Advanced Message Queuing Protocol (AMQP) JMS Mapping Version 1.0

Working Draft 10

21 August 2020

Specification URIs

This version:

http://docs.oasis-open.org/amqp-bindmap/jms/v1.0/wd10/amqp-bindmap-jms-v1.0-wd10.xml (Authoritative)
http://docs.oasis-open.org/amqp-bindmap/jms/v1.0/wd10/amqp-bindmap-jms-v1.0-wd10.html
http://docs.oasis-open.org/amqp-bindmap/jms/v1.0/wd10/amqp-bindmap-jms-v1.0-wd10.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:

Robert Gemmell (rgemmell@redhat.com), Red Hat

Editors:

Robert Gemmell (rgemmell@redhat.com), Red Hat
Robert Godfrey (rgodfrey@redhat.com), Red Hat

Abstract:

The Java Message Service (JMS) API is a Java API for utilising messaging systems from Java applications. It provides a common way to create, send, receive, and read messages with a particular messaging system without becoming tied to its specific implementation.Whilst JMS defines a vendor-neutral client API, it does not define a wire level protocol, and so JMS client implementations often only interoperate with specific servers and clients from the same vendor. AMQP defines an open internet protocol for messaging, and so a JMS client implementation using AMQP as its wire level protocol would be able to interoporate with server and client implementations from different vendors similarly using AMQP.This document defines a JMS mapping for AMQP such that client implementations may interoperate with each other, and servers may offer specific functionality necessary to support particular JMS 2.0 features.

Status:

This document was last revised or approved by the OASIS Advanced Message Queuing Protocol (AMQP) Bindings and Mappings (AMQP-BINDMAP) TC 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. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=amqp-bindmap#technical.

TC members should send comments on this specification to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at https://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 TC's web page (https://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. 21 August 2020. OASIS Working Draft 10. http://docs.oasis-open.org/amqp-bindmap/jms/v1.0/wd10/amqp-bindmap-jms-v1.0-wd10.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

[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 Connections

2.1 Client Identifier

JMS defines the concept of a *Client Identifier*, often shortened as *ClientID*, which is used to associate a Connection and its objects with state maintained on their behalf. In JMS 1.1, each connection must have a unique identifier, with only a single Connection at a time making use of a particular value. The identifier may be configured administratively on a ConnectionFactory instance that will then apply it to the Connection objects it creates, or it may alternatively be set explicitly on the Connection immediately after creation by the application. Some client implementations generate an identifier if not set via either of these means.

AMQP connections are established between *containers*, each of which MUST have an ID transmitted in the `container-id` field of `open`, so this will be utilised to convey the JMS identifier. JMS clients MUST set the `container-id` field of `open` to the value of the *Client Identifier*.

AMQP permits multiple connections between given containers, whereas JMS only allows a single connection to use a given identifier at a time. As such, an additional mechanism is required to facilitate use of the *container-id* to carry the identifier and also satisfy the JMS requirements. To this end, a connection capability is used to request the peer provide sole use of the container-id being used by the client. If the peer is able to do so, it will supply the capability in return. If it is unable to satisfy the request because the container-id is already in use on another connection, it will close the connection with “invalid-field” error. Because closing the connection requires first opening it, the open frame SHOULD carry a connection property of “amqp:connection-establishment-failed” to hint that the close frame will follow immediately afterwards.If the peer is unable to provide sole use because it does not support the mechanism, this will be visible by it opening the connection without supplying the capability, and it is then the decision of the client whether to proceed or fail immediately.

TODO (content): Update to reflect JMS 2.0 semantics rather than mentioned 1.1

TODO (content): Move to referencing the “Enforcing Connection Uniqueness” spec.

3 Messages

3.1 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].

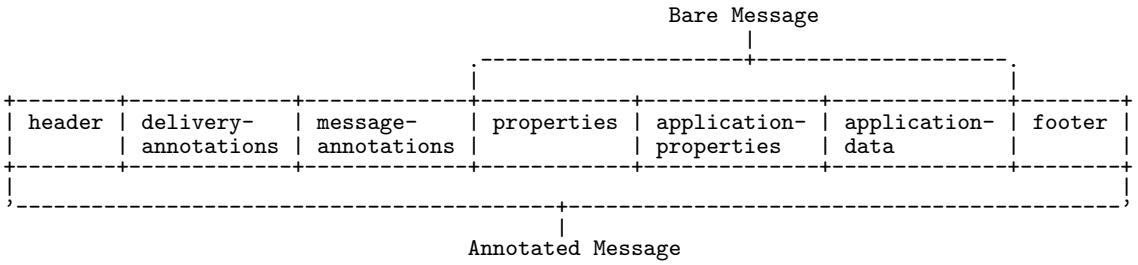

Figure 3.1: AMQP Message Structure

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.

3.2 Mapping JMS Messages To AMQP

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

Figure 3.2: 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.

3.2.1 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:*". This prefix SHOULD be included in the value of the `message-id` field of `properties` set by producing JMS clients when of type `message-id-string` See 3.2.1.1 JMSMessageID And JMSCorrelationID Handling for REQUIRED additional 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 *message-id*, that is `message-id-ulong`, `message-id-uuid`, `message-id-binary` or `message-id-string`. In the case of a `String` this may represent either a `JMSMessageID` value, which begins with "*ID:*", or an application-specific value which does not. See 3.2.1.1 JMSMessageID And JMSCorrelationID Handling for REQUIRED additional 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 5. 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 5. 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 delivery-count-handling for more details on handling of the *delivery-count* value.
JMSType	The `JMSType` header is mapped to the `subject` field of `properties`.

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 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 MUST 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` thus has the value zero, the `absolute-expiry-time` field of `properties` MUST be omitted rather than set to zero. See 6.1 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 direct equivalent field in the AMQP message, however information to convey or synthesize the value can be be carried. See 9. Delivery Delay for additional REQUIRED detail relating to JMSDeliveryTime and requesting/verifying server support for delayed delivery.

3.2.1.1 JMSMessageID And JMSCorrelationID Handling

As indicated in 3.2.1 JMS Headers and 3.3.2 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-uuid`, `message-id-binary` or `message-id-string`. JMS utilises the `String` type for `JMSMessageID` and also primarily for `JMSCorrelationID`, plus optionally `byte[]` for the latter to facilitate support of native correlation id values with certain providers. 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 for an application to retrieve the `JMSMessageID` or `JMSCorrelationID` value from one message and then use it to set the `JMSCorrelationID` value of another message.

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.

The following mapping is defined between a String representing a JMSMessageID value (used as a JMSMessageID or a JMSCorrelationID as appropriate) and the associated AMQP types, facilitating use of the different message-id and correlation-id types and maintaining fidelity of the AMQP type during round-trip through the JMS application layer in String form:

JMSMessageID representation	Description
"ID:AMQP_BINARY:<hex representation of bytes>"	JMSMessageID or JMSCorrelationID representation of a `message-id-binary` value.
"ID:AMQP_UUID:<string representation of uuid>"	JMSMessageID or JMSCorrelationID representation of a `message-id-uuid` value.
"ID:AMQP_ULONG:<string representation of ulong>"	JMSMessageID or JMSCorrelationID representation of a `message-id-ulong` value.
"ID:AMQP_NO_PREFIX:<original-string>"	JMSMessageID representation of a received AMQP message-id of type `message-id-string` which did not start with the "ID:" prefix. May be used to set the JMSCorrelationID.
"ID:AMQP_STRING:<original-string>"	JMSMessageID or JMSCorrelationID representation of a received `message-id-string` value which starts with one of the "ID:AMQP_<type>" escape prefixes used in this table.

The "ID:AMQP_STRING:" prefix exists only to escape `message-id-string` values that represent a *JMSMessageId* (for use as either a *JMSMessageId* or *JMSCorrelationID*) and happen to begin with one of the "ID:AMQP_<type>" prefixes detailed above (including AMQP_STRING itself). It MUST NOT be used otherwise by the client library.

For the "ID:AMQP_BINARY:" 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* value and attempts to identify itself as representing a `message-id-binary`, `message-id-uuid`, or `message-id-ulong` but which can't be converted into the indicated underlying format MUST cause an appropriate exception to be thrown. For example, "ID:AMQP_ULONG:foo" can't be converted to a `message-id-ulong` and so MUST cause an exception. Providing a value beginning "ID:AMQP_ULONG:" in which the remainder contained a leading zero would be similarly be invalid, as would a value with an odd number of characters following the "ID:AMQP_BINARY:" prefix.

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 `JMSMessageID` values derived from the `message-id` field of `properties` of a received message:

AMQP message-id	AMQP type	JMSMessageID
"ID:my-string-id"	message-id-string	"ID:my-string-id"
"non-prefixed-string-id"	message-id-string	"ID:AMQP_NO_PREFIX:non-prefixed-string-id"
<UUID>	message-id-uuid	"ID:AMQP_UUID:<UUID toString>"
42	message-id-ulong	"ID:AMQP_ULONG:42"

0xABCDEF	message-id-binary	“ID:AMQP_BINARY:ABCDEF”
“ID:AMQP_ULONG:string-id”	message-id-string	“ID:AMQP_STRING:ID:AMQP_ULONG:string-id”

A message sent after one of the above `JMSMessageID` values has been used to set its `JMSCorrelationID` value, would give the reverse mapping such that the `correlation-id` field of `properties` contains the same value as found in the `message-id` field of `properties` of the originally received message.

The following table provides examples of the effect of setting various additional application-specific `JMSCorrelationID` values (i.e a string not beginnning with “ID:”) on a message to be sent:

JMSCorrelationID	AMQP type	AMQP correlation-id
“application-specific”	message-id-string	“application-specific”
“AMQP_ULONG:42”	message-id-string	“AMQP_ULONG:42”
“AMQP_ULONG:foo”	message-id-string	“AMQP_ULONG:foo”

The following table provides examples of various `JMSCorrelationID` values derived from the `correlation-id` field of `properties` of a received message:

AMQP correlation-id	AMQP type	JMSCorrelationID
“ID:my-string-id”	message-id-string	“ID:my-string-id”
“non-prefixed-string-id”	message-id-string	“non-prefixed-string-id”
<UUID>	message-id-uuid	“ID:AMQP_UUID:<UUID toString>”
42	message-id-ulong	“ID:AMQP_ULONG:42”
0xABCDEF	message-id-binary	“ID:AMQP_BINARY:ABCDEF”
“ID:AMQP_ULONG:string-id”	message-id-string	“ID:AMQP_STRING:ID:AMQP_ULONG:string-id”
“AMQP_ULONG:foo”	message-id-string	“AMQP_ULONG:foo”

3.2.2 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	This property is set by the client provider on receipt of the message, based on handling of the `delivery-count` field of `header`. See 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 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 provider 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.

3.2.3 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.

3.2.4 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.

3.2.4.1 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.

Note that while byte values MUST be used by sending JMS clients, implementations MUST cope with receiving any integral type for the annotation value.

3.2.4.2 BytesMessage

A `BytesMessage` is encoded using one 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 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.

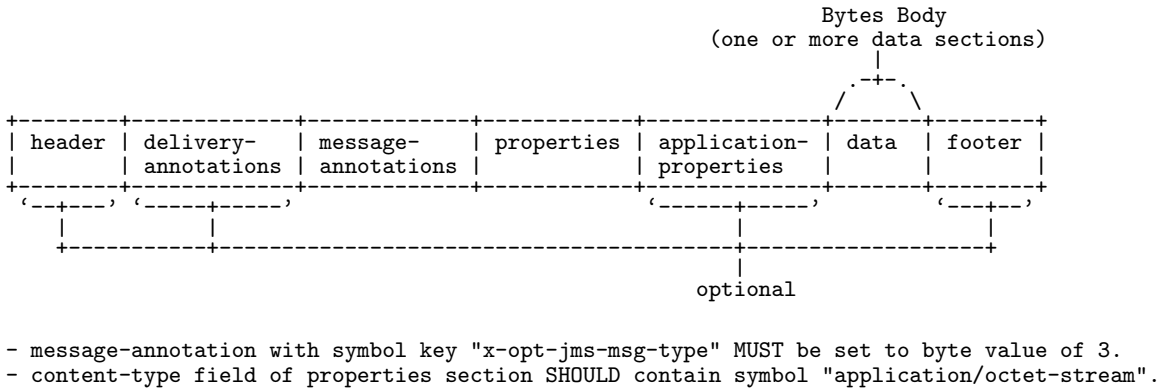

Figure 3.3: AMQP Message Structure of a BytesMessage

3.2.4.3 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-jms-msg-type*" MAY be set to a `byte` value of 5.

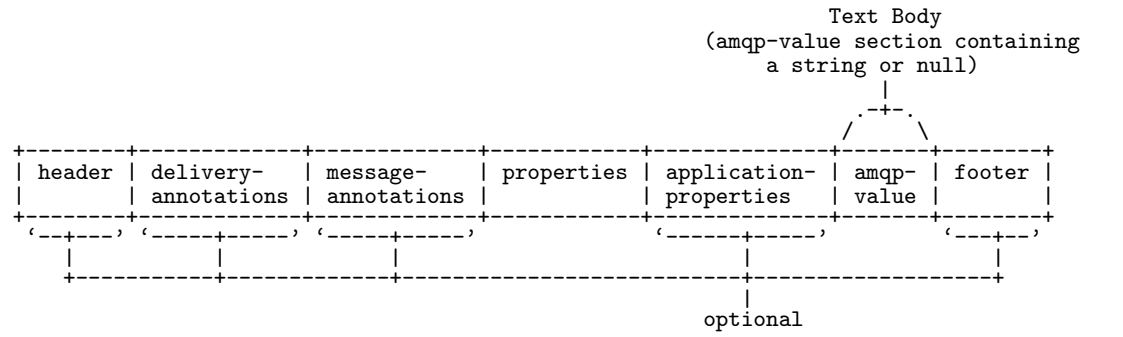

Figure 3.4: AMQP Message Structure of a TextMessage

3.2.4.4 MapMessage

A `MapMessage` body is encoded as a single `amqp-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.

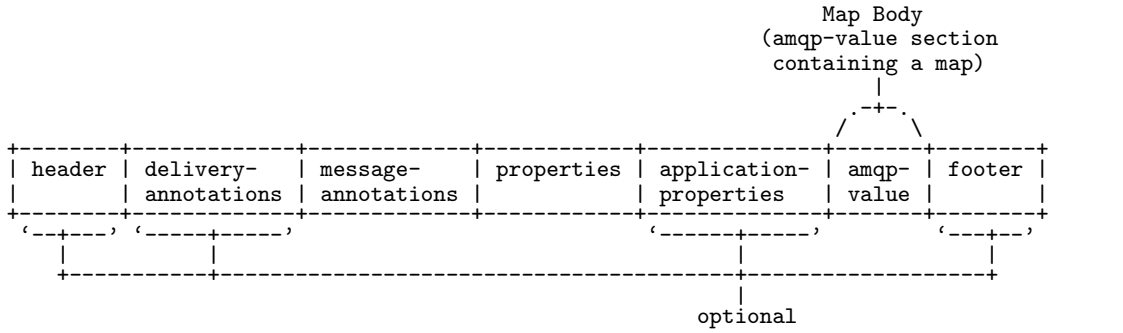

Figure 3.5: AMQP Message Structure of a MapMessage

3.2.4.5 StreamMessage

A `StreamMessage` body is encoded as one or more `amqp-sequence` sections. 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 4.

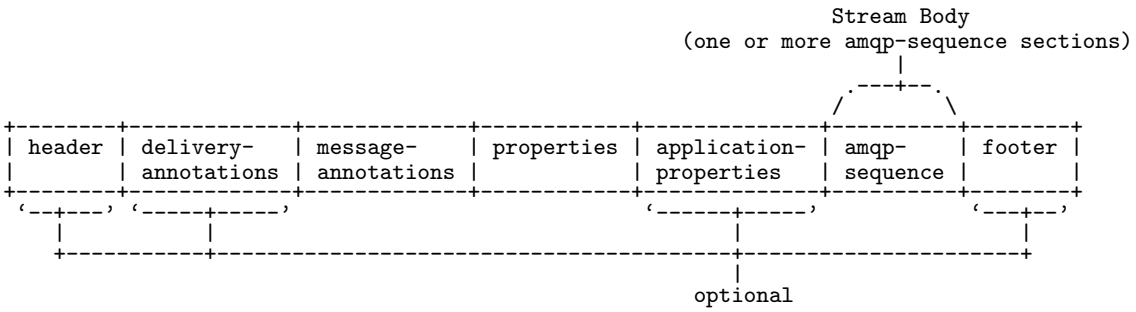

Figure 3.6: AMQP Message Structure of a StreamMessage

3.2.4.6 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, one or more `data` body sections are used, where the content 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 section, each subsequent `data` section MUST contain a continuation of the serialised object content in the previous section. When the object is either not set or explicitly set null, a `data` section containing the serialized *null* MUST be sent. In all cases, the `content-type` field of `properties` MUST contain the `symbol` value "*application/x-java-serialized-object*". The message annotation with `symbol` key of "*x-opt-jms-msg-type*" MUST be set to a `byte` value of 1.

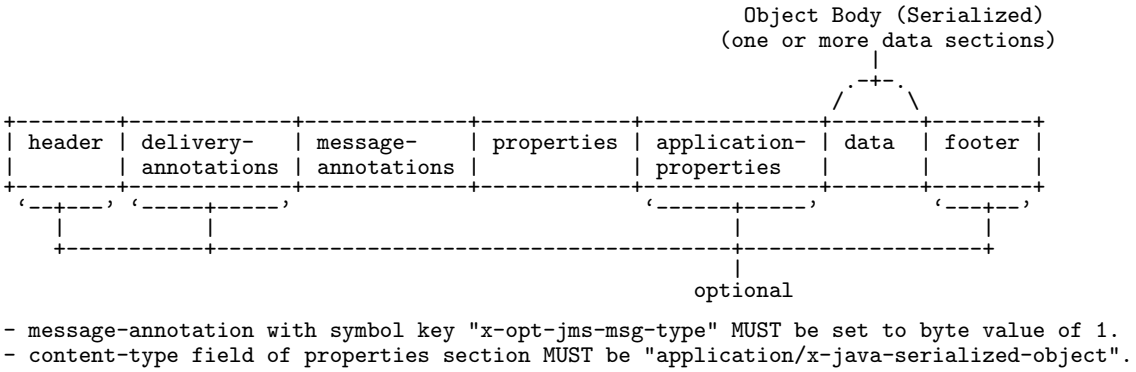

Figure 3.7: AMQP Message Structure of a Java serialized ObjectMessage

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*". When the object is either not set or explicitly set null, an `amqp-value` section containing `null` MUST be sent. The message annotation with `symbol` key of "*x-opt-jms-msg-type*" MUST be set to a `byte` value of 1.

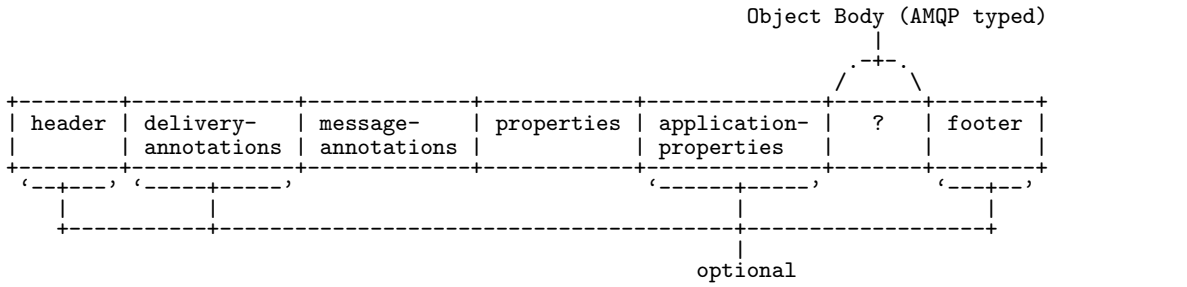

Figure 3.8: AMQP Message Structure of an ObjectMessage using the AMQP type system

3.2.4.7 Message

A `Message` is encoded as a single `amqp-value` section containing `null`. 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.

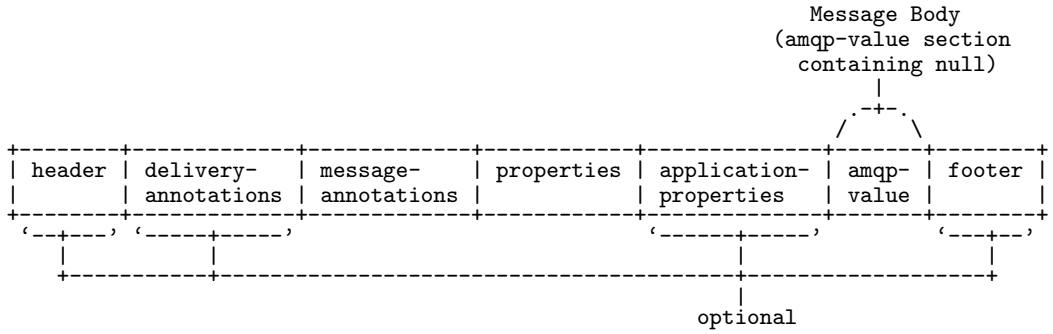

Figure 3.9: AMQP Message Structure of a Message

3.3 Mapping AMQP Messages To JMS

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.

3.3.1 Header Section

Field Name	Description
durable	When receiving a message, the `durable` field of `header` MUST be mapped 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 mapped 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 Epoch.
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 4. JMS Vendor Properties .
delivery-count	This field is mapped to the JMS-defined `JMSXDeliveryCount` property and `JMSRedelivered` header of the Message as follows. 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 delivery-count-handling for more details on handling of this field.

3.3.2 Properties Section

Field Name	Description

message-id	This field is mapped 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`. See 3.2.1.1 JMSMessageID And JMSCorrelationID Handling for REQUIRED additional detail relating to supporting usage of the various AMQP types possible for the `message-id` field of `properties`.
user-id	This field is mapped to the JMS-defined `JMSXUserID` property of the Message. `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 mapped to the `JMSDestination` header of the Message. `JMSDestination` is defined as being of the JMS `Destination` type, while the `to` field of `properties` requires an `address-string`. See 5. 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 is mapped to the `JMSType` header of the Message.
reply-to	This field is mapped to the `JMSReplyTo` header of the Message. `JMSReplyTo` is defined as being of the JMS `Destination` type, while the `reply-to` field of `properties` requires an `address-string`. See 5. 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 mapped 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*, that is `message-id-ulong`, `message-id-uuid`, `message-id-binary` or `message-id-string`. See 3.2.1.1 JMSMessageID And JMSCorrelationID Handling for REQUIRED additional 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, behaviour is left to implementation detail.
content-encoding	This field does not have an equivalent within the JMS specification, behaviour is left to implementation detail.
absolute-expiry-time	This field is mapped to the `JMSExpiration` head of the Message 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` instead if set, see 3.3.1 Header Section for more details.
creation-time	This field is mapped to the `JMSTimestamp` header of the Message. 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 mapped to the JMS-defined `JMSXGroupID` property of the Message.
group-sequence	This field is mapped to the JMS-defined `JMSXGroupSeq` property of the Message. 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 4. JMS Vendor Properties .

3.3.3 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:

- property values with types not defined in the JMS specification

3.3.4 Body Sections

The following sections detail how to determine which type of JMS Message should be used to represent a received 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.

3.3.4.1 Messages With 'x-opt-jms-msg-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 3.2.4.1 Message Type Annotation. If the annotation is not present, the sections which follow should be used to identify the appropriate JMS Message type.

TODO (presentation): Some of these section numbers should be nested but arent. Investigate.

3.3.4.2 Messages Without 'x-opt-jms-msg-type' Annotation

3.3.4.3 Data

Where the "*x-opt-jms-msg-type*" message annotation is not set and one or 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, is set to the `symbol` value "*application/octet-stream*", or is set to a value not determined to represent another message type, then the message MUST be interpreted as a `BytesMessage`.

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`.

When the `content-type` field of `properties` contains a value representing common textual media types as detailed below, the message MUST be interpreted as a `TextMessage`. Where the total length of content in the `data` section(s) is 0, then the return value from the *getText()* method MUST be a Java `String` of length 0.

Top level type	Sub-type
"text"	*
"application"	"xml"
"application"	"xml-dtd"
"application"	Ends with "+xml"
"application"	"json"
"application"	Ends with "+json"
"application"	"javascript"
"application"	"ecmascript"

3.3.4.4 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`.

3.3.4.5 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 `ObjectMessage`.

3.3.4.6 Todo

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

4 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^{32} -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_REPLY_TO_GROUP_ID	Application/ Provider on Receive	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.

5 Destinations

5.1 Destinations On Messages

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-jms-dest*" and "*x-opt-jms-reply-to*" and containing one of the following `byte` values:

Destination Type	Annotation value (type)
Queue	0 (byte)
Topic	1 (byte)
TemporaryQueue	2 (byte)
TemporaryTopic	3 (byte)

Producing JMS clients SHOULD set the "*x-opt-jms-dest*" message annotation on each message sent, and SHOULD set the "*x-opt-jms-reply-to*" message annotation on each message sent that has a `JMSReplyTo` header value.

When receiving an AMQP message which lacks the "*x-opt-jms-dest*" and/or "*x-opt-jms-reply-to*" message annotations, the `JMSDestination` and/or `JMSReplyTo` values respectively SHOULD be constructed using the same `Destination` type derivative as that used when creating the consumer which received the message.

When receiving an AMQP message that 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.

Note that while byte values MUST be used by sending JMS clients, implementations MUST cope with receiving any integral type for the annotation value.

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

5.2 Destinations And Producers/Consumers

When creating producing or consuming entities, links will be established to the remote peer with an appropriate Source or Target address. Some peers may support automatically creating nodes with the appropriate address if they do not exist, while some clients may wish to assert that they have attached to the expected type of node at the given address.

In order to facilitate these actions for the various `Destination` types that JMS supports, type information SHOULD be conveyed when creating producer or consumer links for the application by supplying a terminus capability for the particular Destination type to which the client expects to attach. The following capabilities are defined:

Destination Type	Terminus capability (type)
Queue	queue (symbol)
Topic	topic (symbol)
TemporaryQueue	temporary-queue (symbol)
TemporaryTopic	temporary-topic (symbol)

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

5.3 Temporary Destinations

JMS allows for creation of `TemporaryQueue` and `TemporaryTopic` entities for the lifetime of the parent Connection. Unlike creation of `Queue` and `Topic` objects, JMS does define that creation of these objects at the client result in creation of the node at the peer.

AMQP allows for dynamic creation of peer-named nodes via use of the *dynamic* field on the `source` and `target` types. To create a node with the required lifecycle properties, establish a uniquely named sending link with the `dynamic` field of `target` set *true*, the `expiry-policy` field of `target` set to `symbol` *"link-detach"*, and the `dynamic-node-properties` field of `target` containing the *"lifetime-policy"* `symbol` key mapped to `delete-on-close`. The appropriate capability from 5.2 Destinations And Producers/Consumers MUST be included in the `capabilities` field of `target`.

The creating link will then kept open until the `Connection` is closed, or the *delete()* method is called on the destination object, at which point detaching the link will result destruction of the dynamic node.

As the destination is tied to the life of the `Connection`, the creating link is established on a separate AMQP session not managed by the JMS application.

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

6 Message Producers

6.1 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 3.2.1 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 4. 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 omitted instead rather than populated with a value less than specified by the application.

6.2 Anonymous Producers

JMS producers (e.g `MessageProducer`) can be created with an explicit `Destination`, in which case applications MUST use one of their *send* methods that do not take a destination argument. Alternatively, producers can be created without a particular `Destination`, in which case they MUST be used with their *send* methods that take a destination argument for each message, and are typically described as anonymous producers.

To support the anonymous producer case the concept of an Anonymous Relay node is defined, such that messages sent to the relay node will be relayed to a node at the address given in the `to` field of `properties` of the message. Messages arriving at the relay node that don't contain a valid value for the `to` field of `properties` or are otherwise unable to be relayed successfully will be rejected by the peer. To establish a sending link to the Anonymous Relay node, a link attach is attempted with the `address` field of `target` set to null. Peers supporting the Anonymous Relay functionality advertise this by offering the *ANONYMOUS-RELAY* capability in their Open frame, and clients SHOULD use this ability when offered.

When the ANONYMOUS-RELAY capability is not offered by the peer, clients MAY support the anonymous producer use case through fallback means of establishing sending links to each distinct destination address used by the application, with potential use of link caching mechanisms to improve efficiency.

TODO (content): Update to reference ANONYMOUS-RELAY usage from the Anonymous Terminus spec.

7 Durable Subscriptions

7.1 Subscribe

Applications may create a durable `TopicSubscriber` to receive mesages sent to a Topic while the subscriber is inactive. Each subscription is given a name, unique within the required clientid of the Connection, and only 1 subscriber may be active on the subscription at a time. The act of creating the `TopicSubscriber` is used both to create a subscription if it does not exist, and resume consumption of messages for an existing inactive subscription.

To represent the durable subscription, a consuming link is attached using the subscription name as the link name. The Source `terminus-durability` is set as either *configuration / 1* or *unsettled-state / 2* to indicate the terminus is durable, and a `terminus-expiry-policy` of *never* is used to indicate the terminus does not expire based on a timer.

7.2 Close

Closing a `TopicSubscriber` object does not end the subscription, but rather leaves the subscription in place, accumulating messages sent to the Topic while the subscriber is inactive.

To achieve this, the consuming link for the subscriber is detached without indicating the link should be closed.

7.3 Unsubscribe

An inactive subscription may be removed by calling the Session *unsubscribe* method with the name of the subscription.

A subscription is ended by closing the durable consuming link. To achieve this, a receiving link is attached using the subscription name as the link name, and providing a null Source in the initial attach request. This is necessary because only the subscription name is given at the point of unsubscribe. The broker peer will reattach the named link and respond with an attach containing the actual Source details of the subscription link. The client can then send a closing detach to end the subscription. In the event the named subscription does not exist, the initial attach request will be refused, and the client will treat this as an invalid subscription name.

TODO (content): Merge (or rewrite to coexist) with the following 'shared subscriptions' content.

8 Shared Subscriptions

TODO (content): General rewrite, the content below is direct from overview/notes form text.

TODO (content): Merge with or rewrite to coexist with above 'durable subscription' content.

TODO (presentation): Add links to types etc

The following details a mechanism to support the JMS 2 shared subscriptions additions, augmenting previously existing support for subscription functionality JMS 1.1 offered.

8.1 Short Version

A 'SHARED-SUBS' connection or link capability is used to establish support for the shared subscriptions mechanism.

A named [shared] subscription is identified using the link name.

The subscription identity can be scoped to the AMQP container, i.e. it is unique within the scope of connections using the same container-id, or it can be global, in which case the subscription identity is independent of the container-id and can be used by receivers on any connection with any container-id. A global subscription identity is indicated by adding the symbol 'global' to the Source capabilities.

A subscription may be shared by more than one consuming link, in which case the messages for the subscription are distributed between all the consuming links. To indicate that the link is for a shared subscription the symbol 'shared' is added to Source capabilities.

AMQP requires that link names are unique in a given direction between two containers, identified by the container-id specified in the Open frame. In order to allow multiple links consuming from the same shared subscription on connection(s) with the same container-id a link naming scheme is proposed such that only the part of the link name up to (but excluding) a '|' character is used for establishing the name of the subscription the link refers to. E.g. the link names "mysub|foo" and "mysub|bar" would both be interpreted as referencing a subscription named 'mysub' (which could then be durable or non-durable, and global or container scoped), but as the two names are different they could be used on the same connection or connections with the same container-id. Clients would then utilise this where necessary to ensure unique link names are utilised. The details after the "|" have no semantics associated with them, and are only used to distinguish one link name from another while referencing the same subscription name.

8.2 Intro/Background

JMS 1.1 has volatile and durable single-consumer topic subscriptions. Durable subscriptions have a name, and these names are scoped within a ClientID, with a single Connection being able to use a given ClientID at a time, and a ClientID being mandatory to use a durable subscription.

JMS 2.0 added support for shared volatile and durable subscriptions, such that for a given subscription name multiple consumers can consume from the same subscription to spread processing load. It also makes ClientID optional for these shared subscriptions, with different behaviour given depending on whether a ClientID is set or not. If a ClientID is set, this means that only subscribers on the same Connection can share the subscription. If a ClientID is not set then subscribers from any Connection without a ClientID can share the subscription. Non-shared durable subscriptions require a ClientID as in JMS 1.1 before. Shared durable and non-shared durable subscriptions use a single a name space when a ClientID is set and can't co-exist.

The above essentially gives 5 separate subscription spaces:

- Durable (shared + non-shared) with ClientID
- Shared Volatile with ClientID
- Shared Durable without ClientID
- Shared Volatile without ClientID
- Non-shared Volatile (regardless of ClientID status)

8.3 Existing Non-shared Subscriber Behaviour (used For JMS 1.1):

For single-consumer durable topic subscriptions, the client attaches a receiving link, using the subscription name as the link name, and with a Source having the address of the topic, a terminus durability of 1 or 2, a terminus expiry of never, and filters to reflect selector/no-local. The connections container-id contains the ClientID (which is required for non-shared durable subscriptions) and as links are scoped to container-id this satisfies the ClientID-scoping required for such subscriptions names. Closing (rather than just detaching) the link represents end/unsubscribe of the subscription, and this is utilised by Session#unsubscribe by attaching a receiving link using the subscription name, with a null Source, requiring the server to perform a lookup for an existing subscription using this link name. If none is found then the link is refused and the client throws an exception, otherwise it closes the opened link to provoke unsubscribe.

	Link Name
With a ClientID	mySubscriptionName
Without ClientID	N/A, must have ClientID to use

For single-consumer volatile topic subscriptions, the client attaches a receiving link, using a generated link name, and with a Source having the address of the topic, a terminus durability of 0, a terminus expiry of link-detach, and filters to reflect selector/no-local.

	Link Name
With or without a ClientID	<generated>

8.4 New Shared Subscriber Behaviour

We need a mechanism to support the additional shared (and existing non-shared) subscription behaviours, being able to convey:

- Whether the mechanism is requested / supported / in-use.
- A subscription name (accounting for required name spacing).
- The address of the topic being subscribed to.
- Whether the subscription is durable (and implicitly also how/when it could be/is removed)
- Whether it is shared or not.
- Any selector or no-local (N/A for shared sub) filters.
- We need to be able to signal removal of durable subscriptions given only the subscription name, as that is all Session#unsubscribe gives us.
- Implicitly or explicitly how to handle change of address/filters for a given subscription.

8.5 Inspecting Support For Shared Subs

Since JMS explicitly offers the shared subscriptions via API, and it is actually going to be supported via a layered mechanism, we need a mechanism to detect [lack of] support and avoid confusion where specific servers don't actually support it.

The proposal is to use a "SHARED-SUBS" connection capability or link capability for this. If a server offers the connection capability, it is indicating that any topic address can be used for shared subscriptions. A simple broker might do this for example. On the other hand, a given server might not want/be able to support this for any address but rather only specific addresses. Imagine that a router intermediary is being used to accept connections and facilitate link routing to back end brokers, but additionally also facilitate direct client to client messaging through the router mesh. In this scenario, some addresses may support this behaviour while others might not, and so the router offering the connection capability would be inappropriate.

If the connection capability is not offered by a server, the client can instead use a link capability for detecting support. If the opened link does not offer support for the capability when requested, the client can interpret this as a lack of support and throw an exception accordingly.

8.6 Shared Volatile Subscription

Starting from the non-shared volatile scenario, we additionally need to convey that it is a shared sub, give the subscription name, and whether to share the subscription within a single container or across multiple containers (i.e the ClientID set vs no ClientID scenario).

We need to indicate that we want to share the subscription between multiple subscribers by adding a "shared" capability to the Source. Similarly to the old durable subscription case, we make the link name contain the subscription name. AMQP requires that link names are unique in a given direction between two named containers, however we also desire sharing on a single connection (in JMS, this is actually required to share at all when a ClientID is set). To facilitate this, we need a scheme by which such links can be disambiguated while still retaining the ability to identify the same subscription name. To do this, we propose to consider only the part of the link name up to (but excluding) a '|' character for establishing the identity of the subscription the link refers to, with the remainder being used only to disambiguate the different links from each other. E.g. the link names "mysub|foo" and "mysub|bar" would both be interpreted as referencing subscription name 'mysub' (which could be durable or non-durable, and global or container scoped) but but as the two names are different they could be used on the same connection. The detail after the "|" has no semantics associated with it, and is only used to distinguish one links name from another using the same subscription name, for example by using a counter to index names for a subscription, and qualifiers to aid operators.

For the shared volatile subscription, the link terminus expiry is 'link-detach' and there is no behavioural difference associated with whether the link is detached with closed=true or false, the subscription is considered ended when the last subscriber detaches.

	Link Name
With a ClientID	mySubscriptionName|volatile1 (then 2,3,etc)
Without ClientID	mySubscriptionName|global-volatile1 (then 2,3,etc)

8.7 Shared Durable Subscription

Starting from the non-shared durable scenario. We additionally need to convey that it is a shared sub, and whether to share the subscription within a single container or across multiple containers (i.e the ClientID set vs no ClientID scenario).

We again use the "shared" and "global" source capabilities as outlined for the volatile subscription. Since shared and non-shared durable subscriptions can't coexist with the same name, they use the same initial link name.

Subsequent shared subscribers for a given shared subscription then distinguish their link names with a suffix using the mechanism described above. When a ClientID is not set, the link is qualified with a suffix indicating it is a global subscription link.

For the shared durable subscription, the link terminus expiry is 'never' and the terminus is durable. Detaching the link leaves the subscription in place. For the purposes of ending the subscription, closing the link is used to signal intent to end the subscription (as in the non-shared case). If there are no other active consumers this succeeds, but if there are other active consumer links a detach with error is returned to the client.

	Link Name
With a ClientID	mySubscriptionName
With a ClientID	mySubscriptionName|2 (then 3,etc)
Without ClientID	mySubscriptionName|global
Without ClientID	mySubscriptionName|global2 (then 3,etc)

8.8 Unsubscribe

Closing (detach with closed=true) a link ends the subscription, except if in use by other subscribers (see server enforcement details below). For non-JMS clients this can be any existing link. For JMS clients, a specific new link must be attached knowing only the name of the subscription, as unsubscribe is an explicit operation of the session.

During unsubscribe, if a ClientID is set on the connection, a link named with the subscription name would be used to perform a 'null source lookup' for the subscription, as it is already for the existing non-shared durable subscriptions (see earlier for behaviour outline). If a ClientID is not set, the "|global" suffix will be added as shown previously, e.g:

	Link Name
With a ClientID	mySubscriptionName
Without ClientID	mySubscriptionName|global

Additionally, while using connections that dont have a ClientID set, i.e using global shared subscriptions, the link will have "shared" and "global" desired capabilities added as hints to the server that this is an attempt to attach to a 'global' shared subscription of the appropriate name derived from the link, aiding the server should no link with this name be known by it for the particular client container-id currently in use.

8.9 Server Handling:

Servers will need to handle/provide the following behaviours:

- Durable and volatile subscribers are distinct, may use same subscription names (but not link names) concurrently.
- Container-scoped and 'global' subscribers being distinct, i.e may use same subscription names (but not link names) concurrently.
- Refusing a shared subscriber link using the same link name as for an existing non-shared subscription (and vice versa).
- Refusing a global subscriber link using the same link name as for an existing non-global subscription (and vice versa).
- Changing topic/selector and reinitializing an existing sub with no active subscribers.

- Refusing link trying to change topic/selector for existing sub with active subscribers.
- Refusing to end an otherwise active subscription (e.g unsubscribe from another connection while there is no ClientID set).

9 Delivery Delay

JMS 2.0 introduced support for delivery delay, through which producers may be configured with a delay value so that messages sent by them only become permitted for delivery to consumers at a future point in time.

Support for the the delivery delay mechanism is established through use of a connection capability or a link capability value of *DELAYED_DELIVERY*. Where support for the mechanism can not be established, any attempt to send messages with a non-zero delivery delay MUST fail and result in an exception alerting the application that delayed delivery could not be assured.

JMS clients MUST include *DELAYED_DELIVERY* in the `desired-capabilities` field of `open` in order to request support by the server for the delayed delivery mechanism for any address. If the capability is not present in the `offered-capabilities` field of `open` sent by the server, then the client MUST assume the server does not support delayed delivery unless otherwise established on an individual sending link basis.

If support for the delayed delivery mechanism could not be established for all addresses at a connection level, then the client MUST include *DELAYED_DELIVERY* in the `desired-capabilities` field of `attach` sent for sending links in order to request server support for the delayed delivery mechanism for that specific link. If the capability is not present in the `offered-capabilities` field of `attach` sent by the server, then the client MUST assume the server does not support delayed delivery.

The `JMSDeliveryTime` header is used to convey the earliest point in time at which a message may be delivered to a consumer, represented as the number of milliseconds since the Unix Epoch. The value is set by the producing client during the send operation by adding any configured *delivery delay* value to the time at which the message is sent.

When a message is sent with a delivery delay configured, in order to carry the delivery time information on the AMQP message in an interoperable way, a message annotation with `symbol` key of "*x-opt-delivery-time*" and `long` value MUST be included. If no delivery delay is specified (i.e the value is zero) then the annotation SHOULD be omitted.

When a message is received by a JMS client with the "*x-opt-delivery-time*" annotation, the value MUST be used to populate `JMSDeliveryTime`. If a message is received without the annotation, the client MUST then synthesize the `JMSDeliveryTime` header value from the `JMSTimestamp` header value.

TODO (content): Update to reference the Scheduling Message Delivery spec.

10 Supplementary Definitions

Annotation Name	Reference
x-opt-delivery-time	For further details, see 3.2.1 JMS Headers and 9. Delivery Delay .
x-opt-jms-dest	For further details, see 5. Destinations
x-opt-jms-reply-to	For further details, see 5. Destinations
x-opt-jms-msg-type	For further details, see 3.2.4.1 Message Type Annotation, 3.2.4 Message Body Types and 3.3.4 Body Sections.

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

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