Advanced Message Queuing Protocol (AMQP) JMS Mapping Version 1.0

Working Draft 1

2 December 2013

Specification URIs

This version:

http://docs.oasis-open.org/amqp-bindmap/jms/v1.0/wd01/amqp-bindmap-jms-v1.0-wd01.xml (Authoritative)
http://docs.oasis-open.org/amqp-bindmap/jms/v1.0/wd01/amqp-bindmap-jms-v1.0-wd01.html
http://docs.oasis-open.org/amqp-bindmap/jms/v1.0/wd01/amqp-bindmap-jms-v1.0-wd01.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/amqp-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. 2 December 2013. OASIS Working Draft 1.
http://docs.oasis-open.org/amqp-bindmap/jms/v1.0/wd01/amqp-bindmap-jms-v1.0-wd01.pdf.

Notices:

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

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

[AMQPMSGFORMAT]
AMQP Message Format http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-messaging-v1.0-os.html#section-message-format

[ISO88591]
ISO-8859-1 http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=28245

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].

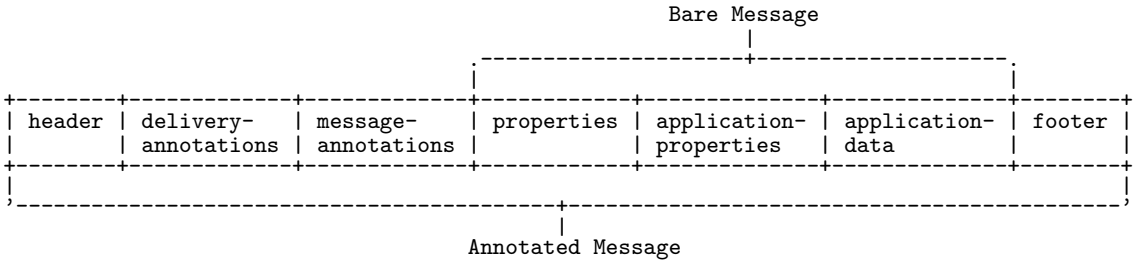

Figure 4.1: AMQP Message Structure

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:

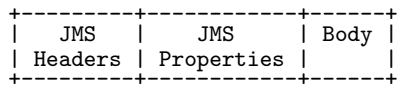

Figure 4.2: JMS Message

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*, such as `message-id-ulong`, `message-id-uuid`, `message-id-binary` or `message-id-string`. *JMSMessageID* values are required to have a prefix of "*ID:*", however this prefix MUST NOT be part of the value of the `message-id` field of `properties` sent by producing JMS clients, and MUST NOT be required receiving JMS clients (i.e. this prefix should be synthesized by the client library). TODO (intent): recommend one AMQP type (e.g string)?
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 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 *message-id*, such as `message-id-ulong`, `message-id-uuid`, `message-id-binary` or `message-id-string`. *JMSMessageID String* values are required to have a prefix of "*ID:*", however when using these as the value for *JMSCorrelationID*, the "*ID:*" prefix MUST NOT be included in the `correlation-id` field of `properties` sent by the producing JMS client and MUST NOT be required by receiving JMS clients (i.e. this prefix must be synthesized by the client library). When setting a *String* value for *JMSCorrelationID* with prefix of "*ID:*", this SHOULD be a valid *JMSMessageID* value from the same provider implementation as message id values from alternative providers may not be accepted. To signal to receiving JMS clients in an interoperable way that the the `correlation-id` field of `properties` represents an application-specific *String* or *byte[]*, 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* or a *byte[]* value, the annotation MUST be set to *true*.
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 should be 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 cannot derive it from address of the target of the link on which the message was sent). *JMSDestination* 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 should be achieved.
JMSDeliveryMode	The *JMSDeliveryMode* header is defined as a Java *int* with two possible values: *1. NON-PERSISTENT 2. 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` should be set to *true*. For *NON-PERSISTENT* messages, the `durable` field of `header` may be set to false (the implicit default for AMQP). 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 7. Delivery Count Handling for more details on handling of the *delivery-count* 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*" should be used, containing a `string` representing the *JMSType* value.
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 *time-to-live* value is specified when sending the message, *JMSExpiration* contains the the computed expiry time. If no *time-to-live* 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 *time-to-live* is supplied, the computed expiration time should be set in the `absolute-expiry-time` field of `properties`. When no *time-to-live* value (or a value of zero) is supplied and *JMSExpiration* should have the value zero, the `absolute-expiry-time` field of `properties` MUST be omitted rather than set to zero.
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.

JMSDeliveryTime *New in JMS 2.0	The *JMSDeliveryTime* header has no equivalent in AMQP. It is defined as a Java *long* representing the earliest time at which the message may 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?

4.2.2 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 ISO-8859-1 [ISO88591] character set.
JMSXAppID	The *JMSXAppID* property is defined as a Java *String* representing the identity of the application sending the message. If supported, this property should 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 7. Delivery Count Handling for more details on handling of the *delivery-count* value.
JMSXGroupID	The *JMSXGroupID* property is equivalent to the `group-id` field of `properties`.

JMSXGroupSeq	The *JMSXGroupSeq* property is equivalent to 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 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 must be 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 may only be of the types given in 2. Mapping JMS Types to AMQP (excluding *char*, which is not allowed in this context). There are no such restrictions on the values within the `application-properties` section.

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 in a defined manner such that Messages which are communicated from one provider to another may be reassembled into the correct message type with full fidelity. Moreover this definition allows for non-JMS producers to create messages of a form where their interpretation by a JMS client can be predicted.

Different Message body formats can be expressed through the use of different types of *application-data* sections within the encoded AMQP message, different values within those sections, and by using fields in the message `properties` section to indicate the nature of the body content.

4.2.4.1 BytesMessage

A *BytesMessage* is encoded using zero or more body sections of type `data`. When `data` sections are included, the `content-type` field of `properties` SHOULD contain the `symbol` value "*application/octet-stream*". The `data` section MAY be omitted when the content is zero-length, in which case the `content-type` field of `properties` MUST contain the `symbol` value "*application/octet-stream*".

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

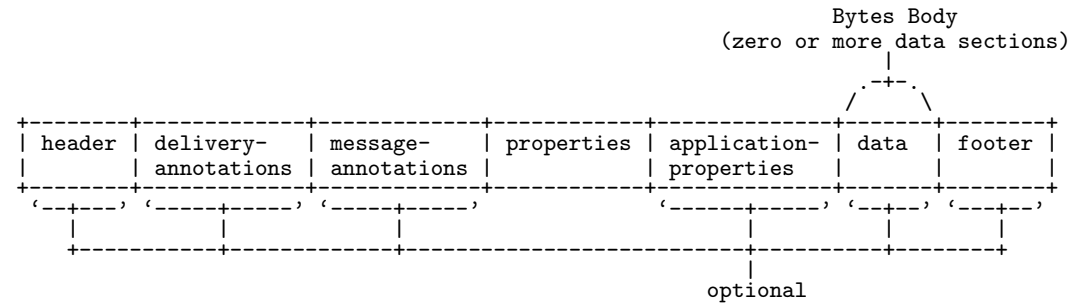

Figure 4.3: AMQP Message Structure of a BytesMessage

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

4.2.4.2 TextMessage

A *TextMessage* is encoded as an `amqp-value` section containing a single encoded `string` or *null*. The `amqp-value` section MAY be omitted when the *TextMessage* body is *null*. If the `amqp-value` section is included, the `content-type` field of `properties` SHOULD NOT be set. If the `amqp-value` section is omitted, the `content-type` field of `properties` MUST contain the `symbol` value "*text/plain*".

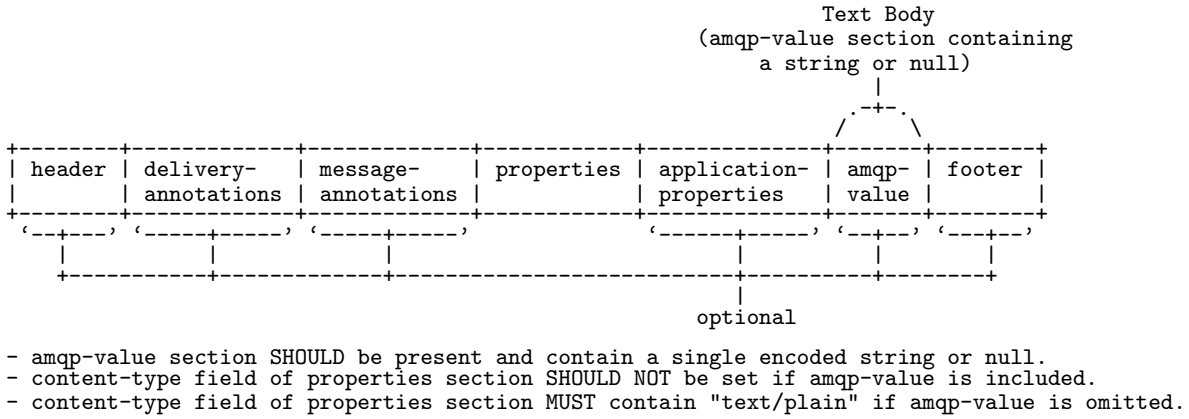

Figure 4.4: AMQP Message Structure of a TextMessage

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

TODO (intent): charset in content-type?

4.2.4.3 MapMessage

A *MapMessage* body is encoded as a single `amqp-value` section containing a single `map` value. As a result, the `content-type` field of `properties` SHOULD NOT be set.

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 should result in an appropriate *JMSException* being thrown.

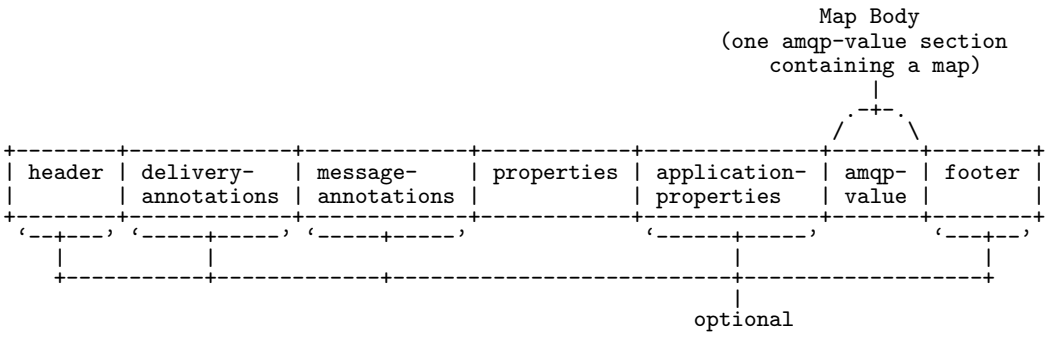

Figure 4.5: AMQP Message Structure of a MapMessage

The JMS Specification defines a number of restrictions on the allowable keys and values for *MapMessage* entries. A key must be of type *String* and the values may be only of the types given in 2. Mapping JMS Types to AMQP . There are no such restrictions on the keys and values in an AMQP `map` value.

TODO (intent): sending non-JMS types?

4.2.4.4 StreamMessage

A *StreamMessage* body is encoded as a single `amqp-value` section containing a single `list`. As a result, the `content-type` field of `properties` SHOULD NOT be set.

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 should result in an appropriate *JMSException* being thrown.

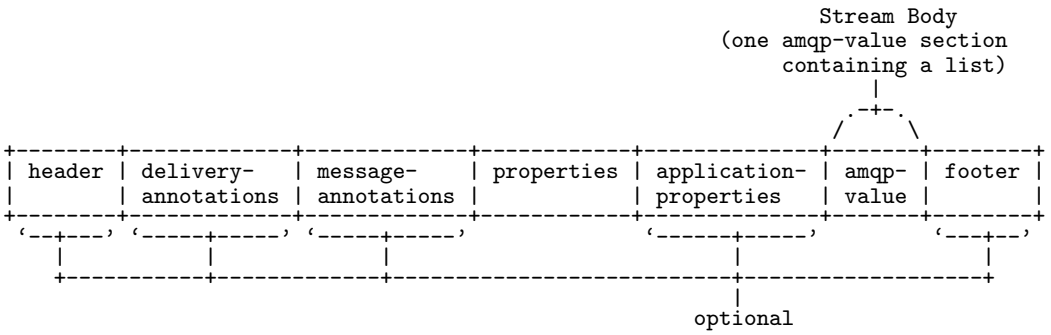

Figure 4.6: AMQP Message Structure of a StreamMessage

The JMS Specification restricts the allowable entry values for a *StreamMessage* entries to be only of the types given in 2. Mapping JMS Types to AMQP . There are no such restrictions on the entries in an AMQP `list` value.

TODO (intent): sending non-JMS types?

4.2.4.5 ObjectMessage

An *ObjectMessage* is encoded using zero or more body sections of type `data`, where the content is either (i) empty, or (ii) a single encoded AMQP `binary` value containing 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 `binary` value holding a continuation of the serialised object content in the previous section. The `data` sections MAY be omitted when the *ObjectMessage* payload is empty or *null*. In all cases, the `content-type` field of `properties` MUST contain the `symbol` value "*application/x-java-serialized-object*".

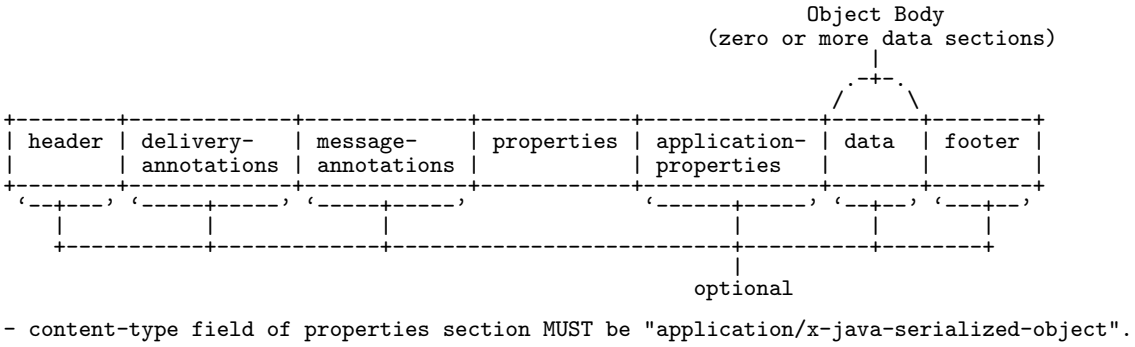

Figure 4.7: AMQP Message Structure of an ObjectMessage

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

TODO (intent): ability to represent objects (e.g maps) using the AMQP type encodings?

4.3 Mapping AMQP Messages To Java

The previous section defined how a Message as defined by the JMS specification should be 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` can be taken to be equivalent to the *JMSDeliveryMode* header of the Message. If the the `durable` field of `header` is set to *false* or unset then the *JMSDeliveryMode* should be taken to be *NON-PERSISTENT*. When the `durable` field of `header` is set to *true* the *JMSDeliveryMode* of the Message should 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* should be set to 9. If If the `priority` field of `header` is unset the the *JMSPriority* should be taken to be *Message.DEFAULT_PRIORITY* (i.e. the value 4).
ttl	This field defines the number of milliseconds for which the message is considered "live". There is no direct equivalent in for the `ttl` field of `header` in the JMS specification, although JMSExpiration is related, and so the vendor property *JMS_AMQP_TTL* should be used. For further details, see 5. JMS Vendor Properties .
first-acquirer	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 should be considered to be true if and only if the `delivery-count` field of `header` has a value greater than 0. See 7. 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*, such as `message-id-ulong`, `message-id-uuid`, `message-id-binary` or `message-id-string`. In order to preserve the type of the `message-id` field of `properties` in situations where the *JMSMessageID* value is used to set a *JMSCorrelationID* value, the returned *JMSMessageID* value SHOULD encode the type information. *JMSMessageID* values MUST have a prefix of "*ID:*", however it is expected that the received `message-id-string` does not include this. The receiving JMS client MUST synthesize this prefix if necessary.
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 ISO-8859-1 [ISO88591] character set.
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 should be 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 should be 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 *message-id*, such as `message-id-ulong`, `message-id-uuid`, `message-id-binary` or `message-id-string`. In order to preserve the type of the `correlation-id` field of `properties` for later use in sending a new AMQP message, the *JMSCorrelationID* value returned SHOULD encode the type information. Where the `boolean` message annotation with `symbol` key of "*x-opt-app-correlation-id*" is either not set on the received message or is false, the the `correlation-id` field of `properties` value MUST be taken to be the id of a message and be formatted as if it were a *JMSMessageID*, that is the client library MUST ensure the returned *JMSCorrelationID* value has prefix "*ID:*" by synthesizing it if necessary.
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 not set, then *JMSExpiration* should have the value zero. If the `absolute-expiry-time` field of `properties` is set, then *JMSExpiration* should have the equivalent Java *long* value, representing the time at which the message expires, in milliseconds since the Unix Epoch.
creation-time	This field is equivalent to the *JMSTimestamp* message header. If the `creation-time` field of `properties` is not set, then *JMSTimestamp* should have the value zero. If the the `creation-time` field of `properties` field is set, then *JMSTimestamp* should 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 *JMSXGroupSeq* message property.
group-sequence	This field is equivalent to the JMS-defined *JMSXGroupID* message property.

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* should be used. For For further details, see 5. JMS Vendor Properties .

4.3.3 Application Properties Section

The `application-properties` section contents are equivalent to the JMS Message *Properties*.

TODO (intent): how to handle receiving (and sending?) 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 type and content of the message body received will influence the particular JMS Message type used to represent the AMQ message.

4.3.7.1 No Body

Where no body sections are received and the `content-type` field of `properties` is either not set, or set to the `symbol` value "*application/octet-stream*" the message should be interpreted as a *BytesMessage* with *zero-length* content.

Where no body sections are received and the `content-type` field of `properties` is set to the `symbol` value "*application/x-java-serialized-object*" the message should be interpreted as an *ObjectMessage* with *null* content.

Where no body sections are received and the `content-type` field of `properties` is set to the `symbol` value "*text/plain*" the message should be interpreted as a *TextMessage* with *null* content.

TODO (intent): charset in content-type?

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

4.3.7.2 Data

Where one or more `data` sections are received and the `content-type` field of `properties` is either not set, or set to the `symbol` value "*application/octet-stream*"the message should be interpreted as a *BytesMessage*.

Where one or more `data` sections are received and the `content-type` field of `properties` is set to the `symbol` value "*application/x-java-serialized-object*" the message should be interpreted as an *ObjectMessage*.

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

TODO (intent): charset in content-type?

4.3.7.3 Amqp-value

Where an `amqp-value` body section is received that contains a `string` value, the message should be interpreted as a *TextMessage*.

Where an `amqp-value` body section is received that contains a `null` value, the message should be interpreted as a *TextMessage* with null content.

Where an `amqp-value` body section is received that contains a `map` value, the message should be interpreted as a *MapMessage*.

Where an `amqp-value` body section is received that contains a `list` value, the message should be interpreted as a *StreamMessage*.

Where an `amqp-value` body section is received that contains a `binary` value, the message should be interpreted as a *BytesMessage*.

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

- Any other typical alternative representations of the JMS message types
- multiple body sections which were not described previously (i.e not data)
- bodies containing non-JMS types

TODO (content): discuss scope for receiving AMQP encoded types as a particular variety of Message or body type (e.g receive a map as a java.util.Map via an ObjectMessage, rather than receiving a MapMessage). The new JMS 2.0 getBody() method both eases and complicates this.

5 JMS Vendor Properties

This document defines the following JMS Vendor Properties.

Property Name	Set By	Description
JMS_AMQP_TTL	Provider on Receive	Optionally used for accessing the `ttl` field of `header`. If set, it MUST be of type *long*.
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	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	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	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	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 must also 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*". For example, a *TemporaryQueue* object value for *JMSDestination* could have its type represented by the value "*queue,temporary*". Leading and trailing whitespace surrounding attribute values MUST be ignored. Empty attribute values MUST be ignored, such as those caused by leading, trailing, or consecutive comma separators.

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.

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

TODO (content): Complete mapping from Destination to address and vice-versa. E.g receiving messages where "to" and/or "reply-to" are set but the annotations are not, i.e. message was not sent from a JMS client.

7 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.

8 Supplementary Definitions

Annotation Name	Reference
x-opt-app-correlation-id	For further details, see 4.2.1 JMS Headers
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

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

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