## Table of Contents

1. AMQP version 0-9-1 specification .................................................................................................................. 6  
   1.1. AMQP-defined Domains .......................................................................................................................... 6  
   1.2. AMQP-defined Constants ......................................................................................................................... 7  
   1.3. Class and Method ID Summaries ............................................................................................................. 8  
   1.4. Class connection ........................................................................................................................................ 10  
      1.4.1. Property and Method Summary ........................................................................................................ 10  
      1.4.2. Methods ............................................................................................................................................... 11  
         1.4.2.1. Method connection.start (ID 10) ................................................................................................. 11  
            1.4.2.1.1. Parameter connection.start.version-major (octet) ......................................................... 12  
            1.4.2.1.2. Parameter connection.start.version-minor (octet) ......................................................... 12  
            1.4.2.1.3. Parameter connection.start.server-properties (peer-properties) ................................ 12  
            1.4.2.1.4. Parameter connection.start.mechanisms (longstr) ........................................................... 12  
            1.4.2.1.5. Parameter connection.start.locale (longstr) ..................................................................... 13  
         1.4.2.2. Method connection.start-ok (ID 11) ............................................................................................ 13  
            1.4.2.2.1. Parameter connection.start-ok.client-properties (peer-properties) ............................ 13  
            1.4.2.2.2. Parameter connection.start-ok.mechanism (shortstr) ...................................................... 13  
            1.4.2.2.3. Parameter connection.start-ok.response (longstr) ............................................................ 13  
            1.4.2.2.4. Parameter connection.start-ok.locale (shortstr) ............................................................... 14  
         1.4.2.3. Method connection.secure (ID 20) ................................................................................................. 14  
            1.4.2.3.1. Parameter connection.secure.challenge (longstr) ............................................................... 14  
         1.4.2.4. Method connection.secure-ok (ID 21) .......................................................................................... 14  
            1.4.2.4.1. Parameter connection.secure-ok.response (longstr) ........................................................... 15  
         1.4.2.5. Method connection.tune (ID 30) .................................................................................................... 15  
            1.4.2.5.1. Parameter connection.tune.channel-max (short) ................................................................. 15  
            1.4.2.5.2. Parameter connection.tune.frame-max (long) ................................................................... 16  
            1.4.2.5.3. Parameter connection.tune.heartbeat (short) ..................................................................... 16  
         1.4.2.6. Method connection.tune-ok (ID 31) .............................................................................................. 16  
            1.4.2.6.1. Parameter connection.tune-ok.channel-max (short) .......................................................... 16  
            1.4.2.6.2. Parameter connection.tune-ok.frame-max (long) ............................................................... 17  
            1.4.2.6.3. Parameter connection.tune-ok.heartbeat (short) ................................................................. 17  
         1.4.2.7. Method connection.open (ID 40) ................................................................................................. 17  
            1.4.2.7.1. Parameter connection.open.virtual-host (path) ................................................................. 17  
            1.4.2.7.2. Parameter connection.open.reserved-1 () ....................................................................... 18  
            1.4.2.7.3. Parameter connection.open.reserved-2 () ....................................................................... 18  
         1.4.2.8. Method connection.open-ok (ID 41) ............................................................................................ 18  
            1.4.2.8.1. Parameter connection.open-ok.reserved-1 () .................................................................. 18  
         1.4.2.9. Method connection.close (ID 50) ............................................................................................... 18  
            1.4.2.9.1. Parameter connection.close.reply-code (reply-code) ....................................................... 19  
            1.4.2.9.2. Parameter connection.close.reply-text (reply-text) ......................................................... 19  
            1.4.2.9.3. Parameter connection.close.class-id (class-id) ............................................................... 19  
            1.4.2.9.4. Parameter connection.close.method-id (method-id) ......................................................... 19  
         1.4.2.10. Method connection.close-ok (ID 51) ........................................................................................ 19  
   1.5. Class channel ............................................................................................................................................. 20  
      1.5.1. Property and Method Summary ........................................................................................................ 20  
      1.5.2. Methods ............................................................................................................................................. 21  
         1.5.2.1. Method channel.open (ID 10) .................................................................................................... 21  
            1.5.2.1.1. Parameter channel.open.reserved-1 () ........................................................................ 21
1.5.2.2.Method channel.open-ok (ID 11) ............................................................... 21
  1.5.2.2.1.Parameter channel.open-ok.reserved-1 ( ) ........................................ 22
1.5.2.3.Method channel.flow (ID 20) .................................................................. 22
  1.5.2.3.1.Parameter channel.flow.active (bit) ................................................... 22
1.5.2.4.Method channel.flow-ok (ID 21) ............................................................... 22
  1.5.2.4.1.Parameter channel.flow-ok.active (bit) .............................................. 23
1.5.2.5.Method channel.close (ID 40) ................................................................ 23
  1.5.2.5.1.Parameter channel.close.reply-code (reply-code) ............................. 24
  1.5.2.5.2.Parameter channel.close.reply-text (reply-text) ............................... 24
  1.5.2.5.3.Parameter channel.close.class-id (class-id) ...................................... 24
  1.5.2.5.4.Parameter channel.close.method-id (method-id) .............................. 24
  1.5.2.6.Method channel.close-ok (ID 41) .......................................................... 24

1.6.Class exchange................................................................................................. 24
  1.6.1.Property and Method Summary ............................................................... 25
  1.6.2.Methods .................................................................................................... 26
    1.6.2.1.Method exchange.declare (ID 10) ....................................................... 26
      1.6.2.1.1.Parameter exchange.declare.reserved-1 ( ) ................................. 27
      1.6.2.1.2.Parameter exchange.declare.exchange (exchange-name) .......... 27
      1.6.2.1.3.Parameter exchange.declare.type (shortstr) ............................. 27
      1.6.2.1.4.Parameter exchange.declare.passive (bit) ................................. 27
      1.6.2.1.5.Parameter exchange.declare.durable (bit) .................................. 27
      1.6.2.1.6.Parameter exchange.declare.reserved-2 ( ) .............................. 27
      1.6.2.1.7.Parameter exchange.declare.reserved-3 ( ) .............................. 28
      1.6.2.1.8.Parameter exchange.declare.no-wait (no-wait) ......................... 28
      1.6.2.1.9.Parameter exchange.declare.arguments (table) ......................... 28
    1.6.2.2.Method exchange.declare-ok (ID 11) .................................................. 28
    1.6.2.3.Method exchange.delete (ID 20) ......................................................... 28
      1.6.2.3.1.Parameter exchange.delete.reserved-1 ( ) ................................ 29
      1.6.2.3.2.Parameter exchange.delete.exchange (exchange-name) .......... 29
      1.6.2.3.3.Parameter exchange.delete.if-unused (bit) ................................. 29
      1.6.2.3.4.Parameter exchange.delete.no-wait (no-wait) ........................... 29
    1.6.2.4.Method exchange.delete-ok (ID 21) .................................................... 29

1.7.Class queue .................................................................................................... 29
  1.7.1.Property and Method Summary ............................................................... 30
  1.7.2.Methods .................................................................................................... 31
    1.7.2.1.Method queue.declare (ID 10) .......................................................... 31
      1.7.2.1.1.Parameter queue.declare.reserved-1 ( ) ..................................... 32
      1.7.2.1.2.Parameter queue.declare.queue (queue-name) ............................ 32
      1.7.2.1.3.Parameter queue.declare.passive (bit) ....................................... 32
      1.7.2.1.4.Parameter queue.declare.durable (bit) ....................................... 32
      1.7.2.1.5.Parameter queue.declare.exclusive (bit) .................................. 32
      1.7.2.1.6.Parameter queue.declare.auto-delete (bit) ................................. 33
      1.7.2.1.7.Parameter queue.declare.no-wait (no-wait) ............................... 33
      1.7.2.1.8.Parameter queue.declare.arguments (table) ............................... 33
    1.7.2.2.Method queue.declare-ok (ID 11) ....................................................... 33
      1.7.2.2.1.Parameter queue.declare-ok.queue (queue-name) ....................... 34
      1.7.2.2.2.Parameter queue.declare-ok.message-count (message-count) ... 34
      1.7.2.2.3.Parameter queue.declare-ok.consumer-count (long) ................. 34
    1.7.2.3.Method queue.bind (ID 20) ................................................................ 34
      1.7.2.3.1.Parameter queue.bind.reserved-1 ( ) ......................................... 35
1.7.2.3.2. Parameter queue.bind queue (queue-name) ................................................................. 35
1.7.2.3.3. Parameter queue.bind exchange (exchange-name) .................................................. 35
1.7.2.3.4. Parameter queue.bind routing-key (shortstr) .......................................................... 35
1.7.2.3.5. Parameter queue.bind no-wait (no-wait) ................................................................. 36
1.7.2.3.6. Parameter queue.bind arguments (table) ................................................................. 36
1.7.2.4. Method queue.bind-ok (ID 21) ....................................................................................... 36
1.7.2.5. Method queue.unbind (ID 50) ....................................................................................... 36
  1.7.2.5.1. Parameter queue.unbind.reserved-1 () ................................................................. 37
  1.7.2.5.2. Parameter queue.unbind.queue (queue-name) ....................................................... 37
  1.7.2.5.3. Parameter queue.unbind.exchange (exchange-name) ........................................... 37
  1.7.2.5.4. Parameter queue.unbind.routing-key (shortstr) .................................................. 37
  1.7.2.5.5. Parameter queue.unbind.arguments (table) .......................................................... 37
1.7.2.6. Method queue.unbind-ok (ID 51) .................................................................................. 37
1.7.2.7. Method queue.purge (ID 30) ......................................................................................... 38
  1.7.2.7.1. Parameter queue.purge.reserved-1 () ................................................................... 38
  1.7.2.7.2. Parameter queue.purge.queue (queue-name) ......................................................... 38
  1.7.2.7.3. Parameter queue.purge.no-wait (no-wait) ............................................................. 38
1.7.2.8. Method queue.purge-ok (ID 31) .................................................................................... 39
  1.7.2.8.1. Parameter queue.purge-ok.message-count (message-count) ............................... 39
1.7.2.9. Method queue.delete (ID 40) ....................................................................................... 39
  1.7.2.9.1. Parameter queue.delete.reserved-1 () ................................................................ 40
  1.7.2.9.2. Parameter queue.delete.queue (queue-name) ....................................................... 40
  1.7.2.9.3. Parameter queue.delete.if-unused (bit) ................................................................. 40
  1.7.2.9.4. Parameter queue.delete.if-empty (bit) .................................................................. 40
  1.7.2.9.5. Parameter queue.delete.no-wait (no-wait) ............................................................. 40
1.7.2.10. Method queue.delete-ok (ID 41) ............................................................................... 40
  1.7.2.10.1. Parameter queue.delete-ok.message-count (message-count) .......................... 40
1.8 Class basic
1.8.1. Property and Method Summary .................................................................................. 42
1.8.2. Properties .................................................................................................................. 44
  1.8.2.1. Property basic.content-type (shortstr) ................................................................. 44
  1.8.2.2. Property basic.content-encoding (shortstr) ......................................................... 44
  1.8.2.3. Property basic.headers (table) ............................................................................... 44
  1.8.2.4. Property basic.delivery-mode (octet) ................................................................. 44
  1.8.2.5. Property basic.priority (octet) ............................................................................... 45
  1.8.2.6. Property basic.correlation-id (shortstr) ................................................................. 45
  1.8.2.7. Property basic.reply-to (shortstr) ....................................................................... 45
  1.8.2.8. Property basic.expiration (shortstr) ................................................................. 45
  1.8.2.9. Property basic.message-id (shortstr) ................................................................. 45
  1.8.2.10. Property basic.timestamp (timestamp) .............................................................. 45
  1.8.2.11. Property basic.type (shortstr) ............................................................................. 45
  1.8.2.12. Property basic.user-id (shortstr) ...................................................................... 45
  1.8.2.13. Property basic.app-id (shortstr) ....................................................................... 45
  1.8.2.14. Property basic.reserved (shortstr) .................................................................... 45
1.8.3. Methods .................................................................................................................. 46
  1.8.3.1. Method basic.qos (ID 10) ..................................................................................... 46
    1.8.3.1.1. Parameter basic.qos.prefetch-size (long) ....................................................... 46
    1.8.3.1.2. Parameter basic.qos.prefetch-count (short) .................................................. 46
    1.8.3.1.3. Parameter basic.qos.global (bit) ................................................................... 47
  1.8.3.2. Method basic.qos-ok (ID 11) .................................................................................. 47
1.8.3.3. Method basic.consume (ID 20) .......................................................... 47
  1.8.3.3.1. Parameter basic.consume.consumer-tag (consumer-tag) .......... 48
  1.8.3.3.2. Parameter basic.consume.queue (queue-name) .................. 48
  1.8.3.3.3. Parameter basic.consume.consumer-tag (consumer-tag) .......... 48
  1.8.3.3.4. Parameter basic.consume.no-local (no-local) ................. 48
  1.8.3.3.5. Parameter basic.consume.no-ack (no-ack) .................. 48
  1.8.3.3.6. Parameter basic.consume.exclusive (bit) .................. 48
  1.8.3.3.7. Parameter basic.consume.no-wait (no-wait) .............. 49
  1.8.3.3.8. Parameter basic.consume.arguments (table) .......... 49
  1.8.3.3.9. Parameter basic.consume.consumer-tag (consumer-tag) .......... 49
  1.8.3.3.10. Parameter basic.consume.reserved-1 () .................. 50
  1.8.3.3.11. Parameter basic.consume.queue (queue-name) ............. 50
  1.8.3.3.12. Parameter basic.consume.no-local (no-local) ............ 50
  1.8.3.3.13. Parameter basic.consume.no-ack (no-ack) .......... 50
  1.8.3.3.14. Parameter basic.consume.exclusive (bit) .......... 50
  1.8.3.3.15. Parameter basic.consume.no-wait (no-wait) .......... 50
  1.8.3.3.16. Parameter basic.consume.arguments (table) .......... 50

1.8.3.4. Method basic.consume-ok (ID 21) ........................................... 49
  1.8.3.4.1. Parameter basic.consume-ok.consumer-tag (consumer-tag) .......... 49

1.8.3.5. Method basic.cancel (ID 30) .................................................. 49
  1.8.3.5.1. Parameter basic.cancel.consumer-tag (consumer-tag) .......... 50
  1.8.3.5.2. Parameter basic.cancel.no-wait (no-wait) .......... 50

1.8.3.6. Method basic.cancel-ok (ID 31) ............................................ 50
  1.8.3.6.1. Parameter basic.cancel-ok.consumer-tag (consumer-tag) .......... 50

1.8.3.7. Method basic.publish (ID 40) .................................................. 51
  1.8.3.7.1. Parameter basic.publish.reserved-1 () .................. 51
  1.8.3.7.2. Parameter basic.publish.exchange (exchange-name) .......... 51
  1.8.3.7.3. Parameter basic.publish.routing-key (shortstr) .......... 51
  1.8.3.7.4. Parameter basic.publish.mandatory (bit) .................. 52
  1.8.3.7.5. Parameter basic.publish.immediate (bit) .......... 52

1.8.3.8. Method basic.return (ID 50) .................................................. 52
  1.8.3.8.1. Parameter basic.return.reply-code (reply-code) .......... 52
  1.8.3.8.2. Parameter basic.return.reply-text (reply-text) .......... 53
  1.8.3.8.3. Parameter basic.return.exchange (exchange-name) .......... 53
  1.8.3.8.4. Parameter basic.return.routing-key (shortstr) .......... 53

1.8.3.9. Method basic.deliver (ID 60) .................................................. 53
  1.8.3.9.1. Parameter basic.deliver.consumer-tag (consumer-tag) .......... 53
  1.8.3.9.2. Parameter basic.deliver.delivery-tag (delivery-tag) .......... 54
  1.8.3.9.3. Parameter basic.deliver.redelivered (redelivered) .......... 54
  1.8.3.9.4. Parameter basic.deliver.exchange (exchange-name) .......... 54
  1.8.3.9.5. Parameter basic.deliver.routing-key (shortstr) .......... 54

1.8.3.10. Method basic.get (ID 70) ...................................................... 54
  1.8.3.10.1. Parameter basic.get.reserved-1 () .................. 55
  1.8.3.10.2. Parameter basic.get.queue (queue-name) .......... 55
  1.8.3.10.3. Parameter basic.get.no-ack (no-ack) .......... 55

1.8.3.11. Method basic.get-ok (ID 71) .................................................. 55
  1.8.3.11.1. Parameter basic.get-ok.delivery-tag (delivery-tag) .......... 56
  1.8.3.11.2. Parameter basic.get-ok.redelivered (redelivered) .......... 56
  1.8.3.11.3. Parameter basic.get-ok.exchange (exchange-name) .......... 56
  1.8.3.11.4. Parameter basic.get-ok.routing-key (shortstr) .......... 56
  1.8.3.11.5. Parameter basic.get-ok.message-count (message-count) .......... 56

1.8.3.12. Method basic.get-empty (ID 72) .......................................... 56
  1.8.3.12.1. Parameter basic.get-empty.reserved-1 () .................. 57

1.8.3.13. Method basic.ack (ID 80) ...................................................... 57
  1.8.3.13.1. Parameter basic.ack.delivery-tag (delivery-tag) .......... 57
  1.8.3.13.2. Parameter basic.ack.multiple (bit) .......... 57

1.8.3.14. Method basic.reject (ID 90) .................................................. 57
  1.8.3.14.1. Parameter basic.reject.delivery-tag (delivery-tag) .......... 58
1. AMQP version 0-9-1 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>[Label] Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bit</td>
<td>bit</td>
<td>[single bit]</td>
</tr>
<tr>
<td>class-id</td>
<td>short</td>
<td></td>
</tr>
<tr>
<td>consumer-tag</td>
<td>shortstr</td>
<td>[consumer tag] Identifier for the consumer, valid within the current channel.</td>
</tr>
<tr>
<td>delivery-tag</td>
<td>longlong</td>
<td>[server-assigned delivery tag] The server-assigned and channel-specific delivery tag</td>
</tr>
<tr>
<td>exchange-name</td>
<td>shortstr</td>
<td>[exchange name] The exchange name is a client-selected string that identifies the exchange for publish methods.</td>
</tr>
<tr>
<td>long</td>
<td>long</td>
<td>[32-bit integer]</td>
</tr>
<tr>
<td>longlong</td>
<td>longlong</td>
<td>[64-bit integer]</td>
</tr>
<tr>
<td>longstr</td>
<td>longstr</td>
<td>[long string]</td>
</tr>
<tr>
<td>message-count</td>
<td>long</td>
<td>[number of messages in queue] The number of messages in the queue, which will be zero for newly-declared queues. This is the number of messages present in the queue, and committed if the channel on which they were published is transacted, that are not waiting acknowledgement.</td>
</tr>
<tr>
<td>method-id</td>
<td>short</td>
<td></td>
</tr>
<tr>
<td>no-ack</td>
<td>bit</td>
<td>[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 assumes the delivery will succeed and immediately dequeues it. This functionality may increase performance but at the cost of reliability. Messages can get lost if a client dies before they are delivered to the application.</td>
</tr>
<tr>
<td>no-local</td>
<td>bit</td>
<td>[do not deliver own messages] If the no-local field is set the server will not send messages to the connection that published them.</td>
</tr>
<tr>
<td>no-wait</td>
<td>bit</td>
<td>[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.</td>
</tr>
<tr>
<td>octet</td>
<td>octet</td>
<td>[single octet]</td>
</tr>
<tr>
<td>path</td>
<td>shortstr</td>
<td>Unconstrained.</td>
</tr>
<tr>
<td>peer-properties</td>
<td>table</td>
<td>This table provides a set of peer properties, used for identification, debugging, and general information.</td>
</tr>
<tr>
<td>queue-name</td>
<td>shortstr</td>
<td>[queue name] The queue name identifies the queue within the vhost. In methods where the queue name may be blank, and that has no specific significance, this refers to the 'current' queue for the channel, meaning the last queue that the client declared on the channel. If the client did not declare a queue, and the method needs a queue name, this will result in a 502 (syntax error) channel exception.</td>
</tr>
<tr>
<td>redelivered</td>
<td>bit</td>
<td>[message is being redelivered] This indicates that the message has been</td>
</tr>
</tbody>
</table>
### 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Error type</th>
<th>[Label] Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame-method</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>frame-header</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>frame-body</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>frame-heartbeat</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>frame-min-size</td>
<td>4096</td>
<td></td>
<td></td>
</tr>
<tr>
<td>frame-end</td>
<td>206</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reply-success</td>
<td>200</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>content-too-large</td>
<td>311</td>
<td>channel</td>
<td>The client attempted to transfer content larger than the server could accept at the present time. The client may retry at a later time.</td>
</tr>
<tr>
<td>no-consumers</td>
<td>313</td>
<td>channel</td>
<td>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.</td>
</tr>
<tr>
<td>connection-forced</td>
<td>320</td>
<td>connection</td>
<td>An operator intervened to close the connection for some reason. The client may retry at some later date.</td>
</tr>
<tr>
<td>invalid-path</td>
<td>402</td>
<td>connection</td>
<td>The client tried to work with an unknown virtual host.</td>
</tr>
<tr>
<td>access-refused</td>
<td>403</td>
<td>channel</td>
<td>The client attempted to work with a server entity to which it has no access due to security settings.</td>
</tr>
<tr>
<td>not-found</td>
<td>404</td>
<td>channel</td>
<td>The client attempted to work with a server entity that does not exist.</td>
</tr>
</tbody>
</table>
| resource-locked | 405  | channel    | The client attempted to work with a server entity to which it has no access.
### Error Type Descriptions

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Error type</th>
<th>[Label] Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>precondition-failed</td>
<td>406</td>
<td>channel</td>
<td>The client requested a method that was not allowed because some precondition failed.</td>
</tr>
<tr>
<td>frame-error</td>
<td>501</td>
<td>connection</td>
<td>The sender sent a malformed frame that the recipient could not decode. This strongly implies a programming error in the sending peer.</td>
</tr>
<tr>
<td>syntax-error</td>
<td>502</td>
<td>connection</td>
<td>The sender sent a frame that contained illegal values for one or more fields. This strongly implies a programming error in the sending peer.</td>
</tr>
<tr>
<td>command-invalid</td>
<td>503</td>
<td>connection</td>
<td>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.</td>
</tr>
<tr>
<td>channel-error</td>
<td>504</td>
<td>connection</td>
<td>The client attempted to work with a channel that had not been correctly opened. This most likely indicates a fault in the client layer.</td>
</tr>
<tr>
<td>unexpected-frame</td>
<td>505</td>
<td>connection</td>
<td>The peer sent a frame that was not expected, usually in the context of a content header and body. This strongly indicates a fault in the peer’s content processing.</td>
</tr>
<tr>
<td>resource-error</td>
<td>506</td>
<td>connection</td>
<td>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.</td>
</tr>
<tr>
<td>not-allowed</td>
<td>530</td>
<td>connection</td>
<td>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.</td>
</tr>
<tr>
<td>not-implemented</td>
<td>540</td>
<td>connection</td>
<td>The client tried to use functionality that is not implemented in the server.</td>
</tr>
<tr>
<td>internal-error</td>
<td>541</td>
<td>connection</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 1.3. Class and Method ID Summaries

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

<table>
<thead>
<tr>
<th>Class</th>
<th>ID</th>
<th>Short class description</th>
<th>Method</th>
<th>ID</th>
<th>Short method description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connection</td>
<td>10</td>
<td>work with socket connections</td>
<td>start</td>
<td>10</td>
<td>start connection negotiation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>start-ok</td>
<td>11</td>
<td>select security mechanism and locale</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>secure</td>
<td>20</td>
<td>security mechanism challenge</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>secure-ok</td>
<td>21</td>
<td>security mechanism response</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tune</td>
<td>30</td>
<td>propose connection tuning parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>tune-ok</td>
<td>31</td>
<td>negotiate connection tuning parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>open</td>
<td>40</td>
<td>open connection to virtual host</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>open-ok</td>
<td>41</td>
<td>signal that connection is ready</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>close</td>
<td>50</td>
<td>request a connection close</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>close-ok</td>
<td>51</td>
<td>confirm a connection close</td>
</tr>
<tr>
<td>channel</td>
<td>20</td>
<td>work with channels</td>
<td>open</td>
<td>10</td>
<td>open a channel for use</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>open-ok</td>
<td>11</td>
<td>signal that the channel is ready</td>
</tr>
<tr>
<td>Class</td>
<td>ID</td>
<td>Short class description</td>
<td>Method</td>
<td>ID</td>
<td>Short method description</td>
</tr>
<tr>
<td>-------</td>
<td>----</td>
<td>-------------------------</td>
<td>--------</td>
<td>----</td>
<td>--------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>flow</td>
<td>20</td>
<td>enable/disable flow from peer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>flow-ok</td>
<td>21</td>
<td>confirm a flow method</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>close</td>
<td>40</td>
<td>request a channel close</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>close-ok</td>
<td>41</td>
<td>confirm a channel close</td>
</tr>
<tr>
<td>exchange</td>
<td>40</td>
<td>work with exchanges</td>
<td>declare</td>
<td>10</td>
<td>verify exchange exists, create if needed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>declare-ok</td>
<td>11</td>
<td>confirm exchange declaration</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>delete</td>
<td>20</td>
<td>delete an exchange</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>delete-ok</td>
<td>21</td>
<td>confirm deletion of an exchange</td>
</tr>
<tr>
<td>queue</td>
<td>50</td>
<td>work with queues</td>
<td>declare</td>
<td>10</td>
<td>declare queue, create if needed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>declare-ok</td>
<td>11</td>
<td>confirms a queue definition</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bind</td>
<td>20</td>
<td>bind queue to an exchange</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bind-ok</td>
<td>21</td>
<td>confirm bind successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>unbind</td>
<td>50</td>
<td>unbind a queue from an exchange</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>unbind-ok</td>
<td>51</td>
<td>confirm unbind successful</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>purge</td>
<td>30</td>
<td>purge a queue</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>purge-ok</td>
<td>31</td>
<td>confirms a queue purge</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>delete</td>
<td>40</td>
<td>delete a queue</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>delete-ok</td>
<td>41</td>
<td>confirm deletion of a queue</td>
</tr>
<tr>
<td>basic</td>
<td>60</td>
<td>work with basic content</td>
<td>qos</td>
<td>10</td>
<td>specify quality of service</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>qos-ok</td>
<td>11</td>
<td>confirm the requested qos</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>consume</td>
<td>20</td>
<td>start a queue consumer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>consume-ok</td>
<td>21</td>
<td>confirm a new consumer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cancel</td>
<td>30</td>
<td>end a queue consumer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cancel-ok</td>
<td>31</td>
<td>confirm a cancelled consumer</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>publish</td>
<td>40</td>
<td>publish a message</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>return</td>
<td>50</td>
<td>return a failed message</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>deliver</td>
<td>60</td>
<td>notify the client of a consumer message</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>get</td>
<td>70</td>
<td>direct access to a queue</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>get-ok</td>
<td>71</td>
<td>provide client with a message</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>get-empty</td>
<td>72</td>
<td>indicate no messages available</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ack</td>
<td>80</td>
<td>acknowledge one or more messages</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>reject</td>
<td>90</td>
<td>reject an incoming message</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>recover-async</td>
<td>100</td>
<td>redeliver unacknowledged messages</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>recover</td>
<td>110</td>
<td>redeliver unacknowledged messages</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>recover-ok</td>
<td>111</td>
<td>confirm recovery</td>
</tr>
<tr>
<td>tx</td>
<td>90</td>
<td>work with transactions</td>
<td>select</td>
<td>10</td>
<td>select standard transaction mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>select-ok</td>
<td>11</td>
<td>confirm transaction mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>commit</td>
<td>20</td>
<td>commit the current transaction</td>
</tr>
</tbody>
</table>
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
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):

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Sync. Response</th>
<th>Short description</th>
<th>S</th>
<th>C</th>
<th>Field Name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>10</td>
<td>start-ok</td>
<td>start connection negotiation</td>
<td>Y</td>
<td>version-major</td>
<td>octet</td>
<td>protocol major version</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>version-minor</td>
<td>octet</td>
<td>protocol minor version</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>server-properties</td>
<td>peer-properties</td>
<td>server properties</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mechanisms</td>
<td>longstr</td>
<td>available security mechanisms</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>locales</td>
<td>longstr</td>
<td>available message locales</td>
<td></td>
</tr>
<tr>
<td>start-ok</td>
<td>11</td>
<td></td>
<td>select security mechanism and locale</td>
<td>Y</td>
<td>client-properties</td>
<td>peer-properties</td>
<td>client properties</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mechanism</td>
<td>shortstr</td>
<td>selected security mechanism</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>response</td>
<td>longstr</td>
<td>security response data</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>locale</td>
<td>shortstr</td>
<td>selected message locale</td>
<td></td>
</tr>
<tr>
<td>secure</td>
<td>20</td>
<td>secure-ok</td>
<td>security mechanism challenge</td>
<td>Y</td>
<td>challenge</td>
<td>longstr</td>
<td>security challenge data</td>
<td></td>
</tr>
<tr>
<td>secure-ok</td>
<td>21</td>
<td></td>
<td>security mechanism</td>
<td>Y</td>
<td>response</td>
<td>longstr</td>
<td>security response data</td>
<td></td>
</tr>
</tbody>
</table>

Commit ID: 21
Confirm a successful commit

Rollback ID: 30
Abandon the current transaction

Rollback-Ok ID: 31
Confirm successful rollback
<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Sync. Response</th>
<th>Short description</th>
<th>S</th>
<th>C</th>
<th>Field Name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tune</td>
<td>30</td>
<td>tune-ok</td>
<td>propose connection tuning parameters</td>
<td>Y</td>
<td></td>
<td>channel-max</td>
<td>short</td>
<td>proposed maximum channels</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>frame-max</td>
<td>long</td>
<td>proposed maximum frame size</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>heartbeat</td>
<td>short</td>
<td>desired heartbeat delay</td>
</tr>
<tr>
<td>tune-ok</td>
<td>31</td>
<td></td>
<td>negotiate connection tuning parameters</td>
<td>Y</td>
<td></td>
<td>channel-max</td>
<td>short</td>
<td>negotiated maximum channels</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>frame-max</td>
<td>long</td>
<td>negotiated maximum frame size</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>heartbeat</td>
<td>short</td>
<td>desired heartbeat delay</td>
</tr>
<tr>
<td>open</td>
<td>40</td>
<td>open-ok</td>
<td>open connection to virtual host</td>
<td>Y</td>
<td></td>
<td>virtual-host</td>
<td>path</td>
<td>virtual host name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>open-ok</td>
<td>41</td>
<td></td>
<td>signal that connection is ready</td>
<td>Y</td>
<td></td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>close</td>
<td>50</td>
<td>close-ok</td>
<td>request a connection close</td>
<td>Y</td>
<td>Y</td>
<td>reply-code</td>
<td>reply-code</td>
<td>reply-code</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>reply-text</td>
<td>reply-text</td>
<td>reply-text</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>class-id</td>
<td>class-id</td>
<td>failing method class</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>method-id</td>
<td>method-id</td>
<td>failing method ID</td>
</tr>
<tr>
<td>close-ok</td>
<td>51</td>
<td></td>
<td>confirm a connection close</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td>[ No parameters defined for this method ]</td>
</tr>
</tbody>
</table>

## 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(s) **connection.start-ok**

**Number of parameters**: 5

**Label**: start connection negotiation

**Parameter Summary**:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>version-major</td>
<td>octet</td>
<td>protocol major version</td>
</tr>
<tr>
<td>2</td>
<td>version-minor</td>
<td>octet</td>
<td>protocol minor version</td>
</tr>
<tr>
<td>3</td>
<td>server-properties</td>
<td>peer-properties</td>
<td>server properties</td>
</tr>
</tbody>
</table>
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 respond with a valid protocol header and then close the socket connection.

  **Test scenario:** The client sends a protocol header containing an invalid protocol name. The server MUST respond by sending a valid protocol header and then 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. 2.0. The server must respond with a protocol header indicating its supported protocol version, e.g. 1.0.

- If the client cannot handle the protocol version suggested by the server it MUST close the socket connection without sending any further data.

  **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 without sending any further data.

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

**Ordinal:** 1

**Domain:** octet

**Label:** protocol major version

The major version number can take any value from 0 to 99 as defined in the AMQP specification.

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

**Ordinal:** 2

**Domain:** octet

**Label:** protocol minor version

The minor version number can take any value from 0 to 99 as defined in the AMQP specification.

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

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>client-properties</td>
<td>peer-properties</td>
<td>client properties</td>
</tr>
<tr>
<td>2</td>
<td>mechanism</td>
<td>shortstr</td>
<td>selected security mechanism</td>
</tr>
<tr>
<td>3</td>
<td>response</td>
<td>longstr</td>
<td>security response data</td>
</tr>
<tr>
<td>4</td>
<td>locale</td>
<td>shortstr</td>
<td>selected message locale</td>
</tr>
</tbody>
</table>

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(s) connection.secure-ok
Number of parameters: 1
Label: security mechanism challenge

Parameter Summary:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>challenge</td>
<td>longstr</td>
<td>security challenge data</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>response</td>
<td>longstr</td>
<td>security response data</td>
</tr>
</tbody>
</table>

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(s) connection.tune-ok
Number of parameters: 3
Label: propose connection tuning parameters

Parameter Summary:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>channel-max</td>
<td>short</td>
<td>proposed maximum channels</td>
</tr>
<tr>
<td>2</td>
<td>frame-max</td>
<td>long</td>
<td>proposed maximum frame size</td>
</tr>
<tr>
<td>3</td>
<td>heartbeat</td>
<td>short</td>
<td>desired heartbeat delay</td>
</tr>
</tbody>
</table>

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

Specifies highest channel number that the server permits. Usable channel numbers are in the range 1..channel-max. Zero indicates no specified limit.

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, including frame header and end-byte. 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:**

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>channel-max</td>
<td>short</td>
<td>negotiated maximum channels</td>
</tr>
<tr>
<td>2</td>
<td>frame-max</td>
<td>long</td>
<td>negotiated maximum frame size</td>
</tr>
<tr>
<td>3</td>
<td>heartbeat</td>
<td>short</td>
<td>desired heartbeat delay</td>
</tr>
</tbody>
</table>

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(s) connection.open-ok
Number of parameters: 3
Label: open connection to virtual host

Parameter Summary:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>virtual-host</td>
<td>path</td>
<td>virtual host name</td>
</tr>
<tr>
<td>2</td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>reserved-2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.reserved-1 ()**

Ordinal: 2

Domain:

1.4.2.7.3. **Parameter connection.open.reserved-2 ()**

Ordinal: 3

Domain:

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:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

1.4.2.8.1. **Parameter connection.open-ok.reserved-1 ()**

Ordinal: 1

Domain:

1.4.2.9. **Method connection.close (ID 50)**

ID: 50

Method accepted by: Server, Client

Synchronous: Yes; expected response is from method(s) connection.close-ok

Number of parameters: 4
**Label:** request a connection close

**Parameter Summary:**

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reply-code</td>
<td>reply-code</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>reply-text</td>
<td>reply-text</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>class-id</td>
<td>class-id</td>
<td>failing method class</td>
</tr>
<tr>
<td>4</td>
<td>method-id</td>
<td>method-id</td>
<td>failing method ID</td>
</tr>
</tbody>
</table>

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 methods except Close and Close-OK MUST be discarded. The response to receiving a Close after sending Close must be to send Close-Ok.

1.4.2.9.1. **Parameter connection.close.reply-code (reply-code)**

**Ordinal:** 1

**Domain:** reply-code

1.4.2.9.2. **Parameter connection.close.reply-text (reply-text)**

**Ordinal:** 2

**Domain:** reply-text

1.4.2.9.3. **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.

1.4.2.9.4. **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.
1.4.2.10. 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
use-channel = C:FLOW S:FLOW-OK / S:FLOW C:FLOW-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):

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Sync. Response</th>
<th>Short description</th>
<th>S</th>
<th>C</th>
<th>Field Name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>10</td>
<td>open-ok</td>
<td>open a channel for use</td>
<td>Y</td>
<td></td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>open-ok</td>
<td>11</td>
<td></td>
<td>signal that the channel is ready</td>
<td></td>
<td>Y</td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flow</td>
<td>20</td>
<td>flow-ok</td>
<td>enable/disable flow from peer</td>
<td>Y</td>
<td>Y</td>
<td>active</td>
<td>bit</td>
<td>start/stop content frames</td>
</tr>
<tr>
<td>flow-ok</td>
<td>21</td>
<td></td>
<td>confirm a flow method</td>
<td>Y</td>
<td>Y</td>
<td>active</td>
<td>bit</td>
<td>current flow setting</td>
</tr>
<tr>
<td>close</td>
<td>40</td>
<td>close-ok</td>
<td>request a channel close</td>
<td>Y</td>
<td>Y</td>
<td>reply-code</td>
<td>reply-code</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>reply-text</td>
<td>reply-text</td>
<td></td>
</tr>
</tbody>
</table>
### 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(s) `channel.open-ok`  
**Number of parameters:** 1  
**Label:** open a channel for use

**Parameter Summary:**

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reserved-1</td>
<td></td>
</tr>
</tbody>
</table>

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.reserved-1`()

**Ordinal:** 1  
**Domain:**

#### 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:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

1.5.2.2.1. Parameter channel.open-ok.reserved-1 ()

Ordinal: 1
Domain:

1.5.2.3. Method channel.flow (ID 20)

ID: 20
Method accepted by: Server, Client
Synchronous: Yes; expected response is from method(s) channel.flow-ok
Number of parameters: 1
Label: enable/disable flow from peer

Parameter Summary:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>active</td>
<td>bit</td>
<td>start/stop content frames</td>
</tr>
</tbody>
</table>

This method asks the peer to pause or restart the flow of content data sent by a consumer. 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. It does not affect contents returned by Basic.Get-Ok methods.

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 server.

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:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>active</td>
<td>bit</td>
<td>current flow setting</td>
</tr>
</tbody>
</table>

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(s) *channel.close-ok*
Number of parameters: 4
Label: request a channel close

Parameter Summary:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reply-code</td>
<td>reply-code</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>reply-text</td>
<td>reply-text</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>class-id</td>
<td>class-id</td>
<td>failing method class</td>
</tr>
<tr>
<td>4</td>
<td>method-id</td>
<td>method-id</td>
<td>failing method ID</td>
</tr>
</tbody>
</table>

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 methods except Close and Close-OK MUST be discarded. The response to receiving a Close after sending Close must be to send Close-Ok.

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.

1.6. Class exchange

Exchanges match and distribute messages across queues. Exchanges can be configured in the server or declared 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 declares 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.

1.6.1. Property and Method Summary

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

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Sync.</th>
<th>Response</th>
<th>Short Description</th>
<th>S</th>
<th>C</th>
<th>Field Name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>declare</td>
<td>10</td>
<td>declare-ok</td>
<td>verify exchange exists, create if needed</td>
<td>Y</td>
<td></td>
<td>reserved-1</td>
<td></td>
<td>exchange-name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>ID</td>
<td>Sync. Response</td>
<td>Short description</td>
<td>S</td>
<td>C</td>
<td>Field Name</td>
<td>Domain</td>
<td>Short Description</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>----</td>
<td>----------------</td>
<td>-------------------</td>
<td>---</td>
<td>---</td>
<td>------------</td>
<td>--------</td>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>shortstr</td>
<td></td>
<td>exchange type</td>
<td></td>
</tr>
<tr>
<td>passive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>bit</td>
<td></td>
<td>do not create exchange</td>
<td></td>
</tr>
<tr>
<td>durable</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>bit</td>
<td></td>
<td>request a durable exchange</td>
<td></td>
</tr>
<tr>
<td>reserved-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reserved-3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>no-wait</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>no-wait</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arguments</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>table</td>
<td></td>
<td>arguments for declaration</td>
<td></td>
</tr>
</tbody>
</table>

**declare-ok** 11 confirm exchange declaration Y [ No parameters defined for this method ]

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Sync. Response</th>
<th>Short description</th>
<th>S</th>
<th>C</th>
<th>Field Name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>type</td>
<td>shortstr</td>
<td>exchange type</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>passive</td>
<td>bit</td>
<td>do not create exchange</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>durable</td>
<td>bit</td>
<td>request a durable exchange</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>reserved-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>reserved-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>no-wait</td>
<td>no-wait</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>arguments</td>
<td>table</td>
<td>arguments for declaration</td>
</tr>
</tbody>
</table>

**delete** 20 delete-ok delete an exchange Y [ No parameters defined for this method ]

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Sync. Response</th>
<th>Short description</th>
<th>S</th>
<th>C</th>
<th>Field Name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete-ok</td>
<td>21</td>
<td></td>
<td>confirm deletion of an exchange</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>[ No parameters defined for this method ]</td>
</tr>
</tbody>
</table>

### 1.6.2. Methods

#### 1.6.2.1. Method exchange.declare (ID 10)

**ID:** 10

**Method accepted by:** Server

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

**Number of parameters:** 9

**Label:** verify exchange exists, create if needed

**Parameter Summary:**

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>type</td>
<td>shortstr</td>
<td>exchange type</td>
</tr>
<tr>
<td>4</td>
<td>passive</td>
<td>bit</td>
<td>do not create exchange</td>
</tr>
<tr>
<td>5</td>
<td>durable</td>
<td>bit</td>
<td>request a durable exchange</td>
</tr>
<tr>
<td>6</td>
<td>reserved-2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>reserved-3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>no-wait</td>
<td>no-wait</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>arguments</td>
<td>table</td>
<td>arguments for declaration</td>
</tr>
</tbody>
</table>
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 declares as many exchanges as it can until the server reports an error; the number of exchanges successfully declared must be at least sixteen.

1.6.2.1.1. Parameter exchange.declare.reserved-1 ()

Ordinal: 1
Domain:

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

Ordinal: 2
Domain: exchange-name

1.6.2.1.3. 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.

1.6.2.1.4. Parameter exchange.declare.passive (bit)

Ordinal: 4
Domain: bit
Label: do not create exchange

If set, the server will reply with Declare-Ok if the exchange already exists with the same name, and raise an error if not. The client can use this to check whether an exchange exists without modifying the server state. When set, all other method fields except name and no-wait are ignored. A declare with both passive and no-wait has no effect. Arguments are compared for semantic equivalence.

1.6.2.1.5. 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.

1.6.2.1.6. Parameter exchange.declare.reserved-2 ()
Ordinal: 6
Domain:

1.6.2.1.7. Parameter exchange.declare.reserved-3 ()
Ordinal: 7
Domain:

1.6.2.1.8. Parameter exchange.declare.no-wait (no-wait)
Ordinal: 8
Domain: no-wait

1.6.2.1.9. 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.

1.6.2.2. 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.

1.6.2.3. Method exchange.delete (ID 20)
ID: 20
Method accepted by: Server
Synchronous: Yes; expected response is from method(s) exchange.delete-ok
Number of parameters: 4
Label: delete an exchange

Parameter Summary:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>if-unused</td>
<td>bit</td>
<td>delete only if unused</td>
</tr>
<tr>
<td>4</td>
<td>no-wait</td>
<td>no-wait</td>
<td></td>
</tr>
</tbody>
</table>

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

1.6.2.3.1. Parameter exchange.delete.reserved-1 ()
Ordinal: 1
Domain: 

1.6.2.3.2. Parameter exchange.delete.exchange (exchange-name)
Ordinal: 2
Domain: exchange-name

1.6.2.3.3. 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.

1.6.2.3.4. Parameter exchange.delete.no-wait (no-wait)
Ordinal: 4
Domain: no-wait

1.6.2.4. 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.

1.7. **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:UNBIND S:UNBIND-OK
/ C:PURGE S:PURGE-OK
/ C:DELETE S:DELETE-OK
```

1.7.1. **Property and Method Summary**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Sync. Response</th>
<th>Short description</th>
<th>S</th>
<th>C</th>
<th>Field Name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>declare</td>
<td>10</td>
<td>declare-ok</td>
<td>declare queue,</td>
<td>Y</td>
<td></td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>create if needed</td>
<td></td>
<td></td>
<td>queue</td>
<td>queue-name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>passive bit</td>
<td></td>
<td></td>
<td>do not create queue</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>durable bit</td>
<td></td>
<td></td>
<td>request a durable queue</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>exclusive bit</td>
<td></td>
<td></td>
<td>request an exclusive queue</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>auto-delete bit</td>
<td></td>
<td></td>
<td>auto-delete queue when unused</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>no-wait</td>
<td></td>
<td></td>
<td>no-wait</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>arguments table</td>
<td></td>
<td></td>
<td>arguments for declaration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>declare-ok</td>
<td>11</td>
<td></td>
<td>confirms a queue definition</td>
<td>Y</td>
<td></td>
<td>queue</td>
<td>queue-name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>message-count</td>
<td></td>
<td></td>
<td>message-count</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>consumer-count</td>
<td></td>
<td></td>
<td>long</td>
<td></td>
<td>number of consumers</td>
</tr>
<tr>
<td>bind</td>
<td>20</td>
<td>bind-ok</td>
<td>bind queue to an exchange</td>
<td>Y</td>
<td></td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>queue</td>
<td></td>
<td></td>
<td>queue-name</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>exchange</td>
<td></td>
<td></td>
<td>exchange-name</td>
<td></td>
<td>name of the exchange to bind to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>routing-key</td>
<td></td>
<td></td>
<td>shortstr</td>
<td></td>
<td>message routing key</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>no-wait</td>
<td></td>
<td></td>
<td>no-wait</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>arguments table</td>
<td></td>
<td></td>
<td>arguments for binding</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bind-ok</td>
<td>21</td>
<td></td>
<td>confirm bind successful</td>
<td>Y</td>
<td></td>
<td>[ No parameters defined for this method ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unbind</td>
<td>50</td>
<td>unbind-ok</td>
<td>unbind a queue from an</td>
<td>Y</td>
<td></td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 1.7.2. Methods

#### 1.7.2.1. Method queue.declare (ID 10)

**ID:** 10  

**Method accepted by:** Server  

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

**Number of parameters:** 8  

**Label:** declare queue, create if needed  

**Parameter Summary:**

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>queue</td>
<td>queue-name</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>passive</td>
<td>bit</td>
<td>do not create queue</td>
</tr>
<tr>
<td>4</td>
<td>durable</td>
<td>bit</td>
<td>request a durable queue</td>
</tr>
<tr>
<td>5</td>
<td>exclusive</td>
<td>bit</td>
<td>request an exclusive queue</td>
</tr>
<tr>
<td>6</td>
<td>auto-delete</td>
<td>bit</td>
<td>auto-delete queue when unused</td>
</tr>
<tr>
<td>7</td>
<td>no-wait</td>
<td>no-wait</td>
<td></td>
</tr>
</tbody>
</table>
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-declared queue to the default exchange, which is an exchange of type 'direct' and use the queue name as the routing key.
  
  **Test scenario:** Client declares 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 declare as many queues as it can until the server reports an error. The resulting count must at least be 256.

1.7.2.1.1. **Parameter queue.declare.reserved-1 ()**

Ordinal: 1

Domain:

1.7.2.1.2. **Parameter queue.declare.queue (queue-name)**

Ordinal: 2

Domain: queue-name

1.7.2.1.3. **Parameter queue.declare.passive (bit)**

Ordinal: 3

Domain: bit

Label: do not create queue

If set, the server will reply with Declare-Ok if the queue already exists with the same name, and raise an error if not. The client can use this to check whether a queue exists without modifying the server state. When set, all other method fields except name and no-wait are ignored. A declare with both passive and no-wait has no effect. Arguments are compared for semantic equivalence.

1.7.2.1.4. **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
1.7.2.1.5. **Parameter queue.declare.exclusive (bit)**

**Ordinal:** 5  
**Domain:** bit  
**Label:** request an exclusive queue

Exclusive queues may only be accessed by the current connection, and are deleted when that connection closes. Passive declaration of an exclusive queue by other connections are not allowed.

1.7.2.1.6. **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. The 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. Applications can explicitly delete auto-delete queues using the Delete method as normal.

1.7.2.1.7. **Parameter queue.declare.no-wait (no-wait)**

**Ordinal:** 7  
**Domain:** no-wait

1.7.2.1.8. **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.

1.7.2.2. **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:**

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This method confirms a Declare method and confirms the name of the queue, essential for automatically-named queues.

1.7.2.2.1. **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.

1.7.2.2.2. **Parameter queue.declare-ok.message-count (message-count)**

**Ordinal:** 2

**Domain:** message-count

1.7.2.2.3. **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.

1.7.2.3. **Method queue.bind (ID 20)**

**ID:** 20

**Method accepted by:** Server

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

**Number of parameters:** 6

**Label:** bind queue to an exchange

**Parameter Summary:**

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>queue</td>
<td>queue-name</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>exchange</td>
<td>exchange-name</td>
<td>name of the exchange to bind to</td>
</tr>
<tr>
<td>4</td>
<td>routing-key</td>
<td>shortstr</td>
<td>message routing key</td>
</tr>
<tr>
<td>5</td>
<td>no-wait</td>
<td>no-wait</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>arguments</td>
<td>table</td>
<td>arguments for binding</td>
</tr>
</tbody>
</table>
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).

- A server MUST not deliver the same message more than once to a queue, even if the queue has multiple bindings that match the message.
  Test scenario: A client declares a named queue and binds it using multiple bindings to the amq.topic exchange. The client then publishes a message that matches all its bindings.

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

- Bindings of durable queues to durable exchanges are automatically durable and the server MUST restore such bindings after a server restart.
  Test scenario: A server declares 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.

- 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 declares a named queue and attempts to bind it to 4 different exchanges.

1.7.2.3.1. Parameter queue.bind.reserved-1 ()

Ordinal: 1
Domain:

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

Ordinal: 2
Domain: queue-name
Specifies the name of the queue to bind.

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

Ordinal: 3
Domain: exchange-name
Label: name of the exchange to bind to
1.7.2.3.4. **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.

1.7.2.3.5. **Parameter queue.bind.no-wait (no-wait)**

**Ordinal:** 5  
**Domain:** no-wait

1.7.2.3.6. **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.

1.7.2.4. **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.

1.7.2.5. **Method queue.unbind (ID 50)**

**ID:** 50  
**Method accepted by:** Server  
**Synchronous:** Yes; expected response is from method(s) queue.unbind-ok  
**Number of parameters:** 5  
**Label:** unbind a queue from an exchange

**Parameter Summary:**
This method unbinds a queue from an exchange.

**Guidelines for implementers:**
- If a unbind fails, the server MUST raise a connection exception.

### 1.7.2.5.1. Parameter `queue.unbind.reserved-1()`

**Ordinal:** 1  
**Domain:** 

### 1.7.2.5.2. Parameter `queue.unbind.queue(queue-name)`

**Ordinal:** 2  
**Domain:** `queue-name`  
Specifies the name of the queue to unbind.

### 1.7.2.5.3. Parameter `queue.unbind.exchange(exchange-name)`

**Ordinal:** 3  
**Domain:** `exchange-name`  
The name of the exchange to unbind from.

### 1.7.2.5.4. Parameter `queue.unbind.routing-key(shortstr)`

**Ordinal:** 4  
**Domain:** `shortstr`  
**Label:** routing key of binding  
Specifies the routing key of the binding to unbind.

### 1.7.2.5.5. Parameter `queue.unbind.arguments(table)`

**Ordinal:** 5  
**Domain:** `table`  
**Label:** arguments of binding  
Specifies the arguments of the binding to unbind.
1.7.2.6. **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.

1.7.2.7. **Method queue.purge (ID 30)**

ID: 30

Method accepted by: Server

Synchronous: Yes; expected response is from method(s) `queue.purge-ok`

Number of parameters: 3

Label: purge a queue

Parameter Summary:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>queue</td>
<td>queue-name</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>no-wait</td>
<td>no-wait</td>
<td></td>
</tr>
</tbody>
</table>

This method removes all messages from a queue which are not awaiting acknowledgment.

Guidelines for implementers:

- 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.

1.7.2.7.1. **Parameter queue.purge.reserved-1 ()**

Ordinal: 1

Domain:

1.7.2.7.2. **Parameter queue.purge.queue (queue-name)**

Ordinal: 2

Domain: queue-name

Specifies the name of the queue to purge.
1.7.2.7.3.  Parameter `queue.purge.no-wait` (no-wait)

Ordinal: 3
Domain: no-wait

1.7.2.8.  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:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>message-count</code></td>
<td><code>message-count</code></td>
<td></td>
</tr>
</tbody>
</table>

This method confirms the purge of a queue.

1.7.2.8.1.  Parameter `queue.purge-ok.message-count` (message-count)

Ordinal: 1
Domain: message-count

Reports the number of messages purged.

1.7.2.9.  Method `queue.delete` (ID 40)

ID: 40
Method accepted by: Server
Synchronous: Yes; expected response is from method(s) `queue.delete-ok`
Number of parameters: 5
Label: delete a queue

Parameter Summary:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>reserved-1</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><code>queue</code></td>
<td><code>queue-name</code></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td><code>if-unused</code></td>
<td><code>bit</code></td>
<td>delete only if unused</td>
</tr>
<tr>
<td>4</td>
<td><code>if-empty</code></td>
<td><code>bit</code></td>
<td>delete only if empty</td>
</tr>
<tr>
<td>5</td>
<td><code>no-wait</code></td>
<td><code>no-wait</code></td>
<td></td>
</tr>
</tbody>
</table>

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.

1.7.2.9.1. Parameter queue.delete.reserved-1 ()

Ordinal: 1
Domain:

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

Ordinal: 2
Domain: queue-name
Specifies the name of the queue to delete.

1.7.2.9.3. 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 not delete it but raises a channel exception instead.

1.7.2.9.4. 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.

1.7.2.9.5. Parameter queue.delete.no-wait (no-wait)

Ordinal: 5
Domain: no-wait

1.7.2.10. 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:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>message-count</td>
<td>message-count</td>
<td></td>
</tr>
</tbody>
</table>

This method confirms the deletion of a queue.

#### 1.7.2.10.1. Parameter queue.delete-ok.message-count (message-count)

**Ordinal:** 1

**Domain:** message-count

Reports the number of messages deleted.

### 1.8. Class basic

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

**Class Grammar:**

```plaintext
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
/ C:RECOVER-ASYNC
/ C:RECOVER S:RECOVER-OK
```

### 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:** Declare 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:** Declare 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 un-acknowledged delivery of Basic content, i.e. consumers with the no-ack field set to TRUE.
- The server MUST support explicitly acknowledged delivery of Basic content, i.e. consumers with the no-ack field set to FALSE.

**Test scenario:** Declare 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.

### 1.8.1. Property and Method Summary

Class `basic` defines the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>content-type</td>
<td>shortstr</td>
<td>MIME content type</td>
</tr>
<tr>
<td>content-encoding</td>
<td>shortstr</td>
<td>MIME content encoding</td>
</tr>
<tr>
<td>headers</td>
<td>table</td>
<td>message header field table</td>
</tr>
<tr>
<td>delivery-mode</td>
<td>octet</td>
<td>non-persistent (1) or persistent (2)</td>
</tr>
<tr>
<td>priority</td>
<td>octet</td>
<td>message priority, 0 to 9</td>
</tr>
<tr>
<td>correlation-id</td>
<td>shortstr</td>
<td>application correlation identifier</td>
</tr>
<tr>
<td>reply-to</td>
<td>shortstr</td>
<td>address to reply to</td>
</tr>
<tr>
<td>expiration</td>
<td>shortstr</td>
<td>message expiration specification</td>
</tr>
<tr>
<td>message-id</td>
<td>shortstr</td>
<td>application message identifier</td>
</tr>
<tr>
<td>timestamp</td>
<td>timestamp</td>
<td>message timestamp</td>
</tr>
<tr>
<td>type</td>
<td>shortstr</td>
<td>message type name</td>
</tr>
<tr>
<td>user-id</td>
<td>shortstr</td>
<td>creating user id</td>
</tr>
</tbody>
</table>
### AMQP Specification (XML-derived)

#### Table: Short Description

<table>
<thead>
<tr>
<th>Name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>app-id</td>
<td>shortstr</td>
<td>creating application id</td>
</tr>
<tr>
<td>reserved</td>
<td>shortstr</td>
<td>reserved, must be empty</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Sync. Response</th>
<th>Short description</th>
<th>S</th>
<th>C</th>
<th>Field Name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qos</td>
<td>10</td>
<td>qos-ok</td>
<td>specify quality of service</td>
<td>Y</td>
<td></td>
<td>prefetch-size</td>
<td>long</td>
<td>prefetch window in octets</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>prefetch-count</td>
<td>short</td>
<td>prefetch window in messages</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>global</td>
<td>bit</td>
<td>apply to entire connection</td>
</tr>
<tr>
<td>qos-ok</td>
<td>11</td>
<td></td>
<td>confirm the requested qos</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td>[ No parameters defined for this method ]</td>
</tr>
<tr>
<td>consume</td>
<td>20</td>
<td>consume-ok</td>
<td>start a queue consumer</td>
<td>Y</td>
<td></td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>queue</td>
<td>queue-name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>consumer-tag</td>
<td>consumer-tag</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>no-local</td>
<td>no-local</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>no-ack</td>
<td>no-ack</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>exclusive</td>
<td>bit</td>
<td>request exclusive access</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>no-wait</td>
<td>no-wait</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>arguments</td>
<td>table</td>
<td>arguments for declaration</td>
</tr>
<tr>
<td>consume-ok</td>
<td>21</td>
<td></td>
<td>confirm a new consumer</td>
<td>Y</td>
<td></td>
<td>consumer-tag</td>
<td>consumer-tag</td>
<td></td>
</tr>
<tr>
<td>cancel</td>
<td>30</td>
<td>cancel-ok</td>
<td>end a queue consumer</td>
<td>Y</td>
<td></td>
<td>consumer-tag</td>
<td>consumer-tag</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>no-wait</td>
<td>no-wait</td>
<td></td>
</tr>
<tr>
<td>cancel-ok</td>
<td>31</td>
<td></td>
<td>confirm a cancelled consumer</td>
<td>Y</td>
<td></td>
<td>consumer-tag</td>
<td>consumer-tag</td>
<td></td>
</tr>
<tr>
<td>publish</td>
<td>40</td>
<td>publish</td>
<td>publish a message</td>
<td>Y</td>
<td></td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
<td></td>
<td>exchange</td>
<td>exchange-name</td>
<td>Message routing key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>routing-key</td>
<td>shortstr</td>
<td></td>
<td></td>
<td>mandatory</td>
<td>bit</td>
<td>indicate mandatory routing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>immediate</td>
<td>bit</td>
<td></td>
<td></td>
<td></td>
<td>bit</td>
<td>request immediate delivery</td>
</tr>
<tr>
<td>return</td>
<td>50</td>
<td>return</td>
<td>return a failed message</td>
<td>Y</td>
<td></td>
<td>reply-code</td>
<td>reply-code</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
<td></td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>routing-key</td>
<td>shortstr</td>
<td></td>
<td></td>
<td>routing-key</td>
<td>shortstr</td>
<td>Message routing key</td>
</tr>
<tr>
<td>deliver</td>
<td>60</td>
<td>notify the client of a consumer message</td>
<td>Y</td>
<td></td>
<td>consumer-tag</td>
<td>consumer-tag</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>delivery-tag</td>
<td>delivery-tag</td>
<td></td>
<td></td>
<td>delivery-tag</td>
<td>delivery-tag</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>redelivered</td>
<td>redelivered</td>
<td></td>
<td></td>
<td>redelivered</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
<td></td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>ID</td>
<td>Sync. Response</td>
<td>Short description</td>
<td>S</td>
<td>C</td>
<td>Field Name</td>
<td>Domain</td>
<td>Short Description</td>
</tr>
<tr>
<td>------------</td>
<td>----</td>
<td>----------------</td>
<td>-------------------</td>
<td>---</td>
<td>---</td>
<td>------------</td>
<td>------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>get</td>
<td>70</td>
<td>get-ok</td>
<td>direct access to a queue</td>
<td>Y</td>
<td></td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>get-ok</td>
<td>71</td>
<td></td>
<td>provide client with a message</td>
<td>Y</td>
<td></td>
<td>delivery-tag</td>
<td>delivery-tag</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>redelivered</td>
<td>redelivered</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>routing-key</td>
<td>shortstr</td>
<td>Message routing key</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>message-count</td>
<td>message-count</td>
<td></td>
</tr>
<tr>
<td>get-empty</td>
<td>72</td>
<td></td>
<td>indicate no messages available</td>
<td>Y</td>
<td></td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ack</td>
<td>80</td>
<td></td>
<td>acknowledge one or more messages</td>
<td>Y</td>
<td></td>
<td>delivery-tag</td>
<td>delivery-tag</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>multiple</td>
<td></td>
<td></td>
<td>bit</td>
<td>acknowledge multiple messages</td>
<td></td>
</tr>
<tr>
<td>reject</td>
<td>90</td>
<td></td>
<td>reject an incoming message</td>
<td>Y</td>
<td></td>
<td>delivery-tag</td>
<td>delivery-tag</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>requeue</td>
<td></td>
<td></td>
<td>bit</td>
<td>requeue the message</td>
<td></td>
</tr>
<tr>
<td>recover-async</td>
<td>100</td>
<td></td>
<td>redeliver unacknowledge d messages</td>
<td>Y</td>
<td></td>
<td>requeue</td>
<td>bit</td>
<td>requeue the message</td>
</tr>
<tr>
<td>recover</td>
<td>110</td>
<td></td>
<td>redeliver unacknowledge d messages</td>
<td>Y</td>
<td></td>
<td>requeue</td>
<td>bit</td>
<td>requeue the message</td>
</tr>
<tr>
<td>recover-ok</td>
<td>111</td>
<td></td>
<td>confirm recovery</td>
<td>Y</td>
<td></td>
<td>[ No parameters defined for this method ]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.8.2. Properties

1.8.2.1. Property basic.content-type (shortstr)

Domain: shortstr
Label: MIME content type

1.8.2.2. Property basic.content-encoding (shortstr)

Domain: shortstr
Label: MIME content encoding

1.8.2.3. Property basic.headers (table)

Domain: table
Label: message header field table
1.8.2.4.  *Property basic.delivery-mode (octet)*

- **Domain:** octet
- **Label:** non-persistent (1) or persistent (2)

1.8.2.5.  *Property basic.priority (octet)*

- **Domain:** octet
- **Label:** message priority, 0 to 9

1.8.2.6.  *Property basic.correlation-id (shortstr)*

- **Domain:** shortstr
- **Label:** application correlation identifier

1.8.2.7.  *Property basic.reply-to (shortstr)*

- **Domain:** shortstr
- **Label:** address to reply to

1.8.2.8.  *Property basic.expiration (shortstr)*

- **Domain:** shortstr
- **Label:** message expiration specification

1.8.2.9.  *Property basic.message-id (shortstr)*

- **Domain:** shortstr
- **Label:** application message identifier

1.8.2.10.  *Property basic.timestamp (timestamp)*

- **Domain:** timestamp
- **Label:** message timestamp

1.8.2.11.  *Property basic.type (shortstr)*

- **Domain:** shortstr
- **Label:** message type name

1.8.2.12.  *Property basic.user-id (shortstr)*

- **Domain:** shortstr
- **Label:** creating user id

1.8.2.13.  *Property basic.app-id (shortstr)*

- **Domain:** shortstr
Label: creating application id

1.8.2.14. Property basic.reserved (shortstr)

Domain: shortstr

Label: reserved, must be empty

1.8.3. Methods

1.8.3.1. Method basic.qos (ID 10)

ID: 10

Method accepted by: Server

Synchronous: Yes; expected response is from method(s) basic.qos-ok

Number of parameters: 3

Label: specify quality of service

Parameter Summary:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>prefetch-size</td>
<td>long</td>
<td>prefetch window in octets</td>
</tr>
<tr>
<td>2</td>
<td>prefetch-count</td>
<td>short</td>
<td>prefetch window in messages</td>
</tr>
<tr>
<td>3</td>
<td>global</td>
<td>bit</td>
<td>apply to entire connection</td>
</tr>
</tbody>
</table>

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.

1.8.3.1.1. 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.

1.8.3.1.2. 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.

1.8.3.1.3. 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.

1.8.3.2. 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.

1.8.3.3. Method basic.consume (ID 20)

ID: 20
Method accepted by: Server
Synchronous: Yes; expected response is from method(s) basic.consume-ok
Number of parameters: 8
Label: start a queue consumer

Parameter Summary:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>queue</td>
<td>queue-name</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>consumer-tag</td>
<td>consumer-tag</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>no-local</td>
<td>no-local</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>no-ack</td>
<td>no-ack</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>exclusive</td>
<td>bit</td>
<td>request exclusive access</td>
</tr>
<tr>
<td>7</td>
<td>no-wait</td>
<td>no-wait</td>
<td></td>
</tr>
</tbody>
</table>
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 declared 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:** Declare 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.

### 1.8.3.3.1. Parameter `basic.consume.reserved-1` ()

**Ordinal:** 1

**Domain:**

### 1.8.3.3.2. Parameter `basic.consume.queue` (queue-name)

**Ordinal:** 2

**Domain:** queue-name

Specifies the name of the queue to consume from.

### 1.8.3.3.3. 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 channel, so two clients can use the same consumer tags. If this field is empty the server will generate a unique tag.

### 1.8.3.3.4. Parameter `basic.consume.no-local` (no-local)

**Ordinal:** 4

**Domain:** no-local

### 1.8.3.3.5. Parameter `basic.consume.no-ack` (no-ack)

**Ordinal:** 5

**Domain:** no-ack

### 1.8.3.3.6. 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.

1.8.3.3.7. Parameter basic.consume.no-wait (no-wait)
Ordinal: 7
Domain: no-wait

1.8.3.8. Parameter basic.consume.arguments (table)
Ordinal: 8
Domain: table
Label: arguments for declaration
A set of arguments for the consume. The syntax and semantics of these arguments depends on the server implementation.

1.8.3.4. 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:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>consumer-tag</td>
<td>consumer-tag</td>
<td></td>
</tr>
</tbody>
</table>

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.

1.8.3.4.1. 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.

1.8.3.5. Method basic.cancel (ID 30)
ID: 30
Method accepted by: Server
**Synchronous:** Yes; expected response is from method(s) **basic.cancel-ok**

**Number of parameters:** 2

**Label:** end a queue consumer

**Parameter Summary:**

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>consumer-tag</td>
<td>consumer-tag</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>no-wait</td>
<td>no-wait</td>
<td></td>
</tr>
</tbody>
</table>

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.

**1.8.3.5.1. Parameter basic.cancel.consumer-tag (consumer-tag)**

**Ordinal:** 1

**Domain:** consumer-tag

**1.8.3.5.2. Parameter basic.cancel.no-wait (no-wait)**

**Ordinal:** 2

**Domain:** no-wait

**1.8.3.6. 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:**

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>consumer-tag</td>
<td>consumer-tag</td>
<td></td>
</tr>
</tbody>
</table>

This method confirms that the cancellation was completed.
1.8.3.6.1. Parameter basic.cancel-ok.consumer-tag (consumer-tag)

Ordinal: 1
Domain: consumer-tag

1.8.3.7. Method basic.publish (ID 40)

ID: 40
Method accepted by: Server
Synchronous: No
Number of parameters: 5
Label: publish a message

Parameter Summary:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>routing-key</td>
<td>shortstr</td>
<td>Message routing key</td>
</tr>
<tr>
<td>4</td>
<td>mandatory</td>
<td>bit</td>
<td>indicate mandatory routing</td>
</tr>
<tr>
<td>5</td>
<td>immediate</td>
<td>bit</td>
<td>request immediate delivery</td>
</tr>
</tbody>
</table>

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.

1.8.3.7.1. Parameter basic.publish.reserved-1 ()

Ordinal: 1
Domain:

1.8.3.7.2. 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.

1.8.3.7.3. 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.

1.8.3.7.4. 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.

1.8.3.7.5. 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.

1.8.3.8. Method basic.return (ID 50)

ID: 50
Method accepted by: Client
Synchronous: No
Number of parameters: 4
Label: return a failed message

Parameter Summary:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reply-code</td>
<td>reply-code</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>reply-text</td>
<td>reply-text</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>routing-key</td>
<td>shortstr</td>
<td>Message routing key</td>
</tr>
</tbody>
</table>

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.

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

Ordinal: 1
Domain: reply-code

**1.8.3.8.2. Parameter basic.return.reply-text (reply-text)**

Ordinal: 2

Domain: reply-text

**1.8.3.8.3. Parameter basic.return.exchange (exchange-name)**

Ordinal: 3

Domain: exchange-name

Specifies the name of the exchange that the message was originally published to. May be empty, meaning the default exchange.

**1.8.3.8.4. 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.

**1.8.3.9. 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:**

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>consumer-tag</td>
<td>consumer-tag</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>delivery-tag</td>
<td>delivery-tag</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>redelivered</td>
<td>redelivered</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>routing-key</td>
<td>shortstr</td>
<td>Message routing key</td>
</tr>
</tbody>
</table>

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.

1.8.3.9.1. **Parameter basic.deliver.consumer-tag (consumer-tag)**

Ordinal: 1  
Domain: consumer-tag

1.8.3.9.2. **Parameter basic.deliver.delivery-tag (delivery-tag)**

Ordinal: 2  
Domain: delivery-tag

1.8.3.9.3. **Parameter basic.deliver.redelivered (redelivered)**

Ordinal: 3  
Domain: redelivered

1.8.3.9.4. **Parameter basic.deliver.exchange (exchange-name)**

Ordinal: 4  
Domain: exchange-name  
Specifies the name of the exchange that the message was originally published to. May be empty, indicating the default exchange.

1.8.3.9.5. **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.

1.8.3.10. **Method basic.get (ID 70)**

ID: 70  
Method accepted by: Server  
Synchronous: Yes; expected response is from method(s) **basic.get-ok, basic.get-empty**  
Number of parameters: 3  
Label: direct access to a queue

Parameter Summary:
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.

1.8.3.10.1. Parameter basic.get.reserved-1 ()

Ordinal: 1
Domain:

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

Ordinal: 2
Domain: queue-name
Specifies the name of the queue to get a message from.

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

Ordinal: 3
Domain: no-ack

1.8.3.11. 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:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>delivery-tag</td>
<td>delivery-tag</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>redelivered</td>
<td>redelivered</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>routing-key</td>
<td>shortstr</td>
<td>Message routing key</td>
</tr>
<tr>
<td>5</td>
<td>message-count</td>
<td>message-count</td>
<td></td>
</tr>
</tbody>
</table>

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.

1.8.3.11.1.  Parameter basic.get-ok.delivery-tag (delivery-tag)
Ordinal: 1
Domain: delivery-tag

1.8.3.11.2.  Parameter basic.get-ok.redelivered (redelivered)
Ordinal: 2
Domain: redelivered

1.8.3.11.3.  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.

1.8.3.11.4.  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.

1.8.3.11.5.  Parameter basic.get-ok.message-count (message-count)
Ordinal: 5
Domain: message-count

1.8.3.12.  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:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>reserved-1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This method tells the client that the queue has no messages available for the client.

1.8.3.12.1. Parameter basic.get-empty.reserved-1 ()

Ordinal: 1
Domain:

1.8.3.13. 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:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>delivery-tag</td>
<td>delivery-tag</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>multiple</td>
<td>bit</td>
<td>acknowledge multiple messages</td>
</tr>
</tbody>
</table>

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.

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

Ordinal: 1
Domain: delivery-tag

1.8.3.13.2. 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.

1.8.3.14. Method basic.reject (ID 90)

ID: 90
Method accepted by: Server
Synchronous: No
Number of parameters: 2
**Label:** reject an incoming message

**Parameter Summary:**

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>delivery-tag</td>
<td>delivery-tag</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>requeue</td>
<td>bit</td>
<td>requeue the message</td>
</tr>
</tbody>
</table>

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.

- The client MUST NOT use this method as a means of selecting messages to process.

  **Test scenario:** TODO.

1.8.3.14.1.  **Parameter basic.reject.delivery-tag (delivery-tag)**

**Ordinal:** 1

**Domain:** delivery-tag

1.8.3.14.2.  **Parameter basic.reject.requeue (bit)**

**Ordinal:** 2

**Domain:** bit

**Label:** requeue the message

If requeue is true, the server will attempt to requeue the message. If requeue is false or the requeue attempt fails the messages are discarded or dead-lettered.

1.8.3.15.  **Method basic.recover-async (ID 100)**

**ID:** 100

**Method accepted by:** Server

**Synchronous:** No

**Number of parameters:** 1

**Label:** redeliver unacknowledged messages

**Parameter Summary:**
This method asks the server to redeliver all unacknowledged messages on a specified channel. Zero or more messages may be redelivered. This method is deprecated in favour of the synchronous Recover/Recover-Ok.

Guidelines for implementers:
- The server MUST set the redelivered flag on all messages that are resent.
  Test scenario: TODO.

1.8.3.15.1. Parameter basic.recover-async.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.

1.8.3.16. Method basic.recover (ID 110)

ID: 110
Method accepted by: Server
Synchronous: No
Number of parameters: 1
Label: redeliver unacknowledged messages

Parameter Summary:

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Parameter name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>requeue</td>
<td>bit</td>
<td>requeue the message</td>
</tr>
</tbody>
</table>

This method asks the server to redeliver all unacknowledged messages on a specified channel. Zero or more messages may be redelivered. This method replaces the asynchronous Recover.

Guidelines for implementers:
- The server MUST set the redelivered flag on all messages that are resent.
  Test scenario: TODO.

1.8.3.16.1. 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.

### 1.8.3.17. Method basic.recover-ok (ID 111)

ID: 111

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm recovery

This method acknowledges a Basic.Recover method.

### 1.9. Class tx

The Tx class allows publish and ack operations to be batched into atomic units of work. The intention is that all publish and ack requests issued within a transaction will complete successfully or none of them will. Servers SHOULD implement atomic transactions at least where all publish or ack requests affect a single queue. Transactions that cover multiple queues may be non-atomic, given that queues can be created and destroyed asynchronously, and such events do not form part of any transaction. Further, the behaviour of transactions with respect to the immediate and mandatory flags on Basic.Publish methods is not defined.

**Class Grammar:**

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

**Guidelines for implementers:**

- Applications MUST NOT rely on the atomicity of transactions that affect more than one queue.
- Applications MUST NOT rely on the behaviour of transactions that include messages published with the immediate option.
- Applications MUST NOT rely on the behaviour of transactions that include messages published with the mandatory option.

### 1.9.1. Property and Method Summary

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

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Sync. Response</th>
<th>Short description</th>
<th>S</th>
<th>C</th>
<th>Field Name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>select</td>
<td>10</td>
<td>select-ok</td>
<td>select standard transaction mode</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>[ No parameters defined for this method ]</td>
</tr>
</tbody>
</table>
### 1.9.2. Methods

#### 1.9.2.1. Method tx.select (ID 10)

**ID:** 10

**Method accepted by:** Server

**Synchronous:** Yes; expected response is from method(s) 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.

#### 1.9.2.2. 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.

#### 1.9.2.3. Method tx.commit (ID 20)

**ID:** 20

**Method accepted by:** Server
Synchronous: Yes; expected response is from method(s) tx.commit-ok

Number of parameters: 0

Label: commit the current transaction

This method commits all message publications and acknowledgments performed in the current transaction. A new transaction starts immediately after a commit.

Guidelines for implementers:

- The client MUST NOT use the Commit method on non-transacted channels.

  Test scenario: The client opens a channel and then uses Tx.Commit.

  On failure: Constant "precondition-failed" (See AMQP-defined Constants)

1.9.2.4. 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.

1.9.2.5. Method tx.rollback (ID 30)

ID: 30

Method accepted by: Server

Synchronous: Yes; expected response is from method(s) tx.rollback-ok

Number of parameters: 0

Label: abandon the current transaction

This method abandons all message publications and acknowledgments performed in the current transaction. A new transaction starts immediately after a rollback. Note that unacked messages will not be automatically redelivered by rollback; if that is required an explicit recover call should be issued.

Guidelines for implementers:

- The client MUST NOT use the Rollback method on non-transacted channels.

  Test scenario: The client opens a channel and then uses Tx.Rollback.

  On failure: Constant "precondition-failed" (See AMQP-defined Constants)

1.9.2.6. 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.