
**Information technology — Advanced
Message Queuing Protocol (AMQP) v1.0
specification**

*Technologies de l'information — Spécification du protocole avancé de
mise en file d'attente de messages (AMPQ) v1.0*



Reference number
ISO/IEC 19464:2014(E)

AVA Documents
Document Sharing Hub
© ISO/IEC 2014



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2014

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 19464 was prepared by the OASIS Advanced Message Queuing Protocol Technical Committee (as OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0) and was adopted, under the PAS procedure, by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, in parallel with its approval by the national bodies of ISO and IEC.



OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0

OASIS Standard
29 October 2012

Specification URIs

This version:

<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.xml> (Authoritative)
<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html>
<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf>

Previous version:

<http://docs.oasis-open.org/amqp/core/v1.0/csprd01/amqp-core-overview-v1.0-csprd01.xml> (Authoritative)
<http://docs.oasis-open.org/amqp/core/v1.0/csprd01/amqp-core-overview-v1.0-csprd01.html>
<http://docs.oasis-open.org/amqp/core/v1.0/csprd01/amqp-core-complete-v1.0-csprd01.pdf>

Latest version:

<http://docs.oasis-open.org/amqp/core/v1.0/amqp-core-overview-v1.0.xml> (Authoritative)
<http://docs.oasis-open.org/amqp/core/v1.0/amqp-core-overview-v1.0.html>
<http://docs.oasis-open.org/amqp/core/v1.0/amqp-core-complete-v1.0.pdf>

Technical Committee:

OASIS Advanced Message Queuing Protocol (AMQP) TC

Chairs:

Ram Jeyaraman (Ram.Jeyaraman@microsoft.com), Microsoft
Angus Telfer (angus.telfer@inetco.com), INETCO Systems

Editors:

Robert Godfrey (robert.godfrey@jpmorgan.com), JPMorgan Chase & Co.
David Ingham (David.Ingham@microsoft.com), Microsoft
Rafael Schloming (rafaels@redhat.com), Red Hat

Additional artifacts:

This specification consists of the following documents:

- Part 0: Overview - Overview of the AMQP specification
<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.xml> (Authoritative)
<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html>
- Part 1: Types - AMQP type system and encoding
<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-types-v1.0-os.xml> (Authoritative)
<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-types-v1.0-os.html>
- Part 2: Transport - AMQP transport layer
<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-transport-v1.0-os.xml> (Authoritative)
<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-transport-v1.0-os.html>
- Part 3: Messaging - AMQP Messaging Layer
<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-messaging-v1.0-os.xml> (Authoritative)
<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-messaging-v1.0-os.html>
- Part 4: Transactions - AMQP Transactions Layer
<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-transactions-v1.0-os.xml> (Authoritative)
<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-transactions-v1.0-os.html>
- Part 5: Security - AMQP Security Layers
<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-security-v1.0-os.xml> (Authoritative)
<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-security-v1.0-os.html>
- XML Document Type Definition (DTD)
<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp.dtd> (Authoritative)

Related work:

This specification replaces or supersedes:

- AMQP v1.0 Final, 07 October 2011. <http://www.amqp.org/specification/1.0/amqp-org-download>

Abstract:

The Advanced Message Queuing Protocol (AMQP) is an open internet protocol for business messaging. It defines a binary wire-level protocol that allows for the reliable exchange of business messages between two parties. AMQP has a layered architecture and the specification is organized as a set of parts that reflects that architecture. Part 1 defines the AMQP type system and encoding. Part 2 defines the AMQP transport layer, an efficient, binary, peer-to-peer protocol for transporting messages between two processes over a network. Part 3 defines the AMQP message format, with a concrete encoding. Part 4 defines how interactions can be grouped within atomic transactions. Part 5 defines the AMQP security layers.

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/>.

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/ipr.php>).

Citation format:

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

[amqp-core-complete-v1.0]

OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0. 29 October 2012. OASIS Standard.

<http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf>.

Notices:

Copyright © OASIS Open 2012. All Rights Reserved.

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

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

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

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

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

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

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

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

Contents

0 Overview	7
0.1 Introduction	7
0.1.1 Terminology	7
0.1.2 Normative References	7
0.1.3 Non-normative References	9
0.2 Conformance	10
0.3 Acknowledgements	11
0.4 Revision History	13
1 Types	16
1.1 Type System	16
1.1.1 Primitive Types	16
1.1.2 Described Types	16
1.1.3 Composite Types	16
1.1.4 Restricted Types	17
1.2 Type Encodings	17
1.2.1 Fixed Width	19
1.2.2 Variable Width	20
1.2.3 Compound	20
1.2.4 Array	20
1.3 Type Notation	21
1.3.1 Primitive Type Notation	21
1.3.2 Composite Type Notation	22
1.3.3 Descriptor Notation	22
1.3.4 Field Notation	23
1.3.5 Restricted Type Notation	23
1.4 Composite Type Representation	24
1.5 Descriptor Values	25
1.6 Primitive Type Definitions	25
1.6.1 null	25
1.6.2 boolean	25
1.6.3 ubyte	26
1.6.4 ushort	26
1.6.5 uint	26
1.6.6 ulong	26
1.6.7 byte	26
1.6.8 short	27
1.6.9 int	27
1.6.10 long	27
1.6.11 float	27
1.6.12 double	27
1.6.13 decimal32	28
1.6.14 decimal64	28
1.6.15 decimal128	28
1.6.16 char	28
1.6.17 timestamp	28
1.6.18 uuid	29
1.6.19 binary	29
1.6.20 string	29

1.6.21	symbol	29
1.6.22	list	29
1.6.23	map	30
1.6.24	array	30
2	Transport	31
2.1	Transport	31
2.1.1	Conceptual Model	31
2.1.2	Communication Endpoints	32
2.1.3	Protocol Frames	34
2.2	Version Negotiation	35
2.3	Framing	36
2.3.1	Frame Layout	37
2.3.2	AMQP Frames	38
2.4	Connections	38
2.4.1	Opening A Connection	39
2.4.2	Pipelined Open	40
2.4.3	Closing A Connection	40
2.4.4	Simultaneous Close	40
2.4.5	Idle Timeout Of A Connection	41
2.4.6	Connection States	41
2.4.7	Connection State Diagram	42
2.5	Sessions	43
2.5.1	Establishing A Session	44
2.5.2	Ending A Session	44
2.5.3	Simultaneous End	45
2.5.4	Session Errors	45
2.5.5	Session States	46
2.5.6	Session Flow Control	47
2.6	Links	48
2.6.1	Naming A Link	48
2.6.2	Link Handles	49
2.6.3	Establishing Or Resuming A Link	49
2.6.4	Detaching And Reattaching A Link	52
2.6.5	Link Errors	52
2.6.6	Closing A Link	53
2.6.7	Flow Control	53
2.6.8	Synchronous Get	55
2.6.9	Asynchronous Notification	56
2.6.10	Stopping A Link	56
2.6.11	Messages	57
2.6.12	Transferring A Message	57
2.6.13	Resuming Deliveries	60
2.6.14	Transferring Large Messages	61
2.7	Performatives	62
2.7.1	Open	62
2.7.2	Begin	64
2.7.3	Attach	65
2.7.4	Flow	67
2.7.5	Transfer	69
2.7.6	Disposition	71
2.7.7	Detach	72
2.7.8	End	72
2.7.9	Close	73
2.8	Definitions	73

2.8.1	Role	73
2.8.2	Sender Settle Mode	73
2.8.3	Receiver Settle Mode	74
2.8.4	Handle	74
2.8.5	Seconds	74
2.8.6	Milliseconds	74
2.8.7	Delivery Tag	74
2.8.8	Delivery Number	74
2.8.9	Transfer Number	75
2.8.10	Sequence No	75
2.8.11	Message Format	75
2.8.12	IETF Language Tag	75
2.8.13	Fields	75
2.8.14	Error	76
2.8.15	AMQP Error	76
2.8.16	Connection Error	77
2.8.17	Session Error	78
2.8.18	Link Error	78
2.8.19	Constant Definitions	79

3	Messaging	81
3.1	Introduction	81
3.2	Message Format	81
3.2.1	Header	82
3.2.2	Delivery Annotations	83
3.2.3	Message Annotations	84
3.2.4	Properties	84
3.2.5	Application Properties	86
3.2.6	Data	86
3.2.7	AMQP Sequence	86
3.2.8	AMQP Value	86
3.2.9	Footer	86
3.2.10	Annotations	87
3.2.11	Message ID ULong	87
3.2.12	Message ID UUID	87
3.2.13	Message ID Binary	87
3.2.14	Message ID String	87
3.2.15	Address String	87
3.2.16	Constant Definitions	87
3.3	Distribution Nodes	88
3.4	Delivery State	88
3.4.1	Received	89
3.4.2	Accepted	89
3.4.3	Rejected	90
3.4.4	Released	90
3.4.5	Modified	90
3.4.6	Resuming Deliveries Using Delivery States	91
3.5	Sources and Targets	95
3.5.1	Filtering Messages	95
3.5.2	Distribution Modes	95
3.5.3	Source	96
3.5.4	Target	97
3.5.5	Terminus Durability	98
3.5.6	Terminus Expiry Policy	99
3.5.7	Standard Distribution Mode	99

3.5.8	Filter Set	100
3.5.9	Node Properties	100
3.5.10	Delete On Close	100
3.5.11	Delete On No Links	100
3.5.12	Delete On No Messages	101
3.5.13	Delete On No Links Or Messages	101
4	Transactions	102
4.1	Transactional Messaging	102
4.2	Declaring a Transaction	102
4.3	Discharging a Transaction	103
4.4	Transactional Work	104
4.4.1	Transactional Posting	104
4.4.2	Transactional Retirement	105
4.4.3	Transactional Acquisition	106
4.4.4	Interaction Of Settlement With Transactions	107
4.4.4.1	Transactional Posting	107
4.4.4.2	Transactional Retirement	108
4.4.4.3	Transactional Acquisition	108
4.5	Coordination	108
4.5.1	Coordinator	108
4.5.2	Declare	109
4.5.3	Discharge	109
4.5.4	Transaction ID	110
4.5.5	Declared	110
4.5.6	Transactional State	110
4.5.7	Transaction Capability	111
4.5.8	Transaction Error	111
5	Security	112
5.1	Security Layers	112
5.2	TLS	112
5.2.1	Alternative Establishment	113
5.2.2	Constant Definitions	113
5.3	SASL	113
5.3.1	SASL Frames	114
5.3.2	SASL Negotiation	114
5.3.3	Security Frame Bodies	115
5.3.3.1	SASL Mechanisms	115
5.3.3.2	SASL Init	115
5.3.3.3	SASL Challenge	116
5.3.3.4	SASL Response	116
5.3.3.5	SASL Outcome	117
5.3.3.6	SASL Code	117
5.3.4	Constant Definitions	117
	XML Document Type Definition (DTD)	119

Part 0: Overview

0.1 Introduction

The Advanced Message Queuing Protocol is an open internet protocol for business messaging.

AMQP is comprised of several layers. The lowest level defines an efficient, binary, peer-to-peer protocol for transporting messages between two processes over a network. Above this, the messaging layer defines an abstract message format, with concrete standard encoding. Every compliant AMQP process MUST be able to send and receive messages in this standard encoding.

0.1.1 Terminology

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this specification are to be interpreted as described in IETF RFC 2119 [RFC2119].

The authoritative form of the AMQP specification consists of a set of XML source documents. These documents are transformed into PDF and HTML representations for readability. The machine readable version of the AMQP DTD describes the XML used for the authoritative source documents. This DTD includes the definition of the syntax used in the excerpts of XML presented in the PDF and HTML representations.

0.1.2 Normative References

[ASCII]

American National Standards Institute, Inc., *American National Standard for Information Systems, Coded Character Sets - 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII)*, ANSI X3.4-1986, March 26, 1986.

[BCP47]

A. Phillips, Ed., M. Davis, Ed., *Tags for Identifying Languages*. IETF BCP: 47, September 2009.
<http://www.ietf.org/rfc/bcp/bcp47.txt>

[IANAHTTPPARAMS]

IANA (Internet Assigned Numbers Authority), *Hypertext Transfer Protocol (HTTP) Parameters*.
<http://www.iana.org/assignments/http-parameters/http-parameters.xml>

[IANAPEN]

IANA (Internet Assigned Numbers Authority), *Private Enterprise Numbers*.
<http://www.iana.org/assignments/enterprise-numbers>

[IANASUBTAG]

IANA (Internet Assigned Numbers Authority), *Language Subtag Registry*.
<http://www.iana.org/assignments/language-subtag-registry>

[IEEE754]

Standard for Floating-Point Arithmetic. IEEE 754-2008, August 2008.
<http://ieeexplore.ieee.org/servlet/opac?punumber=4610933>

[IEEE1003]

The Single UNIX Specification, Version 4. IEEE Std 1003.1-2008, December 2008.
<http://www.unix.org/version4/>

[RFC1982]

R. Elz, R. Bush, *Serial Number Arithmetic*. IETF RFC 1982, August 1996.

<http://www.ietf.org/rfc/rfc1982.txt>

[RFC2046]

N. Freed, N. Borenstein, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. IETF RFC 2046, November 1996.

<http://www.ietf.org/rfc/rfc2046.txt>

[RFC2119]

S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*. IETF RFC 2119, March 1997.

<http://www.ietf.org/rfc/rfc2119.txt>

[RFC2234]

D. Crocker, Ed., P. Overell, *Augmented BNF for Syntax Specifications: ABNF*. IETF RFC 2234, November 1997.

<http://www.ietf.org/rfc/rfc2234.txt>

[RFC2616]

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*. IETF RFC 2616, June 1999.

<http://www.ietf.org/rfc/rfc2616.txt>

[RFC4122]

P. Leach, M. Mealling, R. Salz, *A Universally Unique Identifier (UUID) URN Namespace*. IETF RFC 4122, July 2005.

<http://www.ietf.org/rfc/rfc4122.txt>

[RFC4366]

S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, T. Wright, *Transport Layer Security (TLS) Extensions*. IETF RFC 4366, April 2006.

<http://www.ietf.org/rfc/rfc4366.txt>

[RFC4422]

A. Melnikov, Ed., K. Zeilenga, Ed., *Simple Authentication and Security Layer (SASL)*. IETF RFC 4422, June 2006.

<http://www.ietf.org/rfc/rfc4422.txt>

[RFC4616]

K. Zeilenga, Ed., *The PLAIN Simple Authentication and Security Layer (SASL) Mechanism*. IETF RFC 4616, August 2006.

<http://www.ietf.org/rfc/rfc4616.txt>

[RFC5246]

T. Dierks, E. Rescorla., *The Transport Layer Security (TLS) Protocol Version 1.2*. IETF RFC 5246, August 2008.

<http://www.ietf.org/rfc/rfc5246.txt>

[RFC5646]

A. Phillips, Ed., M. Davis, Ed., *Tags for Identifying Languages*. IETF RFC 5646, September 2009.

<http://www.ietf.org/rfc/rfc5646.txt>

[RFC5802]

C. Newman, A. Menon-Sen, A. Melnikov, N. Williams, *Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms*. IETF RFC 5802, July 2010.

<http://www.ietf.org/rfc/rfc5802.txt>

[UNICODE6]

The Unicode Consortium. *The Unicode Standard, Version 6.0.0*, (Mountain View, CA: The Unicode Consortium, 2011. ISBN 978-1-936213-01-6)

<http://www.unicode.org/versions/Unicode6.0.0/>

0.1.3 Non-normative References

[AMQPCONNCAP]

AMQP Capabilities Registry: Connection Capabilities

<http://www.amqp.org/specification/1.0/connection-capabilities>

[AMQPCONNPROP]

AMQP Capabilities Registry: Connection Properties

<http://www.amqp.org/specification/1.0/connection-properties>

[AMQPDELANN]

AMQP Capabilities Registry: Delivery Annotations

<http://www.amqp.org/specification/1.0/delivery-annotations>

[AMQPDISTMODE]

AMQP Capabilities Registry: Distribution Modes

<http://www.amqp.org/specification/1.0/distribution-modes>

[AMQPFILTERS]

AMQP Capabilities Registry: Filters

<http://www.amqp.org/specification/1.0/filters>

[AMQPFooter]

AMQP Capabilities Registry: Footer

<http://www.amqp.org/specification/1.0/footer>

[AMQPLINKCAP]

AMQP Capabilities Registry: Link Capabilities

<http://www.amqp.org/specification/1.0/link-capabilities>

[AMQPLINKPROP]

AMQP Capabilities Registry: Link Properties

<http://www.amqp.org/specification/1.0/link-properties>

[AMQPLINKSTATEPROP]

AMQP Capabilities Registry: Link State Properties

<http://www.amqp.org/specification/1.0/link-state-properties>

[AMQPMESSANN]

AMQP Capabilities Registry: Message Annotations

<http://www.amqp.org/specification/1.0/message-annotations>

[AMQPNODEPROP]

AMQP Capabilities Registry: Node Properties

<http://www.amqp.org/specification/1.0/node-properties>

[AMQPSESSCAP]

AMQP Capabilities Registry: Session Capabilities

<http://www.amqp.org/specification/1.0/session-capabilities>

[AMQPSESSPROP]

AMQP Capabilities Registry: Session Properties

<http://www.amqp.org/specification/1.0/session-properties>

[AMQPSOURCECAP]

AMQP Capabilities Registry: Source Capabilities

<http://www.amqp.org/specification/1.0/source-capabilities>

[AMQPTARGETCAP]

AMQP Capabilities Registry: Target Capabilities

<http://www.amqp.org/specification/1.0/target-capabilities>

0.2 Conformance

AMQP defines a wire level protocol for business messaging. The definition allows for all common business messaging behaviors. AMQP does not define a wire-level distinction between “clients” and “brokers”, the protocol is symmetric. It is expected and encouraged that implementations of AMQP will have different capabilities. Expectations of the capabilities of a “client library” are different from expectations of a “broker” which are themselves different from the capabilities of a “router”. As relevant profiles emerge (where appropriate and applicable) these will be formalised.

A conformant implementation **MUST** perform protocol negotiation (see Part 2: section 2.2), and then parse, process, and produce frames in accordance with the format and semantics defined in parts 1 through 5 of this specification.

Conformant implementations **MUST NOT** require the use of any extensions defined outside this document in order to interoperate with any other conformant implementation.

Part 1 of this document defines the type system and type encodings that every conformant implementation **MUST** implement.

Part 2 defines the peer-to-peer transport protocol which operates over TCP. Every conformant implementation of AMQP over TCP **MUST** implement Part 2. Future standards mapping AMQP to protocols other than TCP **MAY** modify or replace Part 2 when AMQP is being used over that protocol. A conformant implementation **MUST** implement Part 2 or a mapping of AMQP to some non-TCP protocol.

Part 2 admits behaviors that might not be appropriate for every implementation. For example a “client library” might not allow for its communication partner to spontaneously attempt to initiate a connection and request messages. Where an implementation does not allow for a behavior the implementation **MUST** respond according to the rules defined within Part 2 of the specification.

Part 3 of this document defines the AMQP Messaging Layer. Every conformant implementation which processes messages **MUST** implement this part of the specification.

Some implementations might not process messages (for example, an implementation acting as a “router” which looks only at the routing information carried by the AMQP Transport layer). Such implementations do not actively implement Part 3, but **MUST NOT** act in ways which violate the rules of this part of the specification.

The Messaging layer admits behaviors that might not be appropriate for all implementations (and within an implementation all behaviors might not be available for all configurations). Where a behavior is not admitted, the implementation **MUST** respond according to the rules defined within this specification.

Part 4 defines the requirements for transactional messaging. Transactional messaging defines two roles, that of the *transactional resource* and that of the *transaction controller*. A conformant implementation **SHOULD** be capable of operating in one of these roles but **MAY** be unable to operate in either (for instance a simplistic client library might have no ability to act as a transaction controller and would not be expected to act as a transactional resource).

It is **RECOMMENDED** that implementations designed to act as messaging intermediaries support the ability to act as a transactional resource. It is **RECOMMENDED** that implementations or re-usable libraries provide Application Programming Interfaces to enable them to act as transactional controllers.

Where a behavior is not admitted, the rules defined in part 4 regarding responses to non-admitted behaviors **MUST** be followed.

Part 5 defines Security Layers to provide an authenticated and/or encrypted transport. Implementations **SHOULD** allow the configuration of appropriate levels of security for the domain in which they are to be deployed.

Conformant implementations acting in the TCP server role are strongly **RECOMMENDED** to implement Part 5: section 5.2 (or Part 5: 5.2.1 Alternative Establishment). Implementations acting in the TCP server role are strongly **RECOMMENDED** to implement Part 5: section 5.3 and to support commonly used SASL mechanisms. In particular such implementations **SHOULD** support the PLAIN [RFC4616] and SCRAM-SHA1 [RFC5802] mechanisms.

Conformant implementations acting in the TCP client role SHOULD be capable of being configured to connect to an implementation in the TCP server role that is following the recommendations above.

0.3 Acknowledgements

The following individuals have contributed significantly towards the creation of this specification and are gratefully acknowledged:

- Michael Bridgen (VMware)
- Xin Chen (Microsoft)
- Raphael Cohn (Individual)
- Allan Cornish (INETCO Systems)
- Robert Godfrey (JPMorgan Chase & Co.)
- David Ingham (Microsoft)
- Andreas Moravec (Deutsche Boerse)
- Andreas Mueller (IIT Software)
- Simon MacMullen (VMware)
- John O'Hara (Individual)
- Rafael Schloming (Red Hat)
- Gordon Sim (Red Hat)
- Angus Telfer (INETCO Systems)

The following individuals were members of the OASIS AMQP Technical Committee during the creation of this specification and their contributions are gratefully acknowledged:

- Sanjay Aiyagari (VMware)
- Brian Albers (Kaazing)
- Matthew Arrott (Individual)
- Roger Bass (Traxian)
- Allan Beck (JPMorgan Chase & Co.)
- Mark Blair (Credit Suisse)
- Paul Blanz (Goldman Sachs)
- Laurie Bryson (JPMorgan Chase & Co.)
- Raphael Cohn (Individual)
- Allan Cornish (INETCO Systems)
- Robin Cover (OASIS)
- Raphael Delaporte (Zenika)
- Brad Drysdale (Kaazing)
- Helena Edelson (VMware)
- Chet Ensign (OASIS)
- John Fallows (Kaazing)

- Paul Fremantle (WSO2)
- Robert Gemmell (JPMorgan Chase & Co.)
- Rob Godfrey (JPMorgan Chase & Co.)
- Steve Hodge (Bank of America)
- Steve Huston (Individual)
- David Ingham (Microsoft)
- Stavros Isaiadis (Bank of America)
- Ram Jeyaraman (Microsoft)
- Bill Kahlert (HCL America)
- James Kirkland (Red Hat)
- Hanno Klein (Deutsche Boerse)
- Theo Kramer (Flame Computing Enterprises)
- Alex Kritikos (Software AG)
- Stanley Lewis (Progress Software)
- Mark Little (Red Hat)
- Colin MacNaughton (Progress Software)
- Mansour Mazinani (SITA SC)
- Shawn McAllister (Solace Systems)
- Donald McGarry (Mitre Corporation)
- Dale Moberg (Axway Software)
- Andreas Moravec (Deutsche Boerse)
- Andreas Mueller (IIT Software)
- Suryanarayanan Nagarajan (Software AG)
- Bob Natale (Mitre Corporation)
- John O'Hara (Individual)
- Kimberly Palko (Red Hat)
- Kenneth Peebles (Red Hat)
- Pierre Queinnec (Zenika)
- Makesh Rao (Cisco Systems)
- Alexis Richardson (VMware)
- Rafael Schloming (Red Hat)
- Dee Schur (OASIS)
- Gordon Sim (Red Hat)
- Sunjeet Singh (INETCO Systems)
- Bruce Snyder (VMware)
- Angus Telfer (INETCO Systems)
- Carl Trieloff (Red Hat)

- Asir Vedamuthu (Microsoft)
- Eamon Walshe (StormMQ)
- Denis Weerasiri (WSO2)
- Jeffrey Wheeler (Huawei Technologies Co.)
- Wang Xuan (Primeton Technologies)
- Prasad Yendluri (Software AG)

0.4 Revision History

2012-10-29 : OASIS Standard

2012-08-07 : Candidate OASIS Standard 01

AMQP-74 : HTML Rendering contains broken cross reference links for fields of primitive types

2012-07-18 : Committee Specification 1

AMQP-75 : Minor corrections to the front matter

2012-05-29 : Committee Specification Public Review Draft 2

2012-05-29 : Committee Specification Draft 2

2012-05-28 : Working Draft 9

AMQP-71 : Update acknowledgements to reflect current TC member list

AMQP-72 : In the PDF, add explicit links to all Parts of the document (including DTD) and state authoritative version

AMQP-73 : add clarity to document composition and XML excerpts

2012-05-15 : Working Draft 8

AMQP-62 : Better explain the role of the XML notation used in the specification and its relation to the wire format

AMQP-63 : Improve the conformance section

AMQP-64 : Fix typo in 2.1, second sentence needs initial capital letter

AMQP-65 : Better explain the relationship between links, connections, channels and sessions

AMQP-66 : Remove non-normative use of RFC2119 keywords

AMQP-67 : Improve presentation of the Revision History of the document

AMQP-69 : Fix rendering of sub- and superscript in HTML

2012-02-21 : Committee Specification Public Review Draft 1

2012-02-21 : Committee Specification Draft 1

2012-02-20 : Working Draft 7

AMQP-56 : Fix links to non-normative AMQP Capabilities and Related Work
AMQP-61 : Fixes to front-matter presentation

2012-02-15 : Working Draft 6

AMQP-58 : Fix capitalization of headings on front page material
AMQP-59 : Add missing acknowledgments
AMQP-60 : Correct PDF ToC entry for DTD

2012-02-13 : Working Draft 5

AMQP-50 : Section 3.2.16 is rendered differently in the html and pdf versions of the draft.
AMQP-51 : Add missing acknowledgments
AMQP-52 : Appendix A DTD is missing in the PDF document.
AMQP-53 : Add Revision History
AMQP-54 : Fix capitalization differences between HTML and PDF on front page
AMQP-55 : Change labeling of DTD on front page
AMQP-57 : Unresolved cross references due to removal of redundant subsection

2012-02-02 : Working Draft 4

AMQP-43 : Review comments from Ram Jeyaraman

2012-02-01 : Working Draft 3

AMQP-29 : Add introductory description of the spec's XML schema
AMQP-36 : Clarify that delivery-count is not a count (in 2.6.7 and 2.7.4)
AMQP-37 : Improve description of snd-settle-mode and rcv-settle-mode
AMQP-38 : Provide more information as to how an incomplete settled map is resolved
AMQP-39 : Clarify settlement on multi-transfer deliveries
AMQP-41 : Rephrase language around keys in annotations
AMQP-42 : Clarify use of error field in rejected outcome
AMQP-44 : Specification should refer to AMQP.org document as related (superseded) work
AMQP-47 : Messaging: Incorporate feedback from Steve Huston's review

2012-01-24 : Working Draft 2

AMQP-7 : Typo in SaslInit.hostname description
AMQP-8 : Slight contradiction in description of hostname in SaslInit and TLS SNI
AMQP-9 : Could we add a repository for node-properties?
AMQP-10 : Could we have a repository for additional supported-dist-modes?
AMQP-11 : Message Properties.content-type MIME types need adjustment to fit new section types
AMQP-12 : Labelling of headers and section numbers as per the AMQP WG document...
AMQP-13 : Typo: Transactions in 4.4.1 transactional Posting "Settles" - > "Settle"

- AMQP-14 : PDF Document should use Sans-Serif font to be consistent with OASIS Standards
- AMQP-15 : Move copyright statement to second line of footer to be in line with OASIS style
- AMQP-16 : References to INETCO should be "INETCO Systems Limited"
- AMQP-17 : Typo in AMQP Open.hostname description
- AMQP-18 : Add figure titles to the HTML presentation
- AMQP-19 : Fix generated (sub-)section titles in the PDF where they contain a dash ("-")
- AMQP-20 : PDF Presentation: use correct "open" and "close" double quotes
- AMQP-21 : Change presentation of cross-references to show only section number, not title
- AMQP-22 : Transport: Incorporate feedback from Steve Huston's review
- AMQP-23 : Abstract/Introduction: Incorporate feedback from Steve Huston's review
- AMQP-24 : Update XML transformation to render "n byte" as "n-byte"
- AMQP-25 : Types: Incorporate feedback from Steve Huston's review
- AMQP-26 : Transport: Incorporate feedback from Steve Huston's review (2)
- AMQP-27 : add missing diagram titles
- AMQP-28 : Confused use of the terms two's-complement" and "signed"
- AMQP-30 : Clarify version negotiation
- AMQP-31 : Figure numbering incorrect in the HTML presentation
- AMQP-32 : Clarify open behavior
- AMQP-33 : Tighten conformance language for 2.4.5
- AMQP-34 : Add description of HDR_EXCH state in 2.4.6
- AMQP-35 : Clarifications to 2.6.12
- AMQP-40 : Incorrect title for IEEE1003 reference
- AMQP-45 : Transactions: Incorporate feedback from Steve Huston's review
- AMQP-46 : Security: Incorporate feedback from Steve Huston's review
- AMQP-48 : Remove capitalization of certain terms within the specification
- AMQP-49 : Description of "dynamic" field refers to non-existent concepts of "session name" and "client-id"

2011-11-08 : Working Draft 1

- AMQP-1 : Make the definition of flow.echo more explicit
- AMQP-2 : Make it explicit that flow.properties is only for a link
- AMQP-3 : specification/overview.xml has a contributor's name in all caps
- AMQP-4 : specification/overview.xml is slightly inconsistent in use of company legal entity names and abbreviations thereof

Part 1: Types

1.1 Type System

The AMQP type system defines a set of commonly used primitive types for interoperable data representation. AMQP values can be annotated with additional semantic information beyond that associated with the primitive type. This allows for the association of an AMQP value with an external type that is not present as an AMQP primitive. For example, a URL is commonly represented as a string, however not all strings are valid URLs, and many programming languages and/or applications define a specific type to represent URLs. The AMQP type system would allow for the definition of a code with which to annotate strings when the value is intended to represent a URL.

1.1.1 Primitive Types

The AMQP type system defines a standard set of primitive types for representing both common scalar values and common collections. The scalar types include booleans, integral numbers, floating point numbers, timestamps, UUIDs, characters, strings, binary data, and symbols. The collection types include arrays (monomorphic), lists (polymorphic), and maps.

1.1.2 Described Types

The primitive types defined by AMQP can directly represent many of the basic types present in most popular programming languages, and therefore can be trivially used to exchange basic data. In practice, however, even the simplest applications have their own set of custom types used to model concepts within the application's domain. In messaging applications these custom types need to be externalized for transmission.

AMQP provides a means to do this by allowing any AMQP type to be annotated with a *descriptor*. A *descriptor* forms an association between a custom type, and an AMQP type. This association indicates that the AMQP type is actually a *representation* of the custom type. The resulting combination of the AMQP type and its descriptor is referred to as a *described type*.

A described type contains two distinct kinds of type information. It identifies both an AMQP type and a custom type (as well as the relationship between them), and so can be understood at two different levels. An application with intimate knowledge of a given domain can understand described types as the custom types they represent, thereby decoding and processing them according to the complete semantics of the domain. An application with no intimate knowledge can still understand the described types as AMQP types, decoding and processing them as such.

1.1.3 Composite Types

AMQP defines a number of *composite types* used for encoding structured data such as frame bodies. A composite type defines a composite value where each constituent value is identified by a well-known named *field*. Each composite type definition includes an ordered sequence of fields, each with a specified name, type, and multiplicity. Composite type definitions also include one or more descriptors (symbolic and/or numeric) for identifying their defined representations. Composite types are formally defined using the XML notation defined in section 1.3.

1.1.4 Restricted Types

AMQP defines the notion of a *restricted type*. This is a new type derived from an existing type where the permitted values of the new type are a subset of the values of the existing type. Restricted types are commonly used programming constructs, the most frequent being “enumerations” which in AMQP terminology are restrictions of the integral types. However, the AMQP notion of restricted types can also represent more open ended restrictions such as a URL which can be thought of as a restriction of the string type.

A restricted type definition might limit the permitted values to either a pre-defined fixed set, or an open-ended set. In the former case, each permitted value is called a *choice* and all possible choices are listed in the formal type definition. In the latter case, the nature of the restriction is specified as text in the formal type definition.

The existing type from which a restricted type is derived is referred to as the *source type* for the restriction. A restricted type might or might not be annotated with a descriptor on the wire depending on the formal definition for the type.

1.2 Type Encodings

An AMQP encoded data stream consists of untyped bytes with embedded constructors. The embedded constructor indicates how to interpret the untyped bytes that follow. Constructors can be thought of as functions that consume untyped bytes from an open ended byte stream and construct a typed value. An AMQP encoded data stream always begins with a constructor.

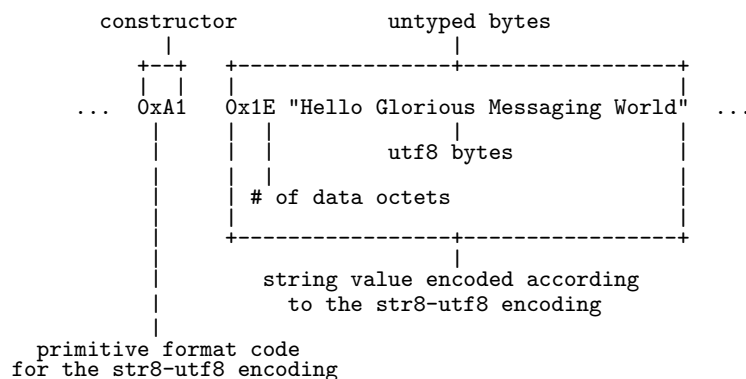


Figure 1.1: Primitive Format Code (String)

An AMQP constructor consists of either a primitive format code, or a described format code. A primitive format code is a constructor for an AMQP primitive type. A described format code consists of a descriptor and a primitive format-code. A descriptor defines how to produce a domain specific type from an AMQP primitive value.

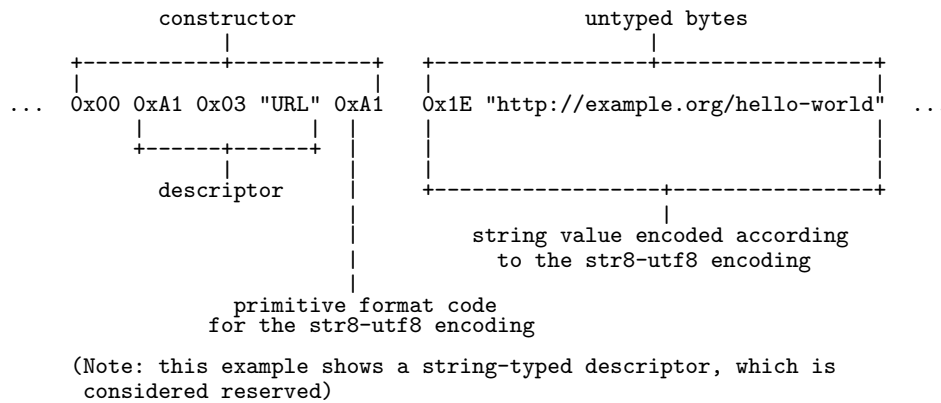


Figure 1.2: Described Format Code (URL)

The descriptor portion of a described format code is itself any valid AMQP encoded value, including other described values. The formal BNF for constructors is given below.

```

constructor = format-code
              / %x00 descriptor constructor

format-code = fixed / variable / compound / array
fixed       = empty / fixed-one / fixed-two / fixed-four
              / fixed-eight / fixed-sixteen
variable    = variable-one / variable-four
compound    = compound-one / compound-four
array       = array-one / array-four

descriptor = value
value      = constructor untyped-bytes
untyped-bytes = *OCTET ; this is not actually *OCTET, the
                  ; valid byte sequences are restricted
                  ; by the constructor

; fixed width format codes
empty      = %x40-4E / %x4F %x00-FF
fixed-one  = %x50-5E / %x5F %x00-FF
fixed-two  = %x60-6E / %x6F %x00-FF
fixed-four = %x70-7E / %x7F %x00-FF
fixed-eight = %x80-8E / %x8F %x00-FF
fixed-sixteen = %x90-9E / %x9F %x00-FF

; variable width format codes
variable-one = %xA0-AE / %xAF %x00-FF
variable-four = %xB0-BE / %xBF %x00-FF

; compound format codes
compound-one = %xC0-CE / %xCF %x00-FF
compound-four = %xD0-DE / %xDF %x00-FF

; array format codes
array-one = %xE0-EE / %xEF %x00-FF
array-four = %xF0-FE / %xFF %x00-FF

```

Figure 1.3: Constructor BNF

Format codes map to one of four different categories: fixed width, variable width, compound and array. Values encoded within each category share the same basic structure parameterized by width. The subcategory within a format-code identifies both the category and width.

Fixed Width

The size of fixed-width data is determined based solely on the subcategory of the format code for the fixed width value.

- Variable Width

The size of variable-width data is determined based on an encoded size that prefixes the data. The width of the encoded size is determined by the subcategory of the format code for the variable width value.
- Compound

Compound data is encoded as a size and a count followed by a polymorphic sequence of *count* constituent values. Each constituent value is preceded by a constructor that indicates the semantics and encoding of the data that follows. The width of the size and count is determined by the subcategory of the format code for the compound value.
- Array

Array data is encoded as a size and count followed by an array element constructor followed by a monomorphic sequence of values encoded according to the supplied array element constructor. The width of the size and count is determined by the subcategory of the format code for the array.

The bits within a format code can be interpreted according to the following layout:

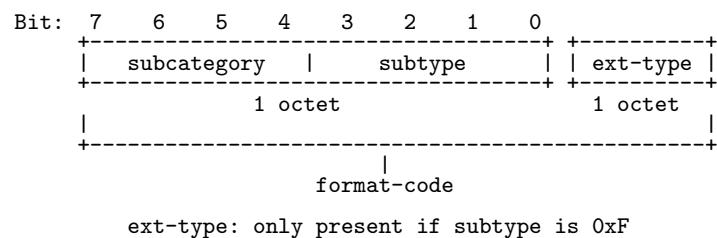


Figure 1.4: Format Code Layout

The following table describes the subcategories of format-codes:

Subcategory	Category	Format
0x4	Fixed Width	Zero octets of data.
0x5	Fixed Width	One octet of data.
0x6	Fixed Width	Two octets of data.
0x7	Fixed Width	Four octets of data.
0x8	Fixed Width	Eight octets of data.
0x9	Fixed Width	Sixteen octets of data.
0xA	Variable Width	One octet of size, 0-255 octets of data.
0xB	Variable Width	Four octets of size, 0-4294967295 octets of data.
0xC	Compound	One octet each of size and count, 0-255 distinctly typed values.
0xD	Compound	Four octets each of size and count, 0-4294967295 distinctly typed values.
0xE	Array	One octet each of size and count, 0-255 uniformly typed values.
0xF	Array	Four octets each of size and count, 0-4294967295 uniformly typed values.

Figure 1.5: Subcategory Formats

Please note, unless otherwise specified, AMQP uses network byte order for all numeric values.

1.2.1 Fixed Width

The width of a specific fixed width encoding can be computed from the subcategory of the format code for the fixed width value:

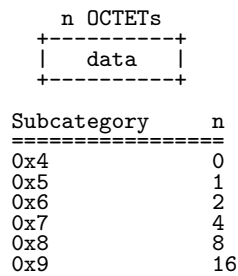


Figure 1.6: Layout of Fixed Width Data Encodings

1.2.2 Variable Width

All variable width encodings consist of a size in octets followed by *size* octets of encoded data. The width of the size for a specific variable width encoding can be computed from the subcategory of the format code:

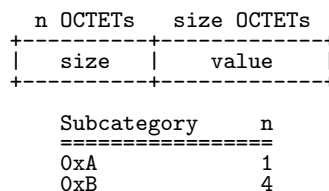


Figure 1.7: Layout of Variable Width Data Encodings

1.2.3 Compound

All compound encodings consist of a size and a count followed by *count* encoded items. The width of the size and count for a specific compound encoding can be computed from the category of the format code:

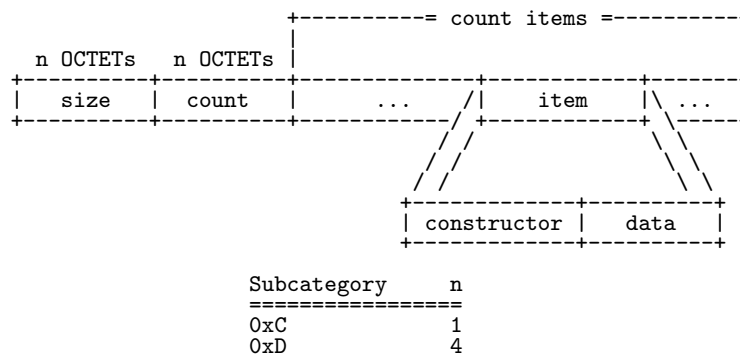


Figure 1.8: Layout of Compound Data Encodings

1.2.4 Array

All array encodings consist of a size followed by a count followed by an element constructor followed by *count* elements of encoded data formatted as REQUIRED by the element constructor:

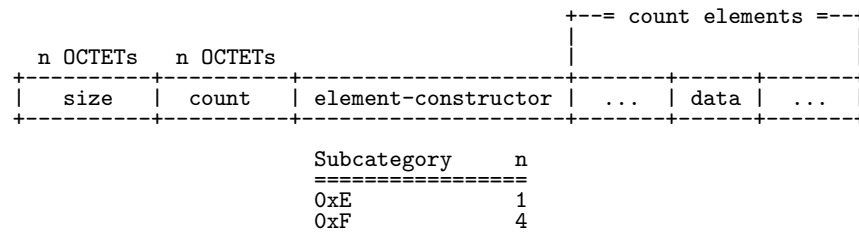


Figure 1.9: Layout of Array Encodings

1.3 Type Notation

Types are formally specified using an XML notation described below. Please note that this XML notation is used only to define the type and is never used to represent encoded data. All encoded AMQP data is binary as described in section 1.2.

The type element formally defines a semantic type used to exchange data on the wire. Every type definition has the following general form. The child elements present depend upon the class of the type definition (primitive, composite, or restricted):

```
<type class="primitive|composite|restricted" name="..." label="..." provides="...">
  ...
</type>
```

Figure 1.10: Type Definitions

The attributes of a type definition specify the following:

- name** The name of the type.
- label** A short description of the type.
- class** A string identifying what class of type is being defined: “primitive”, “composite”, or “restricted”.
- provides** A comma separated list of archetypes (see field element).

There are three different classes of types identified by the “class” attribute: primitive, composite, and restricted.

1.3.1 Primitive Type Notation

A “primitive” type definition will contain one or more encoding elements that describe how the data is formatted on the wire. Primitive types do not contain the descriptor, field, or choice elements.

```
<type class="primitive" name="..." label="..." provides="...">
  <encoding ... > ... </encoding>
  ...
  <encoding ... > ... </encoding>
</type>
```

Figure 1.11: Primitive Type Definitions

The encoding element defines how a primitive type is encoded. The specification defines 4 general categories of encodings: fixed, variable, compound, and array. A specific encoding provides further details of how data is formatted within its general category.

```
<type class="primitive" ... >
  <encoding name="..." code="0xNN" category="fixed|variable|compound|array" width="N" label="..." />
</type>
```

Figure 1.12: Encoding Definitions

name	The name of the encoding.
label	A short description of the encoding.
code	The numeric value that prefixes the encoded data on the wire.
category	The category of the encoding: “fixed”, “variable”, “compound”, or “array”.
width	The width of the encoding or the size/count prefixes depending on the category.

1.3.2 Composite Type Notation

A “composite” type definition specifies a new kind of record type, i.e., a type composed of a fixed number of fields whose values are each of a specific type. A composite type does not have a new encoding, but is sent on the wire as a list annotated by a specific descriptor value that identifies it as a representation of the defined composite type. A composite type definition will contain a descriptor and zero or more field elements. Composite types do not contain encoding or choice elements. The source attribute of a composite type will always be “list”, other values are reserved for future use.

```
<type class="composite" name="..." label="..." provides="...">
  <descriptor name="..." code="..." />
  <field name="..." ... > ... </field>
  <field name="..." ... > ... </field>
</type>
```

Figure 1.13: Composite Type Definitions

1.3.3 Descriptor Notation

The descriptor element specifies what annotation is used to identify encoded values as representations of a described type.

```
<type class="composite|restricted" ...>
  <!-- domain-id:descriptor-id -->
  <descriptor name="domain:name" code="0xNNNNNNNN:0xNNNNNNNN" />
</type>
```

Figure 1.14: Descriptor Definitions

name	A symbolic name for the representation.
code	The numeric value.

1.3.4 Field Notation

The field element identifies a field within a composite type. Every field has the following attributes:

```
<type class="composite" ...>
  <field name="..." type="..." requires="..." default="..." label="..."
    mandatory="true|false"
    multiple="true|false" />
  ...
</type>
```

Figure 1.15: Field Definitions

name	A name that uniquely identifies the field within the type.
type	The type of the field. This attribute defines the range of values that are permitted to appear in this field. It might name a specific type, in which case the values are restricted to that type, or it might be the special character "*", in which case a value of any type is permitted. In the latter case the range of values might be further restricted by the requires attribute.
requires	A comma separated list of archetypes. Field values are restricted to types providing at least one of the specified archetypes.
default	A default value for the field if no value is encoded.
label	A short description of the field.
mandatory	"true" iff a non null value for the field is always encoded.
multiple	"true" iff the field can have multiple values of its specified type.

1.3.5 Restricted Type Notation

A "restricted" type definition specifies a new kind of type whose values are restricted to a subset of the values that might be represented with another type. The source attribute identifies the base type from which a restricted type is derived. A restricted type can have a descriptor element in which case it is identified by a descriptor on the wire. The values permitted by a restricted type can be specified either via documentation, or directly limited to a fixed number of values by the choice elements contained within the definition.

```
<type class="restricted" name="..." label="..." provides="...">
  <descriptor name="..." code="..." />
  <choice name="..." value="..." />
  ...
  <choice name="..." value="..." />
</type>
```

Figure 1.16: Restricted Type Definitions

The choice element identifies a legal value for a restricted type. The choice element has the following attributes:

```
<type class="restricted" ...>
  <choice name="..." value="..." />
  ...
</type>
```

Figure 1.17: Choice Definitions

name	A name for the value.
value	The permitted value.

1.4 Composite Type Representation

AMQP composite types are represented as a described list. Each element in the list is positionally correlated with the fields listed in the composite type definition. The permitted element values are determined by the type specification and multiplicity of the corresponding field definitions. When the trailing elements of the list representation are null, they MAY be omitted. The descriptor of the list indicates the specific composite type being represented.

```
<type class="composite" name="book" label="example composite type">
  <doc>
    <p>An example composite type.</p>
  </doc>

  <descriptor name="example:book:list" code="0x00000003:0x00000002"/>

  <field name="title" type="string" mandatory="true" label="title of the book"/>
  <field name="authors" type="string" multiple="true"/>
  <field name="isbn" type="string" label="the ISBN code for the book"/>
</type>
```

Figure 1.18: Example Composite Type

The *mandatory* attribute of a field description controls whether a null element value is permitted in the representation.

The *multiple* attribute of a field description controls whether multiple element values are permitted in the representation. A single element of the type specified in the field description is always permitted. Multiple values are represented by the use of an array where the type of the elements in the array is the type defined in the field definition. Note that a null value and a zero-length array (with a correct type for its elements) both describe an absence of a value and MUST be treated as semantically identical.

A field which is defined as both multiple and mandatory MUST contain at least one value (i.e. for such a field both *null* and an array with no entries are invalid).

The described list shown below is an example composite value of the *book* type defined above. A trailing null element corresponding to the absence of an ISBN value is depicted in the example, but can optionally be omitted according to the encoding rules.

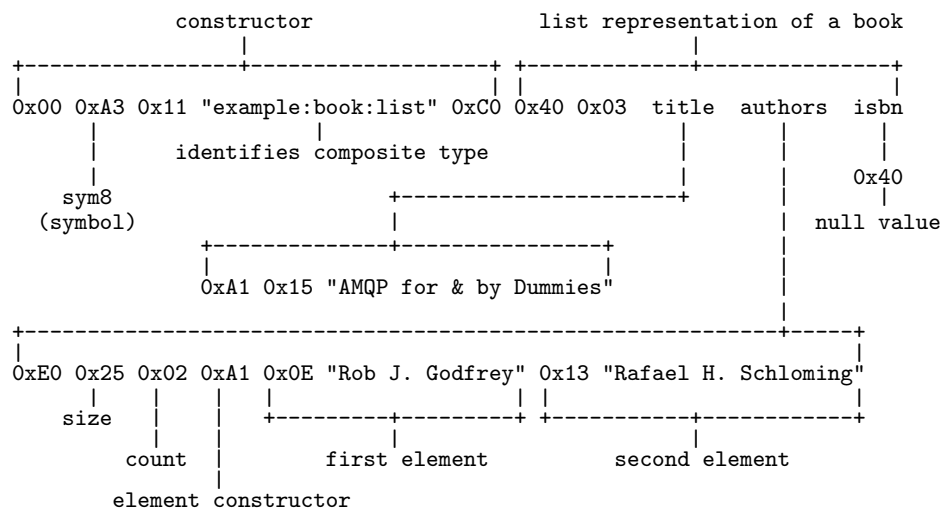


Figure 1.19: Example Composite Value

1.5 Descriptor Values

Descriptor values other than symbolic (`symbol`) or numeric (`ulong`) are, while not syntactically invalid, reserved - this includes numeric types other than `ulong`. To allow for users of the type system to define their own descriptors without collision of descriptor values, an assignment policy for symbolic and numeric descriptors is given below.

The namespace for both symbolic and numeric descriptors is divided into distinct domains. Each domain has a defined symbol and/or 4 byte numeric id assigned by the AMQP working group. For numeric ids the assigned domain-id will be equal to the IANA Private Enterprise Number (PEN) of the requesting organisation [IANAPEN] with domain-id 0 reserved for descriptors defined in the AMQP specification.

Descriptors are then assigned within each domain according to the following rules:

symbolic descriptors

`<domain>:<name>`

numeric descriptors

`(domain-id << 32) | descriptor-id`

1.6 Primitive Type Definitions

1.6.1 null

Indicates an empty value.

```

<type name="null" class="primitive">
  <encoding code="0x40" category="fixed" width="0" label="the null value"/>
</type>

```

1.6.2 boolean

Represents a true or false value.

```
<type name="boolean" class="primitive">
  <encoding code="0x56" category="fixed" width="1"
    label="boolean with the octet 0x00 being false and octet 0x01 being true"/>
  <encoding name="true" code="0x41" category="fixed" width="0" label="the boolean value true"/>
  <encoding name="false" code="0x42" category="fixed" width="0" label="the boolean value false"/>
</type>
```

1.6.3 ubyte

Integer in the range 0 to $2^8 - 1$ inclusive.

```
<type name="ubyte" class="primitive">
  <encoding code="0x50" category="fixed" width="1" label="8-bit unsigned integer"/>
</type>
```

1.6.4 ushort

Integer in the range 0 to $2^{16} - 1$ inclusive.

```
<type name="ushort" class="primitive">
  <encoding code="0x60" category="fixed" width="2"
    label="16-bit unsigned integer in network byte order"/>
</type>
```

1.6.5 uint

Integer in the range 0 to $2^{32} - 1$ inclusive.

```
<type name="uint" class="primitive">
  <encoding code="0x70" category="fixed" width="4"
    label="32-bit unsigned integer in network byte order"/>
  <encoding name="smalluint" code="0x52" category="fixed" width="1"
    label="unsigned integer value in the range 0 to 255 inclusive"/>
  <encoding name="uint0" code="0x43" category="fixed" width="0" label="the uint value 0"/>
</type>
```

1.6.6 ulong

Integer in the range 0 to $2^{64} - 1$ inclusive.

```
<type name="ulong" class="primitive">
  <encoding code="0x80" category="fixed" width="8"
    label="64-bit unsigned integer in network byte order"/>
  <encoding name="smallulong" code="0x53" category="fixed" width="1"
    label="unsigned long value in the range 0 to 255 inclusive"/>
  <encoding name="ulong0" code="0x44" category="fixed" width="0" label="the ulong value 0"/>
</type>
```

1.6.7 byte

Integer in the range $-(2^7)$ to $2^7 - 1$ inclusive.

```
<type name="byte" class="primitive">
  <encoding code="0x51" category="fixed" width="1"
    label="8-bit two's-complement integer"/>
</type>
```

1.6.8 short

Integer in the range $-(2^{15})$ to $2^{15} - 1$ inclusive.

```
<type name="short" class="primitive">
  <encoding code="0x61" category="fixed" width="2"
    label="16-bit two's-complement integer in network byte order"/>
</type>
```

1.6.9 int

Integer in the range $-(2^{31})$ to $2^{31} - 1$ inclusive.

```
<type name="int" class="primitive">
  <encoding code="0x71" category="fixed" width="4"
    label="32-bit two's-complement integer in network byte order"/>
  <encoding name="smallint" code="0x54" category="fixed" width="1"
    label="8-bit two's-complement integer"/>
</type>
```

1.6.10 long

Integer in the range $-(2^{63})$ to $2^{63} - 1$ inclusive.

```
<type name="long" class="primitive">
  <encoding code="0x81" category="fixed" width="8"
    label="64-bit two's-complement integer in network byte order"/>
  <encoding name="smalllong" code="0x55" category="fixed" width="1"
    label="8-bit two's-complement integer"/>
</type>
```

1.6.11 float

32-bit floating point number (IEEE 754-2008 binary32).

```
<type name="float" class="primitive">
  <encoding name="ieee-754" code="0x72" category="fixed" width="4" label="IEEE 754-2008 binary32"/>
</type>
```

A 32-bit floating point number (IEEE 754-2008 binary32 [IEEE754]).

1.6.12 double

64-bit floating point number (IEEE 754-2008 binary64).

```
<type name="double" class="primitive">
  <encoding name="ieee-754" code="0x82" category="fixed" width="8" label="IEEE 754-2008 binary64"/>
</type>
```

A 64-bit floating point number (IEEE 754-2008 binary64 [IEEE754]).

1.6.13 decimal32

32-bit decimal number (IEEE 754-2008 decimal32).

```
<type name="decimal32" class="primitive">
  <encoding name="ieee-754" code="0x74" category="fixed" width="4"
    label="IEEE 754-2008 decimal32 using the Binary Integer Decimal encoding"/>
</type>
```

A 32-bit decimal number (IEEE 754-2008 decimal32 [IEEE754]).

1.6.14 decimal64

64-bit decimal number (IEEE 754-2008 decimal64).

```
<type name="decimal64" class="primitive">
  <encoding name="ieee-754" code="0x84" category="fixed" width="8"
    label="IEEE 754-2008 decimal64 using the Binary Integer Decimal encoding"/>
</type>
```

A 64-bit decimal number (IEEE 754-2008 decimal64 [IEEE754]).

1.6.15 decimal128

128-bit decimal number (IEEE 754-2008 decimal128).

```
<type name="decimal128" class="primitive">
  <encoding name="ieee-754" code="0x94" category="fixed" width="16"
    label="IEEE 754-2008 decimal128 using the Binary Integer Decimal encoding"/>
</type>
```

A 128-bit decimal number (IEEE 754-2008 decimal128 [IEEE754]).

1.6.16 char

A single Unicode character.

```
<type name="char" class="primitive">
  <encoding name="utf32" code="0x73" category="fixed" width="4"
    label="a UTF-32BE encoded Unicode character"/>
</type>
```

A UTF-32BE encoded Unicode character [UNICODE6].

1.6.17 timestamp

An absolute point in time.

```
<type name="timestamp" class="primitive">
  <encoding name="ms64" code="0x83" category="fixed" width="8"
    label="64-bit two's-complement integer representing milliseconds since the unix epoch"/>
</type>
```

Represents an approximate point in time using the Unix time `t` [IEEE1003] encoding of UTC, but with a precision of milliseconds. For example, 1311704463521 represents the moment 2011-07-26T18:21:03.521Z.

1.6.18 uuid

A universally unique identifier as defined by RFC-4122 section 4.1.2 .

```
<type name="uuid" class="primitive">
  <encoding code="0x98" category="fixed" width="16"
    label="UUID as defined in section 4.1.2 of RFC-4122"/>
</type>
```

UUID is defined in section 4.1.2 of RFC-4122 [RFC4122].

1.6.19 binary

A sequence of octets.

```
<type name="binary" class="primitive">
  <encoding name="vbin8" code="0xa0" category="variable" width="1"
    label="up to 2^8 - 1 octets of binary data"/>
  <encoding name="vbin32" code="0xb0" category="variable" width="4"
    label="up to 2^32 - 1 octets of binary data"/>
</type>
```

1.6.20 string

A sequence of Unicode characters.

```
<type name="string" class="primitive">
  <encoding name="str8-utf8" code="0xa1" category="variable" width="1"
    label="up to 2^8 - 1 octets worth of UTF-8 Unicode (with no byte order mark)"/>
  <encoding name="str32-utf8" code="0xb1" category="variable" width="4"
    label="up to 2^32 - 1 octets worth of UTF-8 Unicode (with no byte order mark)"/>
</type>
```

A string represents a sequence of Unicode characters as defined by the Unicode V6.0.0 standard [UNICODE6].

1.6.21 symbol

Symbolic values from a constrained domain.

```
<type name="symbol" class="primitive">
  <encoding name="sym8" code="0xa3" category="variable" width="1"
    label="up to 2^8 - 1 seven bit ASCII characters representing a symbolic value"/>
  <encoding name="sym32" code="0xb3" category="variable" width="4"
    label="up to 2^32 - 1 seven bit ASCII characters representing a symbolic value"/>
</type>
```

Symbols are values from a constrained domain. Although the set of possible domains is open-ended, typically the both number and size of symbols in use for any given application will be small, e.g. small enough that it is reasonable to cache all the distinct values. Symbols are encoded as ASCII characters [ASCII].

1.6.22 list

A sequence of polymorphic values.

```

<type name="list" class="primitive">
  <encoding name="list0" code="0x45" category="fixed" width="0"
    label="the empty list (i.e. the list with no elements)"/>
  <encoding name="list8" code="0xc0" category="compound" width="1"
    label="up to 2^8 - 1 list elements with total size less than 2^8 octets"/>
  <encoding name="list32" code="0xd0" category="compound" width="4"
    label="up to 2^32 - 1 list elements with total size less than 2^32 octets"/>
</type>

```

1.6.23 map

A polymorphic mapping from distinct keys to values.

```

<type name="map" class="primitive">
  <encoding name="map8" code="0xc1" category="compound" width="1"
    label="up to 2^8 - 1 octets of encoded map data"/>
  <encoding name="map32" code="0xd1" category="compound" width="4"
    label="up to 2^32 - 1 octets of encoded map data"/>
</type>

```

A map is encoded as a compound value where the constituent elements form alternating key value pairs.

item 0	item 1	item n-1	item n
key 1	val 1	..	key n/2
	val n/2		

Figure 1.20: Layout of Map Encoding

Map encodings MUST contain an even number of items (i.e. an equal number of keys and values). A map in which there exist two identical key values is invalid. Unless known to be otherwise, maps MUST be considered to be ordered, that is, the order of the key-value pairs is semantically important and two maps which are different only in the order in which their key-value pairs are encoded are not equal.

1.6.24 array

A sequence of values of a single type.

```

<type name="array" class="primitive">
  <encoding name="array8" code="0xe0" category="array" width="1"
    label="up to 2^8 - 1 array elements with total size less than 2^8 octets"/>
  <encoding name="array32" code="0xf0" category="array" width="4"
    label="up to 2^32 - 1 array elements with total size less than 2^32 octets"/>
</type>

```

Part 2: Transport

2.1 Transport

2.1.1 Conceptual Model

An AMQP network consists of *nodes* connected via *links*. Nodes are named entities responsible for the safe storage and/or delivery of *messages*. Messages can originate from, terminate at, or be relayed by nodes.

A link is a unidirectional route between two nodes. A link attaches to a node at a *terminus*. There are two kinds of terminus: *sources* and *targets*. A terminus is responsible for tracking the state of a particular stream of incoming or outgoing messages. Sources track outgoing messages and targets track incoming messages. Messages only travel along a link if they meet the entry criteria at the source.

As a message travels through an AMQP network, the responsibility for safe storage and delivery of the message is transferred between the nodes it encounters. The *link protocol* (defined in section 2.6) manages the transfer of responsibility between the source and target.

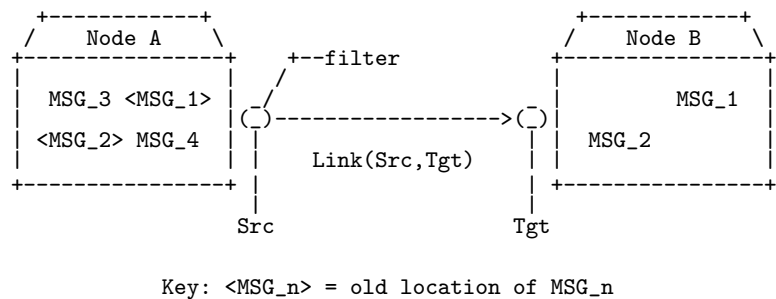


Figure 2.1: Message Transfer between Nodes

Nodes exist within a *container*. Examples of containers are *brokers* and *client* applications. Each container MAY hold many nodes. Examples of AMQP nodes are *producers*, *consumers*, and *queues*. Producers and consumers are the elements within an application that generate and process messages. Queues are entities that store and forward messages.

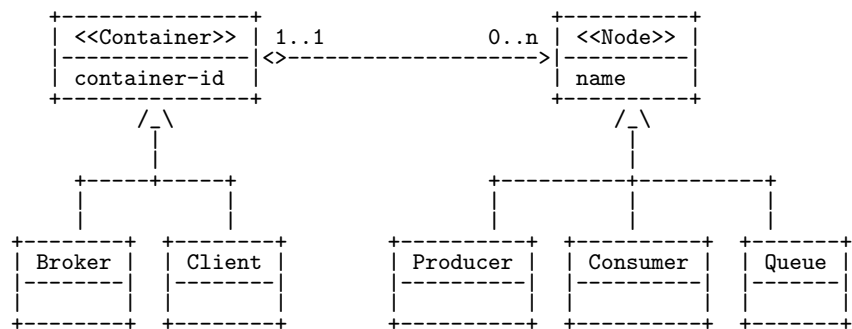


Figure 2.2: Class Diagram of Concrete Containers and Nodes

2.1.2 Communication Endpoints

The AMQP transport specification defines a peer-to-peer protocol for transferring messages between nodes in an AMQP network. This portion of the specification is not concerned with the internal workings of any sort of node, and only deals with the mechanics of unambiguously transferring a message from one node to another.

In order for communication to occur between nodes in different containers a connection needs be established.

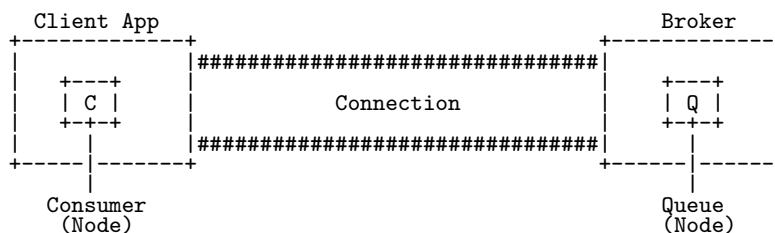


Figure 2.3: An AMQP Connection

An AMQP connection consists of a full-duplex, reliably ordered sequence of *frames*. A frame is the unit of work carried on the wire. Connections have a negotiated maximum frame size allowing byte streams to be easily defragmented into complete frame bodies representing the independently parsable units formally defined in section 2.7. The precise requirement for a connection is that if the n^{th} frame arrives, all frames prior to n MUST also have arrived. It is assumed connections are transient and can fail for a variety of reasons resulting in the loss of an unknown number of frames, but they are still subject to the aforementioned ordered reliability criteria. This is similar to the guarantee that TCP or SCTP provides for byte streams, and the specification defines a framing system used to parse a byte stream into a sequence of frames for use in establishing an AMQP connection (see section 2.3).

An AMQP connection is divided into a negotiated number of independent unidirectional *channels*. Each frame is marked with the channel number indicating its parent channel, and the frame sequence for each channel is multiplexed into a single frame sequence for the connection.

An AMQP *session* correlates two unidirectional channels to form a bidirectional, sequential conversation between two containers. Sessions provide a flow control scheme based on the number of *transfer frames* transmitted. Since frames have a maximum size for a given connection, this provides flow control based on the number of bytes transmitted and can be used to optimize performance.

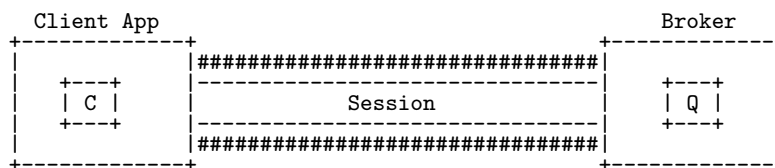


Figure 2.4: An AMQP Session

A single connection MAY have multiple independent sessions active simultaneously, up to the negotiated channel limit. Both connections and sessions are modeled by each peer as *endpoints* that store local and last known remote state regarding the connection or session in question.

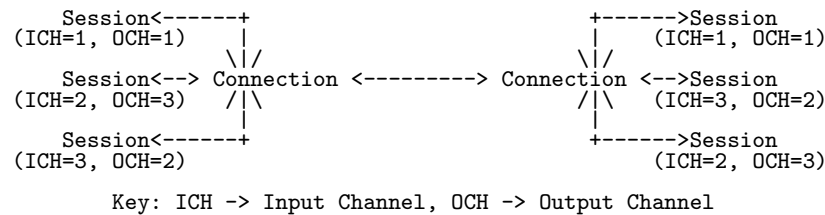


Figure 2.5: Session & Connection Endpoints

In order to transfer messages between nodes (e.g., to move messages from a queue to a consumer) a *link* needs to be established between the nodes. A link is a unidirectional route between two nodes. A link attaches to a node at a terminus. There are two kinds of terminus: sources and targets. A terminus is responsible for tracking the state of a particular stream of incoming or outgoing messages. Sources track outgoing messages and targets track incoming messages. Messages only travel along a link if they meet the entry criteria at the source.

Links provide a credit-based flow control scheme based on the number of messages transmitted, allowing applications to control which nodes to receive messages from at a given point (e.g., to explicitly fetch a message from a given queue).

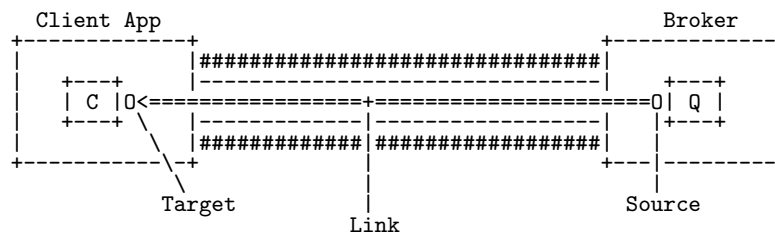


Figure 2.6: An AMQP Link

Sessions provide the context for communication between sources and targets. A *link endpoint* associates a terminus with a *session endpoint*. Within a session, the link protocol (defined in section 2.6) is used to establish links between sources and targets and to transfer messages across them. A single session can be simultaneously associated with any number of links.

Links are named, and the state at the termini can live longer than the connection on which they were established.



Figure 2.7: Connection Loss

The retained state at the termini can be used to reestablish the link on a new connection (and session) with precise control over delivery guarantees (e.g., ensuring “exactly once” delivery).

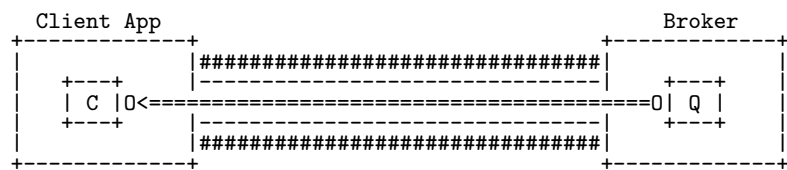


Figure 2.8: Link Recovery

The diagram below shows the relationship between the three AMQP communication endpoints, Connection, Session and Link.

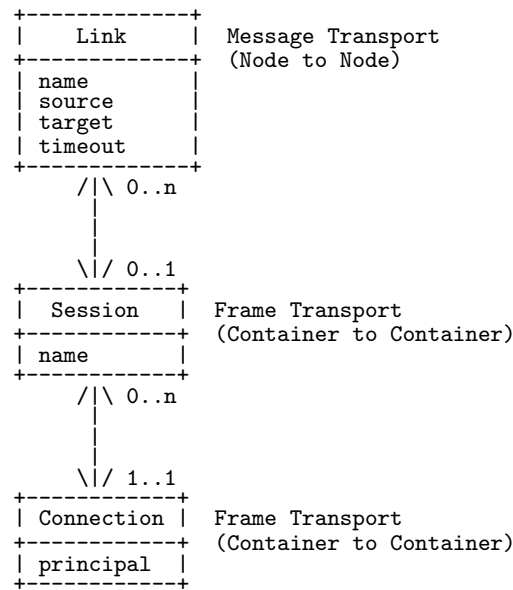


Figure 2.9: Class Diagram of Communication Endpoints

2.1.3 Protocol Frames

The protocol consists of nine frame body types that are formally defined in section 2.7. The following table lists the frame bodies and defines which endpoints handle them.