Compatibility and Conformance

The AMQP specification is open and free to all users and implementers. RabbitMQ implements version 0-9-1 of the specification today, with legacy support for version 0-8 and 0-9. RabbitMQ will perform protocol negotiation with clients implementing 0-9-1, 0-9 and 0-8, in accordance with the specification. These three versions are extremely similar. The 0-9-1 specification clarifies some of the definitions of version 0-8 and 0-9 and makes it easier for multiple implementations to interoperate. RabbitMQ implements a number of **extensions** to the specification.

RabbitMQ implements AMQP 1.0 via an experimental **plugin**. However, AMQP 1.0 is a completely different protocol than AMQP 0-9-1 and hence not a suitable replacement for the latter. RabbitMQ will therefore continue to support AMQP 0-9-1 (and its close relatives 0-9 and 0-8) indefinitely. RabbitMQ is unlikely ever to support AMQP 0-10.

The 0-9-1 (with and without extensions), 0-9 and 0-8 specifications are linked to below for your convenience. We recommend reading them if you want to learn more about AMQP. Please see our **documentation** and, in particular, our **AMQP 0-9-1 Reference Guide** for more information. Or, you can visit **amqp.org**.

Protocol Version	Documentation (PDF)	Machine-Readable Spec (XML)
AMQP 0-9-1 (incl. extensions)		**Full** | **BSD-licensed**
AMQP 0-9-1	**Specification** | **Generated Doc**	**Full** | **BSD-licensed**
AMQP 0-9	**Specification** | **Generated Doc**	**Full** | **BSD-licensed**
AMQP 0-8	**Specification** | **Generated Doc**	**Full** | **BSD-licensed**

Specification versions supported

The following table describes the version of the AMQP protocol specification implemented by RabbitMQ release 2.0.0 and later:

Component	Implements AMQP protocol version
Server	0-9-1, 0-9, 0-8
Java client	0-9-1
.NET/C# client	0-9-1, 0-9 and 0-8 switchable at runtime
Erlang client	0-9-1

The following table describes the version of the AMQP protocol specification implemented by RabbitMQ release prior to 2.0.0:

Component	Implements AMQP protocol version
Server	0-8
Java client	0-8
.NET/C# client	0-9 and 0-8 switchable at runtime

Differences between versions

Please see the **0-8 to 0-9-1 page**.

Interoperability

Please see the **interoperability page**.

Deprecated classes

The following classes were deprecated in version 0-9-1. RabbitMQ does not implement these classes at all, including when the broker is connected to a version 0-8 client, or when the .NET client is configured to use 0-8.

- access
- dtx
- file
- stream
- test
- tunnel

See also the detailed specification compatibility tables below.

Classes from the AMQP specification, version 0-9-1

The following table describes the current implementation status of the various AMQP protocol message classes.

Current Status	Class	Notes
ok	connection	

ok	channel	
ok	exchange	
ok	queue	
ok	basic	
partial	tx	See **notes on tx support**

Methods from the AMQP specification, version 0-9-1

The following table describes the current implementation status of the various AMQP protocol methods in each class.

Current Status	Method	Notes
ok	connection.start	
ok	connection.start-ok	
ok	connection.secure	
ok	connection.secure-ok	
ok	connection.tune	
ok	connection.tune-ok	
ok	connection.open	
ok	connection.open-ok	
ok	connection.close	
ok	connection.close-ok	
ok	channel.open	
ok	channel.open-ok	
partial	channel.flow	active=false is not supported by the server. Limiting prefetch with `basic.qos` provides much better control.
ok	channel.flow-ok	
ok	channel.close	
ok	channel.close-ok	
ok	exchange.declare	
ok	exchange.declare-ok	
partial	exchange.delete	We have made exchange.delete into an idempotent assertion that the exchange must not exist, in the same way that exchange.declare asserts that it must.
ok	exchange.delete-ok	
ok	queue.declare	
ok	queue.declare-ok	The consumer-count parameter is the count of all consumers, rather than only active consumers, as mandated by the specification. The former is more useful to applications.
ok	queue.bind	
ok	queue.bind-ok	
partial	queue.unbind	We have made queue.unbind into an idempotent assertion that the binding must not exist, in the same way that queue.bind asserts that it must.
ok	queue.unbind-ok	
ok	queue.purge	
ok	queue.purge-ok	
partial	queue.delete	We have made queue.delete into an idempotent assertion that the queue must not exist, in the same way that queue.declare asserts that it must.
ok	queue.delete-ok	
partial	basic.qos	The server supports per-consumer and per-channel limits. The `global` flag is given different semantics from those in the specification. See **consumer prefetch** for more information. Prefetch size limits are not implemented.
ok	basic.qos-ok	
partial	basic.consume	The no-local parameter is not implemented. The value of this parameter is ignored and no attempt is made to prevent a consumer from receiving messages that were published on the same connection.
ok	basic.consume-ok	
ok	basic.cancel	
ok	basic.cancel-ok	
ok	basic.publish	
ok	basic.return	
ok	basic.deliver	

ok	basic.get	
ok	basic.get-ok	
ok	basic.get-empty	
ok	basic.ack	
partial	basic.reject	The server discards the message when requeue=false, and requeues it when requeue=true. No attempt is made to prevent redelivery to the same client. The server does not interrupt the sending of message content of a rejected message, i.e. the message is always delivered in full to the client.
partial	basic.recover	Recovery with requeue=false is not supported.
ok	tx.select	
ok	tx.select-ok	
ok	tx.commit	
ok	tx.commit-ok	
ok	tx.rollback	
ok	tx.rollback-ok	

Rules from the AMQP specification, version 0-9-1

The Reference column contains the class or domain, method, field and rule name where present.

Current Status	Type	Actor	Reference	Text
ok	MUST NOT		delivery-tag / channel-local	The delivery tag is valid only within the channel from which the message was received. I.e. a client MUST NOT receive a message on one channel and then acknowledge it on another.
ok	MUST NOT		delivery-tag / non-zero	The server MUST NOT use a zero value for delivery tags. Zero is reserved for client use, meaning "all messages so far received".
does	SHOULD		redelivered / implementation	The server SHOULD try to signal redelivered messages when it can. When redelivering a message that was not successfully acknowledged, the server SHOULD deliver it to the original client if possible.
planned	MUST NOT		redelivered / hinting	The client MUST NOT rely on the redelivered field but should take it as a hint that the message may already have been processed. A fully robust client must be able to track duplicate received messages on non-transacted, and locally-transacted channels. ***Notes:*** *The client already conforms, in that it does not rely on the redelivered field, and we plan on adding duplicate tracking in a future release.*
ok	MUST	client	connection / start / protocol-name	If the server cannot support the protocol specified in the protocol header, it MUST respond with a valid protocol header and then close the socket connection.
ok	MUST	client	connection / start / server-support	The server MUST provide a protocol version that is lower than or equal to that requested by the client in the protocol header.
ok	MUST	client	connection / start / client-support	If the client cannot handle the protocol version suggested by the server it MUST close the socket connection without sending any further data.
does	SHOULD	client	connection / start / server-properties / required-fields	The properties SHOULD contain at least these fields: "host", specifying the server host name or address, "product", giving the name of the server product, "version", giving the name of the server version, "platform", giving the name of the operating system, "copyright", if appropriate, and "information", giving other general information.
ok	MUST	client	connection / start / locales / required-support	The server MUST support at least the en_US locale.
does	SHOULD	server	connection / start-ok / client-properties / required-fields	The properties SHOULD contain at least these fields: "product", giving the name of the client product, "version", giving the name of the client version, "platform", giving the name of the operating system, "copyright", if appropriate, and "information", giving other general information.
ok	SHOULD	server	connection / start-ok / mechanism / security	The client SHOULD authenticate using the highest-level security profile it can handle from the list provided by the server.
ok	MUST	server	connection / start-ok / mechanism / validity	If the mechanism field does not contain one of the security mechanisms proposed by the server in the Start method, the server MUST close the connection without sending any further data.
ok	MUST	client	connection /	Until the frame-max has been negotiated, both peers MUST accept

			tune / frame-max / minimum	frames of up to frame-min-size octets large, and the minimum negotiated value for frame-max is also frame-min-size.
planned	MUST	server	connection / tune-ok / channel-max / upper-limit	If the client specifies a channel max that is higher than the value provided by the server, the server MUST close the connection without attempting a negotiated close. The server may report the error in some fashion to assist implementors.
ok	MUST	server	connection / tune-ok / frame-max / minimum	Until the frame-max has been negotiated, both peers MUST accept frames of up to frame-min-size octets large, and the minimum negotiated value for frame-max is also frame-min-size. ***Notes:*** *frame-min-size is 4Kb.*
ok	MUST	server	connection / tune-ok / frame-max / upper-limit	If the client specifies a frame max that is higher than the value provided by the server, the server MUST close the connection without attempting a negotiated close. The server may report the error in some fashion to assist implementors.
ok	MUST	server	connection / open / virtual-host / separation	If the server supports multiple virtual hosts, it MUST enforce a full separation of exchanges, queues, and all associated entities per virtual host. An application, connected to a specific virtual host, MUST NOT be able to access resources of another virtual host.
does	SHOULD	server	connection / open / virtual-host / security	The server SHOULD verify that the client has permission to access the specified virtual host.
ok	MUST	client	connection / close / stability	After sending this method, any received methods except Close and Close-OK MUST be discarded. The response to receiving a Close after sending Close must be to send Close-Ok.
does	SHOULD	client	connection / close-ok / reporting	A peer that detects a socket closure without having received a Close-Ok handshake method SHOULD log the error. ***Notes:*** *Only the server maintains an error log.*
ok	MUST NOT	server	channel / open / state	The client MUST NOT use this method on an already-opened channel.
doesn't	MAY	server	channel / flow / initial-state	When a new channel is opened, it is active (flow is active). Some applications assume that channels are inactive until started. To emulate this behaviour a client MAY open the channel, then pause it.
doesn't	SHOULD	server	channel / flow / bidirectional	When sending content frames, a peer SHOULD monitor the channel for incoming methods and respond to a Channel.Flow as rapidly as possible. ***Notes:*** *The server does not support blocking flow with active=true. Limiting prefetch with* `basic.qos` *provides much better control.*
ok	MAY	server	channel / flow / throttling	A peer MAY use the Channel.Flow method to throttle incoming content data for internal reasons, for example, when exchanging data over a slower connection. ***Notes:*** *Neither the server or the clients automatically issue a Channel.Flow.*
doesn't	MAY	server	channel / flow / expected-behaviour	The peer that requests a Channel.Flow method MAY disconnect and/or ban a peer that does not respect the request. This is to prevent badly-behaved clients from overwhelming a server.
ok	MUST	client	channel / close / stability	After sending this method, any received methods except Close and Close-OK MUST be discarded. The response to receiving a Close after sending Close must be to send Close-Ok.
does	SHOULD	client	channel / close-ok / reporting	A peer that detects a socket closure without having received a Channel.Close-Ok handshake method SHOULD log the error. ***Notes:*** *Only the server maintains an error log.*
ok	MUST		exchange / required-types	The server MUST implement these standard exchange types: fanout, direct.
does	SHOULD		exchange / recommended-types	The server SHOULD implement these standard exchange types: topic, headers.
ok	MUST		exchange / required-instances	The server MUST, in each virtual host, pre-declare an exchange instance for each standard exchange type that it implements, where the name of the exchange instance, if defined, is "amq." followed by the exchange type name.
ok	MUST		exchange / default-exchange	The server MUST pre-declare a direct exchange with no public name to act as the default exchange for content Publish methods and for default queue bindings.
ok	MUST NOT		exchange / default-access	The server MUST NOT allow clients to access the default exchange except by specifying an empty exchange name in the Queue.Bind and content Publish methods.
does	MAY		exchange / extensions	The server MAY implement other exchange types as wanted.
does	SHOULD	server	exchange / declare / minimum	The server SHOULD support a minimum of 16 exchanges per virtual host and ideally, impose no limit except as defined by available resources.
ok	MUST,	server	exchange /	Exchange names starting with "amq." are reserved for pre-declared and

	MAY		declare / exchange / reserved	standardised exchanges. The client MAY declare an exchange starting with "amq." if the passive option is set, or the exchange already exists. ***Notes:*** *The server does not prevent exchange names starting with "amq." from being declared. Clients may declare exchanges starting with "amq." without the passive bit set.*
ok	MUST	server	exchange / declare / exchange / syntax	The exchange name consists of a non-empty sequence of these characters: letters, digits, hyphen, underscore, period, or colon. ***Notes:*** *The lexicon is not enforced by the server.*
ok	MUST	server	exchange / declare / type / typed	Exchanges cannot be redeclared with different types. The client MUST not attempt to redeclare an existing exchange with a different type than used in the original Exchange.Declare method.
ok	MUST NOT	server	exchange / declare / type / support	The client MUST NOT attempt to declare an exchange with a type that the server does not support.
ok	MUST	server	exchange / declare / passive / not-found	If set, and the exchange does not already exist, the server MUST raise a channel exception with reply code 404 (not found). ***Notes:*** *(refers to the passive flag)*
ok	MUST	server	exchange / declare / passive / equivalent	If not set and the exchange exists, the server MUST check that the existing exchange has the same values for type, durable, and arguments fields. The server MUST respond with Declare-Ok if the requested exchange matches these fields, and MUST raise a channel exception if not. ***Notes:*** *(refers to the passive flag)*
ok	MUST	server	exchange / declare / durable / support	The server MUST support both durable and transient exchanges.
failing	MUST NOT	server	exchange / delete / exchange / exists	The client MUST NOT attempt to delete an exchange that does not exist. ***Notes:*** *We have made exchange.delete into an idempotent assertion that the exchange must not exist, in the same way that exchange.declare asserts that it must.*
ok	MUST NOT	server	exchange / delete / if-unused / in-use	The server MUST NOT delete an exchange that has bindings on it, if the if-unused field is true.
ok	MUST	server	queue / declare / default-binding	The server MUST create a default binding for a newly-declared queue to the default exchange, which is an exchange of type 'direct' and use the queue name as the routing key.
does	SHOULD	server	queue / declare / minimum-queues	The server SHOULD support a minimum of 256 queues per virtual host and ideally, impose no limit except as defined by available resources.
does	MAY, MUST	server	queue / declare / queue / default-name	The queue name MAY be empty, in which case the server MUST create a new queue with a unique generated name and return this to the client in the Declare-Ok method.
ok	MUST	server	queue / declare / queue / reserved	Queue names starting with "amq." are reserved for pre-declared and standardised queues. The client MAY declare a queue starting with "amq." if the passive option is set, or the queue already exists. ***Notes:*** *The server does not prevent queue names starting with "amq." from being declared. Clients may declare queues starting with "amq." without the passive bit set.*
does	MAY	server	queue / declare / queue / syntax	The queue name can be empty, or a sequence of these characters: letters, digits, hyphen, underscore, period, or colon. ***Notes:*** *The lexicon is not enforced by the server.*
does	MAY	server	queue / declare / passive / passive	The client MAY ask the server to assert that a queue exists without creating the queue if not. If the queue does not exist, the server treats this as a failure.
ok	MUST	server	queue / declare / passive / equivalent	If not set and the queue exists, the server MUST check that the existing queue has the same values for durable, exclusive, auto-delete, and arguments fields. The server MUST respond with Declare-Ok if the requested queue matches these fields, and MUST raise a channel exception if not. ***Notes:*** *(refers to the passive flag)*
ok	MUST	server	queue / declare / durable / persistence	The server MUST recreate the durable queue after a restart.
ok	MUST	server	queue / declare / durable / types	The server MUST support both durable and transient queues.
ok	MUST	server	queue / declare / exclusive / types	The server MUST support both exclusive (private) and non-exclusive (shared) queues.

doesn't	MAY NOT	server	queue / declare / exclusive / exclusive	The client MAY NOT attempt to use a queue that was declared as exclusive by another still-open connection.
ok	MUST	server	queue / declare / auto-delete / pre-existence	The server MUST ignore the auto-delete field if the queue already exists.
ok	MUST	server	queue / bind / duplicates	A server MUST allow ignore duplicate bindings - that is, two or more bind methods for a specific queue, with identical arguments - without treating these as an error.
ok	MUST NOT	server	queue / bind / unique	A server MUST not deliver the same message more than once to a queue, even if the queue has multiple bindings that match the message.
ok	MUST	server	queue / bind / transient-exchange	The server MUST allow a durable queue to bind to a transient exchange.
ok	MUST	server	queue / bind / durable-exchange	Bindings of durable queues to durable exchanges are automatically durable and the server MUST restore such bindings after a server restart.
does	SHOULD	server	queue / bind / binding-count	The server SHOULD support at least 4 bindings per queue, and ideally, impose no limit except as defined by available resources.
ok	MUST	server	queue / bind / queue / queue-known	The client MUST either specify a queue name or have previously declared a queue on the same channel
ok	MUST NOT	server	queue / bind / queue / must-exist	The client MUST NOT attempt to bind a queue that does not exist.
ok	MUST NOT	server	queue / bind / exchange / exchange-existence	A client MUST NOT be allowed to bind a queue to a non-existent exchange.
ok	MUST	server	queue / bind / exchange / default-exchange	The server MUST accept a blank exchange name to mean the default exchange.
ok	MUST	server	queue / bind / routing-key / direct-exchange-key-matching	If a message queue binds to a direct exchange using routing key K and a publisher sends the exchange a message with routing key R, then the message MUST be passed to the message queue if K = R.
ok	MUST	server	queue / unbind / 01	If a unbind fails, the server MUST raise a connection exception.
ok	MUST	server	queue / unbind / queue / queue-known	The client MUST either specify a queue name or have previously declared a queue on the same channel
failing	MUST NOT	server	queue / unbind / queue / must-exist	The client MUST NOT attempt to unbind a queue that does not exist. ***Notes:*** *We have made queue.unbind into an idempotent assertion that the binding must not exist, in the same way that queue.bind asserts that it must.*
failing	MUST NOT	server	queue / unbind / exchange / must-exist	The client MUST NOT attempt to unbind a queue from an exchange that does not exist. ***Notes:*** *We have made queue.unbind into an idempotent assertion that the binding must not exist, in the same way that queue.bind asserts that it must.*
ok	MUST	server	queue / unbind / exchange / default-exchange	The server MUST accept a blank exchange name to mean the default exchange.
ok	MUST NOT	server	queue / purge / 02	The server MUST NOT purge messages that have already been sent to a client but not yet acknowledged.
doesn't	MAY	server	queue / purge / 03	The server MAY implement a purge queue or log that allows system administrators to recover accidentally-purged messages. The server SHOULD NOT keep purged messages in the same storage spaces as the live messages since the volumes of purged messages may get very large.
ok	MUST	server	queue / purge / queue / queue-known	The client MUST either specify a queue name or have previously declared a queue on the same channel
ok	MUST NOT	server	queue / purge / queue / must-exist	The client MUST NOT attempt to purge a queue that does not exist.
doesn't	SHOULD	server	queue / delete / 01	The server SHOULD use a dead-letter queue to hold messages that were pending on a deleted queue, and MAY provide facilities for a system

				administrator to move these messages back to an active queue.
ok	MUST	server	queue / delete / queue / queue-known	The client MUST either specify a queue name or have previously declared a queue on the same channel
failing	MUST NOT	server	queue / delete / queue / must-exist	The client MUST NOT attempt to delete a queue that does not exist. ***Notes:*** *We have made queue.delete into an idempotent assertion that the queue must not exist, in the same way that queue.declare asserts that it must.*
ok	MUST NOT	server	queue / delete / if-unused / in-use	The server MUST NOT delete a queue that has consumers on it, if the if-unused field is true.
ok	MUST NOT	server	queue / delete / if-empty / not-empty	The server MUST NOT delete a queue that has messages on it, if the if-empty field is true.
does	SHOULD		basic / 01	The server SHOULD respect the persistent property of basic messages and SHOULD make a best-effort to hold persistent basic messages on a reliable storage mechanism.
ok	MUST NOT		basic / 02	The server MUST NOT discard a persistent basic message in case of a queue overflow.
doesn't	MAY		basic / 03	The server MAY use the Channel.Flow method to slow or stop a basic message publisher when necessary.
does	MAY		basic / 04	The server MAY overflow non-persistent basic messages to persistent storage.
doesn't	MAY		basic / 05	The server MAY discard or dead-letter non-persistent basic messages on a priority basis if the queue size exceeds some configured limit.
planned	MUST		basic / 06	The server MUST implement at least 2 priority levels for basic messages, where priorities 0-4 and 5-9 are treated as two distinct levels.
doesn't	MAY		basic / 07	The server MAY implement up to 10 priority levels.
ok	MUST		basic / 08	The server MUST deliver messages of the same priority in order irrespective of their individual persistence.
ok	MUST		basic / 09	The server MUST support un-acknowledged delivery of Basic content, i.e. consumers with the no-ack field set to TRUE.
ok	MUST		basic / 10	The server MUST support explicitly acknowledged delivery of Basic content, i.e. consumers with the no-ack field set to FALSE.
ok	MUST	server	basic / qos / prefetch-size / 01	The server MUST ignore this setting when the client is not processing any messages - i.e. the prefetch size does not limit the transfer of single messages to a client, only the sending in advance of more messages while the client still has one or more unacknowledged messages. ***Notes:*** *(refers to prefetch-size)*
ok	MUST NOT	server	basic / qos / prefetch-count / 01	The server may send less data in advance than allowed by the client's specified prefetch windows but it MUST NOT send more.
does	SHOULD	server	basic / consume / 01	The server SHOULD support at least 16 consumers per queue, and ideally, impose no limit except as defined by available resources.
ok	MUST NOT	server	basic / consume / consumer-tag / 01	The client MUST NOT specify a tag that refers to an existing consumer.
ok	MUST	server	basic / consume / consumer-tag / 02	The consumer tag is valid only within the channel from which the consumer was created. I.e. a client MUST NOT create a consumer in one channel and then use it in another.
doesn't	MAY NOT	server	basic / consume / exclusive / 01	The client MAY NOT gain exclusive access to a queue that already has active consumers.
ok	MUST	server	basic / cancel / 01	If the queue does not exist the server MUST ignore the cancel method, so long as the consumer tag is valid for that channel.
ok	MUST NOT	server	basic / publish / exchange / must-exist	The client MUST NOT attempt to publish a content to an exchange that does not exist.
ok	MUST	server	basic / publish / exchange / default-exchange	The server MUST accept a blank exchange name to mean the default exchange.
ok	MUST	server	basic / publish / exchange / 02	If the exchange was declared as an internal exchange, the server MUST raise a channel exception with a reply code 403 (access refused).
doesn't	MAY	server	basic / publish / exchange / 03	The exchange MAY refuse basic content in which case it MUST raise a channel exception with reply code 540 (not implemented).
does	SHOULD	server	basic / publish /	The server SHOULD implement the mandatory flag.

			mandatory / 01	
doesn't	SHOULD	server	basic / publish / immediate / 01	The server SHOULD implement the immediate flag. ***Notes:*** *The server does not support the immediate flag.* ***message TTLs*** *of 0 offer an alternative.*
planned	SHOULD	client	basic / deliver / 01	The server SHOULD track the number of times a message has been delivered to clients and when a message is redelivered a certain number of times - e.g. 5 times - without being acknowledged, the server SHOULD consider the message to be unprocessable (possibly causing client applications to abort), and move the message to a dead letter queue.
ok	MUST	server	basic / ack / multiple / exists	The server MUST validate that a non-zero delivery-tag refers to a delivered message, and raise a channel exception if this is not the case. On a transacted channel, this check MUST be done immediately and not delayed until a Tx.Commit. Specifically, a client MUST not acknowledge the same message more than once.
planned	SHOULD	server	basic / reject / 01	The server SHOULD be capable of accepting and process the Reject method while sending message content with a Deliver or Get-Ok method. I.e. the server should read and process incoming methods while sending output frames. To cancel a partially-send content, the server sends a content body frame of size 1 (i.e. with no data except the frame-end octet).
ok	SHOULD	server	basic / reject / 02	The server SHOULD interpret this method as meaning that the client is unable to process the message at this time.
ok	MUST NOT	server	basic / reject / 03	The client MUST NOT use this method as a means of selecting messages to process.
planned	MUST NOT	server	basic / reject / requeue / 01	The server MUST NOT deliver the message to the same client within the context of the current channel. The recommended strategy is to attempt to deliver the message to an alternative consumer, and if that is not possible, to move the message to a dead-letter queue. The server MAY use more sophisticated tracking to hold the message on the queue and redeliver it to the same client at a later stage.
ok	MUST	server	basic / recover-async / 01	The server MUST set the redelivered flag on all messages that are resent.
ok	MUST	server	basic / recover / 01	The server MUST set the redelivered flag on all messages that are resent.
ok	MUST NOT		tx / not multiple queues	Applications MUST NOT rely on the atomicity of transactions that affect more than one queue.
ok	MUST NOT		tx / not immediate	Applications MUST NOT rely on the behaviour of transactions that include messages published with the immediate option.
ok	MUST NOT		tx / not mandatory	Applications MUST NOT rely on the behaviour of transactions that include messages published with the mandatory option.
ok	MUST NOT	server	tx / commit / transacted	The client MUST NOT use the Commit method on non-transacted channels.
ok	MUST NOT	server	tx / rollback / transacted	The client MUST NOT use the Rollback method on non-transacted channels.

Rules from the AMQP specification, version 0-9-1 (PDF)

The rules listed below are from the PDF version of the 0-9-1 specification, wherever MUST, SHOULD or MAY appear in the text.

Current Status	Type	Actor	Reference	Text
does, doesn't	SHOULD, SHOULD		1.4.1	Protocol constants are shown as upper-case names. AMQP implementations SHOULD use these names when defining and using constants in source code and documentation. Property names, method arguments, and frame fields are shown as lower-case names. AMQP implementations SHOULD use these names consistently in source code and documentation. ***Notes:*** *Property names, method arguments, and frame fields legitimately appear as camel-case in some contexts.*
ok	MUST		2.2.4	There is no hand-shaking for errors on connections that are not fully open. Following successful protocol header negotiation, [...] and prior to sending or receiving Open or Open-Ok, a peer that detects an error MUST close the socket without sending any further data.
does	MAY		2.3.3	The server MAY host multiple protocols on the same port. ***Notes:*** *The server supports 0-8, 0-9 and 0-9-1 on the same port.*
doesn't	MAY		2.3.3	Agreed limits MAY enable both parties to pre-allocate key buffers, avoiding deadlocks.
ok	MUST		2.3.3	Every incoming frame either obeys the agreed limits, and so is "safe", or exceeds them, in which case the other party IS faulty and MUST be disconnected.
ok	MUST		2.3.3	The server MUST tell the client what limits it proposes.

doesn't	MAY		2.3.3	The client responds and MAY reduce those limits for its connection.
does	MAY		2.3.5.2	The data [for content frames] can be any size, and MAY be broken into several (or many) chunks, each forming a "content body frame".
ok	MUST, MUST		2.3.7	A connection or channel is considered “open” for the client when it has sent Open, and for the server when it has sent Open-Ok. From this point onwards a peer that wishes to close the channel or connection MUST do so using a hand-shake protocol [...]. When a peer decides to close a channel or connection, it sends a Close method. The receiving peer MUST respond to a Close with a Close-Ok, and then both parties can close their channel or connection.
does	MUST NOT, MUST NOT		3.1.1	The server MUST NOT modify message content bodies that it receives and passes to consumer applications. [...] [The server] MUST NOT remove or modify existing information. ***Notes:*** *"BCC" headers are removed from properties after routing.*
doesn't	MAY		3.1.1	The server MAY add information to content headers [...]
ok	MUST		3.1.2	Each connection MUST BE associated with a single virtual host.
doesn't	MAY		3.1.2	[The] authorization scheme used MAY be unique to each virtual host.
ok	MUST, MUST		3.1.3.1	The server MUST implement the direct exchange type and MUST pre-declare within each virtual host at least two direct exchanges: one named amq.direct, and one with no public name that serves as the default exchange for Publish methods. [...] [All] message queues MUST BE automatically bound to the nameless exchange using the message queue's name as routing key.
ok	MUST		3.1.3.3	The routing key used for a topic exchange MUST consist of zero or more words delimited by dots.
does, ok	SHOULD, MUST		3.1.3.3	The server SHOULD implement the topic exchange type and in that case, the server MUST pre-declare within each virtual host at least one topic exchange, named amq.topic.
does, ok	SHOULD, MUST		3.1.3.4	The server SHOULD implement the headers exchange type and in that case, the server MUST pre-declare within each virtual host at least one headers exchange, named amq.match.
ok	MUST		3.1.3.4	All non-normative exchange types MUST be named starting with "x-".
does	MAY NOT		3.1.4	Note that in the presence of multiple readers from a queue, or client transactions, or use of priority fields, or use of message selectors, or implementation-specific delivery optimisations the queue MAY NOT exhibit true FIFO characteristics. ***Notes:*** *FIFO characteristics are guaranteed under the conditions specified in section 4.7.*
ok	MUST		3.1.10	The server and client MUST respect [the specified naming] conventions
does	SHOULD		3.2.1	The AMQP methods may define specific minimal values (such as numbers of consumers per message queue) for interoperability reasons. These minima are defined in the description of each class. Conforming AMQP implementations SHOULD implement reasonably generous values for such fields, the minima is only intended for use on the least capable platforms.
does, doesn't	SHOULD, MAY		3.2.1	The sending peer SHOULD wait for the specific reply method [after sending a synchronous request], but MAY implement this asynchronously
ok	MUST		4.2.2	The client MUST start a new connection by sending a protocol header.
doesn't	MAY		4.2.2	The server MAY accept non-AMQP protocols such as HTTP. ***Notes:*** *The core broker accepts only AMQP. Plugins exist for other protocols.*
ok	MUST		4.2.2	If the server does not recognise the first 5 octets of data on the socket, or does not support the specific protocol version that the client requests, it MUST write a valid protocol header to the socket, then flush the socket (to ensure the client application will receive the data) and then close the socket connection.
does	MAY		4.2.2	The server MAY print a diagnostic message [during failed protocol negotiation] to assist debugging. ***Notes:*** *Relevant information will be written to the server log.*
doesn't	MAY		4.2.2	The client MAY detect the server protocol version by attempting to connect with its highest supported version and reconnecting with a lower version if it receives such information back from the server.
ok	MUST		4.2.3	The frame-end octet MUST always be the hexadecimal value %xCE.
ok	MUST, MUST, MUST		4.2.3	If a peer receives a frame with a type that is not one of these defined types, it MUST treat this as a fatal protocol error and close the connection without sending any further data on it. When a peer reads a frame it MUST check that the frame-end is valid before attempting to decode the frame. If the frame-end is not valid it MUST treat this as a fatal protocol error and close the connection without sending any further data on it.
does	SHOULD		4.2.3	It SHOULD log information about the [frame decoding] problem, since this indicates an error in either the server or client framing code implementation.
ok	MUST NOT, MUST, MUST,		4.2.3	A peer MUST NOT send frames larger than the agreed-upon size. A peer that receives an oversized frame MUST signal a connection exception with reply code 501 (frame error). The channel number MUST be zero for all heartbeat frames, and for method, header and body frames that refer to the Connection class. A

	MUST, MUST			peer that receives a non-zero channel number for one of these frames MUST signal a connection exception with reply code 503 (command invalid).
ok	MUST NOT		4.2.5.1	Implementers MUST NOT assume that integers encoded in a frame are aligned on memory word boundaries.
ok	MUST		4.2.5.5	Field names MUST start with a letter, '$' or '#' [...]
doesn't, doesn't	SHOULD, SHOULD		4.2.5.5	The server SHOULD validate field names and upon receiving an invalid field name, it SHOULD signal a connection exception with reply code 503 (syntax error).
planned	MUST		4.2.6	A peer that receives an incomplete or badly-formatted content MUST raise a connection exception with reply code 505 (unexpected frame). ***Notes:*** *The given reply code is not returned for all possible ways in which content can be badly-formatted.*
ok	MUST, MUST		4.2.6.1	The class-id [of the content header] MUST match the method frame class id. The peer MUST respond to an invalid class-id by raising a connection exception with reply code 501 (frame error).
ok	MUST NOT, MUST		4.2.6.1	The channel number in content frames MUST NOT be zero. A peer that receives a zero channel number in a content frame MUST signal a connection exception with reply code 504 (channel error).
ok	MUST		4.2.6.2	A peer MUST handle a content body that is split into multiple frames by storing these frames as a single set, and either retransmitting them as-is, broken into smaller frames, or concatenated into a single block for delivery to an application.
ok	MUST		4.2.6.2	Heartbeat frames MUST have a channel number of zero.
ok	MUST		4.2.7	If the peer does not support heartbeating it MUST discard the heartbeat frame without signalling any error or fault. ***Notes:*** *The broker and supported clients do support heartbeat frames.*
does, doesn't	MAY, MAY		4.3	An AMQP peer MAY support multiple channels. The maximum number of channels is defined at connection negotiation, and a peer MAY negotiate this down to 1.
does, doesn't	SHOULD, SHOULD NOT		4.3	Each peer SHOULD balance the traffic on all open channels in a fair fashion. This balancing can be done on a per-frame basis, or on the basis of amount of traffic per channel. A peer SHOULD NOT allow one very busy channel to starve the progress of a less busy channel.
ok	MUST NOT, MUST		4.6	The effects of the request-response MUST NOT be visible on the channel before the response method, and MUST be visible thereafter.
ok	MUST		4.7	The server MUST preserve the order of contents flowing through a single content processing path, unless the redelivered field is set on the Basic.Deliver or Basic.Get-Ok methods, and according to the rules governing the conditions under which that field can be set. ***Notes:*** *The broker makes **stronger guarantees**.*
does	SHOULD		4.10.2	The server SHOULD log all [exceptions during connection negotiation stage] and flag or block clients provoking multiple failures.

Rules from the AMQP specification, version 0-8

The rules listed below relate to features supported by RabbitMQ that have been deprecated.

Current Status	Type	Actor	Reference	Text
doesn't	SHOULD	client		method/connection/redirect: When getting the Connection.Redirect method, the client SHOULD reconnect to the host specified, and if that host is not present, to any of the hosts specified in the known-hosts list.
doesn't	SHOULD	client		method/connection/redirect: When getting the Connection.Redirect method, the client SHOULD reconnect to the host specified, and if that host is not present, to any of the hosts specified in the known-hosts list.
does	MAY	server		domain/known hosts: The server MAY leave this field empty if it knows of no other hosts than itself.