ID specified in the JMQConsumerID property of the ADD_CONSUMER_REPLY message that caused this message to be delivered to the client.

<b>Variable Header Fields</b>	
<b>String</b>	<b>Value</b>
Destination	Destination message was sent to. Syntax defined by the JMQ URI specification for destinations.
MessageID	Optional JMSMessageID header as set by originator.
CorrelationID	Optional JMSCorrelationID header as set by originator.
ReplyTo	Optional JMSReplyTo header as set by originator.
Type	Optional JMSType header as set by originator.

<b>Properties:</b>
MessageProperties as set by originator.

<b>Packet Body</b>	
<b>Message Type</b>	<b>Description</b>
TEXT_MESSAGE	A UTF-8 encoded String. Specifically the byte stream generated by the <code>String.getBytes(encoding)</code> method. Not null terminated. Not padded.
BYTES_MESSAGE	A series of bytes in network byte order (this is the same byte order supported by Java).
MAP_MESSAGE	A serialized <code>java.util.Hashtable</code>
STREAM_MESSAGE	A series of bytes in network byte order (this is the same byte order supported by Java).
OBJECT_MESSAGE	A serialized Java object.
MESSAGE	No message body.

[Back to Summary](#)

---

## ACKNOWLEDGE: C -> S

Acknowledge receipt of one or more messages. This is used by the client to acknowledge receipt of messages using the JMS message acknowledgement model. The ACKNOWLEDGEMENT can optionally request a reply so that the client can verify the server has processed the ACKNOWLEDGEMENT.

With the `JMQAckType` property the client can indicate that the message should be considered "dead" or undeliverable as opposed to acknowledged. Dead messages are handled differently by the system -- for example they may be placed on a Dead Message Queue instead of removed.

**Since:** MQ2.0

**Modified:** MQ3.0, MQ3.6, MQ4.0

**Reply:** ACKNOWLEDGE\_REPLY (optional)

Fixed Header Fields	
Field	Value
packet type	24 (ACKNOWLEDGE)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A Bit	1 = Have server send an ACKNOWLEDGE_REPLY acknowledgement for this packet 0 = Server does not return a reply
I Bit	<b>Unsupported.</b> It is an error to set the I bit on an ACKNOWLEDGEMENT, acknowledgements can not be handled in an indempotent way.
All other bits	Not Applicable
consumer ID	If A bit is set this should be a unique ID that the sender can use to help correlate the ACKNOWLEDGE_REPLY response packet. Whatever value is specified here is copied into the ACKNOWLEDGE_REPLY's consumer ID field by the server.
priority	Not Applicable

Variable Header Fields	
transaction ID	If the packet is in a transaction, this ID is set to the transaction ID returned by the corresponding START_TRANSACTION message.

Properties:

Property	Type	Value	Since
JMQBodyType	Integer	<p>(Optional) Format of packet's body. One of</p> <ul style="list-style-type: none"> <li>● CONSUMERID_I_SYSMESSAGEID(1): Body is a series of marshalled consumer ID / SysMessageIDs with the consumer ID an 'int'. This was the case in V1.0.0 of the protocol</li> <li>● CONSUMERID_L_SYSMESSAGEID(2): Same as the above, but consumer ID is a 'long'.</li> </ul> <p>If not specified and the client is using protocol version 1.0.0 then assume CONSUMERID_I_SYSMESSAGEID(1), else assume CONSUMERID_L_SYSMESSAGEID(2).</p> <p>See BodyType for more information about body types.</p>	<p>MQ3.0.</p> <p><b>Note:</b>In 3.0 this was never used. The broker simple used the client version to determine which body type was being used.</p>
JMQQuantity	Integer	<p>(Optional) Number of acknowledgement blocks in the body. If this parameter is missing then you must compute the number of blocks by dividing the bodysize by the appropriate block size.</p>	<p>MQ3.0</p> <p><b>Note:</b>In 3.0 this was never used. The broker simple used the client version to determine which body type was being used and calculated the number of blocks via the size.</p>
JMQAckType	Integer	<p>(Optional) Indicates the type of acknowledgement being performed:</p> <ul style="list-style-type: none"> <li>● ACKNOWLEDGE(0): Perform normal acknowledgement processing</li> <li>● UNDELIVERABLE(1): The messages specified in the body were not deliverable</li> <li>● DEAD(2): The messages specified in the body should be marked as dead</li> </ul>	<p>MQ3.6</p>



**ACKNOWLEDGE\_REPLY: S -> C**

Server's reply to an ACKNOWLEDGE packet. When a client sends an ACKNOWLEDGE packet to the server it can optionally request a reply by setting the A bit in the ACKNOWLEDGE packet. If the A bit is set, ACKNOWLEDGE\_REPLY will be returned by the server.

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	25 (ACKNOWLEDGE_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

<b>Variable Header Fields</b>	
None	

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQRemote	Boolean	Indicates that a status of GONE was received because a remote broker which was part of a transaction is down	4.1
		<p style="text-align: center;">OK(200) Acknowledge processed.</p> <p style="text-align: center;">NOT_ALLOWED(405) Acknowledgement was performed for a consumer on a session that does not support acknowledgements (for example a NO_ACKNOWLEDGE session)</p> <p style="text-align: center;">ERROR(500) Acknowledge failed. Server had an internal error</p> <p style="text-align: center;">NOT_FOUND(404) The acknowledgements <b>were not</b> successfully processed before</p>	

<p>JMQStatus</p>	<p>Int</p>	<p>takeover. If the message has not been delivered to the consumer associated with this acknowledgment, it should be thrown out (the broker will handle redelivering the message). If the message has been delivered to the consumer associated with this acknowledgement, the transaction <b>must</b> be rolledback.</p> <p>GONE(410) The transaction specified in transactionID does not exist</p> <p>TIMEOUT(408) The transaction specified in transactionID has timed out.</p> <p>PRECONDITION_FAILED(412) The transaction specified in transactionID is not in a state where it can accept messages</p> <p>BAD_REQUEST(400) The acknowledgement request is bad (for example, JMQValidate bit was set but the transactionID was not).</p>	<p>MQ2.0</p> <p>(NOT_ALLOWED_3.6)</p> <p>(NOT_FOUND, GONE, TIMEOUT, PRECONDITION_BAD_REQUEST s</p>
------------------	------------	---	--

<p><b>Packet Body</b></p>
<p>None</p>

[Back to Summary](#)

**ADD\_CONSUMER: C -> S**

Register a consumer on a destination (Topic or Queue).

The ADD\_CONSUMER packet registers a client consumer on a particular Topic or Queue so that messages published to the destination will be delivered to the client.

**Since:** MQ2.0

**Modified:** MQ3.0, MQ3.5, MQ3.6

**Reply:** ADD\_CONSUMER\_REPLY

Fixed Header Fields	
Field	Value
packet type	14 (ADD_CONSUMER)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bits	Not Applicable
priority	Not applicable
consumer ID	A unique ID that the sender can use to help handle the ADD_CONSUMER_REPLY response packet. Whatever value is specified here is copied into the ADD_CONSUMER_REPLY's consumer ID field by the server. <b>Note:</b> this is NOT the ID of the consumer to register interest for. That is specified by the JMQConsumerID property.

Variable Header Fields
None

Properties			
Property	Type	Value	Since

JMQAckMode	Int	<p><b>Deprecated.</b> This has been replaced by <i>JMQAckMode</i> on <i>CREATE_SESSION</i> in 3.5. In fact this property was never supported by the broker.</p> <p>(Optional). Acknowledgement mode this consumer will be using. This may help the server set the redeliver flag set more accurately.</p> <p>1 = AUTO_ACKNOWLEDGE 2 = CLIENT_ACKNOWLEDGE 3 = DUPS_OK_ACKNOWLEDGE</p> <p>Note: if this property is missing everything will still work fine, but the server may set the redeliver flag on messages a little aggressively.</p>	MQ2.0
JMQDestination	String	Topic or Queue Name to register consumer on	MQ2.0
JMQDestType	Int	Type of destination that JMQDestination is. Should be DEST_TYPE_TOPIC or DEST_TYPE_QUEUE optionally OR'd with DEST_TEMP if it is a temporary destination. See DestType for more information.	MQ2.0
JMQConsumerID	Int	<p><b>Deprecated.</b> This parameter is only supported for backwards compatibility with clients using the 1.0 version of the protocol (iMQ 2.0 clients). It is only supported for those clients. This parameter has been replaced by <i>ADD_CONSUMER_REPLY</i> returning the Consumer ID for the client to use. Unique identifier generated by the client. Must be unique in the scope of the client (i.e. it does not need to be unique across all clients). All messages delivered to the client because of this subscription will be tagged with this ID by setting its value in the packet's "consumer ID" field.</p>	MQ2.0
JMQSelector	String	(Optional) JMS Selector.	MQ2.0
JMQDurableName	String	<p>(Optional). Durable Subscription name (i.e. the string that was passed to createDurableSubscriber()).</p> <p>Only required if durable subscription. If this property does not exist assume nondurable subscription.</p> <p>Note that the client ID is not needed since that has already been set on the connection (see SET_CLIENTID).</p>	MQ2.0
JMQNoLocal	Boolean	<p>(Optional). Value of createSubscriber() noLocal variable: True = don't deliver messages published by this connection back to this connection for this subscription. False = do deliver locally published messages.</p> <p>If property is missing assume False.</p>	MQ2.0
JMQReconnect	Boolean	<p>(Optional). True = this consumer had been added before, but it is being added again because of a reconnect to the router.</p> <p>False = normal ADD_CONSUMER -- not caused by a reconnect.</p> <p>If property is missing assume False.</p>	MQ2.0

JMQSessionID	Long	Specifies the session this consumer is being created on. This ID is returned by CREATE_SESSION.	MQ3.5
JMQSize	Int	<p>(Optional) The maximum "prefetch" size (in number of messages) that should be used for the consumer. This value may be overridden by the server based on the configuration of the destination the consumer is being added to. The broker will send this many messages before pausing the flow and waiting for a RESUME from the client.</p> <p>If the property does not exist assume the client does NOT support consumer based flow control (i.e. it is 3.0.1 or earlier).</p> <p>If -1 assume unlimited prefetch size.</p> <p>The actual value the server will be using is returned in the JMQSize property in the ADD_CONSUMER_REPLY packet.</p>	MQ3.5
JMQShare	Boolean	<p>(Optional) Specifies if the consumer may share a subscription or not.</p> <p><b>Durable Subscriptions</b> In 3.5 support for sharing durable subscriptions was added. If this property is "true", then this consumer is willing to share durable subscriptions with other consumers. "False" enforces standard JMS semantics. In this case the subscription is defined to be: ClientID + Durablename.</p> <p><b>Non-Durable Subscriptions</b> In 3.6 support for sharing non-durable subscriptions was added. If this property is "true", then this consumer is willing to share a subscription with other non-durable subscribers. In this case the subscription is defined to be: ClientID + Topicname + Selector</p> <p>In both cases the NoLocal flag must be consistent across the shared subscriptions.</p> <p>Default is "false". Typically this property will have the same value as that used for "JMQShare" in SET_CLIENTID.</p>	MQ3.5 Modified MQ3.6

**Packet Body:** None

---

**ADD\_CONSUMER\_REPLY: S -> C**

Reply to an ADD\_CONSUMER packet.

**Modified:** MQ3.0, MQ3.5

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	15 (ADD_CONSUMER_REPLY),
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler. Note: This is NOT the consumer ID specified in the JMQConsumerID property. Rather the consumer ID set on the packet's fixed header field to use as a correlation ID.

<b>Variable Header Fields</b>
None

Properties:			
Property	Type	Value	Since
JMQConsumerID	Long	Unique identifier generated by the server to identify this consumer. Guaranteed to be unique accross all consumers in a server cluster. All messages delivered to the client because of this subscription will be tagged with this ID by setting it's value in the packet's "consumer ID" field.	MQ3.0
JMQDestType	Int	(Optional). JMQDestType that was passed in ADD_CONSUMER OR'd with the destination's flavor. This basically informs the Client of the destination's flavor which can be used to determine how to set the JMQLock property when sending DELETE_CONSUMER. See DestType for more information. If missing assume default flavor.	MQ3.0
JMQSize	Int	The prefetch size the server is actually going to use. This value will be no larger than what was specified in ADD_CONSUMER. Note that this value is actually a maximum. At times the server may send fewer messages before pausing the flow.  Only specified is consumer flow control is being used.	MQ3.5
JMQStatus	Int	<p>OK(200) Consumer added</p> <p>BAD_REQUEST(400) Invalid selector description</p> <p>FORBIDDEN(403) Client does not have permission to register a consumer on the destination</p> <p>NOT_FOUND(404) Destination was not found</p> <p>NOT_ALLOWED(405) Destination is a Temporary destination that was not created on this connection</p> <p>CONFLICT(409) The Destination has a limit on the number of consumers and adding this consumer would exceed that limit.</p> <p>PRECONDITION_FAILED(412) Request was to add a durable consumer, but a ClientID was not set on the connection.</p> <p>ERROR(500) Internal server error</p>	MQ2.0

**Packet Body:** None

[Back to Summary](#)

---

## ADD\_PRODUCER: C -> S

The ADD\_PRODUCER packet notifies the server that the client is going to produce messages on the specified destination. The server may then grant or deny this request.

**Since:** MQ2.0

**Modified:** MQ3.5

**Reply:** ADD\_PRODUCER\_REPLY

Fixed Header Fields	
Field	Value
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
packet type	18 (ADD_PRODUCER)
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bits	Not Applicable
priority	Not applicable
consumer ID	A unique ID that the sender can use to help handle the ADD_PRODUCER_REPLY response packet. Whatever value is specified here is copied into the ADD_PRODUCER_REPLY's consumer ID field by the server.

Header Strings
None

Properties			
Property	Type	Value	Since
JMQDestination	String	Topic or Queue Name to register producer on. This should be the full canonical destination name including parameters (if any).	MQ2.0
JMQDestType	Int	Type of destination that JMQDestination is. Should be DEST_TYPE_TOPIC or DEST_TYPE_QUEUE optionally OR'd with DEST_TEMP if it is a temporary destination. See DestType for more information.	MQ2.0
JMQSessionID	Long	Specifies the session this consumer is being created on. This ID is returned by CREATE_SESSION.	MQ3.5

**Packet Body:** Not Applicable

---

## ADD\_PRODUCER\_REPLY: S -> C

Reply to an ADD\_PRODUCER packet.

**Modified:** MQ3.5

Fixed Header Fields	
Field	Value
packet type	19 (ADD_PRODUCER_REPLY),
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler. Note: This is NOT the consumer ID specified in the JMQConsumerID property. Rather the consumer ID set on the packet's fixed header field to use as a correlation ID.

Variable Header Fields
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQProducerID	Long	ID to use in producer operations such as RESUME_FLOW and DELETE_PRODUCER.	MQ3.5
JMQBytes	Long	(Optional) Chunking factor (in bytes) client should use for flow control. After sending this many bytes the producer should pause the flow.	MQ3.5
JMQSize	Int	(Optional) Chunking factor (in # msgs) client should use for flow control. After sending this many messages the producer should pause the flow.	MQ3.5
JMQStatus	Int	<p>OK(200) Client may send message to destination</p> <p>FORBIDDEN(403) Client does not have permission to send messages to destination</p> <p>NOT_FOUND(404) Destination does not exist and can't be auto-created. (Note: there is a bug in MQ 3.5 where NOT_ALLOWED(405) is returned instead of NOT_FOUND(404)).</p> <p>CONFLICT(409) The Destination has a limit on the number of producers and adding this producer exceeds that limit. (Note: there is a bug in MQ 3.5 where NOT_ALLOWED(405) is returned instead of CONFLICT(409)).</p> <p>ERROR(500) Internal server error</p>	MQ2.0

**Packet Body:** None

[Back to Summary](#)

**AUTHENTICATE: C -> S**

Authenticate a user with the router. After the HELLO exchange the server sends an AUTHENTICATE\_REQUEST packet to the client. The client must then send an AUTHENTICATE packet to the server.

**Since:** MQ2.0

**Reply:** AUTHENTICATE\_REPLY

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	12 (AUTHENTICATE)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bits	Not Applicable
priority	Not Applicable
consumer ID	A unique ID that the sender can use to help handle the AUTHENTICATE_REPLY response packet. Whatever value is specified here is copied into the AUTHENTICATE_REPLY's consumer ID field by the server.

<b>Variable Header Fields</b>	
None	

Properties:			
Property	Type	Value	Since
JMQAuthType	String	The type of authentication being attempted. For 2.0 we will support two builtin types: "jmqbasic" and "jmqdigest". If the type is "jmqbasic" then user name and obscured passwd is stored in the packet body as described below and will be authenticated by the server. If the type is "jmqdigest" the user name and hashed passwd is stored in the packet body as described below and will be authenticated by the server.	MQ2.0

Packet Body for "jmqbasic" authentication
<p>For basic authentication the AUTHENTICATE packet body consists of two UTF-8 encoded strings. The first is the user name and the second is the passwd which has been BASE64 encoded into a byte[] after being encoded into UTF-8:</p> <pre> 0           1           2           3  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1  +-----+-----+-----+-----+   Length of username string   Username (UTF-8)   +-----+-----+-----+-----+                                 . . .                                 +-----+-----+-----+-----+   Length of passwd data        BASE64 encoded UTF-8 passwd   +-----+-----+-----+-----+                                 . . .                                 +-----+-----+-----+-----+</pre> <p>The first two bytes is a short defining the length of the UTF-8 encoded username that immediately follows. This is compatible with the java.io.DataInputStream.readUTF() and java.io.DataOutputStream.writeUTF().</p> <p>Immediately following the username is two bytes defining the length of the encoded passwd data. The passwd data is a byte[] that contains the BASE64 encoded UTF-8 password string.</p>

Packet Body for "jmqdigest" authentication
Content for jmqdigest authentication packet body

For digest authentication the AUTHENTICATE packet body consists of two values. The first is the user name in clear (UTF-8) text and the second is the MD5 hash of the passwd and a nonce that has previously been provided by the server in the body of the AUTHENTICATE\_REQUEST packet. This is inspired by the HTTP Digest Access Authentication scheme described by RFC2617, but is greatly simplified since we don't have to worry about re-authenticating accross connections, realms, uri's, etc.

The algorithm the client should use for computing the MD5 hashed passwd is as follows:

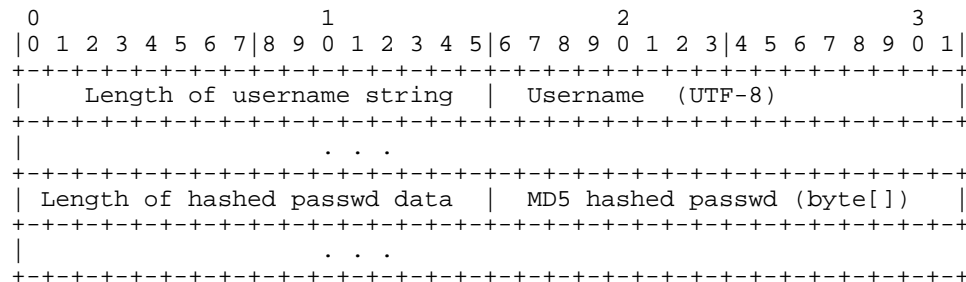
```
MD5( MD5(username + ":" + passwd) + ":" + nonce)
```

Where '+' is the string concatenation operator and MD5() is the MD5 checksum operation. By running MD5 on the passwd first (or more precisely the username + passwd) we allow the server to store MD5(username + ":" + passwd) in its passwd database instead of the clear text passwd.

When the server gets the AUTHENTICATE message it looks up the user's MD5(username + ":" + passwd) value from its passwd database, concatenates the nonce to it, runs this through MD5 and compares the result to what was passed in the AUTHENTICATE message.

See AUTHENTICATE\_REQUEST for a description of how the server should compute the value for JMQNonce. As far as the client is concerned it is an opaque ASCII byte array.

The 128 bit MD5 digest is represented as 32 ASCII printable characters. Each 4 bit value is represented by its hex notation. For example 0000 is '0', 0001 is '1' and 1111 is 'f'. An example digest would be "d87051c6f8ca76e671ba2d52f12ef976". This means the length of the passwd hash should always be 32 bytes, since we only support MD5 at this time.



The first two bytes is a short defining the length of the UTF-8 encoded username that immediately follows. This is compatible with the java.io.DataInputStream.readUTF() and java.io.DataOutputStream.writeUTF().

Immediately following the username is two bytes defining the length of the hashed passwd data. The passwd data is a byte[] that contains the MD5 hashed passwd as defined by the hashing algorithm given above.

**AUTHENTICATE\_REPLY: S -> C**

Server's reply to a an AUTHENTICATE packet.

Fixed Header Fields	
Field	Value
packet type	13 (AUTHENTICATE_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

Variable Header Fields
None

Properties:			
Property	Type	Value	Since
JMQStatus	Int	OK(200) Authorization granted FORBIDDEN(403) Authorization failed ERROR(500) Internal server error	MQ2.0

**Packet Body:** None

[Back to Summary](#)

---

## AUTHENTICATE\_REQUEST: S -> C

Server requests authentication from the client. This will typically immediately follow a HELLO\_REPLY packet.

**Since:** MQ2.0

**Reply:** None. But an AUTHENTICATE message must arrive from the client before the connection can be used for anything else.

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	38 (AUTHENTICATE_REQUEST)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bit fields	Not Applicable
priority	Not applicable
consumer ID	The consumer ID that was set on the HELLO packet that caused this packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

<b>Header Strings</b>
None

<b>Properties</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQAuthType	String	String describing the authentication type the client should use. "jmqbasic" and "jmqdigest" are reserved as is a string starting with "jmq*". If type is "jmqdigest" then the body of this packet is set to a nonce to be used by the client for performing JMQ digest authentication.	MQ2.0
JMQChallenge	Boolean	True if this is the first AUTHENTICATE_REQUEST sent by the server to the client. False if it is a subsequent request (some authentication protocols may require more than one AUTHENTICATE_REQUEST).	MQ2.0

### **Packet Body**

Determined by JMQAuthType. For the built-in authtypes it is as follows:

- jmqbasic: Packet body is empty
- jmqdigest: Packet body is a byte array of ASCII characters uniquely generated for each connection. See AUTHENTICATE for information on how this value is used by the client. It is recommended that the nonce be a 128 bit MD5() digest generated using the following algorithm and represented as 32 ASCII hex bytes:

```
MD5( client-IP + ":" + timestamp + server-seed )
```

Where client-IP is the string representation of the client IP address (dotted IPv4 or colon IPv6), timestamp is a server generated timestamp, and server-seed is some value generated by the server (could be a random number, memory address of an object related to the connection, etc). MD5() is the MD5 message digest function.

The length of the nonce is determined by the packet body size, but will typically be 32 bytes.

[Back to Summary](#)

**BROWSE: C -> S**

Request the server to send the SysMessageIDs of all messages in a particular Queue. This is typically used to implement a QueueBrowser.

As of 3.5 this can also be used to request destination metrics.

**Since:** MQ2.0

**Modified:** MQ3.5

**Reply:** BROWSE\_REPLY

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	26 (BROWSE)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
All other bits	Not Applicable
priority	Not Applicable
consumer ID	A unique ID that the sender can use to help correlate the BROWSE_REPLY response packet. Whatever value is specified here is copied into the BROWSE_REPLY's consumer ID field by the server.

<b>Variable Header Fields</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQDesintation	String	The Queue name to browse.	MQ2.0
JMQSelector	String	The selector to apply to the queue.	MQ2.0
JMQMetrics	Boolean	(Optional) If this property is set to true then the QueueBrowser only wants metrics information. In this case BROWSE_REPLY contains metric information.  <b>Or should we create a GET_METRICS instead??</b>	MQ3.5

<b>Packet Body</b>
None

## BROWSE\_REPLY: S -> C

Server's reply to a BROWSE request.

**Modified:** MQ3.0, MQ3.5

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	27 (BROWSE_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

<b>Variable Header Fields</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQConsumerID	Long	Unique ID that the client should specify in a future DELIVER request when request that the actual message be sent.	MQ3.0
JMQCanCreate	Boolean	Set only if JMQStatus is NOT_FOUND(404). This is used to inform the client if the server will allow the destination to be created via CREATE_DESTINATION (i.e. support auto-create).  True Destination can be created with CREATE_DESTINATION False Destination cannot be created with CREATE_DESTINATION	MQ2.0
JMQMetrics	Boolean	(Optional) Set to 'true' only if JMQMetrics was set on the original request. Otherwise the property is not present. This helps the client detect that the reply contains metric information and not the queue contents.	MQ3.5
JMQStatus	Int	OK(200) Browse successful. Packet body will contain list of SysMessageIDs of messages in Queue (or metric information). BAD_REQUEST(400) Invalid selector FORBIDDEN(403) Client does not have permission to browse destination NOT_FOUND(404) Destination does not exist ERROR(500) Internal server error	MQ2.0

**Packet Body**

If JMQMetrics is absent or 'false' then the message body is a series of SysMessageIDs, one for each message in the Queue. The exact format of the SysMessageID is described in the DELIVER packet.

If JMQMetrics is 'true' then the message body contains a serialized HashMap containing metric information using the same name/value pairs as described in the JMS Broker Monitoring functional spec.

[Back to Summary](#)

---

## CREATE\_DESTINATION: C -> S

The CREATE\_DESTINATION packet requests that the server create a destination. It is used by the client to create temporary destinations (i.e. TemporaryQueue and TemporaryTopic) and to auto-create normal destinations (which may not be allowed by the server).

Since a client is not allowed to produce to a non-existent destination it is acceptable for the client to always send a CREATE\_DESTINATION before ADD\_PRODUCER. But the client must be prepared to handle the error conditions returned by CREATE\_DESTINATION\_REPLY.

**Since:** MQ2.0

**Reply:** CREATE\_DESTINATION\_REPLY

Fixed Header Fields	
Field	Value
packet type	34 (CREATE_DESTINATION)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bits	Not Applicable
priority	Not applicable
consumer ID	A unique ID that the sender can use to help handle the CREATE_DESTINATION_REPLY response packet. Whatever value is specified here is copied into the CREATE_DESTINATION_REPLY's consumer ID field by the server.

Variable Header Fields	
None	

Properties			
Property	Type	Value	Since
JMQDestination	String	Topic or Queue Name to create. This should be the full canonical destination name.	MQ2.0
JMQDestType	Int	Type of destination that JMQDestination is. Should be DEST_TYPE_TOPIC or DEST_TYPE_QUEUE optionally OR'd with DEST_TEMP if it is a temporary destination. See DestType for more information. If the destination is temporary it will be destroyed automatically when the connection that created the destination is closed.	MQ2.0

**Packet Body:** Not Applicable

---

## CREATE\_DESTINATION\_REPLY: S -> C

Server's reply to a CREATE\_DESTINATION packet

**Modified:** MQ3.0

Fixed Header Fields	
Field	Value
packet type	35 (CREATE_DESTINATION_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

Variable Header Fields
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQDestType	Int	(Optional). JMQDestType that was passed in CREATE_DESTINATION OR'd with the destination's flavor. This basically informs the Client of the destination's flavor. See DestType for more information. If missing assume default flavor.	MQ3.0
JMQStatus	Int	OK(200) Destination was created. FORBIDDEN(403) Client is not authorized to create destination CONFLICT(409) Destination already exists ERROR(500) Internal server error	MQ2.0

[Back to Summary](#)

**CREATE\_SESSION: C -> S**

Requests the broker to create a session and allocate a session ID. This session ID is used on subsequent session operations, and is set on other packets, such as CREATE\_CONSUMER, to associate the object with a session.

**Since:** MQ3.5

**Modified:** MQ3.6

**Reply:** CREATE\_SESSION\_REPLY

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	68 (CREATE_SESSION)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bits	Not Applicable
priority	Not applicable
consumer ID	A unique ID that the sender can use to help handle the CREATE_SESSION_REPLY response packet. Whatever value is specified here is copied into the CREATE_SESSION_REPLY's consumer ID field by the server.

<b>Variable Header Fields</b>	
None	

Properties			
Property	Type	Value	Since
JMQAckMode	Int	(Optional) Acknowledgement mode this Session will be using. This may help the server set the redeliver flag more accurately. One of 0=NONE 1=AUTO_ACKNOWLEDGE 2=CLIENT_ACKNOWLEDGE 3=DUPS_OK_ACKNOWLEDGE 32768=NO_ACKNOWLEDGE Default: NONE If NONE then everything will still work fine, but the server may set the redeliver flag on messages a little aggressively. NONE may also be used for transacted sessions.	MQ3.5 (NO_ACKNOWLEDGE since MQ3.6)

**Packet Body:** Not Applicable

---

## CREATE\_SESSION\_REPLY: S -> C

Server's reply to a CREATE\_SESSION packet

Fixed Header Fields	
Field	Value
packet type	69 (CREATE_SESSION_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

Variable Header Fields
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQSessionID	Long	A unique identifier for this session. Used for other operations on the session, and used when creating objects on the session (i.e. with ADD_CONSUMER).	MQ3.5
JMQStatus	Int	OK(200) Session was created. BAD_REQUEST(400) JMQAckMode is invalid. NOT_ALLOWED(405) JMQAckMode is not allowed (e.g. because of licensing restrictions). ERROR(500) Internal server error	MQ3.5

[Back to Summary](#)

**DELETE\_CONSUMER: C -> S**

Removes a consumer from a destination (Topic or Queue).

The DELETE\_CONSUMER packet removes a client consumer that was previously added with ADD\_CONSUMER. If the JMQDurableName and JMQClientID properties are provided the corresponding durable subscription is also removed.

**Since:** MQ2.0

**Modified:** MQ3.0, MQ3.5

**Reply:** DELETE\_CONSUMER\_REPLY

Fixed Header Fields	
Field	Value
packet type	16 (DELETE_CONSUMER)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bits	Not Applicable
priority	Not applicable
consumer ID	A unique ID that the sender can use to help handle the DELETE_CONSUMER_REPLY packet. Whatever value is specified here is copied into the DELETE_CONSUMER_REPLY's cosumer ID field by the server.

Header Strings
None

Properties:			
Property	Type	Value	Since

JMQConsumerID	Long	Consumer to delete. This is the JMQConsumerID returned in the ADD_CONSUMER_REPLY packet. Optional if deleting a durable subscription that we do not have a registered consumer on. Otherwise required.	MQ3.0
JMQConsumerID	Int	<b>Deprecated.</b> <i>This parameter is only supported for backwards compatibility with clients using the 1.0 version of the protocol (iMQ 2.0 clients). It is only supported for those clients. This parameter has been replaced by JMQConsumerID Long.</i> Consumer to delete. This is the JMQConsumerID used in the ADD_CONSUMER packet. Optional if deleting a durable subscription that we do not have a registered consumer on.	MQ2.0
JMQDurableName	String	Durable subscription to remove. Only required if removing a durable subscription.	MQ2.0
JMQClientID	String	ClientID of durable subscription to remove. Only required if removing a durable subscription.	MQ2.0
JMQBlock	Boolean	(Optional). <b>True</b> to require that DELETE_CONSUMER_REPLY is returned only after the consumer has been completely removed from the server. <b>False</b> if the server can send DELETE_CONSUMER_REPLY earlier. For most destination flavors the client doesn't care exactly when the consumer is removed. But for "single receiver" queues it does, since it may want to immediately add a new consumer on the queue -- and it can't until the old one has been completely removed.	MQ3.0

JMQBodyType	Integer	<p>(Optional) Format of packet's body. One of</p> <ul style="list-style-type: none"> <li>● NONE(0)</li> <li>● SYSMESSAGEID(3)</li> </ul> <p>If not specified assume NONE. If SYSMESSAGEID(3) then the body contains one marshalled SysMessageID. This SysMessageID identifies the last message that the consumer had seen but not yet acknowledged. This helps the broker set the redeliver flag more accurately on messages that need to be redelivered. Note that to have this take effect JMQSessionID must have been specified when the consumer was created.</p> <p>See BodyType for more information about body types.</p>	MQ3.5 (Note: this property appeared in the 3.0 spec, but was never implemented by the broker or client.).
-------------	---------	--	---

#### Packet Body

(Optional) Varies depending on the value of JMQBodyType.  
If JMQBodyType is NONE(0) then there is no packet body.  
If JMQBodyType is SYSMESSAGEID(3) then the body contains one marshalled SysMessageIDs. The format is the same as that used for the DELIVER packet.

## DELETE\_CONSUMER\_REPLY: S -> C

Server's reply to a DELETE\_CONSUMER packet

Fixed Header Fields	
Field	Value
packet type	17 (DELETE_CONSUMER_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

Variable Header Fields
None

Properties:			
Property	Type	Value	Since
JMQStatus	Int	OK(200) Consumer deleted FORBIDDEN(403) Client does not have permission to delete consumer NOT_FOUND(404) Consumer was not found CONFLICT(409) Durable subscription is in use PRECONDITION_FAILED(412) Destination that this consumer was on no longer exists. ERROR(500) Internal server error	MQ2.0

[Back to Summary](#)

---

**RECREATE\_CONSUMER: B -> C**

Tell client runtime to recreate a consumer.

This protocol is used to inform client runtime to recreate a consumer. Client runtime should use DELETE\_CONSUMER and ADD\_CONSUMER to recreate the consumer.

**Since:** MQ4.1

**Modified:**

**Reply:** None

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	80 (RECREATE_CONSUMER)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bits	Not Applicable
priority	Not applicable
consumer ID	Not Applicable

<b>Header Strings</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQConsumerID	Long	The consumer to be recreated. This is the JMQConsumerID returned in the ADD_CONSUMER_REPLY packet. Required.	MQ4.1
JMQSessionStopped	Boolean	The consumer's Session is stopped if true. The session can be restarted by protocol STOP	MQ4.1

<b>Packet Body</b>
None

[Back to Summary](#)

**DELETE\_PRODUCER: C -> S**

Removes a producer from a destination (Topic or Queue).

The DELETE\_PRODUCER packet removes a client producer that was previously added with ADD\_PRODUCER.

**Since:** MQ3.5

**Reply:** DELETE\_PRODUCER\_REPLY

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	66 (DELETE_PRODUCER)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bits	Not Applicable
priority	Not applicable
consumer ID	A unique ID that the sender can use to help handle the DELETE_CONSUMER_REPLY packet. Whatever value is specified here is copied into the DELETE_CONSUMER_REPLY's consumer ID field by the server.

<b>Header Strings</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQProducerID	Long	Producer to delete. This is the JMQProducerID returned in the ADD_PRODUCER_REPLY packet.	MQ3.5

<b>Packet Body</b>
None

## DELETE\_PRODUCER\_REPLY: S -> C

Server's reply to a DELETE\_PRODUCER packet

Fixed Header Fields	
Field	Value
packet type	67 (DELETE_CONSUMER_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

<b>Variable Header Fields</b>
None

Properties:			
Property	Type	Value	Since
JMQStatus	Int	OK(200) Producer deleted FORBIDDEN(403) Client does not have permission to delete producer NOT_FOUND(404) Producer was not found ERROR(500) Internal server error	MQ3.5

[Back to Summary](#)

**DELIVER: C -> S**

Request the server to deliver the messages that are specified in the packet body. Typically this will be used by the client in to implement QueueBrowsers in which case the message IDs are gotten from a BROWSE\_REPLY.

**Since:** MQ2.0

**Modified:** MQ3.0

**Reply:** DELIVER\_REPLY

Fixed Header Fields	
Field	Value
packet type	42 (DELIVER)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
All other bits	Not Applicable
priority	Not Applicable
consumer ID	A unique ID that the sender can use to help correlate the DELIVER_REPLY response packet. Whatever value is specified here is copied into the DELIVER_REPLY's consumer ID field by the server.

Variable Header Fields
None

Properties:			
Property	Type	Value	Since
JMQConsumerID	Long	Unique ID that the broker will place in the consumer ID fixed header field of each message delivered because of this request.	MQ3.0
JMQConsumerID	Int	<b>Deprecated.</b> <i>This parameter is only supported for backwards compatibility with clients using the 1.0 version of the protocol (iMQ 2.0 clients). It is only supported for those clients. This parameter has been replaced by JMQConsumerID Long.</i> Unique ID that the broker will place in the consumer ID header field of each message delivered because of this request.	MQ2.0



Variable Header Fields
None

Properties:			
Property	Type	Value	Since
JMQStatus	Int	<p>OK(200) Delivery of at least one message is guaranteed</p> <p>NOT_FOUND(404) None of the requested messages exist. No messages will be delivered.</p> <p>ERROR(500) Internal server error</p> <p>If the status of DELIVER_REPLY is OK, then the server will send one or more messages to the client, each message with the same Consumer ID (the one specified in DELIVER) and the last one with the L bit set (for Last message). These messages are not considered "delvered" in the JMS sense. I.e. it is assumed that the client is an observer of the destination (via a QueueBrowser for example) not a consumer.</p> <p>The client may receive messages that have recently expired. In particular if the last message (with the L bit set) has expired, the broker will still send it.</p>	MQ2.0

[Back to Summary](#)

---

## DESTROY\_DESTINATION: C -> S

The DESTROY\_DESTINATION packet requests that the server destroy a destination. This is intended to be used to destroy Temporary destinations (for example when the client calls TemporaryTopic.destroy()). Note that Temporary destinations should be destroyed by the server automatically when the connection that created the destination terminates.

**Since:** MQ2.0

**Reply:** DESTROY\_DESTINATION\_REPLY

Fixed Header Fields	
Field	Value
packet type	36 (DESTROY_DESTINATION)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bits	Not Applicable
priority	Not applicable
consumer ID	A unique ID that the sender can use to help handle the DESTROY_DESTINATION_REPLY response packet. Whatever value is specified here is copied into the DESTROY_DESTINATION_REPLY's consumer ID field by the server.

Header Strings
None

Properties			
Property	Type	Value	Since
JMQDestination	String	Topic or Queue Name to destroy. This should be the full canonical destination name.	MQ2.0
JMQDestType	Int	Type of destination that JMQDestination is. Should be DEST_TYPE_TOPIC or DEST_TYPE_QUEUE OR'd with DEST_TEMP since this message should only be used with temporary destinations. See DestType for more information.	MQ2.0

**Packet Body:** Not Applicable

## DESTROY\_DESTINATION\_REPLY: S -> C

Server's reply to a DESTROY\_DESTINATION packet

Fixed Header Fields	
Field	Value
packet type	37 (DESTROY_DESTINATION_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

Variable Header Fields
None

Properties:			
Property	Type	Value	Since
JMQStatus	Int	OK(200) Destination was destroyed. FORBIDDEN(403) Client is not authorized to destroy destination NOT_FOUND(404) Destination does not exist CONFLICT(409) Destination is in use (has producers or consumers) ERROR(500) Internal server error	MQ2.0

[Back to Summary](#)

**DESTROY\_SESSION: C -> S**

Requests the broker to destroy the session specified by the passed session ID. Session resources are free'd and any unacknowledged messages left on the session are queued for redelivery. Note that the client is still responsible for removing all consumers using DELETE\_CONSUMER before deleting the session

**Since:** MQ3.5

**Reply:** DESTROY\_SESSION\_REPLY

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	70 (DESTROY_SESSION)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bits	Not Applicable
priority	Not applicable
consumer ID	A unique ID that the sender can use to help handle the DESTROY_SESSION_REPLY response packet. Whatever value is specified here is copied into the DESTROY_SESSION_REPLY's consumer ID field by the server.

<b>Variable Header Fields</b>
None

<b>Properties</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQSessionID	Long	ID of Session to destroy. Returned via CREATE_SESSION.	MQ3.5

**Packet Body:** Not Applicable

**DESTROY\_SESSION\_REPLY: S -> C**

Server's reply to a DESTROY\_SESSION packet

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	71 (DESTROY_SESSION_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

<b>Variable Header Fields</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQStatus	Int	OK(200) Session was created. ERROR(500) Internal server error	MQ3.5

[Back to Summary](#)

---

**ERROR: S -> C**

Send an asynchronous error notification to the client.

**Since:** MQ2.0

**Reply:** None

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	30 (ERROR)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
consumer ID, priority	Not Applicable

<b>Header Strings</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQErrorno	Int	An error number indicating the error that has occurred	MQ2.0
JMQTimestamp	Long	Timestamp of when the error occurred	MQ2.0
JMQErrorLevel	Int	Level of severity of the error	MQ2.0
JMQErrorString	String	(Optional) String describing the error	MQ2.0

<b>Packet Body</b>
None

[Back to Summary](#)

---

**GENERATE\_UID: C -> S**

Request the server to generate one or more unique IDs. A Unique ID has following properties:

- It will stay unique for a very long time (years).
- It will be unique across all other IDs returned by the broker (ConsumerIDs, TransactionIDs, ConnectionIDs, etc).
- It will be unique across all brokers in a cluster.

**Since:** MQ3.0

**Reply:** GENERATE\_UID\_REPLY

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	62 (GENERATE_UID)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must generate a reply)
All other bit flags	Not Applicable
consumer ID, priority	Not Applicable

<b>Variable Header Fields</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQQuantity	Integer	Number of ID's to generate. You should keep this to a reasonable level (ie don't generate thousands).	MQ3.0

<b>Packet Body</b>
None

---

**GENERATE\_UID\_REPLY: S -> C**

Server's reply to a HELLO packet

Fixed Header Fields	
Field	Value
packet type	63 (GENERATE_UID_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit flags	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

Variable Header Fields
None

Properties:			
Property	Type	Value	Since
JMQQuantity	Integer	Number of ID's in the body.	MQ3.0
JMQStatus	Int	OK(200) IDs generated ERROR(500) Internal server error	MQ3.0

Packet Body
A sequence of longs. Each long is a unique ID. The number of IDs in the body is specified by the JMQQuantity property.

[Back to Summary](#)

---

**GET\_LICENSE: C -> S**

Get license information from the server.

**Since:** MQ3.6

**Reply:** GET\_LICENSE\_REPLY

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	76 (GET_LICENSE)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bits	Not Applicable
priority	Not applicable
consumer ID	A unique ID that the sender can use to help handle the GET_LICENSE_REPLY response packet. Whatever value is specified here is copied into the GET_LICENSE_REPLY's consumer ID field by the server. <b>Note:</b> this is NOT the ID of the consumer to register interest for. That is specified by the JMQConsumerID property.

<b>Variable Header Fields</b>
None

<b>Properties</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
None			

**Packet Body:** None

---

**GET\_LICENSE\_REPLY: S -> C**

Reply to a GET\_LICENSE packet.

Since: MQ3.6

Fixed Header Fields	
Field	Value
packet type	77 (GET_LICENSE_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler. Note: This is NOT the consumer ID specified in the JMQConsumerID property. Rather the consumer ID set on the packet's fixed header field to use as a correlation ID.

Variable Header Fields
None

Properties:			
Property	Type	Value	Since
JMQStatus	Int	OK(200) License info returned ERROR(500) Internal server error	MQ3.6
JMQLicense	String	The license being used by the server. This is the short license name like "try", "unl", etc.	MQ3.6
JMQLicenseDesc	String	A description of the license being used by the server. This is a longer description like: "Sun Java(tm) System Message Queue 3.5 SP1 Enterprise Edition"	MQ3.6
*	String	One or more license properties with their values. Typically all license properties are returned. The license properties are guaranteed not to start with "JMQ", so the caller knows that any packet property that does not start with "JMQ" is a license property.	MQ3.6

**Packet Body:** None

[Back to Summary](#)

**GOODBYE: C -> S****GOODBYE: S -> C**

Notify the server that the client is going away.

Notify the client that the server is going away.

The server should only send a GOODBYE to the client when it is forcibly closing the connection. Otherwise a connection is typically terminated by the client saying GOODBYE and the server sending GOODBYE\_REPLY.

**Since:** MQ2.0

**Modified:** MQ3.0, MQ3.5

**Reply:** Optional GOODBYE\_REPLY from server only (i.e. when the client sends GOODBYE to the server, the server will respond with a GOODBYE\_REPLY if the 'A' bit is set. If the server sends a GOODBYE to the client, there is no response from the client.

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	28 (GOODBYE)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
packet type	28 (GOODBYE)
A bit	Optional. When the client is sending GOODBYE to the server it may set the 'A' bit if it desires a reply from the server. If the 'A' bit is not set, the server must not send a reply. When the server is sending GOODBYE to the client the 'A' bit is not applicable.
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bits	Not Applicable
priority	Not Applicable
consumer ID	If the a bit is set, this should be a unique ID that the sender can use to help correlate the GOODBYE_REPLY response packet. Whatever value is specified here is copied into the GOODBYE_REPLY's consumer ID field by the server.

Header Strings
None

Properties:			
Property	Type	Value	Since
JMQBlock	Boolean	(Optional) <b>True</b> GOODBYE_REPLY is not sent until ???? <b>False</b> GOODBYE_REPLY may be sent sooner. If not present defaults to false.	MQ3.0
JMQConnectionID	Long	(Optional) Informs the broker that it may clean up all resources associated with this connection, and that no reconnect (HELLO with a connection ID) will occur. For a typical client shutdown this should be set.	MQ3.0
JMQExit	Boolean	(Optional) Only used for S -> C messages. Informs the client that it should exit, and not attempt reconnects or failovers.	MQ3.5
JMQGoodbyeReason	Integer	(Optional) Only used for S -> C messages.  Tells the client why the broker is going away (if known), one of:  <ul style="list-style-type: none"> <li>● 1 - SHUTDOWN</li> <li>● 2 - RESTART</li> <li>● 3 - KILLED</li> <li>● 4 - ERROR</li> <li>● 5 - CLIENT_CLOSED</li> </ul>	MQ4.0

Packet Body
None

## GOODBYE\_REPLY: S -> C

Server's reply to a GOODBYE packet

Fixed Header Fields	
Field	Value
packet type	29 (GOODBYE_REPLY),
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

Variable Header Fields
None

Properties:			
Property	Type	Value	Since
JMQStatus	Int	OK(200) Goodbye successful.	MQ3.0
JMQGoodbyeReason	Integer	(Optional) Only used for S -> C messages.  Tells the client why the broker is going away (if known), one of: <ul style="list-style-type: none"> <li>● 1 - SHUTDOWN</li> <li>● 2 - RESTART</li> <li>● 3 - KILLED</li> <li>● 4 - ERROR</li> <li>● 5 - CLIENT_CLOSED</li> </ul>	MQ4.0

[Back to Summary](#)

**HELLO: C -> S**

Initiate a connection with the server.

**Since:** MQ2.0

**Modified:** MQ3.0, MQ3.5, MQ3.6 MQ4.0

**Reply:** HELLO\_REPLY.

Also an AUTHENTICATE\_REQUEST may be sent immediately after the HELLO\_REPLY.

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	10 (HELLO)
version, size, timestamp, sequence number, property offset, property size, encryption, source port, source IP	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bits	Not Applicable
priority	Not Applicable
consumer ID	A unique ID that the sender can use to help handle the HELLO_REPLY response packet. Whatever value is specified here is copied into the HELLO_REPLY's consumer ID field by the server.

<b>Variable Header Fields</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQProtocolLevel	Int	Level of the protocol the client is prepared to speak. For example 80=0.8, 100=1.0, 101=1.0.1, 351=3.5.1. You should derive this number from the version number given on the introductory page, see the table on that page for what protocol version was supported by what MQ version.	MQ2.0
JMQVersion	String	The product version string of the client. This will be something like "3.0.1", "3.5", etc.	MQ2.0

JMQPriority	Int	(Optional). 0-9 priority level. This is a hint to the server specifying the priority of service this client should receive. The server may choose to ignore this.	MQ2.0
JMQSafeMode	Boolean	(Optional). True for the server to deliver JMS messages to this connection in "safe" mode. This means the server will not deliver a JMS message to the connection until the previous message has been acknowledged by the client. This may be valuable for small clients that will rely on the broker to hold unacknowledged messages. False if the server should deliver messages without waiting for previous messages to be acknowledged. (Default)	MQ2.0
JMQRBufferSize	Int	<b>Deprecated</b> Replaced by JMQSize (Optional). Number of JMS messages server may write to connection before pausing until the next RESUME_FLOW packet is received from the client. 0 for unlimited (no flow control). If this property is missing then the server uses an appropriate default value. See RESUME_FLOW for more information. <i>Note: this is still being used by the 3.0 client.</i>	MQ2.0
JMQSize	Int	(Optional). Number of JMS messages server may write to connection before pausing until the next RESUME_FLOW packet is received from the client. 0 Don't send any message -1 Unlimited [Not Set] Unlimited See RESUME_FLOW for more information. <b>Warning!</b> This should have been "JMQCount", and may be changed in a future release. <i>Note: this is NOT being used by the 3.0 client.</i>	MQ3.0
JMQConnectionID	Long	(Optional). If this is a reconnection to the server then this should contain the JMQConnectionID that was returned in HELLO_REPLY.	MQ3.0
JMQReconnectable	Boolean	(Optional). Default is false. If true, then the connection is a failover connection and the broker should perform any extra bookkeeping for that connection.	MQ3.5
JMQUserAgent	String	(Optional) A form user agent string provided by the client. The string may be used to enforce some licensing restrictions. For example the C client may not be allowed to connect to a non-EE edition server. Some example syntax is:  S1MQ/3.5 (JMS; SunOS 5.8 sun4u) S1MQ/3.5 (C; Linux i686) S1MQ/3.5 (JMS; Windows NT 5.0)	MQ3.5

JMQClusterID	String	(Optional). Used during reconnection to an HA cluster, this property indicates the clusterid used during the initial connection. (it is used to catch misconfigurations). This value should be the same as the JMQClusterID returned by HELLO_REPLY for the initial connection.	MQ4.0
JMQStoreSession	Long	(Optional). Used during reconnection to an HA cluster. This property is the store session id associated with the initial connection and is used to help the client identify what broker it should connect to during a fail over. This value should be the same as the JMQStoreSession returned by HELLO_REPLY for the initial connection.	MQ4.0

<b>Packet Body</b>
None

**HELLO\_REPLY: S -> C**

Server's reply to a HELLO packet

Fixed Header Fields	
Field	Value
packet type	11 (HELLO_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

<b>Variable Header Fields</b>
None

Properties:			
Property	Type	Value	Since
		OK(200) Server is responsive and will handle requests	

JMQStatus

Int

MQ2.0

NOT_ALLOWED(405)	Connection request can't be accepted due to some restriction (for example use of a license restricted feature). (Since 3.6)
UNAVAILABLE(503)	Server is too busy or a connection limit has been reached. Server is closing connection
BAD_VERSION(505)	Server does not support version of protocol specified
ERROR(500)	Internal server error
BAD_REQUEST(400)	JMQClusterID passed in HELLO does not match the cluster associated with the broker (misconfiguration).
MOVED_PERMANENTLY(301)	Another broker has taken over the passed in JMQStoreSession. When this error is returned, <b>JMQStoreOwner</b> will also be set. This is <b>not</b> a fatal connection error. The broker does not close the connection and it can be continued to be used by clients.
NOT_FOUND(404)	A failure has occurred.  The broker who owns JMQStoreSession can not be found. This is <b>not</b> a fatal connection error. The broker does not close the connection and it can be continued to be used by clients.

		<p>TIMEOUT(408)</p> <p>Takeover is in-process, client should RETRY.</p> <p>The takeover of the store associated with <b>JMQStoreSession</b> has not completed. The client should try again in a few seconds.</p> <p>This is <b>not</b> a fatal connection error. The broker does not close the connection and it can be continued to be used by clients.</p>	
JMQConnectionID	Long	A unique identifier for this connection. Should be passed in the HELLO packet on re-connects.	MQ3.0
JMQLicense	String	(Optional) The license being used by the server. This is the short license name like "try", "unl", etc.	MQ3.6
JMQLicenseDesc	String	(Optional) A description of the license being used by the server. This is a longer description like: "Sun Java(tm) System Message Queue 3.5 SP1 Enterprise Edition"	MQ3.6
JMQProtocolLevel	Int	<p>(Optional) States the level of protocol the server will speak on the connection.</p> <p>If status is OK, then this will be the protocol level passed on the HELLO. (note that in earlier versions of the protocol this was only returned if status was BAD_VERSION. If absent, the client should assume the protocol version specified in HELLO).</p> <p>If status is BAD_VERSION and the protocol level passed on HELLO was higher than the broker could handle, then this is the highest protocol level the broker supports.</p> <p>If status is BAD_VERSION and the protocol level passed on HELLO was lower than the broker could handle, then this is the lowest protocol level the broker supports.</p> <p>If status is BAD_VERSION then client must resend HELLO until it negotiates a valid version.</p>	MQ2.0
JMQVersion	String	The product version string of the server. If missing assume "2.0"	MQ3.0
JMQMaxMsgBytes	Long	(Optional) The maximum message size the broker can handle. Please don't send anything larger than this.	MQ3.0

JMQBytes	Long	(Optional) The maximum number of message bytes the client may write to this connection before pausing and waiting for a RESUME_FLOW from the broker. 0 Don't send any message -1 Unlimited [Not Set] Unlimited	MQ3.0
JMQSize	Integer	(Optional) The maximum number messages the client may write to this connection before pausing and waiting for a RESUME_FLOW from the broker. 0 Don't send any message -1 Unlimited [Not Set] Unlimited  <i><b>Warning!</b> This should have been "JMQCount", and may be changed in a future release.</i>	MQ3.0
JMQService	String	(Optional) The name of the service this connection is to.	MQ3.5
JMQHA	Boolean	(Optional). Indicates whether or not the client is connecting into to an HA cluster. (if unset, the client is not connecting to an HA cluster).	MQ4.0
JMQClusterID	String	(Optional). The ClusterID is a unique id assigned by the administrator to an HA cluster of brokers. This value is returned by HELLO_REPLY in HA clusters and should be sent to the broker during any future Reconnection attempts.	MQ4.0
JMQStoreSession	Long	(Optional). When a client connects into an HA cluster, it connects to a specific store session (an ID associated with a stream of messages sent to a broker). During a failover, message ordering is maintained in an HA cluster ONLY when the producing client follows the store session (which may be taken over by a new broker). This value should be returned to the broker on HELLO when a client attempt to reconnect during a failover.	MQ4.0
JMQBrokerList	String	(Optional). Comma repeated list of URL's for all brokers currently operating in the cluster.	MQ4.0
JMQStoreOwner	String	(Optional). Only set when the status code returned is MOVED_PERMANENTLY. This is the URL to the broker who has taken over the store identified by the JMQStoreSession passed in on HELLO.	MQ4.0

[Back to Summary](#)

---

**PING: S -> C**

**PING: C -> S**

Check the health of the client, broker, and the connection in between.

**Since:** MQ3.0 (S -> C)

**Since:** MQ3.5 (C -> S)

**Reply:** PING\_REPLY Typically only for C -> S packets.

Fixed Header Fields	
Field	Value
packet type	54 (PING)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	0 S -> C 1 C -> S
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bits	Not Applicable
priority	Not applicable
consumer ID	A unique ID that the sender can use to help handle the PING_REPLY response packet. Whatever value is specified here is copied into the PING_REPLY's consumer ID field by the server.

Variable Header Fields
None

Properties			
Property	Type	Value	Since
None			

**Packet Body:** Not Applicable

---

**PING\_REPLY: S -> C**

Server's reply to a PING packet

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	55 (CREATE_SESSION_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

<b>Variable Header Fields</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQStatus	Int	OK(200) Broker is OK ERROR(500) Broker is not OK	MQ3.5

[Back to Summary](#)

---

## REDELIVER: C -> S

Request the server to redeliver the messages that are specified in the packet body. Typically this will be used in by the client in two cases:

1. When `Session.recover()` is called. In this case the client will request the server to resend the unacknowledged messages specified in the packet body. The client should set the `JMQSetRedelivered` property to `True` so that the server sets the R (redelivered) bit on each redelivered message.
2. When `Session.rollback()` is called on a transacted session. This behaves as #1, but the client sets the `JMQSetRedelivered` property to `False` so that the redelivered state of the messages are left unchanged by the server.

In either case a message may not be redelivered if it has expired. Also, messages with higher priority may arrive before the requested messages are redelivered.

**Since:** MQ2.0

**Modified:** MQ3.0

**Reply:** None

Fixed Header Fields	
Field	Value
packet type	32 (REDELIVER)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit flags	Not Applicable
consumer ID, priority	Not Applicable

Variable Header Fields
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQSetRedelivered	Boolean	True : Server should set R (redelivered) bit on all redelivered messages. False: Server should leave the R bit unchanged on all redelivered messages. If this property is missing it is assumed to be False.	MQ2.0
JMQBodyType	Integer	(Optional) Format of packet's body. One of <ul style="list-style-type: none"> <li>● CONSUMERID_I_SYSMESSAGEID(1): Body is a series of marshalled consumer ID / SysMessageIDs with the consumer ID an 'int'. This was the case in V1.0.0 of the protocol</li> <li>● CONSUMERID_L_SYSMESSAGEID(2): Same as the above, but consumer ID is a 'long'.</li> </ul> <p>If not specified and the client is using protocol version 1.0.0 then assume CONSUMERID_I_SYSMESSAGEID(1), else assume CONSUMERID_L_SYSMESSAGEID(2).</p> <p>See BodyType for more information about body types.</p>	MQ3.0
JMQQuantity	Integer	<b>New.</b> (Optional) Number of acknowledgement blocks in the body. If this parameter is missing then you must compute the number of blocks by dividing the bodysize by the appropriate block size.	MQ3.0
JMQTransactionID	long	The transaction id (if any) associated with this redeliver request	

### **Packet Body**

The REDELIVER packet body is identical to that of the ACKNOWLEDGE packet. Each "acknowledgement block" specifies a message to redeliver.

[Back to Summary](#)

---

**RESUME\_FLOW: C -> S****RESUME\_FLOW: S -> C**

In MQ 2.0 the only flow control supported was flow from the broker to the client (message consumption). In 3.0 we added support for controlling flow from the client to the broker (message production). Both of these were done at a connection level. I.e. the flow of all messages through a connection was paused/resumed.

In MQ 3.5 we introduced a finer grained flow control. In this case there are separate flows for each consumer and each producer.

Flow control for message consumption is handled like this:

1. Broker sets the 'F' bit on the last JMS message packet written to the connection if the flow is paused due to connection flow control limits having been reached.
2. Broker sets the 'C' bit on the last JMS message packet written to the connection if the flow is paused due to consumer based flow control limits having been reached. Note that both the 'F' and the 'C' bits can be set if the flow was paused because both connection and consumer limits were reached.
3. When the client sees the 'C' and/or 'F' bit set it sends a RESUME\_FLOW to the broker when it is ready to get more messages. If the RESUME is for consumer based flow control then the JMQConsumerID property is set to indicate the consumer to resume flow for. If JMQConsumerID is not set, then the entire connection is resumed. The JMQSize property on the RESUME\_FLOW message specifies how many messages the client is prepared to accept before the broker should pause the flow again.

Flow control for message production is handled like this:

1. The client sets the 'F' bit on the last JMS message packet written to the connection if the flow is paused due to connection flow control limits having been reached. , or producer limits having been reached.
2. The client sets the 'C' bit on the last JMS message packet written to the connection if the flow is paused due to producer flow control limits having been reached. In this case the client also sets the PRODUCERID field (new in 3.5) on the last JMS message packet This lets the broker know which producer to RESUME. Note that both the 'F' and the 'C' bits can be set if the flow was paused because both connection and producer limits were reached.
3. When the broker sees the 'F' bit set on an incoming message it sends a RESUME\_FLOW to the client when it is ready to get more messages. Since this is resuming the connection, JMQProducerID is not set on RESUME\_FLOW.
4. When the broker sees the 'C' bit set on an incoming message it sends a RESUME\_FLOW to the client when it is ready to get more messages. Also, the incoming message's PRODUCERID field is copied into the JMQProducerID property of the RESUME\_FLOW message so that just the particular producer is resumed.

The initial value for JMQBytes, JMQSize and JMQMaxMsgBytes can be specified in the HELLO and HELLO\_REPLY packets.

**Since:** MQ2.0

**Modified:** MQ3.0, MQ3.5

**Reply:** None

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	(52) RESUME_FLOW
version, size, timestamp, sequence number, property offset, property size, encryption	As defined in the JMQ packet specification
source port	The port number that this packet originated from.
source IP	The IP address that this packet originated from.
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bit flags	Not Applicable
consumer ID	Not Applicable
priority	Not Applicable

<b>Header Strings</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQRBufferSize	Int	<b>Deprecated.</b> Replaced by JMQSize. Number of JMS messages server should write to connection before pausing until the next RESUME_FLOW packet is received from the client. 0 for unlimited (no flow control). If this property is missing then the server continues to use the value from the previous RESUME_FLOW or HELLO message. Optional.	MQ2.0
JMQSize	Int	(Optional). The maximum number of JMS messages that may be written to the connection before pausing until the next RESUME_FLOW packet is received. For producer flow control this should be considered the number of message <i>per producer</i> . 0 Don't send any message -1 Unlimited [Not Set] Unlimited  <b>Warning!</b> This should have been "JMQLimit", and may be changed in a future release.	MQ3.0
JMQBytes	Long	(Optional) The maximum number of message bytes that may be written to the connection before pausing until the next RESUME_FLOW is received. For producer flow control this should be considered the number of bytes <i>per producer</i> . 0 Don't send any message -1 Unlimited [Not Set] Unlimited	MQ3.0
JMQMaxMsgBytes	Long	(Optional) The maximum single message size that may be written to the connection.	MQ3.0
JMQProducerID	Long	(Optional) The producer the client may resume flow on. Only used for S -> C messages.	MQ3.5
JMQConsumerID	Long	(Optional) The producer the broker may resume flow for. Only used for C -> S messages.	MQ3.5

<b>Packet Body</b>
None

[Back to Summary](#)

---

**RESUME\_FLOW: C -> S****RESUME\_FLOW: S -> C**

In MQ 2.0 the only flow control supported was flow from the broker to the client (message consumption). In 3.0 we added support for controlling flow from the client to the broker (message production). Both of these were done at a connection level. I.e. the flow of all messages through a connection was paused/resumed.

In MQ 3.5 we introduced a finer grained flow control. In this case there are separate flows for each consumer and each producer.

Flow control for message consumption is handled like this:

1. Broker sets the 'F' bit on the last JMS message packet written to the connection if the flow is paused due to connection flow control limits having been reached.
2. Broker sets the 'C' bit on the last JMS message packet written to the connection if the flow is paused due to consumer based flow control limits having been reached. Note that both the 'F' and the 'C' bits can be set if the flow was paused because both connection and consumer limits were reached.
3. When the client sees the 'C' and/or 'F' bit set it sends a RESUME\_FLOW to the broker when it is ready to get more messages. If the RESUME is for consumer based flow control then the JMQConsumerID property is set to indicate the consumer to resume flow for. If JMQConsumerID is not set, then the entire connection is resumed. The JMQSize property on the RESUME\_FLOW message specifies how many messages the client is prepared to accept before the broker should pause the flow again.

Flow control for message production is handled like this:

1. The client sets the 'F' bit on the last JMS message packet written to the connection if the flow is paused due to connection flow control limits having been reached. , or producer limits having been reached.
2. The client sets the 'C' bit on the last JMS message packet written to the connection if the flow is paused due to producer flow control limits having been reached. In this case the client also sets the PRODUCERID field (new in 3.5) on the last JMS message packet This lets the broker know which producer to RESUME. Note that both the 'F' and the 'C' bits can be set if the flow was paused because both connection and producer limits were reached.
3. When the broker sees the 'F' bit set on an incoming message it sends a RESUME\_FLOW to the client when it is ready to get more messages. Since this is resuming the connection, JMQProducerID is not set on RESUME\_FLOW.
4. When the broker sees the 'C' bit set on an incoming message it sends a RESUME\_FLOW to the client when it is ready to get more messages. Also, the incoming message's PRODUCERID field is copied into the JMQProducerID property of the RESUME\_FLOW message so that just the particular producer is resumed.

The initial value for JMQBytes, JMQSize and JMQMaxMsgBytes can be specified in the HELLO and HELLO\_REPLY packets.

**Since:** MQ2.0

**Modified:** MQ3.0, MQ3.5

**Reply:** None

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	(52) RESUME_FLOW
version, size, timestamp, sequence number, property offset, property size, encryption	As defined in the JMQ packet specification
source port	The port number that this packet originated from.
source IP	The IP address that this packet originated from.
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bit flags	Not Applicable
consumer ID	Not Applicable
priority	Not Applicable

<b>Header Strings</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQRBufferSize	Int	<b>Deprecated.</b> Replaced by JMQSize. Number of JMS messages server should write to connection before pausing until the next RESUME_FLOW packet is received from the client. 0 for unlimited (no flow control). If this property is missing then the server continues to use the value from the previous RESUME_FLOW or HELLO message. Optional.	MQ2.0
JMQSize	Int	(Optional). The maximum number of JMS messages that may be written to the connection before pausing until the next RESUME_FLOW packet is received. For producer flow control this should be considered the number of message <i>per producer</i> . 0 Don't send any message -1 Unlimited [Not Set] Unlimited  <b>Warning!</b> This should have been "JMQCount", and may be changed in a future release.	MQ3.0
JMQBytes	Long	(Optional) The maximum number of message bytes that may be written to the connection before pausing until the next RESUME_FLOW is received. For producer flow control this should be considered the number of bytes <i>per producer</i> . 0 Don't send any message -1 Unlimited [Not Set] Unlimited	MQ3.0
JMQMaxMsgBytes	Long	(Optional) The maximum single message size that may be written to the connection.	MQ3.0
JMQProducerID	Long	(Optional) The producer the client may resume flow on. Only used for S -> C messages.	MQ3.5
JMQConsumerID	Long	(Optional) The producer the broker may resume flow for. Only used for C -> S messages.	MQ3.5

<b>Packet Body</b>
None

[Back to Summary](#)

**SET\_CLIENTID: C -> S**

Set (or unset) the ClientID for the current connection with the broker.

The SET\_CLIENTID packet verifies the uniqueness of a clientID and associates it with the connection.

**Since:** MQ2.0

**Modified:** MQ3.5, MQ3.6

**Reply:** SET\_CLIENTID\_REPLY

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	50 (SET_CLIENTID)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bit flags	Not Applicable
priority	Not applicable
consumer ID	A unique ID that the sender can use to help correlate the SET_CLIENTID_REPLY response packet. Whatever value is specified here is copied into the SET_CLIENTID_REPLY's consumer ID field by the server.

<b>Variable Header Fields</b>	
None	

Properties			
Property	Type	Value	Since
JMQClientID	String	Client ID to verify and set. If this property is not set, and there is a clientID currently associated with the connection then the clientID is unset. The clientID should be unset only after all Sessions are closed on a connection, otherwise non-deterministic behavior may occur.	MQ2.0 (unset since 3.6)
JMQShare	Boolean	(Optional) Specifies if the clientID may be shared with other connections or not. If "true" the ClientID may be shared with other connections (which is NOT the standard JMS semantics). If "false" ClientID may NOT be shared (standard JMS semantics). Default is "false".	MQ3.5
JMQNamespace	String	<p>(Optional) Ignored if JMQShare is false. Specifies the range of scope for clientID sharing. The server must ensure that JMQClientID is unique for all requests within the same namespace -- it may only be shared between requests with different namespaces (or between requests with no namespace specified).</p> <p>In particular SET_CLIENTID will return CONFLICT if any of the following are true:</p> <ol style="list-style-type: none"> <li>1. JMQClientID is already active and JMQClientID was allocated with JMQShared=false</li> <li>2. JMQClientID is already active and JMQClientID was allocated with JMQShared=true and JMQClientID was allocated with JMQNamespace set to a value that is equivalent to JMQNamespace set on the new request.</li> <li>3. JMQClientID is already active and JMQClientID was allocated with JMQShared=true and JMQClientID was allocated with JMQNamespace NOT set and JMQNamespace is set on the new request</li> </ol>	MQ3.6

**Packet Body:** Not Applicable

---

### SET\_CLIENTID\_REPLY: S -> C

Server's reply to a SET\_CLIENTID\_REPLY packet

**Reply:** None

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
packet type	50 (SET_CLIENTID_REPLY)
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

<b>Variable Header Fields</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQStatus	Int	<p>ClientID is unique and not in use OR The ClientID is in use, but is shareable (ie OK(200) JMQShare was true on the request that initially set the ClientID) and JMQShare is true on this request.</p> <p>BAD_REQUEST(400) Invalid ClientID</p> <p>CONFLICT(409) ClientID is already in use, and is not shareable.</p> <p>ERROR(500) Internal server error</p>	MQ2.0

[Back to Summary](#)

**START: C -> S**

Tell the router to start sending JMS messages to this connection.

**Since:** MQ2.0

**Modified:** MQ3.5

**Reply:** None

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	20 (START)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bit flags	Not Applicable
consumer ID, priority	Not Applicable

<b>Header Strings</b>
None

<b>Properties:</b>			
JMQSessionID	Long	(Optional) Instead of starting the entire connection, start just the specified session.	MQ3.5

<b>Packet Body</b>
None

[Back to Summary](#)

---

**START: C -> S**

Tell the router to start sending JMS messages to this connection.

**Since:** MQ2.0

**Modified:** MQ3.5

**Reply:** None

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	20 (START)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bit flags	Not Applicable
consumer ID, priority	Not Applicable

<b>Header Strings</b>
None

<b>Properties:</b>			
JMQSessionID	Long	(Optional) Instead of starting the entire connection, start just the specified session.	MQ3.5

<b>Packet Body</b>
None

[Back to Summary](#)

---

**VERIFY\_DESTINATION: C -> S**

The VERIFY\_DESTINATION packet requests that the server verify a destination and (optionally) a selector. It does not alter the state of the server.

**Since:** MQ2.0

**Reply:** VERIFY\_DESTINATION\_REPLY

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	23 (STOP_REPLY)
packet type	40 (VERIFY_DESTINATION)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
All other bit flags	Not Applicable
priority, hop TTL	Not applicable
consumer ID	A unique ID that the sender can use to help handle the VERIFY_DESTINATION_REPLY response packet. Whatever value is specified here is copied into the VERIFY_DESTINATION_REPLY's consumer ID field by the server.

<b>Variable Header Fields</b>
None

<b>Properties</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQDestination	String	Topic or Queue Name to verify.	MQ2.0
JMQSelector	String	(Optional). Selector to verify.	MQ2.0
JMQDestType	Int	Type of destination that JMQDestination is. Should be DEST_TYPE_TOPIC or DEST_TYPE_QUEUE optionally OR'd with DEST_TEMP if it is a temporary destination. See DestType for more information.	MQ2.0?

**Packet Body:** Not Applicable

---

**VERIFY\_DESTINATION\_REPLY: S -> C**

Server's reply to a VERIFY\_DESTINATION packet

Fixed Header Fields	
Field	Value
packet type	41 (VERIFY_DESTINATION_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

Variable Header Fields
None

Properties:			
Property	Type	Value	Since
JMQStatus	Int	OK(200) Destination is valid BAD_REQUEST(400) Selector is invalid FORBIDDEN(403) Client is not authorized to access destination NOT_FOUND(404) Destination was not found ERROR(500) Internal server error	MQ2.0
JMQCanCreate	Boolean	Set only if JMQStatus is NOT_FOUND(404). This is used to inform the client if the server will allow the destination to be created via CREATE_DESTINATION (i.e. support auto-create). True Destination can be created with CREATE_DESTINATION False Destination cannot be created with CREATE_DESTINATION	MQ2.0
JMQDestType	Int	(Optional). JMQDestType that was passed in ADD_CONSUMER OR'd with the destination's flavor. This basically informs the Client of the destination's flavor. See DestType for more information. If missing assume default flavor.	MQ3.0

[Back to Summary](#)

**START\_TRANSACTION: C -> S**

Initiates a transaction with the server. This packet specifies a transaction ID that the client will tag onto all JMS message packets produced in the transaction, and all ACKNOWLEDGE packets generated during the transaction. This transaction ID is also used when the transaction is committed or rolled back.

**Since:** MQ2.0

**Modified:** MQ3.0, MQ3.5, MQ4.0

**Reply:** START\_TRANSACTION\_REPLY

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	44 (START_TRANSACTION)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bit flags	Not Applicable
priority	Not Applicable
consumer ID	A unique ID that the sender can use to help correlate the START_TRANSACTION_REPLY response packet. Whatever value is specified here is copied into the START_TRANSACTION_REPLY's consumer ID field by the server.

<b>Variable Header Fields</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>

JMQTransactionID	Int	<p><b>Deprecated.</b> This parameter is only supported for backwards compatibility with clients using the 1.0 version of the protocol (iMQ 2.0 clients). It is only supported for those clients. This parameter has been replaced by <code>START_TRANSACTION_REPLY</code> returning the Transaction ID for the client to use. Transaction ID for transaction. Generated by the client. Only uniqueness criteria is that the client's Connection must not have any other open transaction with this ID. So basically this ID corresponds to a "session ID". All messages sent by the client that are in this transaction should be stamped with the transactionID specified by JMQTransactionID.</p>	MQ2.0
JMQXAFlags	Int	<p>(Optional). Required for XATransactions. Must not be present for local transactions. One of:</p> <ul style="list-style-type: none"> <li>● XAResource.TMRESUME: If resuming a suspended transaction.</li> <li>● XAResource.TMNOFLAGS: If starting a new transaction.</li> <li>● XAResource.TMJJOIN: If joining a transaction after a failover (optional). .</li> </ul>	MQ3.0
JMQSessionID	Long	<p>Specifies the session this transaction is being started on if this is a local transaction. Used only for local (non XA) transactions. This ID is returned by <code>CREATE_SESSION</code>.</p>	MQ3.5

<p>JMQAutoRollback</p>	<p>int</p>	<p>(Optional) Indicated when transactions should be auto rollback:</p> <table border="1" data-bbox="602 380 1279 1264"> <thead> <tr> <th>#</th> <th>Flag</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>ALL</td> <td>rollback all transactions which have not be committed on broker failure.</td> </tr> <tr> <td>2</td> <td>NOT_PREPARED</td> <td>rollback all transactions which have not be PREPARED on broker failure.</td> </tr> <tr> <td>3</td> <td>NEVER</td> <td>never rollback transactions on broker failure (JMQLifetime must be set).  <b>NOTE:</b> NEVER is not targeted to be implemented in the 4.0 release. Setting JMQAutoRollback to NEVER will result in an error code of NOT_IMPLEMENTED</td> </tr> <tr> <td><i>unset</i></td> <td></td> <td>XA transactions default to ALL or NOT_PREPARED depending on the setting of imq.transaction.autorollback. Local transactions default to ALL</td> </tr> </tbody> </table>	#	Flag	Description	1	ALL	rollback all transactions which have not be committed on broker failure.	2	NOT_PREPARED	rollback all transactions which have not be PREPARED on broker failure.	3	NEVER	never rollback transactions on broker failure (JMQLifetime must be set).  <b>NOTE:</b> NEVER is not targeted to be implemented in the 4.0 release. Setting JMQAutoRollback to NEVER will result in an error code of NOT_IMPLEMENTED	<i>unset</i>		XA transactions default to ALL or NOT_PREPARED depending on the setting of imq.transaction.autorollback. Local transactions default to ALL	<p>MQ4.0</p>
#	Flag	Description																
1	ALL	rollback all transactions which have not be committed on broker failure.																
2	NOT_PREPARED	rollback all transactions which have not be PREPARED on broker failure.																
3	NEVER	never rollback transactions on broker failure (JMQLifetime must be set).  <b>NOTE:</b> NEVER is not targeted to be implemented in the 4.0 release. Setting JMQAutoRollback to NEVER will result in an error code of NOT_IMPLEMENTED																
<i>unset</i>		XA transactions default to ALL or NOT_PREPARED depending on the setting of imq.transaction.autorollback. Local transactions default to ALL																
<p>JMQSessionLess</p>	<p>Boolean</p>	<p>(Optional) Indicates whether a transaction should be rollback when its associated Session or connection is closed.</p> <ul style="list-style-type: none"> <li>● TRUE - transaction is not rollback</li> <li>● FALSE - transaction is rollback</li> <li>● <i>unset</i> - XA transactions are not rollback, other transactions are</li> </ul>	<p>MQ4.0</p>															
<p>JMQLifetime</p>	<p>Long</p>	<p>(Optional) Indicates the maximum lifetime of a non-PREPARED transaction before it will be automatically rolled back. (0 indicated never expire).  This property is not supported in Hawk (it is added for future changes)</p>	<p>MQ4.0</p>															

**Packet Body**

Required for XA transactions only. Format is a marshalled Xid

**START\_TRANSACTION\_REPLY: S -> C**

Server's reply to a START\_TRANSACTION packet

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	45 (START_TRANSACTION_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

**Variable Header Fields**

None

**Properties:**

<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQTransactionID	Long	Transaction ID for transaction. The client should use this ID for all future operations on this transaction (whether XA or not). The client should set this ID on every JMS Message Packet and ACKNOWLEDGE packet that is part of a transaction.	MQ3.0
JMQStatus	Int	OK(200) Transaction started CONFLICT(409) TransactionID in use ERROR(500) Internal server error NOT_MODIFIED(304) During a redeliver, the transaction was already STARTED NOT_IMPLEMENTED(501) Returned if the setting for JMQAutoRollback is not supported by this broker	MQ2.0

[Back to Summary](#)

---

## COMMIT\_TRANSACTION: C -> S

Commit a transaction. The server will send all messages in the transaction and process all acknowledgements in the transaction. In other words the server will send all JMS messages that had been tagged with the specified transaction ID and process all acknowledgement requests that had been tagged with the specified transaction id.

Note that when the client receives the COMMIT\_REPLY the server may not have completed processing all messages and acknowledgements -- but it is promising that the processing will occur as an atomic operation.

**Since:** MQ2.0

**Modified:** MQ3.0

**Reply:** COMMIT\_TRANSACTION\_REPLY

Fixed Header Fields	
Field	Value
packet type	46 (COMMIT_TRANSACTION)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bits	Not Applicable
priority	Not Applicable
consumer ID	If A bit is set this should be a unique ID that the sender can use to help correlate the COMMIT_TRANSACTION_REPLY response packet. Whatever value is specified here is copied into the COMMIT_TRANSACTION_REPLY's consumer ID field by the server.

Variable Header Fields	
None	

Properties:			
Property	Type	Value	Since
JMQTransactionID	Long	Transaction ID to commit. This should correspond to the ID returned by START_TRANSACTION_REPLY. All messages and acknowledgements produced between the START_TRANSACTION and COMMIT_TRANSACTION are processed by the server.	MQ3.0
JMQXAFlags	Int	(Optional). Required for XATransactions. Must not be present for local transactions. One of: <ul style="list-style-type: none"> <li>● XAResource.TMONEPHASE: One phase commit. Transaction need not be in PREPARED state.</li> <li>● XAResource.TMNOFLAGS: Two phase commit. Transaction must be in PREPARED state</li> </ul>	MQ3.0
JMQTransactionID	Int	<b>Deprecated.</b> <i>This parameter is only supported for backwards compatibility with clients using the 1.0 version of the protocol (iMQ 2.0 clients). It is only supported for those clients. This parameter has been replaced by JMQTransactionID Long.</i> Transaction ID to commit. This should correspond to the ID specified in the START_TRANSACTION message. All messages and acknowledgements produced between the START_TRANSACTION and COMMIT_TRANSACTION are processed by the server.	MQ2.0
Packet Body			
Required for XA transactions only. Format is a marshalled Xid			

## COMMIT\_TRANSACTION\_REPLY: S -> C

Server's reply to a COMMIT\_TRANSACTION packet

Fixed Header Fields	
Field	Value
packet type	47 (COMMIT_TRANSACTION_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

Variable Header Fields
None

Properties:			
Property	Type	Value	Since
JMQRemote	Boolean	Remote brokers are associated in this transaction	4.1
JMQStatus	Int	<p>OK(200) Transaction committed</p> <p>BAD_REQUEST(400) Invalid TransactionID</p> <p>NOT_FOUND(404) Unknown TransactionID</p> <p>Transaction is not in a state that can be committed.</p> <p>PRECONDITION_FAILED(412)</p> <p>GONE(410) A Remote broker associated with the transaction is down</p> <p>ERROR(500) Internal server error</p> <p>TIMEOUT(408) transaction was rolledback back because JMLifetime was reached.</p>	MQ2.0 (MQ4.1)
JMQTransactionID	Long	JMQTransactionID that was committed. Will typically match the ID passed in the request.	MQ3.0

[Back to Summary](#)

---

## ROLLBACK\_TRANSACTION: C -> S

Rollback a transaction. The server will discard all messages and acknowledgements in the transaction and release the transaction ID. In other words the server will discard all JMS messages that had been tagged with the specified transaction ID and discard all acknowledgement requests that had been tagged with the specified transaction id.

**Since:** MQ2.0

**Modified:** MQ3.0

**Reply:** (Optional) ROLLBACK\_TRANSACTION\_REPLY

Fixed Header Fields	
Field	Value
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
packet type	48 (ROLLBACK_TRANSACTION)
A Bit	0 = Do not send ROLLBACK_TRANSACTION_REPLY. This is typically the case for non-XA transactions. 1 = Send a ROLLBACK_TRANSACTION_REPLY. Required for XA transactions.
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bit flags	Not Applicable
consumer ID	Optional. If A bit is set, then consumer ID is set to a value that allows the client to correlate the response with this request.
priority	Not Applicable

Header Strings
None

Properties:			
Property	Type	Value	Since

JMQTransactionID	Long	Transaction ID to rollback. This should correspond to the ID returned in the START_TRANSACTION_REPLY packet. All messages and acknowledgements generated between the START_TRANSACTION and ROLLBACK_TRANSACTION are discarded. Required for JMS Transactions. Optional but highly desired for XA transactions.	MQ3.0
JMQTransactionID	Int	<b>Deprecated.</b> <i>This parameter is only supported for backwards compatibility with clients using the 1.0 version of the protocol (iMQ 2.0 clients). It is only supported for those clients. This parameter has been replaced by JMQTransactionID Long.</i> Transaction ID to rollback. This should correspond to the ID specified in the START_TRANSACTION message. All messages and acknowledgements generated between the START_TRANSACTION and ROLLBACK_TRANSACTION are discarded.	MQ2.0
JMQRedeliver	Boolean	(Optional) True : If the transaction contains acknowledgements, then automatically trigger a redelivery of the messages associated with the acknowledgements (as opposed to the more typical case where the client requests a redelivery using REDELIVER). If the destination is a queue (or a shared durable) then it is not defined which consumer will get the redelivered messages. If the destination is a topic, then an attempt is made to redeliver the messages to the consumer that generated the original acknowledgements. If that's not possible, then no redelivery is triggered. The redelivered messages will be in order with respect to each other, but they may be delivered out-of-order with respect to messages already sent to the consuming client.  This option is typically used only with XA transactions by the JMS JCA resource adapter.  False: Do not trigger a redeliver -- client must request it explicitly using REDELIVER  Default: False	MQ3.5sp1
JMQSetRedelivered	Boolean	(Optional) Only applicable if JMQRedeliver is 'true'. True : Server should set R (redelivered) bit on all redelivered messages. False: Server should leave the R bit unchanged on all redelivered messages. Default: True	MQ3.5sp1

**Packet Body**

Required for XA transactions only. Format is a marshalled Xid

**ROLLBACK\_TRANSACTION\_REPLY: S -> C**

Server's reply to a ROLLBACK\_TRANSACTION packet

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	49 (ROLLBACK_TRANSACTION_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

**Variable Header Fields**

None

**Properties:**

<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQStatus	Int	OK(200) Transaction rolled-back BAD_REQUEST(400) Invalid TransactionID NOT_FOUND(404) Unknown TransactionID PRECONDITION_FAILED(412) Transaction is not in a state that can be rolled-back. ERROR(500) Internal server error	MQ2.0
JMQTransactionID	Long	JMQTransactionID that was rolledback. Will typically match the ID passed in the request.	MQ3.0

**Body:** None

[Back to Summary](#)

---

**END\_TRANSACTION: C -> S**

Ends or suspends an XA transaction on the server.

**Since:** MQ3.0

**Reply:** END\_TRANSACTION\_REPLY

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	58 (END_TRANSACTION)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bit flags	Not Applicable
priority	Not Applicable
consumer ID	A unique ID that the sender can use to help correlate the END_TRANSACTION_REPLY response packet. Whatever value is specified here is copied into the END_TRANSACTION_REPLY's consumer ID field by the server.

<b>Variable Header Fields</b>	
None	

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQTransactionID	Long	(Optional) Transaction ID to prepare. This is the id that was returned by START_TRANSACTION. If this parameter is missing then the Xid in the body of this packet will be used to identify the transaction. It is strongly recommended that the JMQTransactionID is used as it is faster.	MQ3.0
JMQXAFlags	Int	One of: <ul style="list-style-type: none"> <li>● XAResource.TMSUSPEND: If suspending a transaction.</li> <li>● XAResource.TMFAIL: If failing a transaction.</li> <li>● XAResource.TMSUCCESS: If ending a transaction.</li> </ul>	MQ3.0
<b>Packet Body</b>			
Required. Xid of transaction to prepare. Format is: marshalled Xid			

## END\_TRANSACTION\_REPLY: S -> C

Server's reply to an END\_TRANSACTION packet

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	59 (END_TRANSACTION_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.
<b>Variable Header Fields</b>	
None	

Properties:			
Property	Type	Value	Since
JMQStatus	Int	OK(200) Transaction committed BAD_REQUEST(400) Invalid TransactionID NOT_FOUND(404) Unknown TransactionID PRECONDITION_FAILED(412) Transaction is not in a state that can be prepared. ERROR(500) Internal server error NOT_MODIFIED(304) During a redeliver, the transaction was already ENDED TIMEOUT(408) transaction was rolledback back because JMQLifetime was reached.	MQ3.0
JMQTransactionID	Long	JMQTransactionID that was ended. Will typically match the ID passed in the request.	MQ3.0

**Body:** None

[Back to Summary](#)

**PREPARE\_TRANSACTION: C -> S**

Prepares an XA transaction on the server.

**Since:** MQ3.0

**Reply:** PREPARE\_TRANSACTION\_REPLY

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	56 (PREPARE_TRANSACTION)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bit flags	Not Applicable
priority	Not Applicable
consumer ID	A unique ID that the sender can use to help correlate the PREPARE_TRANSACTION_REPLY response packet. Whatever value is specified here is copied into the PREPARE_TRANSACTION_REPLY's consumer ID field by the server.

<b>Variable Header Fields</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQTransactionID	Long	(Optional) Transaction ID to prepare. This is the id that was returned by START_TRANSACTION. If this parameter is missing then the Xid in the body of this packet will be used to identify the transaction. It is strongly recommended that the JMQTransactionID is used as it is faster.	MQ3.0

<b>Packet Body</b>
Required. Xid of transaction to prepare. Format is: marshalled Xid

**PREPARE\_TRANSACTION\_REPLY: S -> C**

Server's reply to a PREPARE\_TRANSACTION packet

Fixed Header Fields	
Field	Value
packet type	57 (START_TRANSACTION_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

Variable Header Fields
None

Properties:			
Property	Type	Value	Since
JMQStatus	Int	<p>OK(200) Transaction committed</p> <p>BAD_REQUEST(400) Invalid TransactionID</p> <p>NOT_FOUND(404) Unknown TransactionID</p> <p>PRECONDITION_FAILED(412) Transaction is not in a state that can be prepared.</p> <p>GONE(500) A remote broker which is associated with this transaction is down.</p> <p>ERROR(500) Internal server error</p> <p>NOT_MODIFIED(304) During a redeliver, the transaction was already PREPARED</p> <p>TIMEOUT(408) transaction was rolledback back because JMQLifetime was reached.</p>	MQ3.0
JMQTransactionID	Long	JMQTransactionID that was prepared. Will typically match the ID passed in the request.	MQ3.0

[Back to Summary](#)

---

## RECOVER\_TRANSACTION: C -> S

Recovers a list of Xid's that are in Prepared state, or check if a particular transaction is in Prepared state.

**Since:** MQ3.0

**Reply:** RECOVER\_TRANSACTION\_REPLY

Fixed Header Fields	
Field	Value
packet type	60 (RECOVER_TRANSACTION)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bit flags	Not Applicable
priority	Not Applicable
consumer ID	A unique ID that the sender can use to help correlate the END_TRANSACTION_REPLY response packet. Whatever value is specified here is copied into the END_TRANSACTION_REPLY's consumer ID field by the server.

Variable Header Fields	
None	

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQTransactionID	Long	(Optional) Transaction ID to recover. This is an id that was returned by START_TRANSACTION. If this property is present and not 0 then the server checks to see if the specified transaction is in PREPARED state. If this property is missing then the request is to recover a list of all Xid's that are in Prepared state (the common XA usage).	MQ3.5
JMQXAFlags	Int	Cursor positioning for scanning Xids. In the current implementation all Xids are returned on a TMSTARTSCAN and 0 are returned on all other requests. If JMQTransactionID is set then this property is ignored. <ul style="list-style-type: none"> <li>● XAResource.TMSTARTSCAN</li> <li>● XAResource.TMENDSCAN</li> <li>● XAResource.TMNOFLAGS</li> </ul>	MQ3.0
<b>Packet Body</b>			
None.			

## RECOVER\_TRANSACTION\_REPLY: S -> C

Server's reply to a RECOVER\_TRANSACTION packet

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	59 (END_TRANSACTION_REPLY)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler.

<b>Variable Header Fields</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQQuantity	Int	Number of Xid's in the body of the packet.	MQ3.0
JMQTransactionID	Long	(Optional) If JMQTransactionID was specified in the request packet, and the transaction was found to be in a Prepared state, then this property is set to the value of JMQTransactionID in the request and JMQStatus is OK. If the transaction is not found, or is not in the prepared state, then this property will not be set, and JMQStatus will be NOT_FOUND.	MQ3.5
JMQStatus	Int	OK(200) Transactions recovered BAD_REQUEST(400) Invalid JMQTransactionID NOT_FOUND(404) Unknown JMQTransactionID ERROR(500) Internal server error	MQ3.0
<b>Packet Body</b>			
A Sequence of 0 or more marshalled Xids. The number of Xid's is specified by the JMQQuantity property. Will only be set if satus is OK.			

[Back to Summary](#)

**VERIFY\_TRANSACTION: C -> S**

Request information about a specific transaction from the broker.

The VERIFY\_TRANSACTION is sent by the client if wants to receive information about the status of a specific transaction.

**Since:** MQ4.0

**Modified:** N/A

**Reply:** VERIFY\_TRANSACTION\_REPLY

Fixed Header Fields	
Field	Value
packet type	80 (VERIFY_TRANSACTION)
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
A bit	1 (server must send a reply)
I Bit	Optional. 1 = this packet is being resent by the client, do not log any errors which are caused because this message is being reprocessed . 0 = this packet is new.
All other bits	Not Applicable
priority	Not applicable
consumer ID	A unique ID that the sender can use to help handle the VERIFY_TRANSACTION_REPLY response packet. Whatever value is specified here is copied into the VERIFY_TRANSACTION_REPLY's consumer ID fieldby the server. <b>Note:</b> this is NOT the ID of the consumer to register interest for. That is specified by the JMQConsumerID property.

Variable Header Fields
None

Properties			
Property	Type	Value	Since
JMQTransactionID	Int	Transaction to lookup.	MQ4.0

**Packet Body:** None

**VERIFY\_TRANSACTION\_REPLY: S -> C**

Reply to an VERIFY\_TRANSACTION packet.

**Modified:** N/A

<b>Fixed Header Fields</b>	
<b>Field</b>	<b>Value</b>
packet type	81 (VERIFY_TRANSACTION_REPLY),
version, size, timestamp, sequence number, source port, sender IP, property offset, property size, encryption	As defined in the JMQ packet specification
All bit fields	Not Applicable
priority	Not Applicable
consumer ID	The consumer ID that was set on the request packet that caused this reply packet to be generated. This lets the recipient of this reply to more easily dispatch this packet to the correct handler. Note: This is NOT the consumer ID specified in the JMQConsumerID property. Rather the consumer ID set on the packet's fixed header field to use as a correlation ID.

<b>Variable Header Fields</b>
None

<b>Properties:</b>			
<b>Property</b>	<b>Type</b>	<b>Value</b>	<b>Since</b>
JMQTransactionID	Int	Transaction which corresponds to information in this packet.	MQ4.0
JMQStatus	Int	OK(200) Information is returned NOT_FOUND(404) The transaction could not be found. On a rollback or commit, this indicates that the action has been completed ERROR(500) Internal server error	MQ4.0

<b>Packet Body</b>			
A Serialized Hashtable which contains information about the resource.			
<b>Name</b>	<b>Type</b>	<b>Since</b>	<b>Description</b>

JMQAutoRollback	Integer	MQ4.0	Rollback Flag set on the transaction: <table border="1" data-bbox="716 348 1370 701"> <thead> <tr> <th>#</th> <th>Flag</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>ALL</td> <td>rollback all transactions which have not be committed on failover</td> </tr> <tr> <td>2</td> <td>NOT_PREPARED</td> <td>rollback all transactions which have not be PREPARED on failover</td> </tr> <tr> <td>3</td> <td>NEVER</td> <td>never rollback transactions on failover</td> </tr> </tbody> </table>	#	Flag	Description	1	ALL	rollback all transactions which have not be committed on failover	2	NOT_PREPARED	rollback all transactions which have not be PREPARED on failover	3	NEVER	never rollback transactions on failover						
#	Flag	Description																			
1	ALL	rollback all transactions which have not be committed on failover																			
2	NOT_PREPARED	rollback all transactions which have not be PREPARED on failover																			
3	NEVER	never rollback transactions on failover																			
JMQXid	String	MQ4.0	The Xid (if any) associated with this transaction																		
JMQSessionLess	Boolean	MQ4.0	Is the transaction associated with a session: <ul style="list-style-type: none"> <li>● TRUE - do not rollback when connection/session closes</li> <li>● FALSE - rollback when connection/session closes</li> </ul>																		
JMQLifetime	Long	MQ4.0	Length of time the transaction will live (in milliseconds). 0 indicates forever.																		
JMQCreateTime	Long	MQ4.0	Time the transaction was started																		
State	Integer	MQ4.0	State of the transaction, one of: <table border="1" data-bbox="716 1096 1370 1518"> <tbody> <tr><td>1</td><td>CREATED</td></tr> <tr><td>2</td><td>STARTED</td></tr> <tr><td>3</td><td>FAILED</td></tr> <tr><td>4</td><td>INCOMPLETE</td></tr> <tr><td>5</td><td>COMPLETE</td></tr> <tr><td>6</td><td>PREPARED</td></tr> <tr><td>7</td><td>COMMITTED</td></tr> <tr><td>8</td><td>ROLLEDBACK</td></tr> <tr><td>9</td><td>TIMEDOUT</td></tr> </tbody> </table> <p>NOTE: in many cases a 404 status code will be returned in the case of ROLLEDBACK or COMMITTED. On a 2 phase commit, this indicates that the previous operation (which ever one) was successful. This is because the broker can not keep around the actual transaction state if the operation successfully completed.</p>	1	CREATED	2	STARTED	3	FAILED	4	INCOMPLETE	5	COMPLETE	6	PREPARED	7	COMMITTED	8	ROLLEDBACK	9	TIMEDOUT
1	CREATED																				
2	STARTED																				
3	FAILED																				
4	INCOMPLETE																				
5	COMPLETE																				
6	PREPARED																				
7	COMMITTED																				
8	ROLLEDBACK																				
9	TIMEDOUT																				

[Back to Summary](#)

## Destination Types in the Swift Protocol

@(#)DestType.html 1.2 05/01/18

A destination in MQ has three attributes:

1. Its type: Queue or Topic
2. Its lifespan: Temporary or not
3. Its flavor: Single, Round Robin, Failover, etc

Not all combinations of attributes are necessarily valid. For example a Round-Robin Topic is not supported. But the mechanism for specifying destinations should be flexible enough to support any combination just in case the need arises in the future.

In JMQ2.0, all interfaces that pass a destination type should use an int that is constructed from OR'ing the appropriate bit fields specified by the `com.sun.messaging.jmq.util.DestType` class.

Examples:

```
A topic:      int type = DestType.DEST_TYPE_TOPIC;
A temporary topic:  int type = DestType.DEST_TYPE_TOPIC |
                   DestType.DEST_TEMP;
A round robin queue:  int type = DestType.DEST_TYPE_QUEUE |
                   DestType.DEST_FLAVOR_RROBIN;
A single subscribr queue:  int type = DestType.DEST_TYPE_QUEUE;
                           or
                           int type = DestType.DEST_TYPE_QUEUE |
                                   DestType.DEST_FLAVOR_SINGLE;
```

As shown with the last example if the queue flavor is missing it will default to the appropriate value.



## Body Types in the iMQ Protocol

@(#)BodyType.html 1.2 05/01/18

Some control packets in the MQ protocol can have different body (payload) formats. These packets use a `JMQBodyType` property to describe the data format. The descriptions here should be considered general guidelines. The definitive descriptions are given in the specifications for the actual packets.

The `com.sun.messaging.jmq.io.PacketType` class defines constants for use in the `JMQBodyType` property. Here they are

<code>NONE(0)</code>	No data in body
<code>CONSUMERID_I_SYSMESSAGEID(1)</code>	A sequence of marshalled Consumer ID / SysMessageID pairs where the Consumer ID is an int. In iMQ2.0 Consumer ID's were ints. After 2.0 they became longs.
<code>CONSUMERID_L_SYSMESSAGEID(2)</code>	A sequence of marshalled Consumer ID / SysMessageID ID pairs where the Consumer ID is a long
<code>SYSMESSAGEID(3)</code>	A sequence of marshalled SysMessageIDs
<code>SESSIONID_SYSMESSAGEID(4)</code>	A single session ID (long) SysMessageID pair