

# Advanced Message Queuing Protocol (AMQP) JMS Mapping Version 1.0

# Working Draft 5 19 May 2014

### **Specification URIs**

#### This version:

http://docs.oasis-open.org/amqp-bindmap/jms/v1.0/wd05/amqp-bindmap-jms-v1.0-wd05.xml (Authoritative) http://docs.oasis-open.org/amqp-bindmap/jms/v1.0/wd05/amqp-bindmap-jms-v1.0-wd05.html http://docs.oasis-open.org/amqp-bindmap/jms/v1.0/wd05/amqp-bindmap-jms-v1.0-wd05.pdf

#### **Previous version:**

N/A

#### Latest version:

http://docs.oasis-open.org/amqp-bindmap/jms/v1.0/amqp-bindmap-jms-v1.0.html http://docs.oasis-open.org/amqp-bindmap/jms/v1.0/amqp-bindmap-jms-v1.0.pdf http://docs.oasis-open.org/amqp-bindmap/jms/v1.0/amqp-bindmap-jms-v1.0.xml (Authoritative)

#### **Technical Committee:**

OASIS Advanced Message Queuing Protocol (AMQP) Bindings and Mappings (AMQP-BINDMAP) TC

#### Chairs:

Steve Huston (shuston@riverace.com), Individual

#### **Editors:**

Robert Gemmell (robert.gemmell@jpmchase.com), JPMorgan Chase & Co. Robert Godfrey (robert.godfrey@jpmorgan.com), JPMorgan Chase & Co.

#### Abstract:

TODO

#### Status:

This document was last revised or approved by the membership of OASIS on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committees/amap-bindmap/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/amqp-bindmap/ipr.php).

#### **Citation format:**

When referencing this specification the following citation format should be used:

#### [amqp-bindmap-jms-v1.0]

Advanced Message Queuing Protocol (AMQP) JMS Mapping Version 1.0. 19 May 2014. OASIS Working Draft 5. http://docs.oasis-open.org/amqp-bindmap/jms/v1.0/wd05/amqp-bindmap-jms-v1.0-wd05.pdf.

#### **Notices:**

Copyright © OASIS Open 2013. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/policies-guidelines/trademark for above guidance.

# **Contents**

1	References	4
2	Mapping JMS Types to AMQP	5
3	Mapping AMQP Types to Java	6
4	Message Structure 4.1 Message Structure 4.2 Mapping JMS Messages To AMQP 4.2.1 JMS Headers 4.2.2 JMS-defined 'JMSX' Properties 4.2.3 JMS Properties 4.2.4 Message Body Types 4.3 Mapping AMQP Messages To Java 4.3.1 Header Section 4.3.2 Properties Section 4.3.3 Application Properties Section 4.3.4 Delivery Annotations Section 4.3.5 Message Annotations Section 4.3.6 Footer Section 4.3.7 Body Sections	7 8 13 14 15 19 20 23 23 23 23
5	JMS Vendor Properties	26
6	Destinations	27
7	Message Producers 7.1 Sending Messages	<b>28</b> 28
8	Delivery Count Handling	29
9	Supplementary Definitions	30

# 1 References

TODO (presentation): move refs to wherever they are meant to go, ensure they are structured correctly, etc.

#### [AMQPMSGFORMAT]

 $\label{local-core-messaging-v1.0-os.} AMQP\ \textit{Message Format}\ \texttt{http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-messaging-v1.0-os.}\ \texttt{html\#section-message-format}$ 

#### [UNICODE63]

The Unicode Consortium. The Unicode Standard, Version 6.3.0, (Mountain View, CA: The Unicode Consortium, 2013. ISBN 978-1-936213-08-5)

http://www.unicode.org/versions/Unicode6.3.0/

# 2 Mapping JMS Types to AMQP

TODO (content): define mapping from JMS types to AMQP types.

# 3 Mapping AMQP Types to Java

TODO (content): define mapping from AMQP types to Java types.

# 4 Messages

### 4.1 Message Structure

Both JMS and AMQP define Message structure in terms of "Header", "Properties" and the message "Body". Unfortunately the definitions of these terms are not consistent. For JMS the Headers refer to a defined set of attributes which are a mix of "immutable" and "mutable" (i.e. some which are invariant over the lifetime of the message, and some which are updated as the message travels from sender to eventual receiver). In contrast JMS Properties are (mostly) application defined message attributes set by the sender and invariant over the message lifetime from sender to receiver. A number of JMS-defined 'JMSX' Properties also exist which live in the same namespace as the application properties.

The AMQP Message is defined as a sequence of "Sections" [AMQPMSGFORMAT].

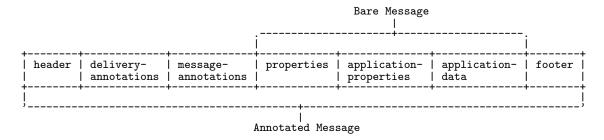


Figure 4.1: AMQP Message Structure

The AMQP header section defines a set of attributes which apply to the message (or rather this particular transfer of the message). These attributes are "mutable" throughout the passage of the message through the AMQP network. The properties section defines "immutable" properties of the message.

## 4.2 Mapping JMS Messages To AMQP

In overview we can say that a JMS Message has the following logical layout:

JMS	JMS	Body	:
	Properties		١
T			т

Figure 4.2: JMS Message

In overview we can say that a JMS Message maps to an AMQP message as follows: The JMS Headers and some JMS-defined 'JMSX' Properties will be stored within the header and properties sections, with occasional aid of additional message-annotations. JMS Properties set by applications will be stored in the application-properties section, including some JMS-defined 'JMSX' Properties. If no such properties are set, the application-properties section MAY be omitted. The Message body will be stored in *application-data* section(s) with type dependent on the particular JMS Message type in use.

TODO (content): do we enable setting (and thus describe here) delivery-annotations or footer details?

### 4.2.1 JMS Headers

The following section describes how each of the defined JMS Headers can be mapped to an AMQP Message.

Header Name	Description
JMSMessageID	The JMSMessageID is defined as a Java String identifier for the Message which is set by the implementation during publication. AMQP uses the message-id field of properties for the same purpose, which is defined as being of type providing message-id, that is message-id-ulong, message-id-uuid, message-id-binary Or message-id-string.  Sending JMS clients SHOULD use the message-id-string type for the message-id field of
	properties by default.  JMSMessageID String values are REQUIRED to have a prefix of "ID.". A single instance of this prefix MUST be removed from the value prior to setting the message-id field of properties on producing JMS clients, and MUST be added by receiving JMS clients (i.e. this prefix MUST be synthesized by the client library).
	See 4.2.1.1 JMSMessageID And JMSCorrelationID Handling for REQUIRED detail relating to supporting usage of the various AMQP types possible for the message-id field of properties.
JMSTimestamp	The JMSTimestamp header is defined as a Java long representing the time at which the message was handed off to the provider to send, in milliseconds since the Unix Epoch. That is, the value is set at the originating client and not changed thereafter. AMQP uses the creation-time field of properties for the same purpose.

JMSCorrelationID	The JMSCorrelationID header is defined as a Java String or byte[] used to link one message with another.
	AMQP uses the correlation-id field of properties for the same purpose, which is defined as being of type providing <i>message-id</i> , that is message-id-ulong, message-id-uuid, message-id-binary or message-id-string.
	To signal to receiving JMS clients in an interoperable way that the correlation—id field of properties represents an application-specific String and not a JMSMessageID String, a boolean message annotation with symbol key of "x-opt-app-correlation-id" is used. When setting JMSCorrelationID to a JMSMessageID (i.e. a String with prefix "ID."), the annotation MUST be omitted or set to false. When setting JMSCorrelationID to an application-specific String value, the annotation MUST be set to true.
	JMSMessageID String values are required to have a prefix of "ID.". When using these as the value for JMSCorrelationID, a single instance of this prefix MUST be removed before using the remainder to set the correlation—id field of properties in the producing JMS client and MUST be added by receiving JMS clients (i.e. this prefix MUST be synthesized by the client library).
	See 4.2.1.1 JMSMessageID And JMSCorrelationID Handling for REQUIRED detail relating to supporting usage of the various AMQP types possible for the correlation—id field of properties.  Application-specific JMSCorrelationID values MUST be sent using the message—id—string type.
JMSReplyTo	The JMSReplyTo header is equivalent to the reply-to field of properties.
	JMSReplyTo is defined as being of the JMS Destination type, while the reply-to field of properties requires an address-string. See 6. Destinations for REQUIRED detail as to how conversion between these types is achieved.

JMSDestination	The JMSDestination header is equivalent to the to field of properties.
	Note that producers MUST set the to field of properties explicitly (intermediaries can't derive it from the target address of the link on which the message was sent).
	JMSDestination is defined as being of the JMS Destination type, while the to field of properties requires an address-string. See 6. Destinations for REQUIRED detail as to how conversion between these types is achieved.
JMSDeliveryMode	The JMSDeliveryMode header is defined as a Java int with two possible values: NON_PERSISTENT and PERSISTENT.
	The JMSDeliveryMode header relates to two different aspects of sending a JMS Message as an AMQP message. Firstly, its value is equivalent to the durable field of header. For PERSISTENT messages, the durable field of header MUST be set to true. For NON_PERSISTENT messages, the durable field of header MUST be either set to false or omitted.
	Additionally, the JMSDeliveryMode value relates to the reliability guarantees of the AMQP message transfer, specifically the point at which sent messages are considered settled. For PERSISTENT messages the sender MUST NOT consider the message settled until the point that the sender has received notification of the disposition at the receiver. For NON_PERSISTENT messages on a non-transacted session an implementer MAY choose to send messages considering them settled as soon as they are sent (i.e. with the settled flag set to true on their original transfer).
JMSRedelivered	This header is set by the client provider on receipt of the message, based on handling of the delivery-count field of header.
	See 8. Delivery Count Handling for more details on handling of the <i>delivery-count</i> value.

JMSType	The JMSType field has no equivalent in AMQP. It is a Java String identifier defined with respect to a notional message definition repository in which message type definitions are contained. This definition would perhaps map closest to the descriptor used on a message whose body consisted of a single instance of an AMQP described type, however as such AMQP types carry their own descriptor it does not need to appear in the message headers.
	In order to carry the JMSType value on a message in an interoperable way, a message annotation with symbol key of "x-opt-jms-type" MUST be used, containing a string representing the JMSType value. If JMSType is not set, the annotation MUST be omitted.
JMSExpiration	The JMSExpiration header is defined as a Java long representing the time at which the message expires, in milliseconds since the Unix Epoch. A value for JMSExpiration is set by the provider when sending the message. That is, the value is set at the originating client and not changed thereafter.
	If a non-zero <i>time-to-live</i> value is specified when sending the message, JMSExpiration contains the computed expiry time. If no <i>time-to-live</i> value (or a value of zero) is supplied when sending the message, then JMSExpiration has the value zero.
	AMQP uses the absolute-expiry-time field of properties for the purpose of setting an expiration time. When a non-zero value <i>time-to-live</i> is supplied, the computed expiration time MUST be set in the absolute-expiry-time field of properties. When no <i>time-to-live</i> value (or a value of zero) is supplied and JMSExpiration thus has the value zero, the absolute-expiry-time field of properties MUST be omitted rather than set to zero.
	See 7.1 Sending Messages for additional REQUIRED detail relating to message expiration.
JMSPriority	The JMSPriority is equivalent to the priority field of header. JMSPriority is specified as being a Java int despite the valid values only being 0-9. AMQP allows the priority to be any valid ubyte value.
	When messages are being sent with a priority of DEFAULT_PRIORITY, the priority field of header SHOULD be omitted.

**JMSDeliveryTime** The JMSDeliveryTime header has no equivalent in AMQP. It is defined as a Java long representing the \*New in JMS 2.0 earliest time at which the message is to be made available for delivery to a consumer, in milliseconds since the Unix Epoch. The value is set at the producing client by adding any provided delivery delay value to the time at which the message is sent. In order to carry the JMSDeliveryTime value on a message in an interoperable way, a message annotation with symbol key of "x-opt-delivery-time" and type timestamp MUST be used if a non-zero delivery delay is specified. If no delivery-delay is specified then the annotation SHOULD be omitted, and receiving JMS clients MUST then synthesize the value via use of the JMSTimestamp header instead. TODO (): define filter to enforce this on the broker? TODO (): define connection-capability for feature?

#### 4.2.1.1 JMSMessageID And JMSCorrelationID Handling

As indicated in 4.2.1 JMS Headers and 4.3.2 Properties Section, AMQP allows for messages to use a variety of types for the *message-id* and *correlation-id* fields of the properties section, specifically message-id-ulong, message-id-unid, message-id-binary or message-id-string. JMS utilises the String type for both JMSMessageID and JMSCorrelationID. Due to this difference in possible types, it is necessary to support controlling and preserving the type information for the underlying fields of the AMQP message when setting and accessing the JMSMessageID and JMSCorrelationID headers of the JMS Message, both for basic interoperability and because it is common to retrieve a message id and then use it as a correlation id value.

To that end, implementing JMS clients MUST support the behaviour described herein for encoding/decoding all JMSMessageID values, and any JMSCorrelationID values which represent a JMSMessageID rather than an application-specific value, in order to influence values sent and received using the underlying *message-id* and *correlation-id* fields of the AMQP properties section.

After the client removes, or before it adds, the synthesized "*ID*." prefix of such values, the following mapping is defined between the associated AMQP types and a String representation thereof:

AMQP type	String Representation
message-id-binary	"AMQP_BINARY: <hex bytes="" of="" representation="">"</hex>
message-id-uuid	"AMQP_UUID: <string of="" representation="" uuid="">"</string>
message-id-ulong	"AMQP_ULONG: <string of="" representation="" ulong="">"</string>
message-id-string	" <original string="">"</original>
message-id-string	"AMQP_STRING: <original string="">"</original>

The AMQP\_STRING encoding prefix exists only to escape message-id-string values that represent a JMSMessageID (for either JMSMessageID or JMSCorrelationID) and would happen to continue with one of the encoding prefixes above (including AMQP\_STRING itself). It MUST NOT be used otherwise by the client library.

For the AMQP\_BINARY encoding prefix, the client MUST return upper-case hex characters when the *getJMSMessageId* and *getJMSCorrelationID* methods of Message are used, but MUST accept both upper-case and lower-case values via the *setJMSMessageID*(String id) and *setJMSCorrelationID*(String id) methods.

When JMSCorrelationID is set using the *setJMSCorrelationID(String id)* method, any value that begins with the "ID:" prefix of a JMSMessageID and attempts to identify itself as an encoded message-id-binary, message-id-uuid, or message-id-ulong but which can't be converted into the indicated format MUST cause an appropriate JMSException to be thrown. For example, "ID:AMQP\_ULONG:foo" can't be converted to a message-id-ulong and so MUST cause an exception.

If implemented, the *getJMSCorrelationIDAsBytes()* method of the Message MUST throw an exception if the type of the correlation-id field of properties is not message-id-binary.

The following table provides examples of various attempts to set JMSCorrelationID, and the associated output including whether the "x-opt-app-correlation-id" is set true to indicate an application-specific value. The mapping for the *getJMSCorrelationID* method is essentially the reverse of this process.

Method call	AMQP type	AMQP value	Annotation
setJMSCorrelationID("ID:foo")	message-id-string	"foo"	X
setJMSCorrelationID("ID:42")	message-id-string	"42"	X
setJMSCorrelationID( "ID:AMQP_STRING:AMQP_ULONG:42")	message-id-string	"AMQP_ULONG:42"	X
setJMSCorrelationID( "ID:AMQP_STRING:AMQP_STRING:foo")	message-id-string	"AMQP_STRING:foo"	X
setJMSCorrelationID("app-spec")	message-id-string	"app-spec"	true
setJMSCorrelationID( "AMQP_ULONG:42")	message-id-string	"AMQP_ULONG:42"	true
setJMSCorrelationID( "AMQP_ULONG:foo")	message-id-string	"AMQP_ULONG:foo"	true
setJMSCorrelationID( "ID:AMQP_ULONG:42")	message-id-ulong	42	X
setJMSCorrelationID( "ID:AMQP_UUID:" + <uuid>)</uuid>	message-id-uuid	<uuid></uuid>	X
setJMSCorrelationID( "ID:AMQP_BINARY:0123ABCD")	message-id-binary	0x0123ABCD	Х
setJMSCorrelationIDAsBytes( 0x0123ABCD)	message-id-binary	0x0123ABCD	X

The behaviour of the *setJMSMessageID* and *getJMSMessageID* methods is similar, with the key distinction that no annotation is necessary since all JMSMessageID values are prefixed "ID:" and there are no corresponding 'AsBytes' alternative methods.

### 4.2.2 JMS-defined 'JMSX' Properties

The following section describes how each of the JMS-defined 'JMSX' Properties can be mapped to an AMQP Message.

Property Name	Description
---------------	-------------

JMSXUserID	The JMSXUserID property is equivalent to the user-id field of properties. The JMSXUserID is specified as String, while the user-id field of properties is specified as type binary.
	To maintain end-to-end fidelity for this property, implementations SHOULD convert between AMQP binary and Java String by using the UTF-8 Unicode [UNICODE63] character encoding.
JMSXAppID	The JMSXAppID property is defined as a Java String representing the identity of the application sending the message. If this property is supported by the client library, it MUST be stored in the application-properties section of the AMQP message.
JMSXDeliveryCount  *Mandatory since JMS 2.0	This property is set by the client provider on receipt of the message, based on handling of the delivery-count field of header.
	See 8. Delivery Count Handling for more details on handling of the <i>delivery-count</i> value.
JMSXGroupID	The JMSXGroupID property is equivalent to the group-id field of properties.
JMSXGroupSeq	The JMSXGroupSeq property is used for the equivalent purpose of the group-sequence field of properties.
	As JMSXGroupSeq is an int and the group-sequence field of properties is an uint, JMSXGroupSeq values in the range $-2^{31}$ to -1 inclusive MUST be mapped to values in the range $2^{31}$ to $2^{32}$ -1 inclusive for the group-sequence field of properties.
JMSXProducerTXID	No standard mapping is provided for JMSXProducerTXID nor is a relation of its semantics to AMQP provided.
JMSXConsumerTXID	No standard mapping is provided for JMSXConsumerTXID nor is a relation of its semantics to AMQP provided. Should the semantics of this property be defined with respect to AMQP it would not affect the on-the-wire encoding as this property is defined to be set by the JMS on receipt of the message at the client.
JMSXRcvTimestamp	This value is (if supported) set by the client provider on receipt of the message, it is not transported on the wire and therefore does not need to be mapped to AMQP.
JMSXState	There is no direct mapping of the JMSXState property to AMQP. It is advised that implementers do not attempt to provide any sort of implementation of this property.

### 4.2.3 JMS Properties

JMS Properties set by applications will typically be stored in the application-properties section, including some JMS-defined 'JMSX' Properties. If no such properties are set, the application-properties section MAY be omitted.

The JMS Specification defines a number of restrictions on the allowable keys and values for JMS Properties. A JMS property key is of type String and, in addition to other naming restrictions, are forbidden to be null or the empty String. Keys in the application-properties section MUST be of type string, thus precluding null values, but impose no other restriction. The value of a JMS property can only be of the types given in 2. Mapping JMS Types to AMQP

(excluding char, which is not allowed in this context). The application-properties section can hold only values of simple types, that is, excluding map, list, and array types.

TODO (intent): Describe mechanism for setting the non-JMS property types (ubyte, uint, etc) to permit full use of application-properties section for interoperability with other AMQP containers.

#### 4.2.4 Message Body Types

JMS defines a number of standard Message body types. These different forms of body each need to be encoded into AMQP message in a defined manner such that JMS Messages which are communicated from one provider to another may be reassembled into the correct message type with full fidelity. Moreover this definition then allows for non-JMS producers to create messages of a form where their handling by a JMS client can be predicted.

The different Message body formats are expressed through the use of different types of *application-data* sections within the AMQP message, different values within those sections, use of fields in the message properties section to indicate the nature of the content, and finally through use of entries in the message-annotations section.

#### 4.2.4.1 Message Type Annotation

In order that it be possible to identify messages as being compatible with the structure defined to be produced by JMS clients implementing this mapping, a message annotation with symbol key of "x-opt-jms-msg-type" is defined to carry the necessary message type information in an interoperable way. The following values have been assigned for each of the existing JMS Message types:

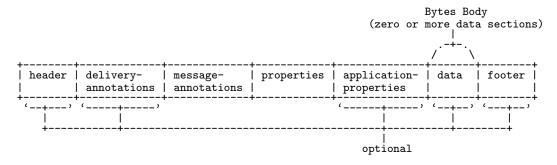
Message Type	Annotation value (type)
Message	0 (byte)
ObjectMessage	1 (byte)
MapMessage	2 (byte)
BytesMessage	3 (byte)
StreamMessage	4 (byte)
TextMessage	5 (byte)

These annotation values may be used by JMS clients implementing this mapping to allow distinguishing their messages from arbitrary AMQP messages of similar structure sent by other AMQP containers, but can similarly be used by those other AMQP containers to produce equivalent messages for consumption by JMS clients.

#### 4.2.4.2 BytesMessage

A BytesMessage is encoded using zero or more body sections of type data. The client SHOULD set the content-type field of properties to contain the symbol value "application/octet-stream". The data section MAY be omitted when the content is zero-length. The message annotation with symbol key of "x-opt-jms-msg-type" MUST be set to a byte value of 3.

The *getBodyLength()* method on BytesMessage MUST return the combined length of the data sections, or 0 if none are present.



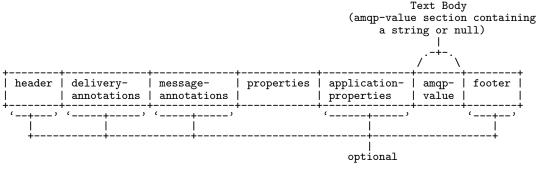
- message-annotation with symbol key "x-opt-jms-msg-type" MUST be set to byte value of 3.
- content-type field of properties section SHOULD contain symbol "application/octet-stream".

Figure 4.3: AMQP Message Structure of a BytesMessage

TODO (intent): confirm we can omit the data section entirely?

#### 4.2.4.3 TextMessage

A TextMessage is encoded as an amqp-value section containing a single encoded string or null. The client SHOULD NOT set the content-type field of properties. The message annotation with symbol key of "x-opt-jmsmsg-type" MAY be set to a byte value of 5.



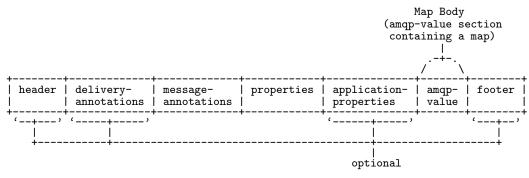
- content-type field of properties section SHOULD NOT be set.
   message-annotation with symbol key "x-opt-jms-msg-type" MAY be set to byte value of 5.

Figure 4.4: AMQP Message Structure of a TextMessage

#### 4.2.4.4 MapMessage

A MapMessage body is encoded as a single amap-value section containing a single map value. The client SHOULD NOT set the content-type field of properties. Any byte[] entries in the MapMessage body MUST be encoded as binary entries in the AMQP map. The message annotation with symbol key of "x-opt-jms-msg-type" MUST be set to a byte value of 2.

Note that this restricts the MapMessage to having at most  $2^{31}$  - 1 entries, and at most  $2^{32}$  - 1 octects of encoded map content. Attempting to send a MapMessage which exceeds these limits MUST result in an appropriate JMSException being thrown.



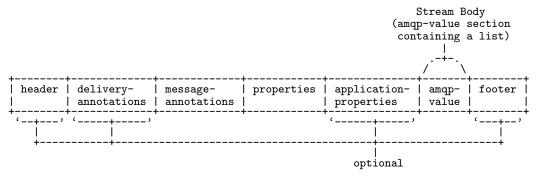
- message-annotation with symbol key "x-opt-jms-msg-type" MUST be set to byte value of 2. content-type field of properties section SHOULD NOT be set.

Figure 4.5: AMQP Message Structure of a MapMessage

#### 4.2.4.5 StreamMessage

A StreamMessage body is encoded as a single amgp-value section containing a single list. The client SHOULD NOT set the content-type field of properties. The message annotation with symbol key of "x-opt-ims-msg-type" MUST be set to a byte value of 4.

Note that this restricts the StreamMessage to having at most  $2^{32}$  - 1 elements, and at most  $2^{32}$  - 1 octects of encoded list content. Attempting to send a StreamMessage which exceeds these limits MUST result in an appropriate JMSException being thrown.



message-annotation with symbol key "x-opt-jms-msg-type" MUST be set to byte value of 4. content-type field of properties section SHOULD NOT be set.

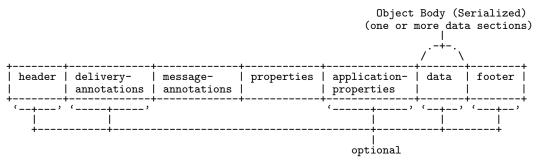
Figure 4.6: AMQP Message Structure of a StreamMessage

#### 4.2.4.6 ObjectMessage

This mapping defines two ways in which an ObjectMessage may be encoded, by default as a series of data sections containing a serialised Java object and alternatively by representing the body components using the AMQP type system directly. This enables composition of AMQP messages with arbitrary body content for increased interoperability with other AMQP containers. JMS clients supporting this mapping MUST support both encoding processes.

To encode an ObjectMessage as serialised Java object data, zero or more data body sections are used, where the content is either (i) empty, or (ii) contains part or all of the serialised object data. If multiple data sections are used, e.g. because the serialised object data exceeds the limits of a single binary value, each subsequent data section MUST contain a continuation of the serialised object content in the previous section. An empty data

section MAY be sent when the ObjectMessage payload is null. In all cases, the content-type field of properties MUST contain the symbol value "application/x-java-serialized-object".



- message-annotation with symbol key "x-opt-jms-msg-type" MUST be set to byte value of 1.
- content-type field of properties section MUST be "application/x-java-serialized-object".

Figure 4.7: AMQP Message Structure of a Java serialized ObjectMessage

TODO (intent): confirm we can omit the data section entirely?

TODO (intent): Describe a way of selecting the method of encoding used for a particular ObjectMessage

TODO (intent): Describe how to encode bodies using AMQP type system, how to handle being able to add components that dont align to the JMS types (e.g ubyte, uint etc), etc.

If using a non-data section, the content-type field of properties SHOULD NOT be set. The client MUST NOT set the content-type field of properties to contain the symbol value "application/x-java-serialized-object".

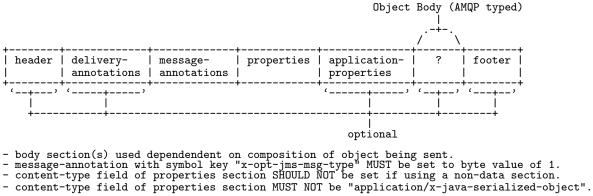
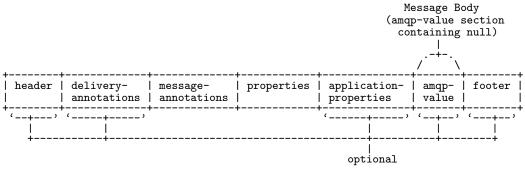


Figure 4.8: AMQP Message Structure of an ObjectMessage using the AMQP type system

TODO (intent): confirm we can omit the body section entirely?

#### 4.2.4.7 Message

A Message is encoded as a single amop-value section containing null, although this MAY be ommitted. The client SHOULD NOT set the content-type field of properties. The message annotation with symbol key of "x-opt-jms-msg-type" MUST be set to a byte value of 0.



- message-annotation with symbol key "x-opt-jms-msg-type" MUST be set to byte value of 0. content-type field of properties section SHOULD NOT be set.
  - Figure 4.9: AMQP Message Structure of a Message

TODO (intent): confirm we can omit the body section entirely?

# 4.3 Mapping AMQP Messages To Java

The previous section defined how a Message as defined by the JMS specification is mapped into AMQP in order to achieve interoperability. In this section the mapping of both these and other arbitrary messages from an AMQP to JMS will defined.

#### 4.3.1 Header Section

Field Name	Description
durable	When receiving a message, the durable field of header MUST be taken to be equivalent to the JMSDeliveryMode header of the Message. If the durable field of header is set to false or isn't set then the JMSDeliveryMode MUST be taken to be NON_PERSISTENT. When the durable field of header is set to true the JMSDeliveryMode of the Message MUST be taken to be PERSISTENT.
priority	This field is equivalent to the JMSPriority header of the Message. JMSPriority is specified as being of type int despite the valid values only being 0-9. AMQP allows for the priority field of header to be any valid ubyte value. When receiving a message with the priority field of header greater than 9, the JMSPriority MUST be set to 9. If the priority field of header is unset then the JMSPriority MUST be taken to be DEFAULT_PRIORITY (i.e. the value 4).

ttl	This field defines the number of milliseconds for which a given message is considered "live". There is no direct equivalent for the ttl field of header in the JMS specification.  If and only if the absolute-expiry-time field of properties is not set, JMSExpiration SHOULD be based on the ttl field of header if set, by summing it with the current time in milliseconds since the Unix
first-acquirer	Epoch.  This field does not have a direct equivalent within the JMS specification, although JMSRedelivered is related, and so vendor property JMS_AMQP_FIRST_ACQUIRER SHOULD be used. For further details, see 5. JMS Vendor Properties .
delivery-count	AMQP uses the delivery-count field of header to track previously failed delivery attempts for a message, with the first delivery attempt having a value of zero, and so on.
	JMSXDeliveryCount is defined as a Java int count of delivery attempts, set by the provider on receive, where the first delivery attempt has value 1, the second has value 2 and so on.
	The value of JMSXDeliveryCount property is thus equal to delivery-count + 1.
	The JMSRedelivered header MUST be considered to be true if and only if the delivery-count field of header has a value greater than 0.
	See 8. Delivery Count Handling for more details on handling of this field.

# 4.3.2 Properties Section

Field Name	Description
------------	-------------

message-id	This field is equivalent to the JMSMessageID header of the Message.
	The JMSMessageID value is a Java String whereas the message-id field of properties is defined as being of type providing message-id, that is message-id-ulong, message-id-uuid, message-id-binary Or message-id-string.
	The JMS client library MUST prefix "ID." to the value of the message-id field of properties before returning it as the JMSMessageID value.
	See 4.2.1.1 JMSMessageID And JMSCorrelationID Handling for REQUIRED detail relating to supporting usage of the various AMQP types possible for the message-id field of properties.
user-id	This field is equivalent to the JMSXUserID header.
	JMSXUserID is specified as being of type String, while the user-id field of properties field is specified as type binary. To maintain end-to-end fidelity for this property implementations SHOULD convert between AMQP binary and Java String by using the UTF-8 Unicode [UNICODE63] character encoding.
to	This field is equivalent to the JMSDestination header.
	JMSDestination is defined as being of the JMS Destination type, while the to field of properties requires an address-string. See 6. Destinations for REQUIRED detail regarding how conversion between these types is achieved if the to field of properties was set.
	If the to field of properties was not set on a received message, the JMSDestination header value SHOULD be derived from the Destination to which the receiving consumer was established.
subject	This field does not have an equivalent within the JMS specification, and so the vendor property JMS_AMQP_SUBJECT SHOULD be used. For further details, see 5. JMS Vendor Properties.
reply-to	This field is equivalent to the JMSReplyTo header.
	JMSReplyTo is defined as being of the JMS Destination type, while the reply-to field of properties requires an address-string. See 6. Destinations for REQUIRED detail regarding how conversion between these types is achieved if the reply-to field of properties was set.

correlation-id	This field is equivalent to the JMSCorrelationID header of the Message.
	The JMSCorrelationID value is a Java String whereas the correlation—id field of properties is defined as being of type providing <i>message-id</i> , that is message—id—ulong, message—id—uuid, message—id—binary Or message—id—string.
	Where the correlation—id field of properties for the received message is of type message—id—string and the boolean message annotation with symbol key of "x-opt-app-correlation-id" is either not set or is false, then the correlation—id field of properties MUST be be formatted as a JMSMessageID, that is the client library MUST prefix "ID:" to the value before returning it as the JMSCorrelationID value.
	See 4.2.1.1 JMSMessageID And JMSCorrelationID Handling for REQUIRED detail relating to supporting usage of the various AMQP types possible for the correlation-id field of properties.
content-type	This field does not have an equivalent within the JMS specification, and so the vendor property JMS_AMQP_CONTENT_TYPE SHOULD be used. For further details, see 5. JMS Vendor Properties.
content-encoding	This field does not have an equivalent within the JMS specification, and so the vendor property JMS_AMQP_CONTENT_ENCODING SHOULD be used. For further details, see 5. JMS Vendor Properties.
absolute-expiry-time	This field is equivalent to the JMSExpiration message header.
	If the absolute-expiry-time field of properties is set, then JMSExpiration MUST have the equivalent Java long value, representing the time at which the message expires, in milliseconds since the Unix Epoch.
	If the absolute-expiry-time field of properties is not set then JMSExpiration SHOULD be based on the ttl field of header if set, see 4.3.1 Header Section for more details.
creation-time	This field is equivalent to the JMSTimestamp message header.
	If the creation-time field of properties is not set, then JMSTimestamp MUST have the value zero. If the creation-time field of properties field is set, then JMSTimestamp MUST have the equivalent Java long value, representing the time at which the message was sent/created, in milliseconds since the Unix Epoch.

group-id	This field is equivalent to the JMS-defined JMSXGroupID message property.
group-sequence	This field is used for the equivalent purpose of the JMSXGroupSeq message property.
	As the group-sequence field of properties is an uint and JMSXGroupSeq is an int, group-sequence values in the range $2^{31}$ to $2^{32}$ -1 inclusive MUST be mapped to JMSXGroupSeq values in the range $-2^{31}$ to -1 inclusive.
reply-to-group-id	This field does not have an equivalent within the JMS specification, and so the vendor property JMS_AMQP_REPLY_TO_GROUP_ID MUST be used. For For further details, see 5. JMS Vendor Properties.

### 4.3.3 Application Properties Section

The application-properties section contents are roughly equivalent to the JMS Message *Properties*, however they differ in the supported types of their contents.

TODO (intent): how to handle receiving the following:

- String property names which do not conform with the JMS restrictions on naming
- property values with types not defined in the JMS specification

### 4.3.4 Delivery Annotations Section

TODO (content):

### 4.3.5 Message Annotations Section

TODO (content):

#### 4.3.6 Footer Section

TODO (content):

### 4.3.7 Body Sections

The following sections detail how to determine which type of JMS Message should be used to represent a receieved AMQP message, based on first identifying whether it is appropriately annotated as corresponding to those produced by JMS clients implementing this mapping, or subsequently by analysing the message structure.

#### 4.3.7.1 Messages With Type Annotation

If the the "x-opt-jms-msg-type" message annotation is present on the received message, its value MUST be used to determine the type of JMS message used to represent the AMQP message, according to the mapping detailed in 4.2.4.1 Message Type Annotation. If the annotation is not present, the sections which follow should be used to identify the appropriate JMS Message type.

#### 4.3.7.2 No Body

Where the message type annotation is not set and no body sections are received, the following should be used to identify the JMS Message type:

If the content-type field of properties is either not set, set to an unknown value, or set to the symbol value "application/octet-stream" the message MUST be interpreted as a BytesMessage with zero-length content.

If the content-type field of properties is set to the symbol value "application/x-java-serialized-object" the message MUST be interpreted as an ObjectMessage with null content.

If the content-type field of properties is set to the symbol value "text/plain" the message MUST be interpreted as a TextMessage with null content.

TODO (intent): Should we cover other specific content types, e.g XML/JSON/Other?

TODO (intent): charset in content-type?

TODO (intent): confirm we can omit the body section entirely?

#### 4.3.7.3 Data

Where the message type annotation is not set and one of more data body sections are received, the following should be used to identify the JMS Message type:

If the content-type field of properties is either not set, set to an unknown value, or set to the symbol value "application/octet-stream" the message MUST be interpreted as a BytesMessage with zero-length content.

If the content-type field of properties is set to the symbol value "application/x-java-serialized-object" the message MUST be interpreted as an ObjectMessage.

If the content-type field of properties is set to the symbol value "text/plain", the message MUST be interpreted as a TextMessage. Where there is only a single data section and it is empty, then the return value from the getText() method MUST be a Java String of length 0.

TODO (intent): Should we cover other specific content types, e.g XML/JSON/Other?

TODO (intent): charset in content-type?

#### 4.3.7.4 Amqp-value

Where the message type annotation is not set and an amqp-value body section is received, the following should be used to identify the JMS Message type:

If the received body section contains a string or null value, the message MUST be interpreted as a TextMessage.

If the received body section contains a binary value, the message MUST be interpreted as a BytesMessage.

For all other amqp-value body section contents, the message MUST be interpreted as an ObjectMessage.

#### 4.3.7.5 Amqp-sequence

Where the message type annotation is not set and one or more amqp-sequence body sections are received, the message MUST be interpreted as an <code>ObjectMessage</code>.

#### 4.3.7.6 Todo

TODO (intent): Discuss how arbitrary AMQP messages will be handled when being represented as an ObjectMessage

# 5 JMS Vendor Properties

This document defines the following JMS Vendor Properties.

Property Name	Set By	Description
JMS_AMQP_TTL	Application	Optionally used for controlling the value of the ttl field of header for the outgoing AMQP message independently from the value normally used due to the JMS Time To Live value applied when sending the message. If set, it MUST be a long property with a value in the range zero to 2 <sup>32</sup> -1. If the property value is zero then the ttl field of header MUST be omitted rather than set to zero.  When setting the ttl field of header by using the JMS_AMQP_TTL property, an entry with this key MUST NOT be included in the application-properties section of
		the transmitted AMQP message.
JMS_AMQP_FIRST_ACQUIRER	Provider on Receive	Optionally used for accessing the first-acquirer field of header. If set, it MUST be of type boolean.
JMS_AMQP_SUBJECT	Application/ Provider on Recieve	Optionally used for setting and/or accessing the subject field of properties. If set, it MUST be of type String.
JMS_AMQP_CONTENT_TYPE	Application/ Provider on Recieve	Optionally used for setting and/or accessing the content-type field of properties to distinguish the content type within the message body where necessary. If set, it MUST be of type String.
JMS_AMQP_CONTENT_ENCODING	Application/ Provider on Recieve	Optionally used for setting and/or accessing the content-encoding field of properties to distinguish the content encoding within the message body where necessary. If set, it MUST be of type String.
JMS_AMQP_REPLY_TO_GROUP_ID	Application/ Provider on Recieve	Optionally used for setting and/or accessing the reply-to-group-id field of properties. If set, it MUST be of type String.

Each implementation MAY, in addition, define its own extension properties but these MUST NOT use AMQP as the "vendor" name, i.e. the additional extension property names MUST NOT begin with "JMS\_AMQP".

TODO (presentation): Decide where this goes, it isn't necessarily a section.

# 6 Destinations

In order to faithfully re-construct the Destination objects used in the JMSDestination and JMSReplyTo headers of a Message following its transmission via AMQP, information regarding the particular type of Destination object also has to be transmitted in an interoperable fashion.

This type information is transferred via message annotations with symbol keys of "x-opt-to-type" and "x-opt-reply-type". The value of these annotation is of type string containing a comma-separated set of destination type attributes. Possible attributes include "queue", "topic", and "temporary". These are used to define the following possible annotation values:

Destination Type	Annotation value
Queue	"queue"
TemporaryQueue	"queue,temporary"
Topic	"topic"
TemporaryTopic	"topic,temporary"

Producing JMS clients SHOULD set the "x-opt-to-type" message annotation on each message sent. Producing JMS clients SHOULD set the "x-opt-reply-type" message annotation on each message sent that has a JMSReplyTo header value.

When recieving an AMQP message the lacks the message annotations outlined above, additional steps are necessary order to ensure that the JMSDestination and/or JMSReplyTo headers can be populated appropriately. If the "x-opt-to-type" and/or "x-opt-reply-type" message annotations are not present, the JMSDestination and/or JMSReplyTo values respectively SHOULD be constructed using the same Destination type derivative as that used to create the consumer which received the message.

When recieving an AMQP message the lacks the to field of properties, receiving JMS clients SHOULD synthesize this by returning the Destination value supplied when creating the consumer which received the message.

TODO (intent): Change the annotations to use a ubyte for efficiency?

TODO (intent): Define that x-opt-reply-type = x-opt-to-type unless both are set, so allowing that only one be set?

TODO (presentation): Decide where this goes, it isn't necessarily a section.

# 7 Message Producers

## 7.1 Sending Messages

JMS producers (e.g MessageProducer) are required to set various headers on a message during the sending operation.

For the JMSExpiration header, specific handling was discussed in 4.2.1 JMS Headers. However, beyond setting the JMSExpiration header with the computed expiration, producing JMS clients need additionally ensure an appropriate value for the ttl field of header on outgoing messages.

If the JMS\_AMQP\_TTL vendor property outlined in 5. JMS Vendor Properties has been set on the Message, its value SHOULD be used to populate the ttl field of header.

If the JMS\_AMQP\_TTL vendor property has not been set and a Time To Live value of 0 is applicable when sending a Message, then producing JMS clients MUST NOT set the ttl field of header, that is it MUST be omitted rather than set to zero.

If the  $JMS\_AMQP\_TTL$  vendor property has not been set, and a non-zero  $Time\ To\ Live\ value\ of\ 2^{32}$ -1 or less is applicable when sending a Message, the ttl field of header MUST be set accordingly by the provider on the AMQP message. If the applicable  $Time\ To\ Live\ value\ exceeds\ 2^{32}$  - 1 then the ttl field of header MUST be ommitted instead rather than populated with a value less than specified by the application.

# 8 Delivery Count Handling

TODO (intent): define handling for delivery-count and its relationship to JMSXDeliveryCount and JMSRedelivered. That is, when to update it based on rollback, recover etc (and how this further depends on the way those methods are actually implemented, i.e locally or by pushing them back to the source). Decide where this goes, it isn't necessarily a section.

# 9 Supplementary Definitions

Annotation Name	Reference
x-opt-app-correlation-id	For further details, see 4.2.1 JMS Headers and 4.3.2 Properties Section
x-opt-jms-type	For further details, see 4.2.1 JMS Headers
x-opt-delivery-time	For further details, see 4.2.1 JMS Headers
x-opt-to-type	For further details, see 6. Destinations
x-opt-reply-type	For further details, see 6. Destinations
x-opt-jms-msg-type	For further details, see 4.2.4.1 Message Type Annotation, 4.2.4 Message Body Types and 4.3.7 Body Sections.

TODO (content): add annotations to registry, back-reference these definitions.

TODO (presentation): Decide where this goes, it isn't necessarily a section.