Table of Contents

1. AMQP version 0-9 specification

This document was automatically generated from the AMQP XML specification. All edits to the content of this file should be directed to the XML file (for content) or the XSLT template (for layout and/or formatting).

1.1. AMQP-defined Domains

The following domains are defined in this specification:

Name	Type	[Label] Description
access-ticket	short	[access ticket granted by server] An access ticket granted by the server for a certain set of access rights within a specific realm. Access tickets are valid within the channel where they were created, and expire when the channel closes.
bit	bit	[single bit]
channel-id	longstr	[unique identifier for a channel]
class-id	short	
consumer-tag	shortstr	[consumer tag] Identifier for the consumer, valid within the current connection.
delivery-tag	longlong	[server-assigned delivery tag] The server-assigned and channel-specific delivery tag
destination	shortstr	[destination for a message] Specifies the destination to which the message is to be transferred. The destination can be empty, meaning the default exchange or consumer.
duration	longlong	[duration in milliseconds]
exchange-name	shortstr	[exchange name] The exchange name is a client-selected string that identifies the exchange for publish methods. Exchange names may consist of any mixture of digits, letters, and underscores. Exchange names are scoped by the virtual host.
known-hosts	shortstr	[list of known hosts] Specifies the list of equivalent or alternative hosts that the server knows about, which will normally include the current server itself. Clients can cache this information and use it when reconnecting to a server after a failure. This field may be empty.
long	long	[32-bit integer]
longlong	longlong	[64-bit integer]
longstr	longstr	[long string]
method-id	short	
no-ack	bit	[no acknowledgement needed] If this field is set the server does not expect acknowledgements for messages. That is, when a message is delivered to the client the server automatically and silently acknowledges it on behalf of the client. This functionality increases performance but at the cost of reliability. Messages can get lost if a client dies before it can deliver them to the application.

Name	Type	[Label] Description
no-local	bit	[do not deliver own messages] If the no-local field is set the server will not send messages to the connection that published them.
octet	octet	[single octet]
offset	longlong	[offset into a message body]
path	shortstr	Must start with a slash "/" and continue with path names separated by slashes. A path name consists of any combination of at least one of [A-Za-z0-9] plus zero or more of [.-_+!=:].
peer-properties	table	This string provides a set of peer properties, used for identification, debugging, and general information.
queue-name	shortstr	[queue name] The queue name identifies the queue within the vhost. Queue names may consist of any mixture of digits, letters, and underscores.
redelivered	bit	[message is being redelivered] This indicates that the message has been previously delivered to this or another client.
reference	longstr	[pointer to a message body]
reject-code	short	[reject code for transfer]
reject-text	shortstr	[informational text for message reject]
reply-code	short	[reply code from server] The reply code. The AMQ reply codes are defined as constants at the start of this formal specification.
reply-text	shortstr	[localised reply text] The localised reply text. This text can be logged as an aid to resolving issues.
security-token	longstr	[security token] Used for authentication, replay prevention, and encrypted bodies.
short	short	[16-bit integer]
shortstr	shortstr	[short string]
table	table	[field table]
timestamp	timestamp	[64-bit timestamp]

1.2. AMQP-defined Constants

Many constants are error codes. Where this is so, they will fall into one of two categories:

- **Channel Errors:** These codes are all associated with failures that affect the current channel but not other channels in the same connection;
- **Connection Errors:** These codes are all associated with failures that preclude any further activity on the connection and require its closing.

The following constants are defined in the specification:

Name	Value	Error type	[Label] Description
frame-method	1		
frame-header	2		
frame-body	3		
frame-oob-method	4		
frame-oob-header	5		
frame-oob-body	6		
frame-trace	7		
frame-heartbeat	8		
frame-min-size	4096		
frame-end	206		
reply-success	200		Indicates that the method completed successfully. This reply code is reserved for future use - the current protocol design does not use positive confirmation and reply codes are sent only in case of an error.
not-delivered	310	channel	The client asked for a specific message that is no longer available. The message was delivered to another client, or was purged from the queue for some other reason.
content-too-large	311	channel	The client attempted to transfer content larger than the server could accept at the present time. The client may retry at a later time.
no-route	312	channel	When the exchange cannot route the result of a .Publish, most likely due to an invalid routing key. Only when the mandatory flag is set.
no-consumers	313	channel	When the exchange cannot deliver to a consumer when the immediate flag is set. As a result of pending data on the queue or the absence of any consumers of the queue.
connection-forced	320	connection	An operator intervened to close the connection for some reason. The client may retry at some later date.
invalid-path	402	connection	The client tried to work with an unknown virtual host.
access-refused	403	channel	The client attempted to work with a server entity to which it has no access due to security settings.
not-found	404	channel	The client attempted to work with a server entity that does not exist.
resource-locked	405	channel	The client attempted to work with a server entity to which it has no access because another client is working with it.
precondition-failed	406	channel	The client requested a method that was not allowed because some precondition failed.
frame-error	501	connection	The client sent a malformed frame that the server could not decode. This strongly implies a programming error in the client.
syntax-error	502	connection	The client sent a frame that contained illegal values for one or more fields. This strongly implies a programming error in the client.

Name	Value	Error type	[Label] Description
command-invalid	503	connection	The client sent an invalid sequence of frames, attempting to perform an operation that was considered invalid by the server. This usually implies a programming error in the client.
channel-error	504	connection	The client attempted to work with a channel that had not been correctly opened. This most likely indicates a fault in the client layer.
resource-error	506	connection	The server could not complete the method because it lacked sufficient resources. This may be due to the client creating too many of some type of entity.
not-allowed	530	connection	The client tried to work with some entity in a manner that is prohibited by the server, due to security settings or by some other criteria.
not-implemented	540	connection	The client tried to use functionality that is not implemented in the server.
internal-error	541	connection	The server could not complete the method because of an internal error. The server may require intervention by an operator in order to resume normal operations.

1.3. Class and Method ID Summaries

The following class and method IDs are defined in the specification:

Class	ID	Short class description	Method	ID	Short method description
connection	10	work with socket connections	start	10	start connection negotiation
			start-ok	11	select security mechanism and locale
			secure	20	security mechanism challenge
			secure-ok	21	security mechanism response
			tune	30	propose connection tuning parameters
			tune-ok	31	negotiate connection tuning parameters
			open	40	open connection to virtual host
			open-ok	41	signal that connection is ready
			redirect	42	redirects client to other server
			close	50	request a connection close
			close-ok	51	confirm a connection close
channel	20	work with channels	open	10	open a channel for use
			open-ok	11	signal that the channel is ready
			flow	20	enable/disable flow from peer
			flow-ok	21	confirm a flow method

Class	ID	Short class description	Method	ID	Short method description
			close	40	request a channel close
			close-ok	41	confirm a channel close
			resume	50	resume an interrupted channel
			ping	60	[WORK IN PROGRESS] initiates a pong
			pong	70	[WORK IN PROGRESS] issued after receiving a ping
			ok	80	[WORK IN PROGRESS] signals normal completion
access	30	work with access tickets	request	10	request an access ticket
			request-ok	11	grant access to server resources
exchange	40	work with exchanges	declare	10	verify exchange exists, create if needed
			declare-ok	11	confirm exchange declaration
			delete	20	delete an exchange
			delete-ok	21	confirm deletion of an exchange
queue	50	work with queues	declare	10	declare queue, create if needed
			declare-ok	11	confirms a queue definition
			bind	20	bind queue to an exchange
			bind-ok	21	confirm bind successful
			unbind	50	unbind a queue from an exchange
			unbind-ok	51	confirm unbind successful
			purge	30	purge a queue
			purge-ok	31	confirms a queue purge
			delete	40	delete a queue
			delete-ok	41	confirm deletion of a queue
basic	60	work with basic content	qos	10	specify quality of service
			qos-ok	11	confirm the requested qos
			consume	20	start a queue consumer
			consume-ok	21	confirm a new consumer
			cancel	30	end a queue consumer
			cancel-ok	31	confirm a cancelled consumer
			publish	40	publish a message
			return	50	return a failed message
			deliver	60	notify the client of a consumer message
			get	70	direct access to a queue

Class	ID	Short class description	Method	ID	Short method description
			get-ok	71	provide client with a message
			get-empty	72	indicate no messages available
			ack	80	acknowledge one or more messages
			reject	90	reject an incoming message
			recover	100	redeliver unacknowledged messages
file	70	work with file content	qos	10	specify quality of service
			qos-ok	11	confirm the requested qos
			consume	20	start a queue consumer
			consume-ok	21	confirm a new consumer
			cancel	30	end a queue consumer
			cancel-ok	31	confirm a cancelled consumer
			open	40	request to start staging
			open-ok	41	confirm staging ready
			stage	50	stage message content
			publish	60	publish a message
			return	70	return a failed message
			deliver	80	notify the client of a consumer message
			ack	90	acknowledge one or more messages
			reject	100	reject an incoming message
stream	80	work with streaming content	qos	10	specify quality of service
			qos-ok	11	confirm the requested qos
			consume	20	start a queue consumer
			consume-ok	21	confirm a new consumer
			cancel	30	end a queue consumer
			cancel-ok	31	confirm a cancelled consumer
			publish	40	publish a message
			return	50	return a failed message
			deliver	60	notify the client of a consumer message
tx	90	work with standard transactions	select	10	select standard transaction mode
			select-ok	11	confirm transaction mode
			commit	20	commit the current transaction
			commit-ok	21	confirm a successful commit
			rollback	30	abandon the current transaction

Class	ID	Short class description	Method	ID	Short method description
			rollback-ok	31	confirm successful rollback
dtx	100	work with distributed transactions	select	10	select standard transaction mode
			select-ok	11	confirm transaction mode
			start	20	start a new distributed transaction
			start-ok	21	confirm the start of a new distributed transaction
tunnel	110	methods for protocol tunnelling	request	10	sends a tunnelled method
message	120	[WORK IN PROGRESS] message transfer	transfer	10	[WORK IN PROGRESS] transfer a message
			consume	20	[WORK IN PROGRESS] start a queue consumer
			cancel	30	[WORK IN PROGRESS] end a queue consumer
			get	40	[WORK IN PROGRESS] direct access to a queue
			recover	50	[WORK IN PROGRESS] redeliver unacknowledged messages
			open	60	[WORK IN PROGRESS] create a reference to an empty message body
			close	70	[WORK IN PROGRESS] close a reference
			append	80	[WORK IN PROGRESS] append to a reference
			checkpoint	90	[WORK IN PROGRESS] checkpoint a message body
			resume	100	[WORK IN PROGRESS] open and resume a checkpointed message
			qos	110	[WORK IN PROGRESS] specify quality of service
			ok	500	[WORK IN PROGRESS] normal completion
			empty	510	[WORK IN PROGRESS] empty queue
			reject	520	[WORK IN PROGRESS] reject a message
			offset	530	[WORK IN PROGRESS] return an offset

1.4. Class connection

The connection class provides methods for a client to establish a network connection to a server, and for

both peers to operate the connection thereafter.

Class Grammar:

```
connection           = open-connection *use-connection close-connection
open-connection      = C:protocol-header
                       S:START C:START-OK
                       *challenge
                       S:TUNE C:TUNE-OK
                       C:OPEN S:OPEN-OK | S:REDIRECT
challenge            = S:SECURE C:SECURE-OK
use-connection       = *channel
close-connection     = C:CLOSE S:CLOSE-OK
                     / S:CLOSE C:CLOSE-OK
```

1.4.1. Property and Method Summary

Class ***connection*** defines the following methods (S = received by server; C = received by client):

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
start	10	start-ok	start connection negotiation		Y	version-major	octet	protocol major version
						version-minor	octet	protocol minor version
						server-properties	peer-properties	server properties
						mechanisms	longstr	available security mechanisms
						locales	longstr	available message locales
start-ok	11		select security mechanism and locale	Y		client-properties	peer-properties	client properties
						mechanism	shortstr	selected security mechanism
						response	longstr	security response data
						locale	shortstr	selected message locale
secure	20	secure-ok	security mechanism challenge		Y	challenge	longstr	security challenge data
secure-ok	21		security mechanism response	Y		response	longstr	security response data
tune	30	tune-ok	propose connection tuning parameters		Y	channel-max	short	proposed maximum channels
						frame-max	long	proposed maximum frame size

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
						heartbeat	short	desired heartbeat delay
tune-ok	31		negotiate connection tuning parameters	Y		channel-max	short	negotiated maximum channels
						frame-max	long	negotiated maximum frame size
						heartbeat	short	desired heartbeat delay
open	40	open-ok	open connection to virtual host	Y		virtual-host	path	virtual host name
						capabilities	shortstr	required capabilities
						insist	bit	insist on connecting to server
open-ok	41		signal that connection is ready		Y	known-hosts	known-hosts	
redirect	42		redirects client to other server		Y	host	shortstr	server to connect to
						known-hosts	known-hosts	
close	50	close-ok	request a connection close	Y	Y	reply-code	reply-code	
						reply-text	reply-text	
						class-id	class-id	failing method class
						method-id	method-id	failing method ID
close-ok	51		confirm a connection close	Y	Y	[No parameters defined for this method]		

1.4.2. Methods

1.4.2.1. Method connection.start (ID 10)

ID: 10

Method accepted by: Client

Synchronous: Yes; expected response is from method ***connection.start-ok***

Number of parameters: 5

Label: start connection negotiation

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description

1	version-major	octet	protocol major version
2	version-minor	octet	protocol minor version
3	server-properties	peer-properties	server properties
4	mechanisms	longstr	available security mechanisms
5	locales	longstr	available message locales

This method starts the connection negotiation process by telling the client the protocol version that the server proposes, along with a list of security mechanisms which the client can use for authentication.

Guidelines for implementers:

- If the server cannot support the protocol specified in the protocol header, it MUST close the socket connection without sending any response method.

  Test scenario: The client sends a protocol header containing an invalid protocol name. The server must respond by closing the connection.

- The server MUST provide a protocol version that is lower than or equal to that requested by the client in the protocol header.

  Test scenario: The client requests a protocol version that is higher than any valid implementation, e.g. 9.0. The server must respond with a current protocol version, e.g. 1.0.

- If the client cannot handle the protocol version suggested by the server it MUST close the socket connection.

  Test scenario: The server sends a protocol version that is lower than any valid implementation, e.g. 0.1. The client must respond by closing the connection.

1.4.2.1.1. Parameter connection.start.version-major (octet)

Ordinal: 1

Domain: octet

Label: protocol major version

The protocol version, major component, as transmitted in the AMQP protocol header. This, combined with the protocol minor component fully describe the protocol version, which is written in the format major-minor. Hence, with major=1, minor=3, the protocol version would be "1-3".

1.4.2.1.2. Parameter connection.start.version-minor (octet)

Ordinal: 2

Domain: octet

Label: protocol minor version

The protocol version, minor component, as transmitted in the AMQP protocol header. This, combined with the protocol major component fully describe the protocol version, which is written in the format major-minor. Hence, with major=1, minor=3, the protocol version would be "1-3".

1.4.2.1.3. Parameter connection.start.server-properties (peer-properties)

Ordinal: 3

Domain: peer-properties

Label: server properties

1.4.2.1.4. Parameter connection.start.mechanisms (longstr)

Ordinal: 4

Domain: longstr

Label: available security mechanisms

A list of the security mechanisms that the server supports, delimited by spaces.

1.4.2.1.5. Parameter connection.start.locales (longstr)

Ordinal: 5

Domain: longstr

Label: available message locales

A list of the message locales that the server supports, delimited by spaces. The locale defines the language in which the server will send reply texts.

1.4.2.2. Method connection.start-ok (ID 11)

ID: 11

Method accepted by: Server

Synchronous: No

Number of parameters: 4

Label: select security mechanism and locale

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	client-properties	peer-properties	client properties
2	mechanism	shortstr	selected security mechanism

3	response	longstr	security response data
4	locale	shortstr	selected message locale

This method selects a SASL security mechanism.

1.4.2.2.1. Parameter connection.start-ok.client-properties (peer-properties)

Ordinal: 1

Domain: peer-properties

Label: client properties

1.4.2.2.2. Parameter connection.start-ok.mechanism (shortstr)

Ordinal: 2

Domain: shortstr

Label: selected security mechanism

A single security mechanisms selected by the client, which must be one of those specified by the server.

1.4.2.2.3. Parameter connection.start-ok.response (longstr)

Ordinal: 3

Domain: longstr

Label: security response data

A block of opaque data passed to the security mechanism. The contents of this data are defined by the SASL security mechanism.

1.4.2.2.4. Parameter connection.start-ok.locale (shortstr)

Ordinal: 4

Domain: shortstr

Label: selected message locale

A single message locale selected by the client, which must be one of those specified by the server.

1.4.2.3. Method connection.secure (ID 20)

ID: 20

Method accepted by: Client

Synchronous: Yes; expected response is from method ***connection.secure-ok***

Number of parameters: 1

Label: security mechanism challenge

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	challenge	longstr	security challenge data

The SASL protocol works by exchanging challenges and responses until both peers have received sufficient information to authenticate each other. This method challenges the client to provide more information.

1.4.2.3.1. Parameter connection.secure.challenge (longstr)

Ordinal: 1

Domain: longstr

Label: security challenge data

Challenge information, a block of opaque binary data passed to the security mechanism.

1.4.2.4. Method connection.secure-ok (ID 21)

ID: 21

Method accepted by: Server

Synchronous: No

Number of parameters: 1

Label: security mechanism response

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	response	longstr	security response data

This method attempts to authenticate, passing a block of SASL data for the security mechanism at the server side.

1.4.2.4.1. Parameter connection.secure-ok.response (longstr)

Ordinal: 1

Domain: longstr

Label: security response data

A block of opaque data passed to the security mechanism. The contents of this data are defined by the SASL security mechanism.

1.4.2.5. Method connection.tune (ID 30)

ID: 30

Method accepted by: Client

Synchronous: Yes; expected response is from method ***connection.tune-ok***

Number of parameters: 3

Label: propose connection tuning parameters

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	channel-max	short	proposed maximum channels
2	frame-max	long	proposed maximum frame size
3	heartbeat	short	desired heartbeat delay

This method proposes a set of connection configuration values to the client. The client can accept and/or adjust these.

1.4.2.5.1. Parameter connection.tune.channel-max (short)

Ordinal: 1

Domain: short

Label: proposed maximum channels

The maximum total number of channels that the server allows per connection. Zero means that the server does not impose a fixed limit, but the number of allowed channels may be limited by available server resources.

1.4.2.5.2. Parameter connection.tune.frame-max (long)

Ordinal: 2

Domain: long

Label: proposed maximum frame size

The largest frame size that the server proposes for the connection. The client can negotiate a lower value.

Zero means that the server does not impose any specific limit but may reject very large frames if it cannot allocate resources for them.

1.4.2.5.3. Parameter connection.tune.heartbeat (short)

Ordinal: 3

Domain: short

Label: desired heartbeat delay

The delay, in seconds, of the connection heartbeat that the server wants. Zero means the server does not want a heartbeat.

1.4.2.6. Method connection.tune-ok (ID 31)

ID: 31

Method accepted by: Server

Synchronous: No

Number of parameters: 3

Label: negotiate connection tuning parameters

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	channel-max	short	negotiated maximum channels
2	frame-max	long	negotiated maximum frame size
3	heartbeat	short	desired heartbeat delay

This method sends the client's connection tuning parameters to the server. Certain fields are negotiated, others provide capability information.

1.4.2.6.1. Parameter connection.tune-ok.channel-max (short)

Ordinal: 1

Domain: short

Label: negotiated maximum channels

The maximum total number of channels that the client will use per connection.

1.4.2.6.2. Parameter connection.tune-ok.frame-max (long)

Ordinal: 2

Domain: long

Label: negotiated maximum frame size

The largest frame size that the client and server will use for the connection. Zero means that the client does not impose any specific limit but may reject very large frames if it cannot allocate resources for them. Note that the frame-max limit applies principally to content frames, where large contents can be broken into frames of arbitrary size.

1.4.2.6.3. Parameter connection.tune-ok.heartbeat (short)

Ordinal: 3

Domain: short

Label: desired heartbeat delay

The delay, in seconds, of the connection heartbeat that the client wants. Zero means the client does not want a heartbeat.

1.4.2.7. Method connection.open (ID 40)

ID: 40

Method accepted by: Server

Synchronous: Yes; expected response is from method ***connection.open-ok***

Number of parameters: 3

Label: open connection to virtual host

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	virtual-host	path	virtual host name
2	capabilities	shortstr	required capabilities
3	insist	bit	insist on connecting to server

This method opens a connection to a virtual host, which is a collection of resources, and acts to separate multiple application domains within a server. The server may apply arbitrary limits per virtual host, such as the number of each type of entity that may be used, per connection and/or in total.

1.4.2.7.1. Parameter connection.open.virtual-host (path)

Ordinal: 1

Domain: path

Label: virtual host name

The name of the virtual host to work with.

1.4.2.7.2. Parameter connection.open.capabilities (shortstr)

Ordinal: 2

Domain: shortstr

Label: required capabilities

The client can specify zero or more capability names, delimited by spaces. The server can use this string to how to process the client's connection request.

1.4.2.7.3. Parameter connection.open.insist (bit)

Ordinal: 3

Domain: bit

Label: insist on connecting to server

In a configuration with multiple collaborating servers, the server may respond to a Connection.Open method with a Connection.Redirect. The insist option tells the server that the client is insisting on a connection to the specified server.

1.4.2.8. Method connection.open-ok (ID 41)

ID: 41

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: signal that connection is ready

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	known-hosts	known-hosts	

This method signals to the client that the connection is ready for use.

1.4.2.8.1. Parameter connection.open-ok.known-hosts (known-hosts)

Ordinal: 1

Domain: known-hosts

1.4.2.9. Method connection.redirect (ID 42)

ID: 42

Method accepted by: Client

Synchronous: No

Number of parameters: 2

Label: redirects client to other server

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	host	shortstr	server to connect to
2	known-hosts	known-hosts	

This method redirects the client to another server, based on the requested virtual host and/or capabilities.

Guidelines for implementers:

- 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.

1.4.2.9.1. Parameter connection.redirect.host (shortstr)

Ordinal: 1

Domain: shortstr

Label: server to connect to

Specifies the server to connect to. This is an IP address or a DNS name, optionally followed by a colon and a port number. If no port number is specified, the client should use the default port number for the protocol.

1.4.2.9.2. Parameter connection.redirect.known-hosts (known-hosts)

Ordinal: 2

Domain: known-hosts

1.4.2.10. Method connection.close (ID 50)

ID: 50

Method accepted by: Server, Client

Synchronous: Yes; expected response is from method ***connection.close-ok***

Number of parameters: 4

Label: request a connection close

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	reply-code	reply-code	
2	reply-text	reply-text	
3	class-id	class-id	failing method class
4	method-id	method-id	failing method ID

This method indicates that the sender wants to close the connection. This may be due to internal conditions (e.g. a forced shut-down) or due to an error handling a specific method, i.e. an exception. When a close is due to an exception, the sender provides the class and method id of the method which caused the exception.

Guidelines for implementers:

- After sending this method any received method except the Close-OK method MUST be discarded.

1.4.2.10.1. Parameter connection.close.reply-code (reply-code)

Ordinal: 1

Domain: reply-code

1.4.2.10.2. Parameter connection.close.reply-text (reply-text)

Ordinal: 2

Domain: reply-text

1.4.2.10.3. Parameter connection.close.class-id (class-id)

Ordinal: 3

Domain: class-id

Label: failing method class

When the close is provoked by a method exception, this is the class of the method.

1.4.2.10.4. Parameter connection.close.method-id (method-id)

Ordinal: 4

Domain: method-id

Label: failing method ID

When the close is provoked by a method exception, this is the ID of the method.

1.4.2.11. Method connection.close-ok (ID 51)

ID: 51

Method accepted by: Server, Client

Synchronous: No

Number of parameters: 0

Label: confirm a connection close

This method confirms a Connection.Close method and tells the recipient that it is safe to release resources for the connection and close the socket.

Guidelines for implementers:

- A peer that detects a socket closure without having received a Close-Ok handshake method SHOULD log the error.

1.5. Class channel

The channel class provides methods for a client to establish a channel to a server and for both peers to operate the channel thereafter.

Class Grammar:

```
channel              = open-channel *use-channel close-channel
open-channel         = C:OPEN S:OPEN-OK
                     / C:RESUME S:OK
use-channel          = C:FLOW S:FLOW-OK
                     / S:FLOW C:FLOW-OK
                     / S:PING C:OK
                     / C:PONG S:OK
                     / C:PING S:OK
                     / S:PONG C:OK
                     / functional-class
close-channel        = C:CLOSE S:CLOSE-OK
                     / S:CLOSE C:CLOSE-OK
```

1.5.1. Property and Method Summary

Class ***channel*** defines the following methods (S = received by server; C = received by client):

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
open	10	open-ok	open a channel for use	Y		out-of-band	shortstr	out-of-band settings
open-ok	11		signal that the channel is ready		Y	channel-id	channel-id	
flow	20	flow-ok	enable/disable flow from peer	Y	Y	active	bit	start/stop content frames
flow-ok	21		confirm a flow method	Y	Y	active	bit	current flow setting
close	40	close-ok	request a channel close	Y	Y	reply-code	reply-code	
						reply-text	reply-text	
						class-id	class-id	failing method class
						method-id	method-id	failing method ID
close-ok	41		confirm a channel close	Y	Y	[No parameters defined for this method]		
resume	50	ok	resume an interrupted channel	Y		channel-id	channel-id	
ping	60	ok	[WORK IN PROGRESS] initiates a pong	Y	Y	[No parameters defined for this method]		
pong	70	ok	[WORK IN PROGRESS] issued after receiving a ping	Y	Y	[No parameters defined for this method]		
ok	80		[WORK IN PROGRESS] signals normal completion	Y	Y	[No parameters defined for this method]		

1.5.2. Methods

1.5.2.1. Method channel.open (ID 10)

ID: 10

Method accepted by: Server

Synchronous: Yes; expected response is from method ***channel.open-ok***

Number of parameters: 1

Label: open a channel for use

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	out-of-band	shortstr	out-of-band settings

This method opens a channel to the server.

Guidelines for implementers:

- The client MUST NOT use this method on an already-opened channel.

  Test scenario: Client opens a channel and then reopens the same channel.

  On failure: Constant "channel-error" (See AMQP-defined Constants)

1.5.2.1.1. Parameter channel.open.out-of-band (shortstr)

Ordinal: 1

Domain: shortstr

Label: out-of-band settings

Configures out-of-band transfers on this channel. The syntax and meaning of this field will be formally defined at a later date.

1.5.2.2. Method channel.open-ok (ID 11)

ID: 11

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: signal that the channel is ready

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	channel-id	channel-id	

This method signals to the client that the channel is ready for use.

1.5.2.2.1. Parameter channel.open-ok.channel-id (channel-id)

Ordinal: 1

Domain: channel-id

1.5.2.3. Method channel.flow (ID 20)

ID: 20

Method accepted by: Server, Client

Synchronous: Yes; expected response is from method ***channel.flow-ok***

Number of parameters: 1

Label: enable/disable flow from peer

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	active	bit	start/stop content frames

This method asks the peer to pause or restart the flow of content data. This is a simple flow-control mechanism that a peer can use to avoid overflowing its queues or otherwise finding itself receiving more messages than it can process. Note that this method is not intended for window control. The peer that receives a disable flow method should finish sending the current content frame, if any, then pause.

Guidelines for implementers:

- 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.
- When sending content frames, a peer SHOULD monitor the channel for incoming methods and respond to a Channel.Flow as rapidly as possible.
- 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.
- 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 broker.

1.5.2.3.1. Parameter channel.flow.active (bit)

Ordinal: 1

Domain: bit

Label: start/stop content frames

If 1, the peer starts sending content frames. If 0, the peer stops sending content frames.

1.5.2.4. Method channel.flow-ok (ID 21)

ID: 21

Method accepted by: Server, Client

Synchronous: No

Number of parameters: 1

Label: confirm a flow method

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	active	bit	current flow setting

Confirms to the peer that a flow command was received and processed.

1.5.2.4.1. Parameter channel.flow-ok.active (bit)

Ordinal: 1

Domain: bit

Label: current flow setting

Confirms the setting of the processed flow method: 1 means the peer will start sending or continue to send content frames; 0 means it will not.

1.5.2.5. Method channel.close (ID 40)

ID: 40

Method accepted by: Server, Client

Synchronous: Yes; expected response is from method ***channel.close-ok***

Number of parameters: 4

Label: request a channel close

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	reply-code	reply-code	
2	reply-text	reply-text	
3	class-id	class-id	failing method class

4	method-id	method-id	failing method ID

This method indicates that the sender wants to close the channel. This may be due to internal conditions (e.g. a forced shut-down) or due to an error handling a specific method, i.e. an exception. When a close is due to an exception, the sender provides the class and method id of the method which caused the exception.

Guidelines for implementers:

- After sending this method any received method except the Close-OK method MUST be discarded.

1.5.2.5.1. Parameter channel.close.reply-code (reply-code)

Ordinal: 1

Domain: reply-code

1.5.2.5.2. Parameter channel.close.reply-text (reply-text)

Ordinal: 2

Domain: reply-text

1.5.2.5.3. Parameter channel.close.class-id (class-id)

Ordinal: 3

Domain: class-id

Label: failing method class

When the close is provoked by a method exception, this is the class of the method.

1.5.2.5.4. Parameter channel.close.method-id (method-id)

Ordinal: 4

Domain: method-id

Label: failing method ID

When the close is provoked by a method exception, this is the ID of the method.

1.5.2.6. Method channel.close-ok (ID 41)

ID: 41

Method accepted by: Server, Client

Synchronous: No

Number of parameters: 0

Label: confirm a channel close

This method confirms a Channel.Close method and tells the recipient that it is safe to release resources for the channel.

Guidelines for implementers:

- A peer that detects a socket closure without having received a Channel.Close-Ok handshake method SHOULD log the error.

1.5.2.7. Method channel.resume (ID 50)

ID: 50

Method accepted by: Server

Synchronous: Yes; expected response is from method ***channel.ok***

Number of parameters: 1

Label: resume an interrupted channel

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	channel-id	channel-id	

This method resume a previously interrupted channel.

1.5.2.7.1. Parameter channel.resume.channel-id (channel-id)

Ordinal: 1

Domain: channel-id

1.5.2.8. Method channel.ping (ID 60)

ID: 60

Method accepted by: Server, Client

Synchronous: Yes; expected response is from method ***channel.ok***

Number of parameters: 0

Label: [WORK IN PROGRESS] initiates a pong

[WORK IN PROGRESS] Request that the recipient issue a pong request.

1.5.2.9. Method channel.pong (ID 70)

ID: 70

Method accepted by: Server, Client

Synchronous: Yes; expected response is from method ***channel.ok***

Number of parameters: 0

Label: [WORK IN PROGRESS] issued after receiving a ping

[WORK IN PROGRESS] Issued after a ping request is received. Note that this is a request issued after receiving a ping, not a response to receiving a ping.

1.5.2.10. Method channel.ok (ID 80)

ID: 80

Method accepted by: Server, Client

Synchronous: No

Number of parameters: 0

Label: [WORK IN PROGRESS] signals normal completion

[WORK IN PROGRESS] Signals normal completion of a method.

1.6. Class access

The protocol control access to server resources using access tickets. A client must explicitly request access tickets before doing work. An access ticket grants a client the right to use a specific set of resources - called a "realm" - in specific ways.

Class Grammar:

```
access              = C:REQUEST S:REQUEST-OK
```

1.6.1. Property and Method Summary

Class ***access*** defines the following methods (S = received by server; C = received by client):

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
request	10	request-ok	request an access ticket	Y		realm	shortstr	name of requested realm
						exclusive	bit	request exclusive access

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
						passive	bit	request passive access
						active	bit	request active access
						write	bit	request write access
						read	bit	request read access
request-ok	11		grant access to server resources		Y	ticket	access-ticket	

1.6.2. Methods

1.6.2.1. Method access.request (ID 10)

ID: 10

Method accepted by: Server

Synchronous: Yes; expected response is from method ***access.request-ok***

Number of parameters: 6

Label: request an access ticket

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	realm	shortstr	name of requested realm
2	exclusive	bit	request exclusive access
3	passive	bit	request passive access
4	active	bit	request active access
5	write	bit	request write access
6	read	bit	request read access

This method requests an access ticket for an access realm. The server responds by granting the access ticket. If the client does not have access rights to the requested realm this causes a connection exception. Access tickets are a per-channel resource.

1.6.2.1.1. Parameter access.request.realm (shortstr)

Ordinal: 1

Domain: shortstr

Label: name of requested realm

Specifies the name of the realm to which the client is requesting access. The realm is a configured server-side object that collects a set of resources (exchanges, queues, etc.). If the channel has already requested an access ticket onto this realm, the previous ticket is destroyed and a new ticket is created with the requested access rights, if allowed.

1.6.2.1.2. Parameter access.request.exclusive (bit)

Ordinal: 2

Domain: bit

Label: request exclusive access

Request exclusive access to the realm, meaning that this will be the only channel that uses the realm's resources.

1.6.2.1.3. Parameter access.request.passive (bit)

Ordinal: 3

Domain: bit

Label: request passive access

Request message passive access to the specified access realm. Passive access lets a client get information about resources in the realm but not to make any changes to them.

1.6.2.1.4. Parameter access.request.active (bit)

Ordinal: 4

Domain: bit

Label: request active access

Request message active access to the specified access realm. Active access lets a client get create and delete resources in the realm.

1.6.2.1.5. Parameter access.request.write (bit)

Ordinal: 5

Domain: bit

Label: request write access

Request write access to the specified access realm. Write access lets a client publish messages to all

exchanges in the realm.

1.6.2.1.6. Parameter access.request.read (bit)

Ordinal: 6

Domain: bit

Label: request read access

Request read access to the specified access realm. Read access lets a client consume messages from queues in the realm.

1.6.2.2. Method access.request-ok (ID 11)

ID: 11

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: grant access to server resources

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	

This method provides the client with an access ticket. The access ticket is valid within the current channel and for the lifespan of the channel.

Guidelines for implementers:

- The client MUST NOT use access tickets except within the same channel as originally granted.

  Test scenario: Client opens two channels, requests a ticket on one channel, and then tries to use that ticket in a second channel.

  On failure: Constant "not-allowed" (See AMQP-defined Constants)

1.6.2.2.1. Parameter access.request-ok.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

1.7. Class exchange

Exchanges match and distribute messages across queues. Exchanges can be configured in the server or created at runtime.

Class Grammar:

```
exchange             = C:DECLARE    S:DECLARE-OK
                     / C:DELETE     S:DELETE-OK
```

Guidelines for implementers:

- The server MUST implement these standard exchange types: fanout, direct.

  Test scenario: Client attempts to declare an exchange with each of these standard types.

- The server SHOULD implement these standard exchange types: topic, headers.

  Test scenario: Client attempts to declare an exchange with each of these standard types.

- 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.

  The server MUST, in each virtual host, pre-declare at least two direct exchange instances: one named "amq.direct", the other with no public name that serves as a default exchange for Publish methods.

  Test scenario: Client creates a temporary queue and attempts to bind to each required exchange instance ("amq.fanout", "amq.direct", "amq.topic", and "amq.headers" if those types are defined).

- 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.

  Test scenario: Client checks that the default exchange is active by specifying a queue binding with no exchange name, and publishing a message with a suitable routing key but without specifying the exchange name, then ensuring that the message arrives in the queue correctly.

- 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.

- The server MAY implement other exchange types as wanted.

1.7.1. Property and Method Summary

Class ***exchange*** defines the following methods (S = received by server; C = received by client):

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
declare	10	declare-ok	verify exchange exists, create if needed	Y		ticket	access-ticket	
						exchange	exchange-name	
						type	shortstr	exchange type
						passive	bit	do not create exchange
						durable	bit	request a durable exchange
						auto-delete	bit	auto-delete when unused
						internal	bit	create internal exchange
						nowait	bit	do not send reply method
						arguments	table	arguments for declaration
declare-ok	11		confirm exchange declaration		Y	[No parameters defined for this method]		
delete	20	delete-ok	delete an exchange	Y		ticket	access-ticket	
						exchange	exchange-name	
						if-unused	bit	delete only if unused
						nowait	bit	do not send a reply method
delete-ok	21		confirm deletion of an exchange		Y	[No parameters defined for this method]		

1.7.2. Methods

1.7.2.1. Method exchange.declare (ID 10)

ID: 10

Method accepted by: Server

Synchronous: Yes; expected response is from method ***exchange.declare-ok***

Number of parameters: 9

Label: verify exchange exists, create if needed

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	exchange	exchange-name	

3	type	shortstr	exchange type
4	passive	bit	do not create exchange
5	durable	bit	request a durable exchange
6	auto-delete	bit	auto-delete when unused
7	internal	bit	create internal exchange
8	nowait	bit	do not send reply method
9	arguments	table	arguments for declaration

This method creates an exchange if it does not already exist, and if the exchange exists, verifies that it is of the correct and expected class.

Guidelines for implementers:

- The server SHOULD support a minimum of 16 exchanges per virtual host and ideally, impose no limit except as defined by available resources.

  Test scenario: The client creates as many exchanges as it can until the server reports an error; the number of exchanges successfully created must be at least sixteen.

1.7.2.1.1. Parameter exchange.declare.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

When a client defines a new exchange, this belongs to the access realm of the ticket used. All further work done with that exchange must be done with an access ticket for the same realm.

1.7.2.1.2. Parameter exchange.declare.exchange (exchange-name)

Ordinal: 2

Domain: exchange-name

1.7.2.1.3. Parameter exchange.declare.type (shortstr)

Ordinal: 3

Domain: shortstr

Label: exchange type

Each exchange belongs to one of a set of exchange types implemented by the server. The exchange types define the functionality of the exchange - i.e. how messages are routed through it. It is not valid or meaningful to attempt to change the type of an existing exchange.

1.7.2.1.4. Parameter exchange.declare.passive (bit)

Ordinal: 4

Domain: bit

Label: do not create exchange

If set, the server will not create the exchange. The client can use this to check whether an exchange exists without modifying the server state.

1.7.2.1.5. Parameter exchange.declare.durable (bit)

Ordinal: 5

Domain: bit

Label: request a durable exchange

If set when creating a new exchange, the exchange will be marked as durable. Durable exchanges remain active when a server restarts. Non-durable exchanges (transient exchanges) are purged if/when a server restarts.

1.7.2.1.6. Parameter exchange.declare.auto-delete (bit)

Ordinal: 6

Domain: bit

Label: auto-delete when unused

If set, the exchange is deleted when all queues have finished using it.

1.7.2.1.7. Parameter exchange.declare.internal (bit)

Ordinal: 7

Domain: bit

Label: create internal exchange

If set, the exchange may not be used directly by publishers, but only when bound to other exchanges. Internal exchanges are used to construct wiring that is not visible to applications.

1.7.2.1.8. Parameter exchange.declare.nowait (bit)

Ordinal: 8

Domain: bit

Label: do not send reply method

If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

1.7.2.1.9. Parameter exchange.declare.arguments (table)

Ordinal: 9

Domain: table

Label: arguments for declaration

A set of arguments for the declaration. The syntax and semantics of these arguments depends on the server implementation. This field is ignored if passive is 1.

1.7.2.2. Method exchange.declare-ok (ID 11)

ID: 11

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm exchange declaration

This method confirms a Declare method and confirms the name of the exchange, essential for automatically-named exchanges.

1.7.2.3. Method exchange.delete (ID 20)

ID: 20

Method accepted by: Server

Synchronous: Yes; expected response is from method ***exchange.delete-ok***

Number of parameters: 4

Label: delete an exchange

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	exchange	exchange-name	
3	if-unused	bit	delete only if unused
4	nowait	bit	do not send a reply method

This method deletes an exchange. When an exchange is deleted all queue bindings on the exchange are

cancelled.

1.7.2.3.1. Parameter exchange.delete.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

1.7.2.3.2. Parameter exchange.delete.exchange (exchange-name)

Ordinal: 2

Domain: exchange-name

1.7.2.3.3. Parameter exchange.delete.if-unused (bit)

Ordinal: 3

Domain: bit

Label: delete only if unused

If set, the server will only delete the exchange if it has no queue bindings. If the exchange has queue bindings the server does not delete it but raises a channel exception instead.

1.7.2.3.4. Parameter exchange.delete.nowait (bit)

Ordinal: 4

Domain: bit

Label: do not send a reply method

If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

1.7.2.4. Method exchange.delete-ok (ID 21)

ID: 21

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm deletion of an exchange

This method confirms the deletion of an exchange.

1.8. Class queue

Queues store and forward messages. Queues can be configured in the server or created at runtime. Queues must be attached to at least one exchange in order to receive messages from publishers.

Class Grammar:

```
queue                = C:DECLARE   S:DECLARE-OK
                     / C:BIND      S:BIND-OK
                     / C:PURGE     S:PURGE-OK
                     / C:DELETE    S:DELETE-OK
```

Guidelines for implementers:

- A server MUST allow any content class to be sent to any queue, in any mix, and queue and deliver these content classes independently. Note that all methods that fetch content off queues are specific to a given content class.

  Test scenario: Client creates an exchange of each standard type and several queues that it binds to each exchange. It must then successfully send each of the standard content types to each of the available queues.

1.8.1. Property and Method Summary

Class ***queue*** defines the following methods (S = received by server; C = received by client):

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
declare	10	declare-ok	declare queue, create if needed	Y		ticket	access-ticket	
						queue	queue-name	
						passive	bit	do not create queue
						durable	bit	request a durable queue
						exclusive	bit	request an exclusive queue
						auto-delete	bit	auto-delete queue when unused
						nowait	bit	do not send a reply method
						arguments	table	arguments for declaration
declare-ok	11		confirms a queue definition		Y	queue	queue-name	
						message-count	long	number of messages in queue
						consumer-count	long	number of consumers

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
bind	20	bind-ok	bind queue to an exchange	Y		ticket	access-ticket	
						queue	queue-name	
						exchange	exchange-name	name of the exchange to bind to
						routing-key	shortstr	message routing key
						nowait	bit	do not send a reply method
						arguments	table	arguments for binding
bind-ok	21		confirm bind successful		Y	[No parameters defined for this method]		
unbind	50	unbind-ok	unbind a queue from an exchange	Y		ticket	access-ticket	
						queue	queue-name	
						exchange	exchange-name	
						routing-key	shortstr	routing key of binding
						arguments	table	arguments of binding
unbind-ok	51		confirm unbind successful		Y	[No parameters defined for this method]		
purge	30	purge-ok	purge a queue	Y		ticket	access-ticket	
						queue	queue-name	
						nowait	bit	do not send a reply method
purge-ok	31		confirms a queue purge		Y	message-count	long	number of messages purged
delete	40	delete-ok	delete a queue	Y		ticket	access-ticket	
						queue	queue-name	
						if-unused	bit	delete only if unused
						if-empty	bit	delete only if empty
						nowait	bit	do not send a reply method
delete-ok	41		confirm deletion of a queue		Y	message-count	long	number of messages purged

1.8.2. Methods

1.8.2.1. Method queue.declare (ID 10)

ID: 10

Method accepted by: Server

Synchronous: Yes; expected response is from method ***queue.declare-ok***

Number of parameters: 8

Label: declare queue, create if needed

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	queue	queue-name	
3	passive	bit	do not create queue
4	durable	bit	request a durable queue
5	exclusive	bit	request an exclusive queue
6	auto-delete	bit	auto-delete queue when unused
7	nowait	bit	do not send a reply method
8	arguments	table	arguments for declaration

This method creates or checks a queue. When creating a new queue the client can specify various properties that control the durability of the queue and its contents, and the level of sharing for the queue.

Guidelines for implementers:

- The server MUST create a default binding for a newly-created queue to the default exchange, which is an exchange of type 'direct' and use the queue name as the routing key.

  Test scenario: Client creates a new queue, and then without explicitly binding it to an exchange, attempts to send a message through the default exchange binding, i.e. publish a message to the empty exchange, with the queue name as routing key.

- The server SHOULD support a minimum of 256 queues per virtual host and ideally, impose no limit except as defined by available resources.

  Test scenario: Client attempts to create as many queues as it can until the server reports an error. The resulting count must at least be 256.

1.8.2.1.1. Parameter queue.declare.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

When a client defines a new queue, this belongs to the access realm of the ticket used. All further work done with that queue must be done with an access ticket for the same realm.

1.8.2.1.2. Parameter queue.declare.queue (queue-name)

Ordinal: 2

Domain: queue-name

1.8.2.1.3. Parameter queue.declare.passive (bit)

Ordinal: 3

Domain: bit

Label: do not create queue

If set, the server will not create the queue. This field allows the client to assert the presence of a queue without modifying the server state.

1.8.2.1.4. Parameter queue.declare.durable (bit)

Ordinal: 4

Domain: bit

Label: request a durable queue

If set when creating a new queue, the queue will be marked as durable. Durable queues remain active when a server restarts. Non-durable queues (transient queues) are purged if/when a server restarts. Note that durable queues do not necessarily hold persistent messages, although it does not make sense to send persistent messages to a transient queue.

1.8.2.1.5. Parameter queue.declare.exclusive (bit)

Ordinal: 5

Domain: bit

Label: request an exclusive queue

Exclusive queues may only be consumed from by the current connection. Setting the 'exclusive' flag always implies 'auto-delete'.

1.8.2.1.6. Parameter queue.declare.auto-delete (bit)

Ordinal: 6

Domain: bit

Label: auto-delete queue when unused

If set, the queue is deleted when all consumers have finished using it. Last consumer can be cancelled either explicitly or because its channel is closed. If there was no consumer ever on the queue, it won't be

deleted.

1.8.2.1.7. Parameter queue.declare.nowait (bit)

Ordinal: 7

Domain: bit

Label: do not send a reply method

If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

1.8.2.1.8. Parameter queue.declare.arguments (table)

Ordinal: 8

Domain: table

Label: arguments for declaration

A set of arguments for the declaration. The syntax and semantics of these arguments depends on the server implementation. This field is ignored if passive is 1.

1.8.2.2. Method queue.declare-ok (ID 11)

ID: 11

Method accepted by: Client

Synchronous: No

Number of parameters: 3

Label: confirms a queue definition

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	queue	queue-name	
2	message-count	long	number of messages in queue
3	consumer-count	long	number of consumers

This method confirms a Declare method and confirms the name of the queue, essential for automatically-named queues.

1.8.2.2.1. Parameter queue.declare-ok.queue (queue-name)

Ordinal: 1

Domain: queue-name

Reports the name of the queue. If the server generated a queue name, this field contains that name.

1.8.2.2.2. Parameter queue.declare-ok.message-count (long)

Ordinal: 2

Domain: long

Label: number of messages in queue

Reports the number of messages in the queue, which will be zero for newly-created queues.

1.8.2.2.3. Parameter queue.declare-ok.consumer-count (long)

Ordinal: 3

Domain: long

Label: number of consumers

Reports the number of active consumers for the queue. Note that consumers can suspend activity (Channel.Flow) in which case they do not appear in this count.

1.8.2.3. Method queue.bind (ID 20)

ID: 20

Method accepted by: Server

Synchronous: Yes; expected response is from method ***queue.bind-ok***

Number of parameters: 6

Label: bind queue to an exchange

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	queue	queue-name	
3	exchange	exchange-name	name of the exchange to bind to
4	routing-key	shortstr	message routing key
5	nowait	bit	do not send a reply method
6	arguments	table	arguments for binding

This method binds a queue to an exchange. Until a queue is bound it will not receive any messages. In a

classic messaging model, store-and-forward queues are bound to a direct exchange and subscription queues are bound to a topic exchange.

Guidelines for implementers:

- 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.

  Test scenario: A client binds a named queue to an exchange. The client then repeats the bind (with identical arguments).

- If a bind fails, the server MUST raise a connection exception.

  Test scenario: TODO

- The server MUST NOT allow a durable queue to bind to a transient exchange.

  Test scenario: A client creates a transient exchange. The client then declares a named durable queue and then attempts to bind the transient exchange to the durable queue.

  On failure: Constant "not-allowed" (See AMQP-defined Constants)

- Bindings for durable queues are automatically durable and the server SHOULD restore such bindings after a server restart.

  Test scenario: A server creates a named durable queue and binds it to a durable exchange. The server is restarted. The client then attempts to use the queue/exchange combination.

- If the client attempts to bind to an exchange that was declared as internal, the server MUST raise a connection exception with reply code 530 (not allowed).

  Test scenario: A client attempts to bind a named queue to an internal exchange.

- The server SHOULD support at least 4 bindings per queue, and ideally, impose no limit except as defined by available resources.

  Test scenario: A client creates a named queue and attempts to bind it to 4 different non-internal exchanges.

1.8.2.3.1. Parameter queue.bind.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

The client provides a valid access ticket giving "active" access rights to the queue's access realm.

1.8.2.3.2. Parameter queue.bind.queue (queue-name)

Ordinal: 2

Domain: queue-name

Specifies the name of the queue to bind. If the queue name is empty, refers to the current queue for the channel, which is the last declared queue.

1.8.2.3.3. Parameter queue.bind.exchange (exchange-name)

Ordinal: 3

Domain: exchange-name

Label: name of the exchange to bind to

1.8.2.3.4. Parameter queue.bind.routing-key (shortstr)

Ordinal: 4

Domain: shortstr

Label: message routing key

Specifies the routing key for the binding. The routing key is used for routing messages depending on the exchange configuration. Not all exchanges use a routing key - refer to the specific exchange documentation. If the queue name is empty, the server uses the last queue declared on the channel. If the routing key is also empty, the server uses this queue name for the routing key as well. If the queue name is provided but the routing key is empty, the server does the binding with that empty routing key. The meaning of empty routing keys depends on the exchange implementation.

1.8.2.3.5. Parameter queue.bind.nowait (bit)

Ordinal: 5

Domain: bit

Label: do not send a reply method

If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

1.8.2.3.6. Parameter queue.bind.arguments (table)

Ordinal: 6

Domain: table

Label: arguments for binding

A set of arguments for the binding. The syntax and semantics of these arguments depends on the exchange class.

1.8.2.4. Method queue.bind-ok (ID 21)

ID: 21

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm bind successful

This method confirms that the bind was successful.

1.8.2.5. Method queue.unbind (ID 50)

ID: 50

Method accepted by: Server

Synchronous: Yes; expected response is from method ***queue.unbind-ok***

Number of parameters: 5

Label: unbind a queue from an exchange

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	queue	queue-name	
3	exchange	exchange-name	
4	routing-key	shortstr	routing key of binding
5	arguments	table	arguments of binding

This method unbinds a queue from an exchange.

Guidelines for implementers:

- If a unbind fails, the server MUST raise a connection exception.

1.8.2.5.1. Parameter queue.unbind.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

The client provides a valid access ticket giving "active" access rights to the queue's access realm.

1.8.2.5.2. Parameter queue.unbind.queue (queue-name)

Ordinal: 2

Domain: queue-name

Specifies the name of the queue to unbind.

1.8.2.5.3. Parameter queue.unbind.exchange (exchange-name)

Ordinal: 3

Domain: exchange-name

The name of the exchange to unbind from.

1.8.2.5.4. Parameter queue.unbind.routing-key (shortstr)

Ordinal: 4

Domain: shortstr

Label: routing key of binding

Specifies the routing key of the binding to unbind.

1.8.2.5.5. Parameter queue.unbind.arguments (table)

Ordinal: 5

Domain: table

Label: arguments of binding

Specifies the arguments of the binding to unbind.

1.8.2.6. Method queue.unbind-ok (ID 51)

ID: 51

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm unbind successful

This method confirms that the unbind was successful.

1.8.2.7. Method queue.purge (ID 30)

ID: 30

Method accepted by: Server

Synchronous: Yes; expected response is from method ***queue.purge-ok***

Number of parameters: 3

Label: purge a queue

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	queue	queue-name	
3	nowait	bit	do not send a reply method

This method removes all messages from a queue. It does not cancel consumers. Purged messages are deleted without any formal "undo" mechanism.

Guidelines for implementers:

- A call to purge MUST result in an empty queue.
- On transacted channels the server MUST not purge messages that have already been sent to a client but not yet acknowledged.
- 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.

1.8.2.7.1. Parameter queue.purge.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

The access ticket must be for the access realm that holds the queue.

1.8.2.7.2. Parameter queue.purge.queue (queue-name)

Ordinal: 2

Domain: queue-name

Specifies the name of the queue to purge. If the queue name is empty, refers to the current queue for the channel, which is the last declared queue.

1.8.2.7.3. Parameter queue.purge.nowait (bit)

Ordinal: 3

Domain: bit

Label: do not send a reply method

If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

1.8.2.8. Method queue.purge-ok (ID 31)

ID: 31

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: confirms a queue purge

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	message-count	long	number of messages purged

This method confirms the purge of a queue.

1.8.2.8.1. Parameter queue.purge-ok.message-count (long)

Ordinal: 1

Domain: long

Label: number of messages purged

Reports the number of messages purged.

1.8.2.9. Method queue.delete (ID 40)

ID: 40

Method accepted by: Server

Synchronous: Yes; expected response is from method ***queue.delete-ok***

Number of parameters: 5

Label: delete a queue

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	

2	queue	queue-name	
3	if-unused	bit	delete only if unused
4	if-empty	bit	delete only if empty
5	nowait	bit	do not send a reply method

This method deletes a queue. When a queue is deleted any pending messages are sent to a dead-letter queue if this is defined in the server configuration, and all consumers on the queue are cancelled.

Guidelines for implementers:

- 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.

1.8.2.9.1. Parameter queue.delete.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

The client provides a valid access ticket giving "active" access rights to the queue's access realm.

1.8.2.9.2. Parameter queue.delete.queue (queue-name)

Ordinal: 2

Domain: queue-name

Specifies the name of the queue to delete. If the queue name is empty, refers to the current queue for the channel, which is the last declared queue.

1.8.2.9.3. Parameter queue.delete.if-unused (bit)

Ordinal: 3

Domain: bit

Label: delete only if unused

If set, the server will only delete the queue if it has no consumers. If the queue has consumers the server does does not delete it but raises a channel exception instead.

1.8.2.9.4. Parameter queue.delete.if-empty (bit)

Ordinal: 4

Domain: bit

Label: delete only if empty

If set, the server will only delete the queue if it has no messages.

1.8.2.9.5. Parameter queue.delete.nowait (bit)

Ordinal: 5

Domain: bit

Label: do not send a reply method

If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

1.8.2.10. Method queue.delete-ok (ID 41)

ID: 41

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: confirm deletion of a queue

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	message-count	long	number of messages purged

This method confirms the deletion of a queue.

1.8.2.10.1. Parameter queue.delete-ok.message-count (long)

Ordinal: 1

Domain: long

Label: number of messages purged

Reports the number of messages purged.

1.9. Class basic

The Basic class provides methods that support an industry-standard messaging model.

Class Grammar:

```
basic                = C:QOS S:QOS-OK
                     / C:CONSUME S:CONSUME-OK
                     / C:CANCEL S:CANCEL-OK
                     / C:PUBLISH content
                     / S:RETURN content
                     / S:DELIVER content
                     / C:GET ( S:GET-OK content / S:GET-EMPTY )
                     / C:ACK
                     / C:REJECT
```

Guidelines for implementers:

- 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.

  Test scenario: Send a persistent message to queue, stop server, restart server and then verify whether message is still present. Assumes that queues are durable. Persistence without durable queues makes no sense.

- The server MUST NOT discard a persistent basic message in case of a queue overflow.

  Test scenario: Create a queue overflow situation with persistent messages and verify that messages do not get lost (presumably the server will write them to disk).

- The server MAY use the Channel.Flow method to slow or stop a basic message publisher when necessary.

  Test scenario: Create a queue overflow situation with non-persistent messages and verify whether the server responds with Channel.Flow or not. Repeat with persistent messages.

- The server MAY overflow non-persistent basic messages to persistent storage.
- The server MAY discard or dead-letter non-persistent basic messages on a priority basis if the queue size exceeds some configured limit.
- 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.

  Test scenario: Send a number of priority 0 messages to a queue. Send one priority 9 message. Consume messages from the queue and verify that the first message received was priority 9.

- The server MAY implement up to 10 priority levels.

  Test scenario: Send a number of messages with mixed priorities to a queue, so that all priority values from 0 to 9 are exercised. A good scenario would be ten messages in low-to-high priority. Consume from queue and verify how many priority levels emerge.

- The server MUST deliver messages of the same priority in order irrespective of their individual persistence.

Test scenario: Send a set of messages with the same priority but different persistence settings to a queue. Consume and verify that messages arrive in same order as originally published.

- The server MUST support automatic acknowledgements on Basic content, i.e. consumers with the no-ack field set to FALSE.

  Test scenario: Create a queue and a consumer using automatic acknowledgements. Publish a set of messages to the queue. Consume the messages and verify that all messages are received.

- The server MUST support explicit acknowledgements on Basic content, i.e. consumers with the no-ack field set to TRUE.

  Test scenario: Create a queue and a consumer using explicit acknowledgements. Publish a set of messages to the queue. Consume the messages but acknowledge only half of them. Disconnect and reconnect, and consume from the queue. Verify that the remaining messages are received.

1.9.1. Property and Method Summary

Class ***basic*** defines the following properties:

Name	Domain	Short Description
content-type	shortstr	MIME content type
content-encoding	shortstr	MIME content encoding
headers	table	message header field table
delivery-mode	octet	non-persistent (1) or persistent (2)
priority	octet	message priority, 0 to 9
correlation-id	shortstr	application correlation identifier
reply-to	shortstr	destination to reply to
expiration	shortstr	message expiration specification
message-id	shortstr	application message identifier
timestamp	timestamp	message timestamp
type	shortstr	message type name
user-id	shortstr	creating user id
app-id	shortstr	creating application id
cluster-id	shortstr	intra-cluster routing identifier

Class ***basic*** defines the following methods (S = received by server; C = received by client):

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
qos	10	qos-ok	specify quality of	Y		prefetch-size	long	prefetch window in octets

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
			service			prefetch-count	short	prefetch window in messages
						global	bit	apply to entire connection
qos-ok	11		confirm the requested qos		Y	[No parameters defined for this method]		
consume	20	consume-ok	start a queue consumer	Y		ticket	access-ticket	
						queue	queue-name	
						consumer-tag	consumer-tag	
						no-local	no-local	
						no-ack	no-ack	
						exclusive	bit	request exclusive access
						nowait	bit	do not send a reply method
						filter	table	arguments for consuming
consume-ok	21		confirm a new consumer		Y	consumer-tag	consumer-tag	
cancel	30	cancel-ok	end a queue consumer	Y		consumer-tag	consumer-tag	
						nowait	bit	do not send a reply method
cancel-ok	31		confirm a cancelled consumer		Y	consumer-tag	consumer-tag	
publish	40		publish a message	Y		ticket	access-ticket	
						exchange	exchange-name	
						routing-key	shortstr	Message routing key
						mandatory	bit	indicate mandatory routing
						immediate	bit	request immediate delivery
return	50		return a failed message		Y	reply-code	reply-code	
						reply-text	reply-text	
						exchange	exchange-name	
						routing-key	shortstr	Message routing key
deliver	60		notify the client of a consumer message		Y	consumer-tag	consumer-tag	
						delivery-tag	delivery-tag	
						redelivered	redelivered	
						exchange	exchange-name	
						routing-key	shortstr	Message routing key

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
get	70	get-ok	direct access to a queue	Y		ticket	access-ticket	
						queue	queue-name	
						no-ack	no-ack	
get-ok	71		provide client with a message		Y	delivery-tag	delivery-tag	
						redelivered	redelivered	
						exchange	exchange-name	
						routing-key	shortstr	Message routing key
						message-count	long	number of messages pending
get-empty	72		indicate no messages available		Y	cluster-id	shortstr	Cluster id
ack	80		acknowledge one or more messages	Y		delivery-tag	delivery-tag	
						multiple	bit	acknowledge multiple messages
reject	90		reject an incoming message	Y		delivery-tag	delivery-tag	
						requeue	bit	requeue the message
recover	100		redeliver unacknowledged messages	Y		requeue	bit	requeue the message

1.9.2. Properties

1.9.2.1. Property basic.content-type (shortstr)

Domain: shortstr

Label: MIME content type

1.9.2.2. Property basic.content-encoding (shortstr)

Domain: shortstr

Label: MIME content encoding

1.9.2.3. Property basic.headers (table)

Domain: table

Label: message header field table

1.9.2.4. Property basic.delivery-mode (octet)

Domain: octet

Label: non-persistent (1) or persistent (2)

1.9.2.5. Property basic.priority (octet)

Domain: octet

Label: message priority, 0 to 9

1.9.2.6. Property basic.correlation-id (shortstr)

Domain: shortstr

Label: application correlation identifier

1.9.2.7. Property basic.reply-to (shortstr)

Domain: shortstr

Label: destination to reply to

1.9.2.8. Property basic.expiration (shortstr)

Domain: shortstr

Label: message expiration specification

1.9.2.9. Property basic.message-id (shortstr)

Domain: shortstr

Label: application message identifier

1.9.2.10. Property basic.timestamp (timestamp)

Domain: timestamp

Label: message timestamp

1.9.2.11. Property basic.type (shortstr)

Domain: shortstr

Label: message type name

1.9.2.12. Property basic.user-id (shortstr)

Domain: shortstr

Label: creating user id

1.9.2.13. Property basic.app-id (shortstr)

Domain: shortstr

Label: creating application id

1.9.2.14. Property basic.cluster-id (shortstr)

Domain: shortstr

Label: intra-cluster routing identifier

1.9.3. Methods

1.9.3.1. Method basic.qos (ID 10)

ID: 10

Method accepted by: Server

Synchronous: Yes; expected response is from method ***basic.qos-ok***

Number of parameters: 3

Label: specify quality of service

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	prefetch-size	long	prefetch window in octets
2	prefetch-count	short	prefetch window in messages
3	global	bit	apply to entire connection

This method requests a specific quality of service. The QoS can be specified for the current channel or for all channels on the connection. The particular properties and semantics of a qos method always depend on the content class semantics. Though the qos method could in principle apply to both peers, it is currently meaningful only for the server.

1.9.3.1.1. Parameter basic.qos.prefetch-size (long)

Ordinal: 1

Domain: long

Label: prefetch window in octets

The client can request that messages be sent in advance so that when the client finishes processing a message, the following message is already held locally, rather than needing to be sent down the channel. Prefetching gives a performance improvement. This field specifies the prefetch window size in octets. The server will send a message in advance if it is equal to or smaller in size than the available prefetch size (and also falls into other prefetch limits). May be set to zero, meaning "no specific limit", although other prefetch limits may still apply. The prefetch-size is ignored if the no-ack option is set.

1.9.3.1.2. Parameter basic.qos.prefetch-count (short)

Ordinal: 2

Domain: short

Label: prefetch window in messages

Specifies a prefetch window in terms of whole messages. This field may be used in combination with the prefetch-size field; a message will only be sent in advance if both prefetch windows (and those at the channel and connection level) allow it. The prefetch-count is ignored if the no-ack option is set.

1.9.3.1.3. Parameter basic.qos.global (bit)

Ordinal: 3

Domain: bit

Label: apply to entire connection

By default the QoS settings apply to the current channel only. If this field is set, they are applied to the entire connection.

1.9.3.2. Method basic.qos-ok (ID 11)

ID: 11

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm the requested qos

This method tells the client that the requested QoS levels could be handled by the server. The requested QoS applies to all active consumers until a new QoS is defined.

1.9.3.3. Method basic.consume (ID 20)

ID: 20

Method accepted by: Server

Synchronous: Yes; expected response is from method ***basic.consume-ok***

Number of parameters: 8

Label: start a queue consumer

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	queue	queue-name	
3	consumer-tag	consumer-tag	
4	no-local	no-local	
5	no-ack	no-ack	
6	exclusive	bit	request exclusive access
7	nowait	bit	do not send a reply method
8	filter	table	arguments for consuming

This method asks the server to start a "consumer", which is a transient request for messages from a specific queue. Consumers last as long as the channel they were created on, or until the client cancels them.

Guidelines for implementers:

- The server SHOULD support at least 16 consumers per queue, and ideally, impose no limit except as defined by available resources.

  Test scenario: Create a queue and create consumers on that queue until the server closes the connection. Verify that the number of consumers created was at least sixteen and report the total number.

1.9.3.3.1. Parameter basic.consume.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

1.9.3.3.2. Parameter basic.consume.queue (queue-name)

Ordinal: 2

Domain: queue-name

Specifies the name of the queue to consume from. If the queue name is null, refers to the current queue for the channel, which is the last declared queue.

1.9.3.3.3. Parameter basic.consume.consumer-tag (consumer-tag)

Ordinal: 3

Domain: consumer-tag

Specifies the identifier for the consumer. The consumer tag is local to a connection, so two clients can use the same consumer tags. If this field is empty the server will generate a unique tag.

1.9.3.3.4. Parameter basic.consume.no-local (no-local)

Ordinal: 4

Domain: no-local

1.9.3.3.5. Parameter basic.consume.no-ack (no-ack)

Ordinal: 5

Domain: no-ack

1.9.3.3.6. Parameter basic.consume.exclusive (bit)

Ordinal: 6

Domain: bit

Label: request exclusive access

Request exclusive consumer access, meaning only this consumer can access the queue.

1.9.3.3.7. Parameter basic.consume.nowait (bit)

Ordinal: 7

Domain: bit

Label: do not send a reply method

If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

1.9.3.3.8. Parameter basic.consume.filter (table)

Ordinal: 8

Domain: table

Label: arguments for consuming

A set of filters for the consume. The syntax and semantics of these filters depends on the providers implementation.

1.9.3.4. Method basic.consume-ok (ID 21)

ID: 21

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: confirm a new consumer

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	consumer-tag	consumer-tag	

The server provides the client with a consumer tag, which is used by the client for methods called on the consumer at a later stage.

1.9.3.4.1. Parameter basic.consume-ok.consumer-tag (consumer-tag)

Ordinal: 1

Domain: consumer-tag

Holds the consumer tag specified by the client or provided by the server.

1.9.3.5. Method basic.cancel (ID 30)

ID: 30

Method accepted by: Server

Synchronous: Yes; expected response is from method ***basic.cancel-ok***

Number of parameters: 2

Label: end a queue consumer

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	consumer-tag	consumer-tag	

2	nowait	bit	do not send a reply method

This method cancels a consumer. This does not affect already delivered messages, but it does mean the server will not send any more messages for that consumer. The client may receive an arbitrary number of messages in between sending the cancel method and receiving the cancel-ok reply.

Guidelines for implementers:

- If the queue does not exist the server MUST ignore the cancel method, so long as the consumer tag is valid for that channel.

  Test scenario: TODO.

1.9.3.5.1. Parameter basic.cancel.consumer-tag (consumer-tag)

Ordinal: 1

Domain: consumer-tag

1.9.3.5.2. Parameter basic.cancel.nowait (bit)

Ordinal: 2

Domain: bit

Label: do not send a reply method

If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

1.9.3.6. Method basic.cancel-ok (ID 31)

ID: 31

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: confirm a cancelled consumer

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	consumer-tag	consumer-tag	

This method confirms that the cancellation was completed.

1.9.3.6.1. Parameter basic.cancel-ok.consumer-tag (consumer-tag)

Ordinal: 1

Domain: consumer-tag

1.9.3.7. Method basic.publish (ID 40)

ID: 40

Method accepted by: Server

Synchronous: No

Number of parameters: 5

Label: publish a message

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	exchange	exchange-name	
3	routing-key	shortstr	Message routing key
4	mandatory	bit	indicate mandatory routing
5	immediate	bit	request immediate delivery

This method publishes a message to a specific exchange. The message will be routed to queues as defined by the exchange configuration and distributed to any active consumers when the transaction, if any, is committed.

1.9.3.7.1. Parameter basic.publish.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

1.9.3.7.2. Parameter basic.publish.exchange (exchange-name)

Ordinal: 2

Domain: exchange-name

Specifies the name of the exchange to publish to. The exchange name can be empty, meaning the default exchange. If the exchange name is specified, and that exchange does not exist, the server will raise a channel exception.

1.9.3.7.3. Parameter basic.publish.routing-key (shortstr)

Ordinal: 3

Domain: shortstr

Label: Message routing key

Specifies the routing key for the message. The routing key is used for routing messages depending on the exchange configuration.

1.9.3.7.4. Parameter basic.publish.mandatory (bit)

Ordinal: 4

Domain: bit

Label: indicate mandatory routing

This flag tells the server how to react if the message cannot be routed to a queue. If this flag is set, the server will return an unroutable message with a Return method. If this flag is zero, the server silently drops the message.

1.9.3.7.5. Parameter basic.publish.immediate (bit)

Ordinal: 5

Domain: bit

Label: request immediate delivery

This flag tells the server how to react if the message cannot be routed to a queue consumer immediately. If this flag is set, the server will return an undeliverable message with a Return method. If this flag is zero, the server will queue the message, but with no guarantee that it will ever be consumed.

1.9.3.8. Method basic.return (ID 50)

ID: 50

Method accepted by: Client

Synchronous: No

Number of parameters: 4

Label: return a failed message

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	reply-code	reply-code	

2	reply-text	reply-text	
3	exchange	exchange-name	
4	routing-key	shortstr	Message routing key

This method returns an undeliverable message that was published with the "immediate" flag set, or an unroutable message published with the "mandatory" flag set. The reply code and text provide information about the reason that the message was undeliverable.

1.9.3.8.1. Parameter basic.return.reply-code (reply-code)

Ordinal: 1

Domain: reply-code

1.9.3.8.2. Parameter basic.return.reply-text (reply-text)

Ordinal: 2

Domain: reply-text

1.9.3.8.3. Parameter basic.return.exchange (exchange-name)

Ordinal: 3

Domain: exchange-name

Specifies the name of the exchange that the message was originally published to.

1.9.3.8.4. Parameter basic.return.routing-key (shortstr)

Ordinal: 4

Domain: shortstr

Label: Message routing key

Specifies the routing key name specified when the message was published.

1.9.3.9. Method basic.deliver (ID 60)

ID: 60

Method accepted by: Client

Synchronous: No

Number of parameters: 5

Label: notify the client of a consumer message

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	consumer-tag	consumer-tag	
2	delivery-tag	delivery-tag	
3	redelivered	redelivered	
4	exchange	exchange-name	
5	routing-key	shortstr	Message routing key

This method delivers a message to the client, via a consumer. In the asynchronous message delivery model, the client starts a consumer using the Consume method, then the server responds with Deliver methods as and when messages arrive for that consumer.

Guidelines for implementers:

- 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.

  Test scenario: TODO.

1.9.3.9.1. Parameter basic.deliver.consumer-tag (consumer-tag)

Ordinal: 1

Domain: consumer-tag

1.9.3.9.2. Parameter basic.deliver.delivery-tag (delivery-tag)

Ordinal: 2

Domain: delivery-tag

1.9.3.9.3. Parameter basic.deliver.redelivered (redelivered)

Ordinal: 3

Domain: redelivered

1.9.3.9.4. Parameter basic.deliver.exchange (exchange-name)

Ordinal: 4

Domain: exchange-name

Specifies the name of the exchange that the message was originally published to.

1.9.3.9.5. Parameter basic.deliver.routing-key (shortstr)

Ordinal: 5

Domain: shortstr

Label: Message routing key

Specifies the routing key name specified when the message was published.

1.9.3.10. Method basic.get (ID 70)

ID: 70

Method accepted by: Server

Synchronous: Yes; expected response is from method ***basic.get-ok***

Number of parameters: 3

Label: direct access to a queue

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	queue	queue-name	
3	no-ack	no-ack	

This method provides a direct access to the messages in a queue using a synchronous dialogue that is designed for specific types of application where synchronous functionality is more important than performance.

1.9.3.10.1. Parameter basic.get.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

1.9.3.10.2. Parameter basic.get.queue (queue-name)

Ordinal: 2

Domain: queue-name

Specifies the name of the queue to consume from. If the queue name is null, refers to the current queue for the channel, which is the last declared queue.

1.9.3.10.3. Parameter basic.get.no-ack (no-ack)

Ordinal: 3

Domain: no-ack

1.9.3.11. Method basic.get-ok (ID 71)

ID: 71

Method accepted by: Client

Synchronous: No

Number of parameters: 5

Label: provide client with a message

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	delivery-tag	delivery-tag	
2	redelivered	redelivered	
3	exchange	exchange-name	
4	routing-key	shortstr	Message routing key
5	message-count	long	number of messages pending

This method delivers a message to the client following a get method. A message delivered by 'get-ok' must be acknowledged unless the no-ack option was set in the get method.

1.9.3.11.1. Parameter basic.get-ok.delivery-tag (delivery-tag)

Ordinal: 1

Domain: delivery-tag

1.9.3.11.2. Parameter basic.get-ok.redelivered (redelivered)

Ordinal: 2

Domain: redelivered

1.9.3.11.3. Parameter basic.get-ok.exchange (exchange-name)

Ordinal: 3

Domain: exchange-name

Specifies the name of the exchange that the message was originally published to. If empty, the message was published to the default exchange.

1.9.3.11.4. Parameter basic.get-ok.routing-key (shortstr)

Ordinal: 4

Domain: shortstr

Label: Message routing key

Specifies the routing key name specified when the message was published.

1.9.3.11.5. Parameter basic.get-ok.message-count (long)

Ordinal: 5

Domain: long

Label: number of messages pending

This field reports the number of messages pending on the queue, excluding the message being delivered. Note that this figure is indicative, not reliable, and can change arbitrarily as messages are added to the queue and removed by other clients.

1.9.3.12. Method basic.get-empty (ID 72)

ID: 72

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: indicate no messages available

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	cluster-id	shortstr	Cluster id

This method tells the client that the queue has no messages available for the client.

1.9.3.12.1. Parameter basic.get-empty.cluster-id (shortstr)

Ordinal: 1

Domain: shortstr

Label: Cluster id

For use by cluster applications, should not be used by client applications.

1.9.3.13. Method basic.ack (ID 80)

ID: 80

Method accepted by: Server

Synchronous: No

Number of parameters: 2

Label: acknowledge one or more messages

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	delivery-tag	delivery-tag	
2	multiple	bit	acknowledge multiple messages

This method acknowledges one or more messages delivered via the Deliver or Get-Ok methods. The client can ask to confirm a single message or a set of messages up to and including a specific message.

1.9.3.13.1. Parameter basic.ack.delivery-tag (delivery-tag)

Ordinal: 1

Domain: delivery-tag

1.9.3.13.2. Parameter basic.ack.multiple (bit)

Ordinal: 2

Domain: bit

Label: acknowledge multiple messages

If set to 1, the delivery tag is treated as "up to and including", so that the client can acknowledge multiple messages with a single method. If set to zero, the delivery tag refers to a single message. If the multiple field is 1, and the delivery tag is zero, tells the server to acknowledge all outstanding messages.

1.9.3.14. Method basic.reject (ID 90)

ID: 90

Method accepted by: Server

Synchronous: No

Number of parameters: 2

Label: reject an incoming message

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	delivery-tag	delivery-tag	
2	requeue	bit	requeue the message

This method allows a client to reject a message. It can be used to interrupt and cancel large incoming messages, or return untreatable messages to their original queue.

Guidelines for implementers:

- 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).
- The server SHOULD interpret this method as meaning that the client is unable to process the message at this time.

  Test scenario: TODO.
- A client MUST NOT use this method as a means of selecting messages to process. A rejected message MAY be discarded or dead-lettered, not necessarily passed to another client.

  Test scenario: TODO.

1.9.3.14.1. Parameter basic.reject.delivery-tag (delivery-tag)

Ordinal: 1

Domain: delivery-tag

1.9.3.14.2. Parameter basic.reject.requeue (bit)

Ordinal: 2

Domain: bit

Label: requeue the message

If this field is zero, the message will be discarded. If this bit is 1, the server will attempt to requeue the message.

1.9.3.15. Method basic.recover (ID 100)

ID: 100

Method accepted by: Server

Synchronous: No

Number of parameters: 1

Label: redeliver unacknowledged messages

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	requeue	bit	requeue the message

This method asks the broker to redeliver all unacknowledged messages on a specified channel. Zero or more messages may be redelivered. This method is only allowed on non-transacted channels.

Guidelines for implementers:

- The server MUST set the redelivered flag on all messages that are resent.

  Test scenario: TODO.

- The server MUST raise a channel exception if this is called on a transacted channel.

  Test scenario: TODO.

1.9.3.15.1. Parameter basic.recover.requeue (bit)

Ordinal: 1

Domain: bit

Label: requeue the message

If this field is zero, the message will be redelivered to the original recipient. If this bit is 1, the server will attempt to requeue the message, potentially then delivering it to an alternative subscriber.

1.10. Class file

The file class provides methods that support reliable file transfer. File messages have a specific set of properties that are required for interoperability with file transfer applications. File messages and acknowledgements are subject to channel transactions. Note that the file class does not provide message browsing methods; these are not compatible with the staging model. Applications that need browsable file transfer should use Basic content and the Basic class.

Class Grammar:

```
file                 = C:QOS S:QOS-OK
                     / C:CONSUME S:CONSUME-OK
                     / C:CANCEL S:CANCEL-OK
                     / C:OPEN S:OPEN-OK C:STAGE content
                     / S:OPEN C:OPEN-OK S:STAGE content
                     / C:PUBLISH
                     / S:DELIVER
                     / S:RETURN
                     / C:ACK
                     / C:REJECT
```

Guidelines for implementers:

- The server MUST make a best-effort to hold file messages on a reliable storage mechanism.
- The server MUST NOT discard a file message in case of a queue overflow. The server MUST use the Channel.Flow method to slow or stop a file message publisher when necessary.
- The server MUST implement at least 2 priority levels for file messages, where priorities 0-4 and 5-9 are treated as two distinct levels. The server MAY implement up to 10 priority levels.
- The server MUST support both automatic and explicit acknowledgements on file content.

1.10.1. Property and Method Summary

Class ***file*** defines the following properties:

Name	Domain	Short Description
content-type	shortstr	MIME content type
content-encoding	shortstr	MIME content encoding
headers	table	message header field table
priority	octet	message priority, 0 to 9
reply-to	shortstr	destination to reply to
message-id	shortstr	application message identifier
filename	shortstr	message filename
timestamp	timestamp	message timestamp
cluster-id	shortstr	intra-cluster routing identifier

Class ***file*** defines the following methods (S = received by server; C = received by client):

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
qos	10	qos-ok	specify quality of	Y		prefetch-size	long	prefetch window in octets

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
			service			prefetch-count	short	prefetch window in messages
						global	bit	apply to entire connection
qos-ok	11		confirm the requested qos		Y	[No parameters defined for this method]		
consume	20	consume-ok	start a queue consumer	Y		ticket	access-ticket	
						queue	queue-name	
						consumer-tag	consumer-tag	
						no-local	no-local	
						no-ack	no-ack	
						exclusive	bit	request exclusive access
						nowait	bit	do not send a reply method
						filter	table	arguments for consuming
consume-ok	21		confirm a new consumer		Y	consumer-tag	consumer-tag	
cancel	30	cancel-ok	end a queue consumer	Y		consumer-tag	consumer-tag	
						nowait	bit	do not send a reply method
cancel-ok	31		confirm a cancelled consumer		Y	consumer-tag	consumer-tag	
open	40	open-ok	request to start staging	Y	Y	identifier	shortstr	staging identifier
						content-size	longlong	message content size
open-ok	41	stage	confirm staging ready	Y	Y	staged-size	longlong	already staged amount
stage	50		stage message content	Y	Y	[No parameters defined for this method]		
publish	60		publish a message	Y		ticket	access-ticket	
						exchange	exchange-name	
						routing-key	shortstr	Message routing key
						mandatory	bit	indicate mandatory routing
						immediate	bit	request immediate delivery
						identifier	shortstr	staging identifier
return	70		return a failed message		Y	reply-code	reply-code	
						reply-text	reply-text	
						exchange	exchange-name	

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
						routing-key	shortstr	Message routing key
deliver	80		notify the client of a consumer message		Y	consumer-tag	consumer-tag	
						delivery-tag	delivery-tag	
						redelivered	redelivered	
						exchange	exchange-name	
						routing-key	shortstr	Message routing key
						identifier	shortstr	staging identifier
ack	90		acknowledge one or more messages	Y		delivery-tag	delivery-tag	
						multiple	bit	acknowledge multiple messages
reject	100		reject an incoming message	Y		delivery-tag	delivery-tag	
						requeue	bit	requeue the message

1.10.2. Properties

1.10.2.1. Property file.content-type (shortstr)

Domain: shortstr

Label: MIME content type

1.10.2.2. Property file.content-encoding (shortstr)

Domain: shortstr

Label: MIME content encoding

1.10.2.3. Property file.headers (table)

Domain: table

Label: message header field table

1.10.2.4. Property file.priority (octet)

Domain: octet

Label: message priority, 0 to 9

1.10.2.5. Property file.reply-to (shortstr)

Domain: shortstr

Label: destination to reply to

1.10.2.6. Property file.message-id (shortstr)

Domain: shortstr

Label: application message identifier

1.10.2.7. Property file.filename (shortstr)

Domain: shortstr

Label: message filename

1.10.2.8. Property file.timestamp (timestamp)

Domain: timestamp

Label: message timestamp

1.10.2.9. Property file.cluster-id (shortstr)

Domain: shortstr

Label: intra-cluster routing identifier

1.10.3. Methods

1.10.3.1. Method file.qos (ID 10)

ID: 10

Method accepted by: Server

Synchronous: Yes; expected response is from method ***file.qos-ok***

Number of parameters: 3

Label: specify quality of service

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	prefetch-size	long	prefetch window in octets
2	prefetch-count	short	prefetch window in messages

3	global	bit	apply to entire connection

This method requests a specific quality of service. The QoS can be specified for the current channel or for all channels on the connection. The particular properties and semantics of a qos method always depend on the content class semantics. Though the qos method could in principle apply to both peers, it is currently meaningful only for the server.

1.10.3.1.1. Parameter file.qos.prefetch-size (long)

Ordinal: 1

Domain: long

Label: prefetch window in octets

The client can request that messages be sent in advance so that when the client finishes processing a message, the following message is already held locally, rather than needing to be sent down the channel. Prefetching gives a performance improvement. This field specifies the prefetch window size in octets. May be set to zero, meaning "no specific limit". Note that other prefetch limits may still apply. The prefetch-size is ignored if the no-ack option is set.

1.10.3.1.2. Parameter file.qos.prefetch-count (short)

Ordinal: 2

Domain: short

Label: prefetch window in messages

Specifies a prefetch window in terms of whole messages. This is compatible with some file API implementations. This field may be used in combination with the prefetch-size field; a message will only be sent in advance if both prefetch windows (and those at the channel and connection level) allow it. The prefetch-count is ignored if the no-ack option is set.

1.10.3.1.3. Parameter file.qos.global (bit)

Ordinal: 3

Domain: bit

Label: apply to entire connection

By default the QoS settings apply to the current channel only. If this field is set, they are applied to the entire connection.

1.10.3.2. Method file.qos-ok (ID 11)

ID: 11

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm the requested qos

This method tells the client that the requested QoS levels could be handled by the server. The requested QoS applies to all active consumers until a new QoS is defined.

1.10.3.3. Method file.consume (ID 20)

ID: 20

Method accepted by: Server

Synchronous: Yes; expected response is from method ***file.consume-ok***

Number of parameters: 8

Label: start a queue consumer

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	queue	queue-name	
3	consumer-tag	consumer-tag	
4	no-local	no-local	
5	no-ack	no-ack	
6	exclusive	bit	request exclusive access
7	nowait	bit	do not send a reply method
8	filter	table	arguments for consuming

This method asks the server to start a "consumer", which is a transient request for messages from a specific queue. Consumers last as long as the channel they were created on, or until the client cancels them.

Guidelines for implementers:

- The server SHOULD support at least 16 consumers per queue, unless the queue was declared as private, and ideally, impose no limit except as defined by available resources.

1.10.3.3.1. Parameter file.consume.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

1.10.3.3.2. Parameter file.consume.queue (queue-name)

Ordinal: 2

Domain: queue-name

Specifies the name of the queue to consume from. If the queue name is null, refers to the current queue for the channel, which is the last declared queue.

1.10.3.3.3. Parameter file.consume.consumer-tag (consumer-tag)

Ordinal: 3

Domain: consumer-tag

Specifies the identifier for the consumer. The consumer tag is local to a connection, so two clients can use the same consumer tags. If this field is empty the server will generate a unique tag.

1.10.3.3.4. Parameter file.consume.no-local (no-local)

Ordinal: 4

Domain: no-local

1.10.3.3.5. Parameter file.consume.no-ack (no-ack)

Ordinal: 5

Domain: no-ack

1.10.3.3.6. Parameter file.consume.exclusive (bit)

Ordinal: 6

Domain: bit

Label: request exclusive access

Request exclusive consumer access, meaning only this consumer can access the queue.

1.10.3.3.7. Parameter file.consume.nowait (bit)

Ordinal: 7

Domain: bit

Label: do not send a reply method

If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

1.10.3.3.8. Parameter file.consume.filter (table)

Ordinal: 8

Domain: table

Label: arguments for consuming

A set of filters for the consume. The syntax and semantics of these filters depends on the providers implementation.

1.10.3.4. Method file.consume-ok (ID 21)

ID: 21

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: confirm a new consumer

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	consumer-tag	consumer-tag	

This method provides the client with a consumer tag which it MUST use in methods that work with the consumer.

1.10.3.4.1. Parameter file.consume-ok.consumer-tag (consumer-tag)

Ordinal: 1

Domain: consumer-tag

Holds the consumer tag specified by the client or provided by the server.

1.10.3.5. Method file.cancel (ID 30)

ID: 30

Method accepted by: Server

Synchronous: Yes; expected response is from method ***file.cancel-ok***

Number of parameters: 2

Label: end a queue consumer

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	consumer-tag	consumer-tag	
2	nowait	bit	do not send a reply method

This method cancels a consumer. This does not affect already delivered messages, but it does mean the server will not send any more messages for that consumer.

1.10.3.5.1. Parameter file.cancel.consumer-tag (consumer-tag)

Ordinal: 1

Domain: consumer-tag

1.10.3.5.2. Parameter file.cancel.nowait (bit)

Ordinal: 2

Domain: bit

Label: do not send a reply method

If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

1.10.3.6. Method file.cancel-ok (ID 31)

ID: 31

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: confirm a cancelled consumer

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	consumer-tag	consumer-tag	

This method confirms that the cancellation was completed.

1.10.3.6.1. Parameter file.cancel-ok.consumer-tag (consumer-tag)

Ordinal: 1

Domain: consumer-tag

1.10.3.7. Method file.open (ID 40)

ID: 40

Method accepted by: Server, Client

Synchronous: Yes; expected response is from method ***file.open-ok***

Number of parameters: 2

Label: request to start staging

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	identifier	shortstr	staging identifier
2	content-size	longlong	message content size

This method requests permission to start staging a message. Staging means sending the message into a temporary area at the recipient end and then delivering the message by referring to this temporary area. Staging is how the protocol handles partial file transfers - if a message is partially staged and the connection breaks, the next time the sender starts to stage it, it can restart from where it left off.

1.10.3.7.1. Parameter file.open.identifier (shortstr)

Ordinal: 1

Domain: shortstr

Label: staging identifier

This is the staging identifier. This is an arbitrary string chosen by the sender. For staging to work correctly the sender must use the same staging identifier when staging the same message a second time after recovery from a failure. A good choice for the staging identifier would be the SHA1 hash of the message properties data (including the original filename, revised time, etc.).

1.10.3.7.2. Parameter file.open.content-size (longlong)

Ordinal: 2

Domain: longlong

Label: message content size

The size of the content in octets. The recipient may use this information to allocate or check available space in advance, to avoid "disk full" errors during staging of very large messages.

1.10.3.8. Method file.open-ok (ID 41)

ID: 41

Method accepted by: Server, Client

Synchronous: Yes; expected response is from method ***file.stage***

Number of parameters: 1

Label: confirm staging ready

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	staged-size	longlong	already staged amount

This method confirms that the recipient is ready to accept staged data. If the message was already partially-staged at a previous time the recipient will report the number of octets already staged.

1.10.3.8.1. Parameter file.open-ok.staged-size (longlong)

Ordinal: 1

Domain: longlong

Label: already staged amount

The amount of previously-staged content in octets. For a new message this will be zero.

1.10.3.9. Method file.stage (ID 50)

ID: 50

Method accepted by: Server, Client

Synchronous: No

Number of parameters: 0

Label: stage message content

This method stages the message, sending the message content to the recipient from the octet offset specified in the Open-Ok method.

1.10.3.10. Method file.publish (ID 60)

ID: 60

Method accepted by: Server

Synchronous: No

Number of parameters: 6

Label: publish a message

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	exchange	exchange-name	
3	routing-key	shortstr	Message routing key
4	mandatory	bit	indicate mandatory routing
5	immediate	bit	request immediate delivery
6	identifier	shortstr	staging identifier

This method publishes a staged file message to a specific exchange. The file message will be routed to queues as defined by the exchange configuration and distributed to any active consumers when the transaction, if any, is committed.

1.10.3.10.1. Parameter file.publish.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

1.10.3.10.2. Parameter file.publish.exchange (exchange-name)

Ordinal: 2

Domain: exchange-name

Specifies the name of the exchange to publish to. The exchange name can be empty, meaning the default exchange. If the exchange name is specified, and that exchange does not exist, the server will raise a channel exception.

1.10.3.10.3. Parameter file.publish.routing-key (shortstr)

Ordinal: 3

Domain: shortstr

Label: Message routing key

Specifies the routing key for the message. The routing key is used for routing messages depending on the

exchange configuration.

1.10.3.10.4. Parameter file.publish.mandatory (bit)

Ordinal: 4

Domain: bit

Label: indicate mandatory routing

This flag tells the server how to react if the message cannot be routed to a queue. If this flag is set, the server will return an unroutable message with a Return method. If this flag is zero, the server silently drops the message.

1.10.3.10.5. Parameter file.publish.immediate (bit)

Ordinal: 5

Domain: bit

Label: request immediate delivery

This flag tells the server how to react if the message cannot be routed to a queue consumer immediately. If this flag is set, the server will return an undeliverable message with a Return method. If this flag is zero, the server will queue the message, but with no guarantee that it will ever be consumed.

1.10.3.10.6. Parameter file.publish.identifier (shortstr)

Ordinal: 6

Domain: shortstr

Label: staging identifier

This is the staging identifier of the message to publish. The message must have been staged. Note that a client can send the Publish method asynchronously without waiting for staging to finish.

1.10.3.11. Method file.return (ID 70)

ID: 70

Method accepted by: Client

Synchronous: No

Number of parameters: 4

Label: return a failed message

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description

1	reply-code	reply-code	
2	reply-text	reply-text	
3	exchange	exchange-name	
4	routing-key	shortstr	Message routing key

This method returns an undeliverable message that was published with the "immediate" flag set, or an unroutable message published with the "mandatory" flag set. The reply code and text provide information about the reason that the message was undeliverable.

1.10.3.11.1. Parameter file.return.reply-code (reply-code)

Ordinal: 1

Domain: reply-code

1.10.3.11.2. Parameter file.return.reply-text (reply-text)

Ordinal: 2

Domain: reply-text

1.10.3.11.3. Parameter file.return.exchange (exchange-name)

Ordinal: 3

Domain: exchange-name

Specifies the name of the exchange that the message was originally published to.

1.10.3.11.4. Parameter file.return.routing-key (shortstr)

Ordinal: 4

Domain: shortstr

Label: Message routing key

Specifies the routing key name specified when the message was published.

1.10.3.12. Method file.deliver (ID 80)

ID: 80

Method accepted by: Client

Synchronous: No

Number of parameters: 6

Label: notify the client of a consumer message

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	consumer-tag	consumer-tag	
2	delivery-tag	delivery-tag	
3	redelivered	redelivered	
4	exchange	exchange-name	
5	routing-key	shortstr	Message routing key
6	identifier	shortstr	staging identifier

This method delivers a staged file message to the client, via a consumer. In the asynchronous message delivery model, the client starts a consumer using the Consume method, then the server responds with Deliver methods as and when messages arrive for that consumer.

Guidelines for implementers:

- 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.

1.10.3.12.1. Parameter file.deliver.consumer-tag (consumer-tag)

Ordinal: 1

Domain: consumer-tag

1.10.3.12.2. Parameter file.deliver.delivery-tag (delivery-tag)

Ordinal: 2

Domain: delivery-tag

1.10.3.12.3. Parameter file.deliver.redelivered (redelivered)

Ordinal: 3

Domain: redelivered

1.10.3.12.4. Parameter file.deliver.exchange (exchange-name)

Ordinal: 4

Domain: exchange-name

Specifies the name of the exchange that the message was originally published to.

1.10.3.12.5. Parameter file.deliver.routing-key (shortstr)

Ordinal: 5

Domain: shortstr

Label: Message routing key

Specifies the routing key name specified when the message was published.

1.10.3.12.6. Parameter file.deliver.identifier (shortstr)

Ordinal: 6

Domain: shortstr

Label: staging identifier

This is the staging identifier of the message to deliver. The message must have been staged. Note that a server can send the Deliver method asynchronously without waiting for staging to finish.

1.10.3.13. Method file.ack (ID 90)

ID: 90

Method accepted by: Server

Synchronous: No

Number of parameters: 2

Label: acknowledge one or more messages

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	delivery-tag	delivery-tag	
2	multiple	bit	acknowledge multiple messages

This method acknowledges one or more messages delivered via the Deliver method. The client can ask to confirm a single message or a set of messages up to and including a specific message.

1.10.3.13.1. Parameter file.ack.delivery-tag (delivery-tag)

Ordinal: 1

Domain: delivery-tag

1.10.3.13.2. Parameter file.ack.multiple (bit)

Ordinal: 2

Domain: bit

Label: acknowledge multiple messages

If set to 1, the delivery tag is treated as "up to and including", so that the client can acknowledge multiple messages with a single method. If set to zero, the delivery tag refers to a single message. If the multiple field is 1, and the delivery tag is zero, tells the server to acknowledge all outstanding messages.

1.10.3.14. Method file.reject (ID 100)

ID: 100

Method accepted by: Server

Synchronous: No

Number of parameters: 2

Label: reject an incoming message

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	delivery-tag	delivery-tag	
2	requeue	bit	requeue the message

This method allows a client to reject a message. It can be used to return untreatable messages to their original queue. Note that file content is staged before delivery, so the client will not use this method to interrupt delivery of a large message.

Guidelines for implementers:

- The server SHOULD interpret this method as meaning that the client is unable to process the message at this time.
- A client MUST NOT use this method as a means of selecting messages to process. A rejected message MAY be discarded or dead-lettered, not necessarily passed to another client.

1.10.3.14.1. Parameter file.reject.delivery-tag (delivery-tag)

Ordinal: 1

Domain: delivery-tag

1.10.3.14.2. Parameter file.reject.requeue (bit)

Ordinal: 2

Domain: bit

Label: requeue the message

If this field is zero, the message will be discarded. If this bit is 1, the server will attempt to requeue the message.

1.11. Class stream

The stream class provides methods that support multimedia streaming. The stream class uses the following semantics: one message is one packet of data; delivery is unacknowledged and unreliable; the consumer can specify quality of service parameters that the server can try to adhere to; lower-priority messages may be discarded in favour of high priority messages.

Class Grammar:

```
stream              = C:QOS S:QOS-OK
                    / C:CONSUME S:CONSUME-OK
                    / C:CANCEL S:CANCEL-OK
                    / C:PUBLISH content
                    / S:RETURN
                    / S:DELIVER content
```

Guidelines for implementers:

- The server SHOULD discard stream messages on a priority basis if the queue size exceeds some configured limit.
- The server MUST implement at least 2 priority levels for stream messages, where priorities 0-4 and 5-9 are treated as two distinct levels. The server MAY implement up to 10 priority levels.
- The server MUST implement automatic acknowledgements on stream content. That is, as soon as a message is delivered to a client via a Deliver method, the server must remove it from the queue.

1.11.1. Property and Method Summary

Class ***stream*** defines the following properties:

Name	Domain	Short Description
content-type	shortstr	MIME content type
content-encoding	shortstr	MIME content encoding
headers	table	message header field table

Name	Domain	Short Description
priority	octet	message priority, 0 to 9
timestamp	timestamp	message timestamp

Class ***stream*** defines the following methods (S = received by server; C = received by client):

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
qos	10	qos-ok	specify quality of service	Y		prefetch-size	long	prefetch window in octets
						prefetch-count	short	prefetch window in messages
						consume-rate	long	transfer rate in octets/second
						global	bit	apply to entire connection
qos-ok	11		confirm the requested qos		Y	[No parameters defined for this method]		
consume	20	consume-ok	start a queue consumer	Y		ticket	access-ticket	
						queue	queue-name	
						consumer-tag	consumer-tag	
						no-local	no-local	
						exclusive	bit	request exclusive access
						nowait	bit	do not send a reply method
						filter	table	arguments for consuming
consume-ok	21		confirm a new consumer		Y	consumer-tag	consumer-tag	
cancel	30	cancel-ok	end a queue consumer	Y		consumer-tag	consumer-tag	
						nowait	bit	do not send a reply method
cancel-ok	31		confirm a cancelled consumer		Y	consumer-tag	consumer-tag	
publish	40		publish a message	Y		ticket	access-ticket	
						exchange	exchange-name	
						routing-key	shortstr	Message routing key
						mandatory	bit	indicate mandatory routing
						immediate	bit	request immediate delivery
return	50		return a failed message		Y	reply-code	reply-code	
						reply-text	reply-text	
						exchange	exchange-name	

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
						routing-key	shortstr	Message routing key
deliver	60		notify the client of a consumer message		Y	consumer-tag	consumer-tag	
						delivery-tag	delivery-tag	
						exchange	exchange-name	
						queue	queue-name	

1.11.2. Properties

1.11.2.1. Property stream.content-type (shortstr)

Domain: shortstr

Label: MIME content type

1.11.2.2. Property stream.content-encoding (shortstr)

Domain: shortstr

Label: MIME content encoding

1.11.2.3. Property stream.headers (table)

Domain: table

Label: message header field table

1.11.2.4. Property stream.priority (octet)

Domain: octet

Label: message priority, 0 to 9

1.11.2.5. Property stream.timestamp (timestamp)

Domain: timestamp

Label: message timestamp

1.11.3. Methods

1.11.3.1. Method stream.qos (ID 10)

ID: 10

Method accepted by: Server

Synchronous: Yes; expected response is from method ***stream.qos-ok***

Number of parameters: 4

Label: specify quality of service

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	prefetch-size	long	prefetch window in octets
2	prefetch-count	short	prefetch window in messages
3	consume-rate	long	transfer rate in octets/second
4	global	bit	apply to entire connection

This method requests a specific quality of service. The QoS can be specified for the current channel or for all channels on the connection. The particular properties and semantics of a qos method always depend on the content class semantics. Though the qos method could in principle apply to both peers, it is currently meaningful only for the server.

1.11.3.1.1. Parameter stream.qos.prefetch-size (long)

Ordinal: 1

Domain: long

Label: prefetch window in octets

The client can request that messages be sent in advance so that when the client finishes processing a message, the following message is already held locally, rather than needing to be sent down the channel. Prefetching gives a performance improvement. This field specifies the prefetch window size in octets. May be set to zero, meaning "no specific limit". Note that other prefetch limits may still apply.

1.11.3.1.2. Parameter stream.qos.prefetch-count (short)

Ordinal: 2

Domain: short

Label: prefetch window in messages

Specifies a prefetch window in terms of whole messages. This field may be used in combination with the prefetch-size field; a message will only be sent in advance if both prefetch windows (and those at the channel and connection level) allow it.

1.11.3.1.3. Parameter stream.qos.consume-rate (long)

Ordinal: 3

Domain: long

Label: transfer rate in octets/second

Specifies a desired transfer rate in octets per second. This is usually determined by the application that uses the streaming data. A value of zero means "no limit", i.e. as rapidly as possible.

1.11.3.1.4. Parameter stream.qos.global (bit)

Ordinal: 4

Domain: bit

Label: apply to entire connection

By default the QoS settings apply to the current channel only. If this field is set, they are applied to the entire connection.

1.11.3.2. Method stream.qos-ok (ID 11)

ID: 11

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm the requested qos

This method tells the client that the requested QoS levels could be handled by the server. The requested QoS applies to all active consumers until a new QoS is defined.

1.11.3.3. Method stream.consume (ID 20)

ID: 20

Method accepted by: Server

Synchronous: Yes; expected response is from method ***stream.consume-ok***

Number of parameters: 7

Label: start a queue consumer

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	queue	queue-name	
3	consumer-tag	consumer-tag	
4	no-local	no-local	
5	exclusive	bit	request exclusive access
6	nowait	bit	do not send a reply method
7	filter	table	arguments for consuming

This method asks the server to start a "consumer", which is a transient request for messages from a specific queue. Consumers last as long as the channel they were created on, or until the client cancels them.

Guidelines for implementers:

- The server SHOULD support at least 16 consumers per queue, unless the queue was declared as private, and ideally, impose no limit except as defined by available resources.
- Streaming applications SHOULD use different channels to select different streaming resolutions. AMQP makes no provision for filtering and/or transforming streams except on the basis of priority-based selective delivery of individual messages.

1.11.3.3.1. Parameter stream.consume.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

1.11.3.3.2. Parameter stream.consume.queue (queue-name)

Ordinal: 2

Domain: queue-name

Specifies the name of the queue to consume from. If the queue name is null, refers to the current queue for the channel, which is the last declared queue.

1.11.3.3.3. Parameter stream.consume.consumer-tag (consumer-tag)

Ordinal: 3

Domain: consumer-tag

Specifies the identifier for the consumer. The consumer tag is local to a connection, so two clients can use

the same consumer tags. If this field is empty the server will generate a unique tag.

1.11.3.3.4. Parameter stream.consume.no-local (no-local)

Ordinal: 4

Domain: no-local

1.11.3.3.5. Parameter stream.consume.exclusive (bit)

Ordinal: 5

Domain: bit

Label: request exclusive access

Request exclusive consumer access, meaning only this consumer can access the queue.

1.11.3.3.6. Parameter stream.consume.nowait (bit)

Ordinal: 6

Domain: bit

Label: do not send a reply method

If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

1.11.3.3.7. Parameter stream.consume.filter (table)

Ordinal: 7

Domain: table

Label: arguments for consuming

A set of filters for the consume. The syntax and semantics of these filters depends on the providers implementation.

1.11.3.4. Method stream.consume-ok (ID 21)

ID: 21

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: confirm a new consumer

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	consumer-tag	consumer-tag	

This method provides the client with a consumer tag which it may use in methods that work with the consumer.

1.11.3.4.1. Parameter stream.consume-ok.consumer-tag (consumer-tag)

Ordinal: 1

Domain: consumer-tag

Holds the consumer tag specified by the client or provided by the server.

1.11.3.5. Method stream.cancel (ID 30)

ID: 30

Method accepted by: Server

Synchronous: Yes; expected response is from method ***stream.cancel-ok***

Number of parameters: 2

Label: end a queue consumer

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	consumer-tag	consumer-tag	
2	nowait	bit	do not send a reply method

This method cancels a consumer. Since message delivery is asynchronous the client may continue to receive messages for a short while after cancelling a consumer. It may process or discard these as appropriate.

1.11.3.5.1. Parameter stream.cancel.consumer-tag (consumer-tag)

Ordinal: 1

Domain: consumer-tag

1.11.3.5.2. Parameter stream.cancel.nowait (bit)

Ordinal: 2

Domain: bit

Label: do not send a reply method

If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

1.11.3.6. Method stream.cancel-ok (ID 31)

ID: 31

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: confirm a cancelled consumer

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	consumer-tag	consumer-tag	

This method confirms that the cancellation was completed.

1.11.3.6.1. Parameter stream.cancel-ok.consumer-tag (consumer-tag)

Ordinal: 1

Domain: consumer-tag

1.11.3.7. Method stream.publish (ID 40)

ID: 40

Method accepted by: Server

Synchronous: No

Number of parameters: 5

Label: publish a message

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	exchange	exchange-name	
3	routing-key	shortstr	Message routing key

4	mandatory	bit	indicate mandatory routing
5	immediate	bit	request immediate delivery

This method publishes a message to a specific exchange. The message will be routed to queues as defined by the exchange configuration and distributed to any active consumers as appropriate.

1.11.3.7.1. Parameter stream.publish.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

1.11.3.7.2. Parameter stream.publish.exchange (exchange-name)

Ordinal: 2

Domain: exchange-name

Specifies the name of the exchange to publish to. The exchange name can be empty, meaning the default exchange. If the exchange name is specified, and that exchange does not exist, the server will raise a channel exception.

1.11.3.7.3. Parameter stream.publish.routing-key (shortstr)

Ordinal: 3

Domain: shortstr

Label: Message routing key

Specifies the routing key for the message. The routing key is used for routing messages depending on the exchange configuration.

1.11.3.7.4. Parameter stream.publish.mandatory (bit)

Ordinal: 4

Domain: bit

Label: indicate mandatory routing

This flag tells the server how to react if the message cannot be routed to a queue. If this flag is set, the server will return an unroutable message with a Return method. If this flag is zero, the server silently drops the message.

1.11.3.7.5. Parameter stream.publish.immediate (bit)

Ordinal: 5

Domain: bit

Label: request immediate delivery

This flag tells the server how to react if the message cannot be routed to a queue consumer immediately. If this flag is set, the server will return an undeliverable message with a Return method. If this flag is zero, the server will queue the message, but with no guarantee that it will ever be consumed.

1.11.3.8. Method stream.return (ID 50)

ID: 50

Method accepted by: Client

Synchronous: No

Number of parameters: 4

Label: return a failed message

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	reply-code	reply-code	
2	reply-text	reply-text	
3	exchange	exchange-name	
4	routing-key	shortstr	Message routing key

This method returns an undeliverable message that was published with the "immediate" flag set, or an unroutable message published with the "mandatory" flag set. The reply code and text provide information about the reason that the message was undeliverable.

1.11.3.8.1. Parameter stream.return.reply-code (reply-code)

Ordinal: 1

Domain: reply-code

1.11.3.8.2. Parameter stream.return.reply-text (reply-text)

Ordinal: 2

Domain: reply-text

1.11.3.8.3. Parameter stream.return.exchange (exchange-name)

Ordinal: 3

Domain: exchange-name

Specifies the name of the exchange that the message was originally published to.

1.11.3.8.4. Parameter stream.return.routing-key (shortstr)

Ordinal: 4

Domain: shortstr

Label: Message routing key

Specifies the routing key name specified when the message was published.

1.11.3.9. Method stream.deliver (ID 60)

ID: 60

Method accepted by: Client

Synchronous: No

Number of parameters: 4

Label: notify the client of a consumer message

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	consumer-tag	consumer-tag	
2	delivery-tag	delivery-tag	
3	exchange	exchange-name	
4	queue	queue-name	

This method delivers a message to the client, via a consumer. In the asynchronous message delivery model, the client starts a consumer using the Consume method, then the server responds with Deliver methods as and when messages arrive for that consumer.

1.11.3.9.1. Parameter stream.deliver.consumer-tag (consumer-tag)

Ordinal: 1

Domain: consumer-tag

1.11.3.9.2. Parameter stream.deliver.delivery-tag (delivery-tag)

Ordinal: 2

Domain: delivery-tag

1.11.3.9.3. Parameter stream.deliver.exchange (exchange-name)

Ordinal: 3

Domain: exchange-name

Specifies the name of the exchange that the message was originally published to.

1.11.3.9.4. Parameter stream.deliver.queue (queue-name)

Ordinal: 4

Domain: queue-name

Specifies the name of the queue that the message came from. Note that a single channel can start many consumers on different queues.

1.12. Class tx

Standard transactions provide so-called "1.5 phase commit". We can ensure that work is never lost, but there is a chance of confirmations being lost, so that messages may be resent. Applications that use standard transactions must be able to detect and ignore duplicate messages.

Class Grammar:

```
tx                    = C:SELECT S:SELECT-OK
                      / C:COMMIT S:COMMIT-OK
                      / C:ROLLBACK S:ROLLBACK-OK
```

Guidelines for implementers:

- An client using standard transactions SHOULD be able to track all messages received within a reasonable period, and thus detect and reject duplicates of the same message. It SHOULD NOT pass these to the application layer.

1.12.1. Property and Method Summary

Class ***tx*** defines the following methods (S = received by server; C = received by client):

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
select	10	select-ok	select standard transaction	Y		[No parameters defined for this method]		

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
			mode					
select-ok	11		confirm transaction mode		Y	[No parameters defined for this method]		
commit	20	commit-ok	commit the current transaction	Y		[No parameters defined for this method]		
commit-ok	21		confirm a successful commit		Y	[No parameters defined for this method]		
rollback	30	rollback-ok	abandon the current transaction	Y		[No parameters defined for this method]		
rollback-ok	31		confirm successful rollback		Y	[No parameters defined for this method]		

1.12.2. Methods

1.12.2.1. Method tx.select (ID 10)

ID: 10

Method accepted by: Server

Synchronous: Yes; expected response is from method ***tx.select-ok***

Number of parameters: 0

Label: select standard transaction mode

This method sets the channel to use standard transactions. The client must use this method at least once on a channel before using the Commit or Rollback methods.

1.12.2.2. Method tx.select-ok (ID 11)

ID: 11

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm transaction mode

This method confirms to the client that the channel was successfully set to use standard transactions.

1.12.2.3. Method tx.commit (ID 20)

ID: 20

Method accepted by: Server

Synchronous: Yes; expected response is from method ***tx.commit-ok***

Number of parameters: 0

Label: commit the current transaction

This method commits all messages published and acknowledged in the current transaction. A new transaction starts immediately after a commit.

1.12.2.4. Method tx.commit-ok (ID 21)

ID: 21

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm a successful commit

This method confirms to the client that the commit succeeded. Note that if a commit fails, the server raises a channel exception.

1.12.2.5. Method tx.rollback (ID 30)

ID: 30

Method accepted by: Server

Synchronous: Yes; expected response is from method ***tx.rollback-ok***

Number of parameters: 0

Label: abandon the current transaction

This method abandons all messages published and acknowledged in the current transaction. A new transaction starts immediately after a rollback.

1.12.2.6. Method tx.rollback-ok (ID 31)

ID: 31

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm successful rollback

This method confirms to the client that the rollback succeeded. Note that if an rollback fails, the server raises a channel exception.

1.13. Class dtx

Distributed transactions provide so-called "2-phase commit". The AMQP distributed transaction model supports the X-Open XA architecture and other distributed transaction implementations. The Dtx class assumes that the server has a private communications channel (not AMQP) to a distributed transaction coordinator.

Class Grammar:

```
dtx                 = C:SELECT S:SELECT-OK
                      C:START S:START-OK
```

1.13.1. Property and Method Summary

Class ***dtx*** defines the following methods (S = received by server; C = received by client):

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
select	10	select-ok	select standard transaction mode	Y		[No parameters defined for this method]		
select-ok	11		confirm transaction mode		Y	[No parameters defined for this method]		
start	20	start-ok	start a new distributed transaction	Y		dtx-identifier	shortstr	transaction identifier
start-ok	21		confirm the start of a new distributed transaction		Y	[No parameters defined for this method]		

1.13.2. Methods

1.13.2.1. Method dtx.select (ID 10)

ID: 10

Method accepted by: Server

Synchronous: Yes; expected response is from method ***dtx.select-ok***

Number of parameters: 0

Label: select standard transaction mode

This method sets the channel to use distributed transactions. The client must use this method at least once on a channel before using the Start method.

1.13.2.2. Method dtx.select-ok (ID 11)

ID: 11

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm transaction mode

This method confirms to the client that the channel was successfully set to use distributed transactions.

1.13.2.3. Method dtx.start (ID 20)

ID: 20

Method accepted by: Server

Synchronous: Yes; expected response is from method ***dtx.start-ok***

Number of parameters: 1

Label: start a new distributed transaction

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	dtx-identifier	shortstr	transaction identifier

This method starts a new distributed transaction. This must be the first method on a new channel that uses the distributed transaction mode, before any methods that publish or consume messages.

1.13.2.3.1. Parameter dtx.start.dtx-identifier (shortstr)

Ordinal: 1

Domain: shortstr

Label: transaction identifier

The distributed transaction key. This identifies the transaction so that the AMQP server can coordinate with the distributed transaction coordinator.

1.13.2.4. Method dtx.start-ok (ID 21)

ID: 21

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm the start of a new distributed transaction

This method confirms to the client that the transaction started. Note that if a start fails, the server raises a channel exception.

1.14. Class tunnel

The tunnel methods are used to send blocks of binary data - which can be serialised AMQP methods or other protocol frames - between AMQP peers.

Class Grammar:

```
tunnel                = C:REQUEST
                      / S:REQUEST
```

1.14.1. Property and Method Summary

Class ***tunnel*** defines the following properties:

Name	Domain	Short Description
headers	table	message header field table
proxy-name	shortstr	identity of tunnelling proxy
data-name	shortstr	name or type of message being tunnelled
durable	octet	message durability indicator

Name	Domain	Short Description
broadcast	octet	message broadcast mode

Class ***tunnel*** defines the following methods (S = received by server; C = received by client):

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
request	10		sends a tunnelled method	Y		meta-data	table	meta data for the tunnelled block

1.14.2. Properties

1.14.2.1. Property tunnel.headers (table)

Domain: table

Label: message header field table

1.14.2.2. Property tunnel.proxy-name (shortstr)

Domain: shortstr

Label: identity of tunnelling proxy

1.14.2.3. Property tunnel.data-name (shortstr)

Domain: shortstr

Label: name or type of message being tunnelled

1.14.2.4. Property tunnel.durable (octet)

Domain: octet

Label: message durability indicator

1.14.2.5. Property tunnel.broadcast (octet)

Domain: octet

Label: message broadcast mode

1.14.3. Methods

1.14.3.1. Method tunnel.request (ID 10)

ID: 10

Method accepted by: Server

Synchronous: No

Number of parameters: 1

Label: sends a tunnelled method

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	meta-data	table	meta data for the tunnelled block

This method tunnels a block of binary data, which can be an encoded AMQP method or other data. The binary data is sent as the content for the Tunnel.Request method.

1.14.3.1.1. Parameter tunnel.request.meta-data (table)

Ordinal: 1

Domain: table

Label: meta data for the tunnelled block

This field table holds arbitrary meta-data that the sender needs to pass to the recipient.

1.15. Class message

[WORK IN PROGRESS] The message class provides methods that support an industry-standard messaging model.

Class Grammar:

```
message              = C:QOS S:OK
                     / C:CONSUME S:OK
                     / C:CANCEL S:OK
                     / C:TRANSFER ( S:OK / S:REJECT )
                     / S:TRANSFER ( C:OK / C:REJECT )
                     / C:GET ( S:OK / S:EMPTY )
                     / C:RECOVER S:OK
                     / C:OPEN S:OK
                     / S:OPEN C:OK
                     / C:APPEND S:OK
```

```
                        / S:APPEND C:OK
                        / C:CLOSE S:OK
                        / S:CLOSE C:OK
                        / C:CHECKPOINT S:OK
                        / S:CHECKPOINT C:OK
                        / C:RESUME S:OFFSET
                        / S:RESUME C:OFFSET
```

Guidelines for implementers:

- The server SHOULD respect the persistent property of messages and SHOULD make a best-effort to hold persistent mess ages on a reliable storage mechanism.

  Test scenario: Send a persistent message to queue, stop server, restart server and then verify whether message is still present. Assumes that queues are durable. Persistence without durable queues makes no sense.

- The server MUST NOT discard a persistent message in case of a queue overflow.

  Test scenario: Create a queue overflow situation with persistent messages and verify that messages do not get lost (presumably the server will write them to disk).

- The server MAY use the Channel.Flow method to slow or stop a message publisher when necessary.

  Test scenario: Create a queue overflow situation with non-persistent messages and verify whether the server responds with Channel.Flow or not. Repeat with persistent messages.

- The server MAY overflow non-persistent messages to persistent storage.

- The server MAY discard or dead-letter non-persistent messages on a priority basis if the queue size exceeds some configured limit.

- The server MUST implement at least 2 priority levels for messages, where priorities 0-4 and 5-9 are treated as two distinct levels.

  Test scenario: Send a number of priority 0 messages to a queue. Send one priority 9 message. Consume messages from the queue and verify that the first message received was priority 9.

- The server MAY implement up to 10 priority levels.

  Test scenario: Send a number of messages with mixed priorities to a queue, so that all priority values from 0 to 9 are exercised. A good scenario would be ten messages in low-to-high priority. Consume from queue and verify how many priority levels emerge.

- The server MUST deliver messages of the same priority in order irrespective of their individual persistence.

  Test scenario: Send a set of messages with the same priority but different persistence settings to a queue. Consume and verify that messages arrive in same order as originally published.

- The server MUST support automatic acknowledgements on messages, i.e. consumers with the no-ack field set to FALSE.

  Test scenario: Create a queue and a consumer using automatic acknowledgements. Publish a set of messages to the queue. Consume the messages and verify that all messages are received.

- The server MUST support explicit acknowledgements on messages, i.e. consumers with the no-ack field set to TRUE.

  Test scenario: Create a queue and a consumer using explicit acknowledgements. Publish a set of messages to the queue. Consume the messages but acknowledge only half of them. Disconnect and reconnect, and consume from the queue. Verify that the remaining messages are received.

1.15.1. Property and Method Summary

Class ***message*** defines the following methods (S = received by server; C = received by client):

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
transfer	10	ok	[WORK IN PROGRESS] transfer a message	Y	Y	ticket	access-ticket	
						destination	destination	
						redelivered	redelivered	
						immediate	bit	request immediate delivery
						ttl	duration	time to live
						priority	octet	message priority, 0 to 9
						timestamp	timestamp	message timestamp
						delivery-mode	octet	non-persistent (1) or persistent (2)
						expiration	timestamp	message expiration time
						exchange	exchange-name	originating exchange
						routing-key	shortstr	message routing key
						message-id	shortstr	application message identifier
						correlation-id	shortstr	application correlation identifier
						reply-to	shortstr	destination to reply to
						content-type	shortstr	MIME content type
						content-encoding	shortstr	MIME content encoding
						user-id	shortstr	creating user id
						app-id	shortstr	creating application id

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
						transaction-id	shortstr	distributed transaction id
						security-token	security-token	
						application-headers	table	application specific headers table
						body	content	message body
consume	20	ok	[WORK IN PROGRESS] start a queue consumer	Y		ticket	access-ticket	
						queue	queue-name	
						destination	destination	incoming message destination
						no-local	no-local	
						no-ack	no-ack	
						exclusive	bit	request exclusive access
						filter	table	arguments for consuming
cancel	30	ok	[WORK IN PROGRESS] end a queue consumer	Y		destination	destination	
get	40	ok	[WORK IN PROGRESS] direct access to a queue	Y		ticket	access-ticket	
						queue	queue-name	
						destination	destination	
						no-ack	no-ack	
recover	50	ok	[WORK IN PROGRESS] redeliver unacknowledged messages	Y		requeue	bit	requeue the message
open	60	ok	[WORK IN PROGRESS] create a reference to an empty message body	Y	Y	reference	reference	
close	70	ok	[WORK IN PROGRESS] close a reference	Y	Y	reference	reference	target reference
append	80	ok	[WORK IN	Y	Y	reference	reference	target reference

Name	ID	Sync. Response	Short description	S	C	Field Name	Domain	Short Description
			PROGRESS] append to a reference			bytes	longstr	data to append
checkpoint	90	ok	[WORK IN PROGRESS] checkpoint a message body	Y	Y	reference	reference	target reference
						identifier	shortstr	checkpoint identifier
resume	100	offset	[WORK IN PROGRESS] open and resume a checkpointed message	Y	Y	reference	reference	target reference
						identifier	shortstr	checkpoint identifier
qos	110	ok	[WORK IN PROGRESS] specify quality of service	Y		prefetch-size	long	prefetch window in octets
						prefetch-count	short	prefetch window in messages
						global	bit	apply to entire connection
ok	500		[WORK IN PROGRESS] normal completion	Y	Y	[No parameters defined for this method]		
empty	510		[WORK IN PROGRESS] empty queue	Y	Y	[No parameters defined for this method]		
reject	520		[WORK IN PROGRESS] reject a message	Y	Y	code	reject-code	
						text	reject-text	
offset	530		[WORK IN PROGRESS] return an offset	Y	Y	value	offset	offset into a reference body

1.15.2. Methods

1.15.2.1. Method message.transfer (ID 10)

ID: 10

Method accepted by: Server, Client

Synchronous: Yes; expected response is from method ***message.ok***

Number of parameters: 22

Label: [WORK IN PROGRESS] transfer a message

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	destination	destination	
3	redelivered	redelivered	
4	immediate	bit	request immediate delivery
5	ttl	duration	time to live
6	priority	octet	message priority, 0 to 9
7	timestamp	timestamp	message timestamp
8	delivery-mode	octet	non-persistent (1) or persistent (2)
9	expiration	timestamp	message expiration time
10	exchange	exchange-name	originating exchange
11	routing-key	shortstr	message routing key
12	message-id	shortstr	application message identifier
13	correlation-id	shortstr	application correlation identifier
14	reply-to	shortstr	destination to reply to
15	content-type	shortstr	MIME content type
16	content-encoding	shortstr	MIME content encoding
17	user-id	shortstr	creating user id
18	app-id	shortstr	creating application id
19	transaction-id	shortstr	distributed transaction id
20	security-token	security-token	
21	application-headers	table	application specific headers table
22	body	content	message body

[WORK IN PROGRESS] This method transfers a message between two peers. When a client uses this method to publish a message to a broker, the destination identifies a specific exchange. The message will then be routed to queues as defined by the exchange configuration and distributed to any active consumers when the transaction, if any, is committed. In the asynchronous message delivery model, the client starts a consumer using the Consume method and passing in a destination, then the broker responds with transfer methods to the specified destination as and when messages arrive for that consumer. If synchronous message delivery is required, the client may issue a get request which on success causes a single message to be transferred to the specified destination. Message acknowledgement is signalled by

the return result of this method.

Guidelines for implementers:

- The recipient MUST NOT return ok before the message has been processed as defined by the QoS settings.

1.15.2.1.1. Parameter message.transfer.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

1.15.2.1.2. Parameter message.transfer.destination (destination)

Ordinal: 2

Domain: destination

Specifies the destination to which the message is to be transferred. The destination can be empty, meaning the default exchange or consumer. If the destination is specified, and that exchange or consumer does not exist, the peer must raise a channel exception.

1.15.2.1.3. Parameter message.transfer.redelivered (redelivered)

Ordinal: 3

Domain: redelivered

1.15.2.1.4. Parameter message.transfer.immediate (bit)

Ordinal: 4

Domain: bit

Label: request immediate delivery

This flag tells the server how to react if the message cannot be routed to a queue consumer immediately. If this flag is set, the server will reject the message. If this flag is zero, the server will queue the message, but with no guarantee that it will ever be consumed.

1.15.2.1.5. Parameter message.transfer.ttl (duration)

Ordinal: 5

Domain: duration

Label: time to live

If this is set to a non zero value then a message expiration time will be computed based on the current time

plus this value. Messages that live longer than their expiration time will be discarded (or dead lettered).

1.15.2.1.6. Parameter message.transfer.priority (octet)

Ordinal: 6

Domain: octet

Label: message priority, 0 to 9

1.15.2.1.7. Parameter message.transfer.timestamp (timestamp)

Ordinal: 7

Domain: timestamp

Label: message timestamp

Set on arrival by the broker.

1.15.2.1.8. Parameter message.transfer.delivery-mode (octet)

Ordinal: 8

Domain: octet

Label: non-persistent (1) or persistent (2)

1.15.2.1.9. Parameter message.transfer.expiration (timestamp)

Ordinal: 9

Domain: timestamp

Label: message expiration time

The expiration header assigned by the broker. After receiving the message the broker sets expiration to the sum of the ttl specified in the publish method and the current time. (ttl = expiration - timestamp)

1.15.2.1.10. Parameter message.transfer.exchange (exchange-name)

Ordinal: 10

Domain: exchange-name

Label: originating exchange

1.15.2.1.11. Parameter message.transfer.routing-key (shortstr)

Ordinal: 11

Domain: shortstr

Label: message routing key

1.15.2.1.12. Parameter message.transfer.message-id (shortstr)

Ordinal: 12

Domain: shortstr

Label: application message identifier

1.15.2.1.13. Parameter message.transfer.correlation-id (shortstr)

Ordinal: 13

Domain: shortstr

Label: application correlation identifier

1.15.2.1.14. Parameter message.transfer.reply-to (shortstr)

Ordinal: 14

Domain: shortstr

Label: destination to reply to

1.15.2.1.15. Parameter message.transfer.content-type (shortstr)

Ordinal: 15

Domain: shortstr

Label: MIME content type

1.15.2.1.16. Parameter message.transfer.content-encoding (shortstr)

Ordinal: 16

Domain: shortstr

Label: MIME content encoding

1.15.2.1.17. Parameter message.transfer.user-id (shortstr)

Ordinal: 17

Domain: shortstr

Label: creating user id

1.15.2.1.18. Parameter message.transfer.app-id (shortstr)

Ordinal: 18

Domain: shortstr

Label: creating application id

1.15.2.1.19. Parameter message.transfer.transaction-id (shortstr)

Ordinal: 19

Domain: shortstr

Label: distributed transaction id

1.15.2.1.20. Parameter message.transfer.security-token (security-token)

Ordinal: 20

Domain: security-token

1.15.2.1.21. Parameter message.transfer.application-headers (table)

Ordinal: 21

Domain: table

Label: application specific headers table

1.15.2.1.22. Parameter message.transfer.body (content)

Ordinal: 22

Domain: content

Label: message body

1.15.2.2. Method message.consume (ID 20)

ID: 20

Method accepted by: Server

Synchronous: Yes; expected response is from method ***message.ok***

Number of parameters: 7

Label: [WORK IN PROGRESS] start a queue consumer

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	queue	queue-name	
3	destination	destination	incoming message destination
4	no-local	no-local	
5	no-ack	no-ack	
6	exclusive	bit	request exclusive access
7	filter	table	arguments for consuming

[WORK IN PROGRESS] This method asks the server to start a "consumer", which is a transient request for messages from a specific queue. Consumers last as long as the channel they were created on, or until the client cancels them.

Guidelines for implementers:

- The server SHOULD support at least 16 consumers per queue, and ideally, impose no limit except as defined by available resources.

  Test scenario: Create a queue and create consumers on that queue until the server closes the connection. Verify that the number of consumers created was at least sixteen and report the total number.

1.15.2.2.1. Parameter message.consume.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

1.15.2.2.2. Parameter message.consume.queue (queue-name)

Ordinal: 2

Domain: queue-name

Specifies the name of the queue to consume from. If the queue name is null, refers to the current queue for the channel, which is the last declared queue.

1.15.2.2.3. Parameter message.consume.destination (destination)

Ordinal: 3

Domain: destination

Label: incoming message destination

Specifies the destination for the consumer. The destination is local to a connection, so two clients can use

the same destination.

1.15.2.2.4. Parameter message.consume.no-local (no-local)

Ordinal: 4

Domain: no-local

1.15.2.2.5. Parameter message.consume.no-ack (no-ack)

Ordinal: 5

Domain: no-ack

1.15.2.2.6. Parameter message.consume.exclusive (bit)

Ordinal: 6

Domain: bit

Label: request exclusive access

Request exclusive consumer access, meaning only this consumer can access the queue.

1.15.2.2.7. Parameter message.consume.filter (table)

Ordinal: 7

Domain: table

Label: arguments for consuming

A set of filters for the consume. The syntax and semantics of these filters depends on the providers implementation.

1.15.2.3. Method message.cancel (ID 30)

ID: 30

Method accepted by: Server

Synchronous: Yes; expected response is from method ***message.ok***

Number of parameters: 1

Label: [WORK IN PROGRESS] end a queue consumer

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	destination	destination	

[WORK IN PROGRESS] This method cancels a consumer. This does not affect already delivered messages, but it does mean the server will not send any more messages for that consumer. The client may receive an arbitrary number of messages in between sending the cancel method and receiving the cancel-ok reply.

Guidelines for implementers:

- If the queue does not exist the server MUST ignore the cancel method, so long as the consumer tag is valid for that channel.

1.15.2.3.1. Parameter message.cancel.destination (destination)

Ordinal: 1

Domain: destination

1.15.2.4. Method message.get (ID 40)

ID: 40

Method accepted by: Server

Synchronous: Yes; expected response is from method ***message.ok***

Number of parameters: 4

Label: [WORK IN PROGRESS] direct access to a queue

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	ticket	access-ticket	
2	queue	queue-name	
3	destination	destination	
4	no-ack	no-ack	

[WORK IN PROGRESS] This method provides a direct access to the messages in a queue using a synchronous dialogue that is designed for specific types of application where synchronous functionality is more important than performance.

1.15.2.4.1. Parameter message.get.ticket (access-ticket)

Ordinal: 1

Domain: access-ticket

1.15.2.4.2. Parameter message.get.queue (queue-name)

Ordinal: 2

Domain: queue-name

Specifies the name of the queue to consume from. If the queue name is null, refers to the current queue for the channel, which is the last declared queue.

1.15.2.4.3. Parameter message.get.destination (destination)

Ordinal: 3

Domain: destination

On normal completion of the get request (i.e. a response of ok). A message will be transferred to the supplied destination.

1.15.2.4.4. Parameter message.get.no-ack (no-ack)

Ordinal: 4

Domain: no-ack

1.15.2.5. Method message.recover (ID 50)

ID: 50

Method accepted by: Server

Synchronous: Yes; expected response is from method ***message.ok***

Number of parameters: 1

Label: [WORK IN PROGRESS] redeliver unacknowledged messages

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	requeue	bit	requeue the message

[WORK IN PROGRESS] This method asks the broker to redeliver all unacknowledged messages on a specified channel. Zero or more messages may be redelivered. This method is only allowed on non-transacted channels.

Guidelines for implementers:

- The server MUST set the redelivered flag on all messages that are resent.
- The server MUST raise a channel exception if this is called on a transacted channel.

1.15.2.5.1. Parameter message.recover.requeue (bit)

Ordinal: 1

Domain: bit

Label: requeue the message

If this field is zero, the message will be redelivered to the original recipient. If this bit is 1, the server will attempt to requeue the message, potentially then delivering it to an alternative subscriber.

1.15.2.6. Method message.open (ID 60)

ID: 60

Method accepted by: Server, Client

Synchronous: Yes; expected response is from method ***message.ok***

Number of parameters: 1

Label: [WORK IN PROGRESS] create a reference to an empty message body

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	reference	reference	

[WORK IN PROGRESS] This method creates a reference. A references provides a means to send a message body into a temporary area at the recipient end and then deliver the message by referring to this temporary area. This is how the protocol handles large message transfers. The scope of a ref is defined to be between calls to open (or resume) and close. Between these points it is valid for a ref to be used from any content data type, and so the receiver must hold onto its contents. Should the channel be closed when a ref is still in scope, the receiver may discard its contents (unless it is checkpointed). A ref that is in scope is considered open.

1.15.2.6.1. Parameter message.open.reference (reference)

Ordinal: 1

Domain: reference

1.15.2.7. Method message.close (ID 70)

ID: 70

Method accepted by: Server, Client

Synchronous: Yes; expected response is from method ***message.ok***

Number of parameters: 1

Label: [WORK IN PROGRESS] close a reference

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	reference	reference	target reference

[WORK IN PROGRESS] This method signals the recipient that no more data will be appended to the reference.

Guidelines for implementers:

- A recipient CANNOT acknowledge a message until its reference is closed (not in scope).

1.15.2.7.1. Parameter message.close.reference (reference)

Ordinal: 1

Domain: reference

Label: target reference

1.15.2.8. Method message.append (ID 80)

ID: 80

Method accepted by: Server, Client

Synchronous: Yes; expected response is from method ***message.ok***

Number of parameters: 2

Label: [WORK IN PROGRESS] append to a reference

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	reference	reference	target reference
2	bytes	longstr	data to append

[WORK IN PROGRESS] This method appends data to a reference.

1.15.2.8.1. Parameter message.append.reference (reference)

Ordinal: 1

Domain: reference

Label: target reference

1.15.2.8.2. Parameter message.append.bytes (longstr)

Ordinal: 2

Domain: longstr

Label: data to append

1.15.2.9. Method message.checkpoint (ID 90)

ID: 90

Method accepted by: Server, Client

Synchronous: Yes; expected response is from method ***message.ok***

Number of parameters: 2

Label: [WORK IN PROGRESS] checkpoint a message body

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	reference	reference	target reference
2	identifier	shortstr	checkpoint identifier

[WORK IN PROGRESS] This method provides a means to checkpoint large message transfer. The sender may ask the recipient to checkpoint the contents of a reference using the supplied identifier. The sender may then resume the transfer at a later point. It is at the discretion of the recipient how much data to save with the checkpoint, and the sender MUST honour the offset returned by the resume method.

1.15.2.9.1. Parameter message.checkpoint.reference (reference)

Ordinal: 1

Domain: reference

Label: target reference

1.15.2.9.2. Parameter message.checkpoint.identifier (shortstr)

Ordinal: 2

Domain: shortstr

Label: checkpoint identifier

This is the checkpoint identifier. This is an arbitrary string chosen by the sender. For checkpointing to work correctly the sender must use the same checkpoint identifier when resuming the message. A good choice for the checkpoint identifier would be the SHA1 hash of the message properties data (including the original filename, revised time, etc.).

1.15.2.10. Method message.resume (ID 100)

ID: 100

Method accepted by: Server, Client

Synchronous: Yes; expected response is from method ***message.offset***

Number of parameters: 2

Label: [WORK IN PROGRESS] open and resume a checkpointed message

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	reference	reference	target reference
2	identifier	shortstr	checkpoint identifier

[WORK IN PROGRESS] This method resumes a reference from the last checkpoint. A reference is considered to be open (in scope) after a resume even though it will not have been opened via the open method during this session.

1.15.2.10.1. Parameter message.resume.reference (reference)

Ordinal: 1

Domain: reference

Label: target reference

1.15.2.10.2. Parameter message.resume.identifier (shortstr)

Ordinal: 2

Domain: shortstr

Label: checkpoint identifier

1.15.2.11. Method message.qos (ID 110)

ID: 110

Method accepted by: Server

Synchronous: Yes; expected response is from method ***message.ok***

Number of parameters: 3

Label: [WORK IN PROGRESS] specify quality of service

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	prefetch-size	long	prefetch window in octets
2	prefetch-count	short	prefetch window in messages
3	global	bit	apply to entire connection

[WORK IN PROGRESS] This method requests a specific quality of service. The QoS can be specified for the current channel or for all channels on the connection. The particular properties and semantics of a qos method always depend on the content class semantics. Though the qos method could in principle apply to both peers, it is currently meaningful only for the server.

1.15.2.11.1. Parameter message.qos.prefetch-size (long)

Ordinal: 1

Domain: long

Label: prefetch window in octets

The client can request that messages be sent in advance so that when the client finishes processing a message, the following message is already held locally, rather than needing to be sent down the channel. Prefetching gives a performance improvement. This field specifies the prefetch window size in octets. The server will send a message in advance if it is equal to or smaller in size than the available prefetch size (and also falls into other prefetch limits). May be set to zero, meaning "no specific limit", although other prefetch limits may still apply. The prefetch-size is ignored if the no-ack option is set.

1.15.2.11.2. Parameter message.qos.prefetch-count (short)

Ordinal: 2

Domain: short

Label: prefetch window in messages

Specifies a prefetch window in terms of whole messages. This field may be used in combination with the prefetch-size field; a message will only be sent in advance if both prefetch windows (and those at the channel and connection level) allow it. The prefetch-count is ignored if the no-ack option is set.

1.15.2.11.3. Parameter message.qos.global (bit)

Ordinal: 3

Domain: bit

Label: apply to entire connection

By default the QoS settings apply to the current channel only. If this field is set, they are applied to the entire connection.

1.15.2.12. Method message.ok (ID 500)

ID: 500

Method accepted by: Server, Client

Synchronous: No

Number of parameters: 0

Label: [WORK IN PROGRESS] normal completion

[WORK IN PROGRESS] Signals the normal completion of a method.

1.15.2.13. Method message.empty (ID 510)

ID: 510

Method accepted by: Server, Client

Synchronous: No

Number of parameters: 0

Label: [WORK IN PROGRESS] empty queue

[WORK IN PROGRESS] Signals that a queue does not contain any messages.

1.15.2.14. Method message.reject (ID 520)

ID: 520

Method accepted by: Server, Client

Synchronous: No

Number of parameters: 2

Label: [WORK IN PROGRESS] reject a message

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description

1	code	reject-code	
2	text	reject-text	

[WORK IN PROGRESS] This response rejects a message. A message may be rejected for a number of reasons.

1.15.2.14.1. Parameter message.reject.code (reject-code)

Ordinal: 1

Domain: reject-code

1.15.2.14.2. Parameter message.reject.text (reject-text)

Ordinal: 2

Domain: reject-text

1.15.2.15. Method message.offset (ID 530)

ID: 530

Method accepted by: Server, Client

Synchronous: No

Number of parameters: 1

Label: [WORK IN PROGRESS] return an offset

Parameter Summary:

Ordinal	Parameter name	Domain	Short Description
1	value	offset	offset into a reference body

[WORK IN PROGRESS] Returns the data offset into a reference body.

1.15.2.15.1. Parameter message.offset.value (offset)

Ordinal: 1

Domain: offset

Label: offset into a reference body