## Table of Contents

1. AMQP version 0-9 specification ................................................................................................................................. 12
   1.1. AMQP-defined Domains ........................................................................................................................................ 12
   1.2. AMQP-defined Constants ......................................................................................................................................... 13
   1.3. Class and Method ID Summaries ............................................................................................................................ 15
   1.4. Class connection ....................................................................................................................................................... 18
      1.4.1. Property and Method Summary ......................................................................................................................... 19
      1.4.2. Methods ............................................................................................................................................................... 20
         1.4.2.1. Method connection.start (ID 10) ................................................................................................................... 20
            1.4.2.1.1. Parameter connection.start.version-major (octet) ................................................................................. 21
            1.4.2.1.2. Parameter connection.start.version-minor (octet) ............................................................................... 21
            1.4.2.1.3. Parameter connection.start.server-properties (peer-properties) .............................................................. 21
            1.4.2.1.4. Parameter connection.start.mechanisms (longstr) .............................................................................. 22
            1.4.2.1.5. Parameter connection.startlocales (longstr) ......................................................................................... 22
         1.4.2.2. Method connection.start-ok (ID 11) ............................................................................................................. 22
            1.4.2.2.1. Parameter connection.start-ok.client-properties (peer-properties) ......................................................... 23
            1.4.2.2.2. Parameter connection.start-ok.mechanism (shortstr) ................................................................. 23
            1.4.2.2.3. Parameter connection.start-ok.response (longstr) ........................................................................ 23
            1.4.2.2.4. Parameter connection.start-ok.locale (shortstr) ................................................................................. 23
         1.4.2.3. Method connection.secure (ID 20) .................................................................................................................. 23
            1.4.2.3.1. Parameter connection.secure.challenge (longstr) .................................................................................. 24
         1.4.2.4. Method connection.secure-ok (ID 21) ............................................................................................................... 24
            1.4.2.4.1. Parameter connection.secure-ok.response (longstr) ........................................................................... 24
         1.4.2.5. Method connection.tune (ID 30) ...................................................................................................................... 25
            1.4.2.5.1. Parameter connection.tune.channel-max (short) ....................................................................................... 25
            1.4.2.5.2. Parameter connection.tune.frame-max (long) ....................................................................................... 25
            1.4.2.5.3. Parameter connection.tune.heartbeat (short) ......................................................................................... 25
         1.4.2.6. Method connection.tune-ok (ID 31) .................................................................................................................. 26
            1.4.2.6.1. Parameter connection.tune-ok.channel-max (short) .............................................................................. 26
            1.4.2.6.2. Parameter connection.tune-ok.frame-max (long) .................................................................................. 26
            1.4.2.6.3. Parameter connection.tune-ok.heartbeat (short) .................................................................................... 27
         1.4.2.7. Method connection.open (ID 40) .................................................................................................................... 27
            1.4.2.7.1. Parameter connection.open.virtual-host (path) ..................................................................................... 27
            1.4.2.7.2. Parameter connection.open.capabilities (shortstr) ............................................................................... 27
            1.4.2.7.3. Parameter connection.open.insist (bit) ..................................................................................................... 28
         1.4.2.8. Method connection.open-ok (ID 41) ................................................................................................................ 28
            1.4.2.8.1. Parameter connection.open-ok.known-hosts (known-hosts) ................................................................. 28
         1.4.2.9. Method connection.redirect (ID 42) ................................................................................................................ 28
            1.4.2.9.1. Parameter connection.redirect.host (shortstr) ....................................................................................... 29
            1.4.2.9.2. Parameter connection.redirect.known-hosts (known-hosts) ................................................................. 29
1.5.1 Property and Method Summary ................................................................. 31
1.5.2 Methods ............................................................................................... 32
  1.5.2.1 Method channel.open (ID 10) .......................................................... 32
    1.5.2.1.1 Parameter channel.open.out-of-band (shortstr) ....................... 33
  1.5.2.2 Method channel.open-ok (ID 11) ...................................................... 33
    1.5.2.2.1 Parameter channel.open-ok.channel-id (channel-id) .......... 33
  1.5.2.3 Method channel.flow (ID 20) ......................................................... 34
    1.5.2.3.1 Parameter channel.flow.active (bit) ....................................... 34
  1.5.2.4 Method channel.flow-ok (ID 21) ..................................................... 34
    1.5.2.4.1 Parameter channel.flow-ok.active (bit) ................................. 35
  1.5.2.5 Method channel.close (ID 40) ....................................................... 35
    1.5.2.5.1 Parameter channel.close.reply-code (reply-code) ................. 36
    1.5.2.5.2 Parameter channel.close.reply-text (reply-text) ................. 36
    1.5.2.5.3 Parameter channel.close.class-id (class-id) ......................... 36
    1.5.2.5.4 Parameter channel.close.method-id (method-id) ................. 36
  1.5.2.6 Method channel.close-ok (ID 41) .................................................. 36
  1.5.2.7 Method channel.resume (ID 50) ..................................................... 37
    1.5.2.7.1 Parameter channel.resume.channel-id (channel-id) ............ 37
  1.5.2.8 Method channel.ping (ID 60) ....................................................... 37
  1.5.2.9 Method channel.pong (ID 70) ....................................................... 37
  1.5.2.10 Method channel.ok (ID 80) ......................................................... 38
1.6 Class access ............................................................................................ 38
  1.6.1 Property and Method Summary .......................................................... 38
  1.6.2 Methods ............................................................................................ 39
  1.6.2.1 Method access.request (ID 10) ..................................................... 39
    1.6.2.1.1 Parameter access.request.realm (shortstr) ......................... 39
    1.6.2.1.2 Parameter access.requestexclusive (bit) ............................ 40
    1.6.2.1.3 Parameter access.request.passive (bit) ............................... 40
    1.6.2.1.4 Parameter access.request.active (bit) ................................. 40
    1.6.2.1.5 Parameter access.request.write (bit) .................................. 40
    1.6.2.1.6 Parameter access.request.read (bit) .................................. 41
  1.6.2.2 Method access.request-ok (ID 11) ............................................... 41
    1.6.2.2.1 Parameter access.request-ok.ticket (access-ticket) .......... 41
1.7 Class exchange ....................................................................................... 41
1.7.1 Property and Method Summary ............................................................................................................................................42
1.7.2 Methods ..............................................................................................................................................................................43
1.7.2.1 Method exchange.declare (ID 10) ....................................................................................................................................43
  1.7.2.1.1 Parameter exchange.declare.ticket (access-ticket) .............................................................................................................44
  1.7.2.1.2 Parameter exchange.declare.exchange (exchange-name) ..................................................................................................44
  1.7.2.1.3 Parameter exchange.declare.type (shortstr) .......................................................................................................................44
  1.7.2.1.4 Parameter exchange.declare.passive (bit)..........................................................................................................................44
  1.7.2.1.5 Parameter exchange.declare.durable (bit) ..........................................................................................................................45
  1.7.2.1.6 Parameter exchange.declare.auto-delete (bit) ......................................................................................................................45
  1.7.2.1.7 Parameter exchange.declare.internal (bit) ..........................................................................................................................45
  1.7.2.1.8 Parameter exchange.declare.nowait (bit) ..........................................................................................................................45
  1.7.2.1.9 Parameter exchange.declare.arguments (table) ..................................................................................................................46
1.7.2.2 Method exchange.declare-ok (ID 11) ....................................................................................................................................46
1.7.3 Method exchange.delete (ID 20) ........................................................................................................................................46
  1.7.2.3.1 Parameter exchange.delete.ticket (access-ticket) ................................................................................................................47
  1.7.2.3.2 Parameter exchange.delete.exchange (exchange-name) ......................................................................................................47
  1.7.2.3.3 Parameter exchange.delete.if-unused (bit) ..........................................................................................................................47
  1.7.2.3.4 Parameter exchange.delete.nowait (bit) ..........................................................................................................................47
1.7.2.4 Method exchange.delete-ok (ID 21) ....................................................................................................................................47
1.8 Class queue ..............................................................................................................................................................................47
  1.8.1 Property and Method Summary ........................................................................................................................................48
  1.8.2 Methods ..............................................................................................................................................................................49
  1.8.2.1 Method queue.declare (ID 10) .............................................................................................................................................49
    1.8.2.1.1 Parameter queue.declare.ticket (access-ticket) .................................................................................................................50
    1.8.2.1.2 Parameter queue.declare.queue (queue-name) .....................................................................................................................50
    1.8.2.1.3 Parameter queue.declare.passive (bit) ...............................................................................................................................51
    1.8.2.1.4 Parameter queue.declare.durable (bit) ...............................................................................................................................51
    1.8.2.1.5 Parameter queue.declare.exclusive (bit) ..............................................................................................................................51
    1.8.2.1.6 Parameter queue.declare.auto-delete (bit) ...........................................................................................................................51
    1.8.2.1.7 Parameter queue.declare.internal (bit) ...............................................................................................................................51
    1.8.2.1.8 Parameter queue.declare.nowait (bit) ...............................................................................................................................52
  1.8.2.2 Method queue.declare-ok (ID 11) .........................................................................................................................................52
    1.8.2.2.1 Parameter queue.declare-ok.queue (queue-name) ..............................................................................................................52
    1.8.2.2.2 Parameter queue.declare-ok.message-count (long) ............................................................................................................53
    1.8.2.2.3 Parameter queue.declare-ok.consumer-count (long) .........................................................................................................53
  1.8.2.3 Method queue.bind (ID 20) ................................................................................................................................................53
    1.8.2.3.1 Parameter queue.bind.ticket (access-ticket) ......................................................................................................................54
    1.8.2.3.2 Parameter queue.bind.queue (queue-name) .......................................................................................................................54
    1.8.2.3.3 Parameter queue.bind.exchange (exchange-name) ............................................................................................................55
    1.8.2.3.4 Parameter queue.bind.routing-key (shortstr) .....................................................................................................................55
    1.8.2.3.5 Parameter queue.bind.nowait (bit) .................................................................................................................................55
1.8.2.3.6. Parameter queue.bind.arguments (table) ................................................................. 55
1.8.2.4. Method queue.bind-ok (ID 21) ................................................................. 55
1.8.2.5. Method queue.unbind (ID 50) ................................................................. 56
1.8.2.5.1. Parameter queue.unbind.ticket (access-ticket) ............................................ 56
1.8.2.5.2. Parameter queue.unbind.queue (queue-name) .......................................... 56
1.8.2.5.3. Parameter queue.unbind.exchange (exchange-name) ................................. 57
1.8.2.5.4. Parameter queue.unbind.routing-key (shortstr) ........................................... 57
1.8.2.5.5. Parameter queue.unbind.arguments (table) ................................................ 57
1.8.2.6. Method queue.unbind-ok (ID 51) ................................................................. 57
1.8.2.7. Method queue.purge (ID 30) ........................................................................ 57
1.8.2.7.1. Parameter queue.purge.ticket (access-ticket) ............................................. 58
1.8.2.7.2. Parameter queue.purge.queue (queue-name) .............................................. 58
1.8.2.7.3. Parameter queue.purge.nowait (bit) ............................................................ 58
1.8.2.8. Method queue.purge-ok (ID 31) .................................................................. 59
1.8.2.8.1. Parameter queue.purge-ok.message-count (long) ....................................... 59
1.8.2.9. Method queue.delete (ID 40) ..................................................................... 59
1.8.2.9.1. Parameter queue.delete.ticket (access-ticket) ........................................... 60
1.8.2.9.2. Parameter queue.delete.queue (queue-name) ............................................ 60
1.8.2.9.3. Parameter queue.delete.if-unused (bit) ....................................................... 60
1.8.2.9.4. Parameter queue.delete.if-empty (bit) ....................................................... 60
1.8.2.9.5. Parameter queue.delete.nowait (bit) .......................................................... 61
1.8.2.10. Method queue.delete-ok (ID 41) ............................................................... 61
1.8.2.10.1. Parameter queue.delete-ok.message-count (long) .................................... 61

1.9 Class basic ........................................................................................................... 61

1.9.1. Property and Method Summary ......................................................................... 63

1.9.2. Properties ....................................................................................................... 65
1.9.2.1. Property basic.content-type (shortstr) .......................................................... 65
1.9.2.2. Property basic.content-encoding (shortstr) ................................................... 65
1.9.2.3. Property basic.headers (table) ....................................................................... 65
1.9.2.4. Property basic.delivery-mode (octet) ............................................................ 65
1.9.2.5. Property basic.priority (octet) ...................................................................... 66
1.9.2.6. Property basic.correlation-id (shortstr) ........................................................... 66
1.9.2.7. Property basic.reply-to (shortstr) .................................................................. 66
1.9.2.8. Property basic.expiration (shortstr) ............................................................... 66
1.9.2.9. Property basic.message-id (shortstr) .............................................................. 66
1.9.2.10. Property basic.timestamp (timestamp) ......................................................... 66
1.9.2.11. Property basic.type (shortstr) ....................................................................... 66
1.9.2.12. Property basic.user-id (shortstr) ................................................................. 66
1.9.2.13. Property basic.app-id (shortstr) ................................................................. 67
1.9.2.14. Property basic.cluster-id (shortstr) ............................................................ 67

1.9.3. Methods ....................................................................................................... 67
1.9.3.1.Method basic.qos (ID 10)
   1.9.3.1.1.Parameter basic.qos.prefetch-size (long) ................................................................. 67
   1.9.3.1.2.Parameter basic.qos.prefetch-count (short) ............................................................... 68
   1.9.3.1.3.Parameter basic.qos.global (bit) ................................................................................. 68
1.9.3.2.Method basic.qos-ok (ID 11) ............................................................................................ 68
1.9.3.3.Method basic.consume (ID 20)
   1.9.3.3.1.Parameter basic.consume.ticket (access-ticket) ........................................................... 69
   1.9.3.3.2.Parameter basic.consume.queue (queue-name) ............................................................. 69
   1.9.3.3.3.Parameter basic.consume.consumer-tag (consumer-tag) ............................................. 70
   1.9.3.3.4.Parameter basic.consume.no-local (no-local) ............................................................... 70
   1.9.3.3.5.Parameter basic.consume.no-ack (no-ack) .................................................................. 70
   1.9.3.3.6.Parameter basic.consume.exclusive (bit) .................................................................... 70
   1.9.3.3.7.Parameter basic.consume.nowait (bit) ....................................................................... 70
   1.9.3.3.8.Parameter basic.consume.filter (table) ..................................................................... 70
1.9.3.4.Method basic.consume-ok (ID 21)
   1.9.3.4.1.Parameter basic.consume-ok.consumer-tag (consumer-tag) ........................................ 71
1.9.3.5.Method basic.cancel (ID 30)
   1.9.3.5.1.Parameter basic.cancel.consumer-tag (consumer-tag) .................................................. 71
   1.9.3.5.2.Parameter basic.cancel.nowait (bit) ......................................................................... 72
1.9.3.6.Method basic.cancel-ok (ID 31)
   1.9.3.6.1.Parameter basic.cancel-ok.consumer-tag (consumer-tag) ........................................... 72
1.9.3.7.Method basic.publish (ID 40)
   1.9.3.7.1.Parameter basic.publish.ticket (access-ticket) ............................................................ 73
   1.9.3.7.2.Parameter basic.publish.exchange (exchange-name) .................................................. 73
   1.9.3.7.3.Parameter basic.publish.routing-key (shortstr) ............................................................ 73
   1.9.3.7.4.Parameter basic.publish.mandatory (bit) .................................................................... 74
   1.9.3.7.5.Parameter basic.publish.immediate (bit) ..................................................................... 74
1.9.3.8.Method basic.return (ID 50)
   1.9.3.8.1.Parameter basic.return.reply-code (reply-code) ......................................................... 75
   1.9.3.8.2.Parameter basic.return.reply-text (reply-text) ........................................................... 75
   1.9.3.8.3.Parameter basic.return.exchange (exchange-name) ................................................... 75
   1.9.3.8.4.Parameter basic.return.routing-key (shortstr) ............................................................ 75
1.9.3.9.Method basic.deliver (ID 60)
   1.9.3.9.1.Parameter basic.deliver.consumer-tag (consumer-tag) ............................................... 76
   1.9.3.9.2.Parameter basic.deliver.delivery-tag (delivery-tag) ..................................................... 76
   1.9.3.9.3.Parameter basic.deliver.redelivered (redelivered) ...................................................... 76
   1.9.3.9.4.Parameter basic.deliver.exchange (exchange-name) .................................................... 76
   1.9.3.9.5.Parameter basic.deliver.routing-key (shortstr) ............................................................ 77
1.9.3.10.Method basic.get (ID 70)
   1.9.3.10.1.Parameter basic.get.ticket (access-ticket) ............................................................... 77
   1.9.3.10.2.Parameter basic.get.queue (queue-name) ................................................................. 77
1.10.1. Parameter basic.get-empty.cluster-id (shortstr) ................................................................. 79
1.10.13. Method basic.ack (ID 80) .................................................................................................. 80
1.10.13.1. Parameter basic.ack.delivery-tag (delivery-tag) ............................................................ 80
1.10.13.2. Parameter basic.ack.multiple (bit) .................................................................................. 80
1.10.14. Method basic.reject (ID 90) ............................................................................................... 81
1.10.14.1. Parameter basic.reject.delivery-tag (delivery-tag) ......................................................... 81
1.10.14.2. Parameter basic.reject.requeue (bit) .............................................................................. 81
1.10.15. Method basic.recover (ID 100) ........................................................................................... 82
1.10.15.1. Parameter basic.recover.requeue (bit) ............................................................................ 82

1.10.2. Properties ............................................................................................................................ 83
1.10.2.1. Property file.content-type (shortstr) ................................................................................ 85
1.10.2.2. Property file.content-encoding (shortstr) ....................................................................... 85
1.10.2.3. Property file.headers (table) ............................................................................................. 85
1.10.2.4. Property file.priority (octet) ............................................................................................ 85
1.10.2.5. Property file.reply-to (shortstr) ......................................................................................... 86
1.10.2.6. Property file.message-id (shortstr) ................................................................................... 86
1.10.2.7. Property file.filename (shortstr) ......................................................................................... 86
1.10.2.8. Property file.timestamp (timestamp) ................................................................................ 86
1.10.2.9. Property file.cluster-id (shortstr) .................................................................................... 86

1.10.3. Methods .............................................................................................................................. 86
1.10.3.1. Method file.qos (ID 10) ................................................................................................... 86
1.10.3.1.1. Parameter file.qos.prefetch-size (long) ........................................................................ 87
1.10.3.1.2. Parameter file.qos.prefetch-count (short) .................................................................... 87
1.10.3.1.3. Parameter file.qos.global (bit) ...................................................................................... 87
1.10.3.2. Method file.qos-ok (ID 11) ............................................................................................... 87
1.10.3.3. Method file.consume (ID 20) ............................................................................................ 88
1.10.3.3.1. Parameter file.consume.ticket (access-ticket) .............................................................. 88
1.10.3.3.2. Parameter file.consume.queue (queue-name) ............................................................... 89
1.10.3.3.3. Parameter file.consume.consumer-tag (consumer-tag) ............................................... 89
1.10.3.3.4. Parameter file.consume.no-local (no-local) ................................................................. 89
1.10.3.3.5. Parameter file.consume.no-ack (no-ack) ..................................................................... 89

1.9.3.11. Method basic.get-ok (ID 71) ............................................................................................ 78
1.9.3.11.1. Parameter basic.get-ok.delivery-tag (delivery-tag) ....................................................... 78
1.9.3.11.2. Parameter basic.get-ok.redelivered (redelivered) ......................................................... 78
1.9.3.11.3. Parameter basic.get-ok.exchange (exchange-name) .................................................... 78
1.9.3.11.4. Parameter basic.get-ok.routing-key (shortstr) .............................................................. 79
1.9.3.11.5. Parameter basic.get-ok.message-count (long) .............................................................. 79
1.9.3.12. Method basic.get-empty (ID 72) ..................................................................................... 79
1.9.3.12.1. Parameter basic.get-empty.cluster-id (shortstr) .......................................................... 79
1.9.3.13. Method basic.ack (ID 80) ................................................................................................ 80
1.9.3.13.1. Parameter basic.ack.delivery-tag (delivery-tag) ............................................................ 80
1.9.3.13.2. Parameter basic.ack.multiple (bit) ................................................................................ 80
1.9.3.14. Method basic.reject (ID 90) ............................................................................................ 81
1.9.3.14.1. Parameter basic.reject.delivery-tag (delivery-tag) ......................................................... 81
1.9.3.14.2. Parameter basic.reject.requeue (bit) ............................................................................ 81
1.9.3.15. Method basic.recover (ID 100) ........................................................................................... 82
1.9.3.15.1. Parameter basic.recover.requeue (bit) ........................................................................... 82

1.9.3.10.3. Parameter basic.get.no-ack (no-ack) ........................................................................... 78

1.9.3.10.1. Property file.qos.global (bit) ...................................................................................... 87
1.9.3.10.2. Property file.qos.prefetch-count (short) .................................................................... 87
1.9.3.10.3. Property file.qos.global (bit) ...................................................................................... 87
1.9.3.11. Method basic.get-ok (ID 71) ............................................................................................ 78
1.9.3.11.1. Parameter basic.get-ok.delivery-tag (delivery-tag) ....................................................... 78
1.9.3.11.2. Parameter basic.get-ok.redelivered (redelivered) ......................................................... 78
1.9.3.11.3. Parameter basic.get-ok.exchange (exchange-name) .................................................... 78
1.9.3.11.4. Parameter basic.get-ok.routing-key (shortstr) .............................................................. 79
1.9.3.11.5. Parameter basic.get-ok.message-count (long) .............................................................. 79
1.9.3.12. Method basic.get-empty (ID 72) ..................................................................................... 79
1.9.3.12.1. Parameter basic.get-empty.cluster-id (shortstr) .......................................................... 79
1.9.3.13. Method basic.ack (ID 80) ................................................................................................ 80
1.9.3.13.1. Parameter basic.ack.delivery-tag (delivery-tag) ............................................................ 80
1.9.3.13.2. Parameter basic.ack.multiple (bit) ................................................................................ 80
1.9.3.14. Method basic.reject (ID 90) ............................................................................................ 81
1.9.3.14.1. Parameter basic.reject.delivery-tag (delivery-tag) ......................................................... 81
1.9.3.14.2. Parameter basic.reject.requeue (bit) ............................................................................ 81
1.9.3.15. Method basic.recover (ID 100) ........................................................................................... 82
1.9.3.15.1. Parameter basic.recover.requeue (bit) ........................................................................... 82

1.10. Class file .................................................................................................................................. 82
1.10.1. Property and Method Summary ........................................................................................ 83
1.10.2. Properties ............................................................................................................................ 85
1.10.2.1. Property file.content-type (shortstr) ................................................................................ 85
1.10.2.2. Property file.content-encoding (shortstr) ....................................................................... 85
1.10.2.3. Property file.headers (table) ............................................................................................. 85
1.10.2.4. Property file.priority (octet) ............................................................................................ 85
1.10.2.5. Property file.reply-to (shortstr) ......................................................................................... 86
1.10.2.6. Property file.message-id (shortstr) ................................................................................... 86
1.10.2.7. Property file.filename (shortstr) ......................................................................................... 86
1.10.2.8. Property file.timestamp (timestamp) ................................................................................ 86
1.10.2.9. Property file.cluster-id (shortstr) .................................................................................... 86
1.10.3. Methods .............................................................................................................................. 86
1.10.3.1. Method file.qos (ID 10) ................................................................................................... 86
1.10.3.1.1. Parameter file.qos.prefetch-size (long) ........................................................................ 87
1.10.3.1.2. Parameter file.qos.prefetch-count (short) .................................................................... 87
1.10.3.1.3. Parameter file.qos.global (bit) ...................................................................................... 87
1.10.3.2. Method file.qos-ok (ID 11) ............................................................................................... 87
1.10.3.3. Method file.consume (ID 20) ............................................................................................ 88
1.10.3.3.1. Parameter file.consume.ticket (access-ticket) .............................................................. 88
1.10.3.3.2. Parameter file.consume.queue (queue-name) ............................................................... 89
1.10.3.3.3. Parameter file.consume.consumer-tag (consumer-tag) ............................................... 89
1.10.3.3.4. Parameter file.consume.no-local (no-local) ................................................................. 89
1.10.3.3.5. Parameter file.consume.no-ack (no-ack) ..................................................................... 89
1.10.3.3.6.Parameter file.consume.exclusive (bit) .................................................. 89
1.10.3.3.7.Parameter file.consume.nowait (bit) ...................................................... 89
1.10.3.3.8.Parameter file.consume.filter (table) ..................................................... 90
1.10.3.4.Method file.consume-ok (ID 21) ............................................................... 90
    1.10.3.4.1.Parameter file.consume-ok.consumer-tag (consumer-tag) ............. 90
1.10.3.5.Method file.cancel (ID 30) ................................................................. 90
    1.10.3.5.1.Parameter file.cancel.consumer-tag (consumer-tag) .................... 91
    1.10.3.5.2.Parameter file.cancel.nowait (bit) ................................................ 91
1.10.3.6.Method file.cancel-ok (ID 31) ............................................................. 91
    1.10.3.6.1.Parameter file.cancel-ok.consumer-tag (consumer-tag) ............... 92
1.10.3.7.Method file.open (ID 40) ................................................................. 92
    1.10.3.7.1.Parameter file.open.identifier (shortstr) ...................................... 92
    1.10.3.7.2.Parameter file.open.content-size (longlong) ................................ 92
1.10.3.8.Method file.open-ok (ID 41) ............................................................... 93
    1.10.3.8.1.Parameter file.open-ok.staged-size (longlong) ............................ 93
1.10.3.9.Method file.stage (ID 50) ................................................................. 93
1.10.3.10.Method file.publish (ID 60) ............................................................. 93
    1.10.3.10.1.Parameter file.publish.ticket (access-ticket) .......................... 94
    1.10.3.10.2.Parameter file.publish.exchange (exchange-name) .................... 94
    1.10.3.10.3.Parameter file.publish.routing-key (shortstr) .......................... 94
    1.10.3.10.4.Parameter file.publish.mandatory (bit) ...................................... 95
    1.10.3.10.5.Parameter file.publish.immediate (bit) ..................................... 95
    1.10.3.10.6.Parameter file.publish.identifier (shortstr) ........................... 95
1.10.3.11.Method file.return (ID 70) ............................................................... 95
    1.10.3.11.1.Parameter file.return.reply-code (reply-code) ........................ 96
    1.10.3.11.2.Parameter file.return.reply-text (reply-text) ........................ 96
    1.10.3.11.3.Parameter file.return.exchange (exchange-name) ...................... 96
    1.10.3.11.4.Parameter file.return.routing-key (shortstr) .......................... 96
1.10.3.12.Method file.deliver (ID 80) ........................................................... 96
    1.10.3.12.1.Parameter file.deliver.consumer-tag (consumer-tag) ............... 97
    1.10.3.12.2.Parameter file.deliver.delivery-tag (delivery-tag) .................. 97
    1.10.3.12.3.Parameter file.deliver.redelivered (redelivered) ..................... 97
    1.10.3.12.4.Parameter file.deliver.exchange (exchange-name) .................... 97
    1.10.3.12.5.Parameter file.deliver.routing-key (shortstr) ........................ 98
    1.10.3.12.6.Parameter file.deliver.identifier (shortstr) .......................... 98
1.10.3.13.Method file.ack (ID 90) ................................................................. 98
    1.10.3.13.1.Parameter file.ack.delivery-tag (delivery-tag) ....................... 98
    1.10.3.13.2.Parameter file.ack.multiple (bit) ........................................... 99
1.10.3.14.Method file.reject (ID 100) ............................................................ 99
    1.10.3.14.1.Parameter file.reject.delivery-tag (delivery-tag) ................... 99
    1.10.3.14.2.Parameter file.reject.requeue (bit) .......................................100

AMQP Specification (XML-derived) 

Page 7 of 139
1.11.Class stream

1.11.1.Property and Method Summary

1.11.2.Properties
  1.11.2.1.Property stream.content-type (shortstr)
  1.11.2.2.Property stream.content-encoding (shortstr)
  1.11.2.3.Property stream.headers (table)
  1.11.2.4.Property stream.priority (octet)
  1.11.2.5.Property stream.timestamp (timestamp)

1.11.3.Methods
  1.11.3.1.Method stream.qos (ID 10)
    1.11.3.1.1.Parameter stream.qos.prefetch-size (long)
    1.11.3.1.2.Parameter stream.qos.prefetch-count (short)
    1.11.3.1.3.Parameter stream.qos.consume-rate (long)
    1.11.3.1.4.Parameter stream.qos.global (bit)
  1.11.3.2.Method stream.qos-ok (ID 11)
  1.11.3.3.Method stream.consume (ID 20)
    1.11.3.3.1.Parameter stream.consume.ticket (access-ticket)
    1.11.3.3.2.Parameter stream.consume.queue (queue-name)
    1.11.3.3.3.Parameter stream.consume.consumer-tag (consumer-tag)
    1.11.3.3.4.Parameter stream.consume.no-local (no-local)
    1.11.3.3.5.Parameter stream.consume.exclusive (bit)
    1.11.3.3.6.Parameter stream.consume.nowait (bit)
    1.11.3.3.7.Parameter stream.consume.filter (table)
  1.11.3.4.Method stream.consume-ok (ID 21)
    1.11.3.4.1.Parameter stream.consume-ok.consumer-tag (consumer-tag)
  1.11.3.5.Method stream.cancel (ID 30)
    1.11.3.5.1.Parameter stream.cancel.consumer-tag (consumer-tag)
    1.11.3.5.2.Parameter stream.cancel.nowait (bit)
  1.11.3.6.Method stream.cancel-ok (ID 31)
    1.11.3.6.1.Parameter stream.cancel-ok.consumer-tag (consumer-tag)
  1.11.3.7.Method stream.publish (ID 40)
    1.11.3.7.1.Parameter stream.publish.ticket (access-ticket)
    1.11.3.7.2.Parameter stream.publish.exchange (exchange-name)
    1.11.3.7.3.Parameter stream.publish.routing-key (shortstr)
    1.11.3.7.4.Parameter stream.publish.mandatory (bit)
    1.11.3.7.5.Parameter stream.publish.immediate (bit)
  1.11.3.8.Method stream.return (ID 50)
    1.11.3.8.1.Parameter stream.return.reply-code (reply-code)
    1.11.3.8.2.Parameter stream.return.reply-text (reply-text)
    1.11.3.8.3.Parameter stream.return.exchange (exchange-name)
    1.11.3.8.4.Parameter stream.return.routing-key (shortstr)
1.11.3.9.Method stream.deliver (ID 60)..........................................................................................111
  1.11.3.9.1.Parameter stream.deliver.consumer-tag (consumer-tag)..............................................111
  1.11.3.9.2.Parameter stream.deliver.delivery-tag (delivery-tag)....................................................111
  1.11.3.9.3.Parameter stream.deliver.exchange (exchange-name)..................................................112
  1.11.3.9.4.Parameter stream.deliver.queue (queue-name).............................................................112

1.12.Class tx........................................................................................................................................112
  1.12.1.Property and Method Summary ..........................................................................................112
  1.12.2.Methods................................................................................................................................113
    1.12.2.1.Method tx.select (ID 10)................................................................................................113
    1.12.2.2.Method tx.select-ok (ID 11)............................................................................................113
    1.12.2.3.Method tx.commit (ID 20)...............................................................................................114
    1.12.2.4.Method tx.commit-ok (ID 21)...........................................................................................114
    1.12.2.5.Method tx.rollback (ID 30)............................................................................................114
    1.12.2.6.Method tx.rollback-ok (ID 31).......................................................................................114

1.13.Class dtx......................................................................................................................................115
  1.13.1.Property and Method Summary ..........................................................................................115
  1.13.2.Methods................................................................................................................................116
    1.13.2.1.Method dtx.select (ID 10)..............................................................................................116
    1.13.2.2.Method dtx.select-ok (ID 11)..........................................................................................116
    1.13.2.3.Method dtx.start (ID 20)................................................................................................116
      1.13.2.3.1.Parameter dtx.start.dtx-identifier (shortstr).................................................................117
    1.13.2.4.Method dtx.start-ok (ID 21)............................................................................................117

1.14.Class tunnel..................................................................................................................................117
  1.14.1.Property and Method Summary ..........................................................................................117
  1.14.2.Properties.............................................................................................................................118
    1.14.2.1.Property tunnel.headers (table).....................................................................................118
    1.14.2.2.Property tunnel.proxy-name (shortstr).........................................................................118
    1.14.2.3.Property tunnel.data-name (shortstr)............................................................................118
    1.14.2.4.Property tunnel.durable (octet).....................................................................................118
    1.14.2.5.Property tunnel.broadcast (octet)................................................................................118
  1.14.3.Methods................................................................................................................................119
    1.14.3.1.Method tunnel.request (ID 10).......................................................................................119
      1.14.3.1.1.Parameter tunnel.request.meta-data (table)...............................................................119

1.15.Class message............................................................................................................................119
  1.15.1.Property and Method Summary ..........................................................................................119
  1.15.2.Methods................................................................................................................................121
    1.15.2.1.Method message.transfer (ID 10)..................................................................................123
      1.15.2.1.1.Parameter message.transfer.ticket (access-ticket).................................................125
      1.15.2.1.2.Parameter message.transfer.destination (destination)..........................................125
      1.15.2.1.3.Parameter message.transfer.redelivered (redelivered).........................................125
      1.15.2.1.4.Parameter message.transfer.immediate (bit)..........................................................125
1.15.2.1.5. Parameter message.transfer.ttl (duration) ................................................................. 125
1.15.2.1.6. Parameter message.transfer.priority (octet) .......................................................... 126
1.15.2.1.7. Parameter message.transfer.timestamp (timestamp) ........................................... 126
1.15.2.1.8. Parameter message.transfer.delivery-mode (octet) .............................................. 126
1.15.2.1.9. Parameter message.transfer.expiration (timestamp) ........................................... 126
1.15.2.1.10. Parameter message.transfer.exchange (exchange-name) .................................... 126
1.15.2.1.11. Parameter message.transfer.routing-key (shortstr) ........................................... 126
1.15.2.1.12. Parameter message.transfer.message-id (shortstr) ............................................ 126
1.15.2.1.13. Parameter message.transfer.correlation-id (shortstr) ........................................... 127
1.15.2.1.14. Parameter message.transfer.reply-to (shortstr) .................................................. 127
1.15.2.1.15. Parameter message.transfer.content-type (shortstr) ........................................... 127
1.15.2.1.16. Parameter message.transfer.content-encoding (shortstr) .................................... 127
1.15.2.1.17. Parameter message.transfer.user-id (shortstr) .................................................... 127
1.15.2.1.18. Parameter message.transfer.app-id (shortstr) ...................................................... 128
1.15.2.1.19. Parameter message.transfer.transaction-id (shortstr) ......................................... 128
1.15.2.1.20. Parameter message.transfer.security-token (security-token) ............................. 128
1.15.2.1.21. Parameter message.transfer.application-headers (table) .................................... 128
1.15.2.1.22. Parameter message.transfer.body (content) .......................................................... 128
1.15.2.2. Method message.consume (ID 20) ................................................................................. 128
  1.15.2.2.1. Parameter message.consume.ticket (access-ticket) ............................................. 129
  1.15.2.2.2. Parameter message.consume.queue (queue-name) .............................................. 129
  1.15.2.2.3. Parameter message.consume.destination (destination) ......................................... 129
  1.15.2.2.4. Parameter message.consume.no-local (no-local) ................................................. 130
  1.15.2.2.5. Parameter message.consume.no-ack (no-ack) ..................................................... 130
  1.15.2.2.6. Parameter message.consume.exclusive (bit) ....................................................... 130
  1.15.2.2.7. Parameter message.consume.filter (table) ........................................................... 130
1.15.2.3. Method message.cancel (ID 30) .................................................................................. 130
  1.15.2.3.1. Parameter message.cancel.destination (destination) ........................................... 131
1.15.2.4. Method message.get (ID 40) ...................................................................................... 131
  1.15.2.4.1. Parameter message.get.ticket (access-ticket) ...................................................... 131
  1.15.2.4.2. Parameter message.get.queue (queue-name) ....................................................... 132
  1.15.2.4.3. Parameter message.get.destination (destination) ................................................ 132
  1.15.2.4.4. Parameter message.get.no-ack (no-ack) ............................................................. 132
1.15.2.5. Method message.recover (ID 50) .................................................................................. 132
  1.15.2.5.1. Parameter message.recover.requeue (bit) ........................................................... 133
1.15.2.6. Method message.open (ID 60) .................................................................................... 133
  1.15.2.6.1. Parameter message.open.reference (reference) ................................................... 133
1.15.2.7. Method message.close (ID 70) ................................................................................. 133
  1.15.2.7.1. Parameter message.close.reference (reference) ................................................... 134
1.15.2.8. Method message.append (ID 80) ............................................................................ 134
  1.15.2.8.1. Parameter message.append.reference (reference) ............................................... 134
1.15.2.8.2. Parameter message.append.bytes (longstr) .......................................................... 135

1.15.2.9. Method message.checkpoint (ID 90) ........................................................................ 135
   1.15.2.9.1. Parameter message.checkpoint.reference (reference) ........................................ 135
   1.15.2.9.2. Parameter message.checkpoint.identifier (shortstr) ..................................... 135

1.15.2.10. Method message.resume (ID 100) ........................................................................ 136
   1.15.2.10.1. Parameter message.resume.reference (reference) ..................................... 136
   1.15.2.10.2. Parameter message.resume.identifier (shortstr) ..................................... 136

1.15.2.11. Method message.qos (ID 110) ............................................................................. 136
   1.15.2.11.1. Parameter message.qos.prefetch-size (long) ............................................. 137
   1.15.2.11.2. Parameter message.qos.prefetch-count (short) ....................................... 137
   1.15.2.11.3. Parameter message.qos.global (bit) ...................................................... 138

1.15.2.12. Method message.ok (ID 500) ............................................................................. 138

1.15.2.13. Method message.empty (ID 510) ........................................................................ 138

1.15.2.14. Method message.reject (ID 520) ........................................................................ 138
   1.15.2.14.1. Parameter message.reject.code (reject-code) ........................................ 139
   1.15.2.14.2. Parameter message.reject.text (reject-text) .......................................... 139

1.15.2.15. Method message.offset (ID 530) ......................................................................... 139
   1.15.2.15.1. Parameter message.offset.value (offset) ............................................. 139
1. AMQP version 0-9 specification

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

1.1. AMQP-defined Domains

The following domains are defined in this specification:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>[Label] Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>access-ticket</td>
<td>short</td>
<td>[access ticket granted by server] An access ticket granted by the server for a certain set of access rights within a specific realm. Access tickets are valid within the channel where they were created, and expire when the channel closes.</td>
</tr>
<tr>
<td>bit</td>
<td>bit</td>
<td>[single bit]</td>
</tr>
<tr>
<td>channel-id</td>
<td>longstr</td>
<td>[unique identifier for a channel]</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 connection.</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>destination</td>
<td>shortstr</td>
<td>[destination for a message] Specifies the destination to which the message is to be transferred. The destination can be empty, meaning the default exchange or consumer.</td>
</tr>
<tr>
<td>duration</td>
<td>longlong</td>
<td>[duration in milliseconds]</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. Exchange names may consist of any mixture of digits, letters, and underscores. Exchange names are scoped by the virtual host.</td>
</tr>
<tr>
<td>known-hosts</td>
<td>shortstr</td>
<td>[list of known hosts] Specifies the list of equivalent or alternative hosts that the server knows about, which will normally include the current server itself. Clients can cache this information and use it when reconnecting to a server after a failure. This field may be empty.</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>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 automatically and silently acknowledges it on behalf of the client. This functionality increases performance but at the cost of reliability. Messages can get lost if a client dies before it can deliver them to the application.</td>
</tr>
</tbody>
</table>
### 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>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>octet</td>
<td>octet</td>
<td>[single octet]</td>
</tr>
<tr>
<td>offset</td>
<td>longlong</td>
<td>[offset into a message body]</td>
</tr>
<tr>
<td>path</td>
<td>shortstr</td>
<td>Must start with a slash &quot;&quot; and continue with path names separated by slashes. A path name consists of any combination of at least one of [A-Za-z0-9] plus zero or more of [._+!=:].</td>
</tr>
<tr>
<td>peer-properties</td>
<td>table</td>
<td>This string 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. Queue names may consist of any mixture of digits, letters, and underscores.</td>
</tr>
<tr>
<td>redelivered</td>
<td>bit</td>
<td>[message is being redelivered] This indicates that the message has been previously delivered to this or another client.</td>
</tr>
<tr>
<td>reference</td>
<td>longstr</td>
<td>[pointer to a message body]</td>
</tr>
<tr>
<td>reject-code</td>
<td>short</td>
<td>[reject code for transfer]</td>
</tr>
<tr>
<td>reject-text</td>
<td>shortstr</td>
<td>[informational text for message reject]</td>
</tr>
<tr>
<td>reply-code</td>
<td>short</td>
<td>[reply code from server] The reply code. The AMQ reply codes are defined as constants at the start of this formal specification.</td>
</tr>
<tr>
<td>reply-text</td>
<td>shortstr</td>
<td>[localised reply text] The localised reply text. This text can be logged as an aid to resolving issues.</td>
</tr>
<tr>
<td>security-token</td>
<td>longstr</td>
<td>[security token] Used for authentication, replay prevention, and encrypted bodies.</td>
</tr>
<tr>
<td>short</td>
<td>short</td>
<td>[16-bit integer]</td>
</tr>
<tr>
<td>shortstr</td>
<td>shortstr</td>
<td>[short string]</td>
</tr>
<tr>
<td>table</td>
<td>table</td>
<td>[field table]</td>
</tr>
<tr>
<td>timestamp</td>
<td>timestamp</td>
<td>[64-bit timestamp]</td>
</tr>
<tr>
<td>Name</td>
<td>Value</td>
<td>Error type</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------</td>
<td>------------</td>
</tr>
<tr>
<td>frame-method</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>frame-header</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>frame-body</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>frame-oob-method</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>frame-oob-header</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>frame-oob-body</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>frame-trace</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>frame-heartbeat</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>frame-min-size</td>
<td>4096</td>
<td></td>
</tr>
<tr>
<td>frame-end</td>
<td>206</td>
<td></td>
</tr>
<tr>
<td>reply-success</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>not-delivered</td>
<td>310</td>
<td>channel</td>
</tr>
<tr>
<td>content-too-large</td>
<td>311</td>
<td>channel</td>
</tr>
<tr>
<td>no-route</td>
<td>312</td>
<td>channel</td>
</tr>
<tr>
<td>no-consumers</td>
<td>313</td>
<td>channel</td>
</tr>
<tr>
<td>connection-forced</td>
<td>320</td>
<td>connection</td>
</tr>
<tr>
<td>invalid-path</td>
<td>402</td>
<td>connection</td>
</tr>
<tr>
<td>access-refused</td>
<td>403</td>
<td>channel</td>
</tr>
<tr>
<td>not-found</td>
<td>404</td>
<td>channel</td>
</tr>
<tr>
<td>resource-locked</td>
<td>405</td>
<td>channel</td>
</tr>
<tr>
<td>precondition-failed</td>
<td>406</td>
<td>channel</td>
</tr>
<tr>
<td>frame-error</td>
<td>501</td>
<td>connection</td>
</tr>
<tr>
<td>syntax-error</td>
<td>502</td>
<td>connection</td>
</tr>
<tr>
<td>Name</td>
<td>Value</td>
<td>Error type</td>
</tr>
<tr>
<td>------------------</td>
<td>-------</td>
<td>------------</td>
</tr>
<tr>
<td>command-invalid</td>
<td>503</td>
<td>connection</td>
</tr>
<tr>
<td>channel-error</td>
<td>504</td>
<td>connection</td>
</tr>
<tr>
<td>resource-error</td>
<td>506</td>
<td>connection</td>
</tr>
<tr>
<td>not-allowed</td>
<td>530</td>
<td>connection</td>
</tr>
<tr>
<td>not-implemented</td>
<td>540</td>
<td>connection</td>
</tr>
<tr>
<td>internal-error</td>
<td>541</td>
<td>connection</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>redirect</td>
<td>42</td>
<td>redirects client to other server</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></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>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>close</td>
<td>40</td>
<td>request a channel close</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>close-ok</td>
<td>41</td>
<td>confirm a channel close</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>resume</td>
<td>50</td>
<td>resume an interrupted channel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ping</td>
<td>60</td>
<td>[WORK IN PROGRESS] initiates a pong</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pong</td>
<td>70</td>
<td>[WORK IN PROGRESS] issued after receiving a ping</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ok</td>
<td>80</td>
<td>[WORK IN PROGRESS] signals normal completion</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>access</td>
<td>30</td>
<td>work with access tickets</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>request</td>
<td>10</td>
<td>request an access ticket</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>request-ok</td>
<td>11</td>
<td>grant access to server resources</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>exchange</td>
<td>40</td>
<td>work with exchanges</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>declare</td>
<td>10</td>
<td>verify exchange exists, create if needed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>declare-ok</td>
<td>11</td>
<td>confirm exchange declaration</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td>20</td>
<td>delete an exchange</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete-ok</td>
<td>21</td>
<td>confirm deletion of an exchange</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>queue</td>
<td>50</td>
<td>work with queues</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>declare</td>
<td>10</td>
<td>declare queue, create if needed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>declare-ok</td>
<td>11</td>
<td>confirms a queue definition</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bind</td>
<td>20</td>
<td>bind queue to an exchange</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bind-ok</td>
<td>21</td>
<td>confirm bind successful</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unbind</td>
<td>50</td>
<td>unbind a queue from an exchange</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unbind-ok</td>
<td>51</td>
<td>confirm unbind successful</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>purge</td>
<td>30</td>
<td>purge a queue</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>purge-ok</td>
<td>31</td>
<td>confirms a queue purge</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td>40</td>
<td>delete a queue</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete-ok</td>
<td>41</td>
<td>confirm deletion of a queue</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>basic</td>
<td>60</td>
<td>work with basic content</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>qos</td>
<td>10</td>
<td>specify quality of service</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>qos-ok</td>
<td>11</td>
<td>confirm the requested qos</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consume</td>
<td>20</td>
<td>start a queue consumer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consume-ok</td>
<td>21</td>
<td>confirm a new consumer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cancel</td>
<td>30</td>
<td>end a queue consumer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cancel-ok</td>
<td>31</td>
<td>confirm a cancelled consumer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>publish</td>
<td>40</td>
<td>publish a message</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>return</td>
<td>50</td>
<td>return a failed message</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deliver</td>
<td>60</td>
<td>notify the client of a consumer message</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>get</td>
<td>70</td>
<td>direct access to a queue</td>
<td></td>
<td></td>
<td></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>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>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</td>
<td>100</td>
<td>redeliver unacknowledged messages</td>
</tr>
<tr>
<td>file</td>
<td>70</td>
<td>work with file 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>open</td>
<td>40</td>
<td>request to start staging</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>open-ok</td>
<td>41</td>
<td>confirm staging ready</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>stage</td>
<td>50</td>
<td>stage message content</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>publish</td>
<td>60</td>
<td>publish a message</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>return</td>
<td>70</td>
<td>return a failed message</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>deliver</td>
<td>80</td>
<td>notify the client of a consumer message</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ack</td>
<td>90</td>
<td>acknowledge one or more messages</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>reject</td>
<td>100</td>
<td>reject an incoming message</td>
</tr>
<tr>
<td>stream</td>
<td>80</td>
<td>work with streaming 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>tx</td>
<td>90</td>
<td>work with standard 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>
<tr>
<td></td>
<td></td>
<td></td>
<td>commit-ok</td>
<td>21</td>
<td>confirm a successful commit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>rollback</td>
<td>30</td>
<td>abandon 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 | S:REDIRECT
challenge = S:SECURE C:SECURE-OK
use-connection = *channel
close-connection = C:CLOSE S:CLOSE-OK
                / S:CLOSE C:CLOSE-OK
```

### 1.4.1. Property and Method Summary

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

<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></td>
<td>version-major</td>
<td>octet</td>
<td>protocol major version</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>version-minor</td>
<td>octet</td>
<td>protocol minor version</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>server-properties</td>
<td>peer-properties</td>
<td>server properties</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mechanisms</td>
<td>longstr</td>
<td>available security mechanisms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>locales</td>
<td>longstr</td>
<td>available message locales</td>
</tr>
<tr>
<td>start-ok</td>
<td>11</td>
<td></td>
<td>select security mechanism and locale</td>
<td>Y</td>
<td></td>
<td>client-properties</td>
<td>peer-properties</td>
<td>client properties</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mechanism</td>
<td>shortstr</td>
<td>selected security mechanism</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>response</td>
<td>longstr</td>
<td>security response data</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>locale</td>
<td>shortstr</td>
<td>selected message locale</td>
</tr>
<tr>
<td>secure</td>
<td>20</td>
<td>secure-ok</td>
<td>security mechanism challenge</td>
<td>Y</td>
<td></td>
<td>challenge</td>
<td>longstr</td>
<td>security challenge data</td>
</tr>
<tr>
<td>secure-ok</td>
<td>21</td>
<td></td>
<td>security mechanism response</td>
<td>Y</td>
<td></td>
<td>response</td>
<td>longstr</td>
<td>security response data</td>
</tr>
<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>
</tbody>
</table>

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

### 1.4.2. Methods

#### 1.4.2.1. Method connection.start (ID 10)

**ID:** 10

**Method accepted by:** Client

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

**Number of parameters:** 5

**Label:** start connection negotiation

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

Guidelines for implementers:

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

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

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

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

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

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

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

**Ordinal:** 1

**Domain:** octet

**Label:** protocol major version

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

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

**Ordinal:** 2

**Domain:** octet

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

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

Ordinal: 3
Domain: peer-properties
Label: server properties

1.4.2.1.4. Parameter connection.start.mechanisms (longstr)

Ordinal: 4
Domain: longstr
Label: available security mechanisms
A list of the security mechanisms that the server supports, delimited by spaces.

1.4.2.1.5. Parameter connection.start.locales (longstr)

Ordinal: 5
Domain: longstr
Label: available message locales
A list of the message locales that the server supports, delimited by spaces. The locale defines the
language in which the server will send reply texts.

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

ID: 11
Method accepted by: Server
Synchronous: No
Number of parameters: 4
Label: select security mechanism and locale

Parameter Summary:

<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>
</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 mechanism selected by the client, which must be one of those specified by the server.

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

Ordinal: 3  
Domain: longstr  
Label: security response data

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

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

Ordinal: 4  
Domain: shortstr  
Label: selected message locale

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

1.4.2.3. **Method connection.secure (ID 20)**

ID: 20  
Method accepted by: Client
Synchronous: Yes; expected response is from method connection.secure-ok

Number of parameters: 1

Label: security mechanism challenge

Parameter Summary:

<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 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

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

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

Ordinal: 2

Domain: long

Label: proposed maximum frame size

The largest frame size that the server proposes for the connection. The client can negotiate a lower value.
Zero means that the server does not impose any specific limit but may reject very large frames if it cannot allocate resources for them.

### 1.4.2.5.3. Parameter connection.tune.heartbeat (short)

**Ordinal:** 3  
**Domain:** short  
**Label:** desired heartbeat delay

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

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

**ID:** 31  
**Method accepted by:** Server  
**Synchronous:** No  
**Number of parameters:** 3  
**Label:** negotiate connection tuning parameters

**Parameter Summary:**

<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 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>capabilities</td>
<td>shortstr</td>
<td>required capabilities</td>
</tr>
<tr>
<td>3</td>
<td>insist</td>
<td>bit</td>
<td>insist on connecting to server</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.capabilities (shortstr)

Ordinal: 2
Domain: shortstr
Label: required capabilities
The client can specify zero or more capability names, delimited by spaces. The server can use this string to how to process the client's connection request.

1.4.2.7.3. Parameter connection.open.insist (bit)

Ordinal: 3
Domain: bit
Label: insist on connecting to server
In a configuration with multiple collaborating servers, the server may respond to a Connection.Open method with a Connection.Redirect. The insist option tells the server that the client is insisting on a connection to the specified server.

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

ID: 41
Method accepted by: Client
Synchronous: No
Number of parameters: 1
Label: signal that connection is ready
Parameter Summary:

<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>known-hosts</td>
<td>known-hosts</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.known-hosts (known-hosts)

Ordinal: 1
**Domain:** known-hosts

### 1.4.2.9. Method connection.redirect (ID 42)

**ID:** 42

**Method accepted by:** Client

**Synchronous:** No

**Number of parameters:** 2

**Label:** redirects client to other server

**Parameter Summary:**

<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>host</td>
<td>shortstr</td>
<td>server to connect to</td>
</tr>
<tr>
<td>2</td>
<td>known-hosts</td>
<td>known-hosts</td>
<td></td>
</tr>
</tbody>
</table>

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

**Guidelines for implementers:**

- When getting the Connection.Redirect method, the client SHOULD reconnect to the host specified, and if that host is not present, to any of the hosts specified in the known-hosts list.

### 1.4.2.9.1. Parameter connection.redirect.host (shortstr)

**Ordinal:** 1

**Domain:** shortstr

**Label:** server to connect to

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

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

**Ordinal:** 2

**Domain:** known-hosts

### 1.4.2.10. Method connection.close (ID 50)

**ID:** 50

**Method accepted by:** Server, Client
**Synchronous:** Yes; expected response is from method `connection.close-ok`

**Number of parameters:** 4

**Label:** request a connection close

**Parameter Summary:**

<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 method except the Close-OK method MUST be discarded.

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

**Ordinal:** 1

**Domain:** reply-code

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

**Ordinal:** 2

**Domain:** reply-text

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

**Ordinal:** 3

**Domain:** class-id

**Label:** failing method class

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

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

**Ordinal:** 4
Domain: method-id
Label: failing method ID

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

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

ID: 51
Method accepted by: Server, Client
Synchronous: No
Number of parameters: 0

Label: confirm a connection close

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

Guidelines for implementers:

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

1.5. **Class channel**

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

Class Grammar:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>channel</td>
<td>open-channel *use-channel close-channel</td>
</tr>
<tr>
<td>open-channel</td>
<td>C:OPEN S:OPEN-OK</td>
</tr>
<tr>
<td></td>
<td>/ C:RESUME S:OK</td>
</tr>
<tr>
<td>use-channel</td>
<td>C:FLOW S:FLOW-OK</td>
</tr>
<tr>
<td></td>
<td>/ S:FLOW C:FLOW-OK</td>
</tr>
<tr>
<td></td>
<td>/ S:PING C:OK</td>
</tr>
<tr>
<td></td>
<td>/ C:PONG S:OK</td>
</tr>
<tr>
<td></td>
<td>/ C:PING S:OK</td>
</tr>
<tr>
<td></td>
<td>/ S:PONG C:OK</td>
</tr>
<tr>
<td></td>
<td>/ functional-class</td>
</tr>
<tr>
<td>close-channel</td>
<td>C:CLOSE S:CLOSE-OK</td>
</tr>
<tr>
<td></td>
<td>/ S:CLOSE C:CLOSE-OK</td>
</tr>
</tbody>
</table>

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></td>
<td>out-of-band</td>
<td>shortstr</td>
</tr>
<tr>
<td>open-ok</td>
<td>11</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td>channel-id</td>
<td>channel-id</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></td>
<td>reply-text</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>class-id</td>
<td>failing method class</td>
</tr>
<tr>
<td>close-ok</td>
<td>41</td>
<td></td>
<td>confirm a channel close</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>method-id</td>
<td>failing method ID</td>
</tr>
<tr>
<td>resume</td>
<td>50</td>
<td>ok</td>
<td>resume an interrupted channel</td>
<td>Y</td>
<td></td>
<td>channel-id</td>
<td>channel-id</td>
<td></td>
</tr>
<tr>
<td>ping</td>
<td>60</td>
<td>ok</td>
<td>[WORK IN PROGRESS] initiates a pong</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td>[ No parameters defined for this method ]</td>
<td></td>
</tr>
<tr>
<td>pong</td>
<td>70</td>
<td>ok</td>
<td>[WORK IN PROGRESS] issued after receiving a ping</td>
<td>Y</td>
<td>Y</td>
<td>[ No parameters defined for this method ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ok</td>
<td>80</td>
<td></td>
<td>[WORK IN PROGRESS] signals normal completion</td>
<td>Y</td>
<td>Y</td>
<td>[ No parameters defined for this method ]</td>
<td></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 `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>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>out-of-band</td>
<td>shortstr</td>
<td>out-of-band settings</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.out-of-band (shortstr)**

Ordinal: 1

Domain: shortstr

Label: out-of-band settings

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

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

ID: 11

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: signal that the channel is ready

Parameter Summary:

<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-id</td>
<td>channel-id</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.channel-id (channel-id)

Ordinal: 1

Domain: channel-id

1.5.2.3. Method channel.flow (ID 20)

ID: 20

Method accepted by: Server, Client

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

Number of parameters: 1

Label: enable/disable flow from peer

Parameter Summary:

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

Guidelines for implementers:

- When a new channel is opened, it is active (flow is active). Some applications assume that channels are inactive until started. To emulate this behaviour a client MAY open the channel, then pause it.
- When sending content frames, a peer SHOULD monitor the channel for incoming methods and respond to a Channel.Flow as rapidly as possible.
- A peer MAY use the Channel.Flow method to throttle incoming content data for internal reasons, for example, when exchanging data over a slower connection.
- The peer that requests a Channel.Flow method MAY disconnect and/or ban a peer that does not respect the request. This is to prevent badly-behaved clients from overwhelming a broker.

1.5.2.3.1. Parameter channel.flow.active (bit)

Ordinal: 1

Domain: bit

Label: start/stop content frames
If 1, the peer starts sending content frames. If 0, the peer stops sending content frames.

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

**ID:** 21

**Method accepted by:** Server, Client

**Synchronous:** No

**Number of parameters:** 1

**Label:** confirm a flow method

**Parameter Summary:**

<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 `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>
</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 method except the Close-OK method MUST be discarded.

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

Ordinal: 1
Domain: reply-code

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

Ordinal: 2
Domain: reply-text

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

Ordinal: 3
Domain: class-id
Label: failing method class

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

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

Ordinal: 4
Domain: method-id
Label: failing method ID

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

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

ID: 41
Method accepted by: Server, Client
Synchronous: No
Number of parameters: 0

Label: confirm a channel close

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

Guidelines for implementers:

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

1.5.2.7.  Method channel.resume (ID 50)

ID: 50

Method accepted by: Server

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

Number of parameters: 1

Label: resume an interrupted channel

Parameter Summary:

<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-id</td>
<td>channel-id</td>
<td></td>
</tr>
</tbody>
</table>

This method resume a previously interrupted channel.

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

Ordinal: 1

Domain: channel-id

1.5.2.8.  Method channel.ping (ID 60)

ID: 60

Method accepted by: Server, Client

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

Number of parameters: 0

Label: [WORK IN PROGRESS] initiates a pong

[WORK IN PROGRESS] Request that the recipient issue a pong request.
1.5.2.9. Method channel.pong (ID 70)

ID: 70

Method accepted by: Server, Client

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

Number of parameters: 0

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

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

1.5.2.10. Method channel.ok (ID 80)

ID: 80

Method accepted by: Server, Client

Synchronous: No

Number of parameters: 0

Label: [WORK IN PROGRESS] signals normal completion

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

1.6. Class access

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

Class Grammar:

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

1.6.1. Property and Method Summary

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

<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>request</td>
<td>10</td>
<td>request-ok</td>
<td>request an access ticket</td>
<td>Y</td>
<td></td>
<td>realm</td>
<td>shortstr</td>
<td>name of requested realm</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>
</tbody>
</table>
### 1.6.2. Methods

#### 1.6.2.1. Method access.request (ID 10)

ID: 10

Method accepted by: Server

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

Number of parameters: 6

Label: request an access ticket

**Parameter Summary:**

<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>realm</td>
<td>shortstr</td>
<td>name of requested realm</td>
</tr>
<tr>
<td>2</td>
<td>exclusive</td>
<td>bit</td>
<td>request exclusive access</td>
</tr>
<tr>
<td>3</td>
<td>passive</td>
<td>bit</td>
<td>request passive access</td>
</tr>
<tr>
<td>4</td>
<td>active</td>
<td>bit</td>
<td>request active access</td>
</tr>
<tr>
<td>5</td>
<td>write</td>
<td>bit</td>
<td>request write access</td>
</tr>
<tr>
<td>6</td>
<td>read</td>
<td>bit</td>
<td>request read access</td>
</tr>
</tbody>
</table>

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

#### 1.6.2.1.1. Parameter access.request.realm (shortstr)

Ordinal: 1
Domain: shortstr

Label: name of requested realm

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

1.6.2.1.2. Parameter access.request.exclusive (bit)

Ordinal: 2

Domain: bit

Label: request exclusive access

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

1.6.2.1.3. Parameter access.request.passive (bit)

Ordinal: 3

Domain: bit

Label: request passive access

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

1.6.2.1.4. Parameter access.request.active (bit)

Ordinal: 4

Domain: bit

Label: request active access

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

1.6.2.1.5. Parameter access.request.write (bit)

Ordinal: 5

Domain: bit

Label: request write access

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

1.6.2.1.6. Parameter access.request.read (bit)

Ordinal: 6
Domain: bit
Label: request read access

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

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

ID: 11
Method accepted by: Client
Synchronous: No
Number of parameters: 1
Label: grant access to server resources

Parameter Summary:

<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>ticket</td>
<td>access-ticket</td>
<td></td>
</tr>
</tbody>
</table>

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

Guidelines for implementers:

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

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

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

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

Ordinal: 1
Domain: access-ticket
1.7. **Class exchange**

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

**Class Grammar:**

<table>
<thead>
<tr>
<th>exchange</th>
<th>= C:DECLARE S:DECLARE-OK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/ C:DELETE S:DELETE-OK</td>
</tr>
</tbody>
</table>

**Guidelines for implementers:**

- The server **MUST** implement these standard exchange types: fanout, direct.
  
  **Test scenario:** Client attempts to declare an exchange with each of these standard types.

- The server **SHOULD** implement these standard exchange types: topic, headers.
  
  **Test scenario:** Client attempts to declare an exchange with each of these standard types.

- The server **MUST**, in each virtual host, pre-declare an exchange instance for each standard exchange type that it implements, where the name of the exchange instance, if defined, is "amq." followed by the exchange type name.

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

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

- The server **MUST** pre-declare a direct exchange with no public name to act as the default exchange for content Publish methods and for default queue bindings.

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

- The server **MUST NOT** allow clients to access the default exchange except by specifying an empty exchange name in the Queue.Bind and content Publish methods.

- The server **MAY** implement other exchange types as wanted.

1.7.1. **Property and Method Summary**

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

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

**ID:** 10

**Method accepted by:** Server

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

**Number of parameters:** 9

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

**Parameter Summary:**

<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>ticket</td>
<td>access-ticket</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>shortstr</td>
<td>exchange type</td>
</tr>
<tr>
<td>passive</td>
<td>bit</td>
<td>do not create exchange</td>
</tr>
<tr>
<td>durable</td>
<td>bit</td>
<td>request a durable exchange</td>
</tr>
<tr>
<td>auto-delete</td>
<td>bit</td>
<td>auto-delete when unused</td>
</tr>
<tr>
<td>internal</td>
<td>bit</td>
<td>create internal exchange</td>
</tr>
<tr>
<td>nowait</td>
<td>bit</td>
<td>do not send reply method</td>
</tr>
<tr>
<td>arguments</td>
<td>table</td>
<td>arguments for declaration</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>declare-ok</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[No parameters defined for this method]
This method creates an exchange if it does not already exist, and if the exchange exists, verifies that it is of the correct and expected class.

**Guidelines for implementers:**

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

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

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

**Ordinal:** 1  
**Domain:** access-ticket

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

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

**Ordinal:** 2  
**Domain:** exchange-name

### 1.7.2.1.3. Parameter exchange.declare.type (shortstr)

**Ordinal:** 3  
**Domain:** shortstr  
**Label:** exchange type

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

Ordinal: 4

Domain: bit

Label: do not create exchange

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

1.7.2.1.5.  Parameter exchange.declare.durable (bit)

Ordinal: 5

Domain: bit

Label: request a durable exchange

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

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

Ordinal: 6

Domain: bit

Label: auto-delete when unused

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

1.7.2.1.7.  Parameter exchange.declare.internal (bit)

Ordinal: 7

Domain: bit

Label: create internal exchange

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

1.7.2.1.8.  Parameter exchange.declare.nowait (bit)

Ordinal: 8

Domain: bit

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

1.7.2.1.9. Parameter exchange.declare.arguments (table)

Ordinal: 9
Domain: table
Label: arguments for declaration
A set of arguments for the declaration. The syntax and semantics of these arguments depends on the server implementation. This field is ignored if passive is 1.

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

ID: 11
Method accepted by: Client
Synchronous: No
Number of parameters: 0
Label: confirm exchange declaration
This method confirms a Declare method and confirms the name of the exchange, essential for automatically-named exchanges.

1.7.2.3. Method exchange.delete (ID 20)

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

<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>ticket</td>
<td>access-ticket</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>nowait</td>
<td>bit</td>
<td>do not send a reply method</td>
</tr>
</tbody>
</table>

This method deletes an exchange. When an exchange is deleted all queue bindings on the exchange are
1.7.2.3.1. **Parameter exchange.delete.ticket (access-ticket)**

*Ordinal*: 1  
*Domain*: access-ticket

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

*Ordinal*: 2  
*Domain*: exchange-name

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

*Ordinal*: 3  
*Domain*: bit  
*Label*: delete only if unused

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

1.7.2.3.4. **Parameter exchange.delete.nowait (bit)**

*Ordinal*: 4  
*Domain*: bit  
*Label*: do not send a reply method

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

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

*ID*: 21  
*Method accepted by*: Client  
*Synchronous*: No  
*Number of parameters*: 0  
*Label*: confirm deletion of an exchange

This method confirms the deletion of an exchange.
1.8. **Class queue**

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

**Class Grammar:**

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

**Guidelines for implementers:**

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

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

### 1.8.1. Property and Method Summary

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

<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, create if needed</td>
<td>Y</td>
<td></td>
<td>ticket</td>
<td>access-ticket</td>
<td>ticket</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>queue</td>
<td>queue-name</td>
<td>Queue name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>passive</td>
<td>bit</td>
<td>passive bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>durable</td>
<td>bit</td>
<td>durable bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>exclusive</td>
<td>bit</td>
<td>exclusive bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>auto-delete</td>
<td>bit</td>
<td>auto-delete bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>nowait</td>
<td>bit</td>
<td>nowait bit</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>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>queue name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>message-count</td>
<td>long</td>
<td>number of messages in queue</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>consumer-count</td>
<td>long</td>
<td>number of consumers</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>bind</td>
<td>20</td>
<td>bind-ok</td>
<td>bind queue to an exchange</td>
<td>Y</td>
<td></td>
<td>ticket</td>
<td>access-ticket</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>exchange</td>
<td>exchange-name</td>
<td>name of the exchange to bind to</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>nowait</td>
<td>bit</td>
<td>do not send a reply method</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>arguments</td>
<td>table</td>
<td>arguments for binding</td>
</tr>
<tr>
<td>bind-ok</td>
<td>21</td>
<td>confirm bind</td>
<td>confirm bind successful</td>
<td>Y</td>
<td></td>
<td>ticket</td>
<td>access-ticket</td>
<td>[ No parameters defined for this method ]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>successful</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>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>routing key of binding</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>nowait</td>
<td>bit</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 of binding</td>
</tr>
<tr>
<td>unbind</td>
<td>50</td>
<td>unbind-ok</td>
<td>unbind a queue from an exchange</td>
<td>Y</td>
<td></td>
<td>ticket</td>
<td>access-ticket</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>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>routing key of binding</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>nowait</td>
<td>bit</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 of binding</td>
</tr>
<tr>
<td>unbind-ok</td>
<td>51</td>
<td>confirm unbind</td>
<td>confirm unbind successful</td>
<td>Y</td>
<td></td>
<td>ticket</td>
<td>access-ticket</td>
<td>[ No parameters defined for this method ]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cancel</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>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>routing key of binding</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>nowait</td>
<td>bit</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 of binding</td>
</tr>
<tr>
<td>purge</td>
<td>30</td>
<td>purge-ok</td>
<td>purge a queue</td>
<td>Y</td>
<td></td>
<td>ticket</td>
<td>access-ticket</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>nowait</td>
<td>bit</td>
<td>do not send a reply method</td>
</tr>
<tr>
<td>purge-ok</td>
<td>31</td>
<td>confirm purge</td>
<td>confirms a queue purge</td>
<td>Y</td>
<td></td>
<td>message-count</td>
<td>long</td>
<td>number of messages purged</td>
</tr>
<tr>
<td>delete</td>
<td>40</td>
<td>delete-ok</td>
<td>delete a queue</td>
<td>Y</td>
<td></td>
<td>ticket</td>
<td>access-ticket</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>if-unused</td>
<td>bit</td>
<td>delete only if unused</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>if-empty</td>
<td>bit</td>
<td>delete only if empty</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>nowait</td>
<td>bit</td>
<td>do not send a reply method</td>
</tr>
<tr>
<td>delete-ok</td>
<td>41</td>
<td>confirm deletion</td>
<td>confirm deletion of a queue</td>
<td>Y</td>
<td></td>
<td>message-count</td>
<td>long</td>
<td>number of messages purged</td>
</tr>
</tbody>
</table>

1.8.2. Methods

1.8.2.1. Method `queue.declare` (ID 10)

ID: 10
Method accepted by: Server

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

Number of parameters: 8

Label: declare queue, create if needed

Parameter Summary:

<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>ticket</td>
<td>access-ticket</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>nowait</td>
<td>bit</td>
<td>do not send a reply method</td>
</tr>
<tr>
<td>8</td>
<td>arguments</td>
<td>table</td>
<td>arguments for declaration</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-created queue to the default exchange, which is an exchange of type 'direct' and use the queue name as the routing key.
  
  Test scenario: Client creates a new queue, and then without explicitly binding it to an exchange, attempts to send a message through the default exchange binding, i.e. publish a message to the empty exchange, with the queue name as routing key.

- The server SHOULD support a minimum of 256 queues per virtual host and ideally, impose no limit except as defined by available resources.
  
  Test scenario: Client attempts to create as many queues as it can until the server reports an error. The resulting count must at least be 256.

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

Ordinal: 1

Domain: access-ticket

When a client defines a new queue, this belongs to the access realm of the ticket used. All further work done with that queue must be done with an access ticket for the same realm.
1.8.2.1.2. **Parameter queue.declare.queue (queue-name)**

Ordinal: 2

Domain: queue-name

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

Ordinal: 3

Domain: bit

Label: do not create queue

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

1.8.2.1.4. **Parameter queue.declare.durable (bit)**

Ordinal: 4

Domain: bit

Label: request a durable queue

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

1.8.2.1.5. **Parameter queue.declare.exclusive (bit)**

Ordinal: 5

Domain: bit

Label: request an exclusive queue

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

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

Ordinal: 6

Domain: bit

Label: auto-delete queue when unused

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

**Ordinal:** 7

**Domain:** bit

**Label:** do not send a reply method

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

1.8.2.1.8. **Parameter queue.declare.arguments (table)**

**Ordinal:** 8

**Domain:** table

**Label:** arguments for declaration

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

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

**ID:** 11

**Method accepted by:** Client

**Synchronous:** No

**Number of parameters:** 3

**Label:** confirms a queue definition

**Parameter Summary:**

<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>queue</td>
<td>queue-name</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>message-count</td>
<td>long</td>
<td>number of messages in queue</td>
</tr>
<tr>
<td>3</td>
<td>consumer-count</td>
<td>long</td>
<td>number of consumers</td>
</tr>
</tbody>
</table>

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

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

**Ordinal:** 1
**Domain:** queue-name

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

### 1.8.2.2. Parameter `queue.declare-ok.message-count (long)`

**Ordinal:** 2

**Domain:** long

**Label:** number of messages in queue

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

### 1.8.2.3. Parameter `queue.declare-ok.consumer-count (long)`

**Ordinal:** 3

**Domain:** long

**Label:** number of consumers

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

### 1.8.3. Method `queue.bind (ID 20)`

**ID:** 20

**Method accepted by:** Server

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

**Number of parameters:** 6

**Label:** bind queue to an exchange

**Parameter Summary:**

<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>ticket</td>
<td>access-ticket</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>nowait</td>
<td>bit</td>
<td>do not send a reply method</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).

- If a bind fails, the server MUST raise a connection exception.
  
  **Test scenario:** TODO

- The server MUST NOT allow a durable queue to bind to a transient exchange.
  
  **Test scenario:** A client creates a transient exchange. The client then declares a named durable queue and then attempts to bind the transient exchange to the durable queue.

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

- Bindings for durable queues are automatically durable and the server SHOULD restore such bindings after a server restart.
  
  **Test scenario:** A server creates a named durable queue and binds it to a durable exchange. The server is restarted. The client then attempts to use the queue/exchange combination.

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

- The server SHOULD support at least 4 bindings per queue, and ideally, impose no limit except as defined by available resources.
  
  **Test scenario:** A client creates a named queue and attempts to bind it to 4 different non-internal exchanges.

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

**Ordinal:** 1

**Domain:** access-ticket

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

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

**Ordinal:** 2

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

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

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

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

**Ordinal:** 4  
**Domain:** shortstr  
**Label:** message routing key

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

### 1.8.2.3.5. Parameter queue.bind.nowait (bit)

**Ordinal:** 5  
**Domain:** bit  
**Label:** do not send a reply method

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

### 1.8.2.3.6. Parameter queue.bind.arguments (table)

**Ordinal:** 6  
**Domain:** table  
**Label:** arguments for binding

A set of arguments for the binding. The syntax and semantics of these arguments depends on the exchange class.
1.8.2.4. Method queue.bind-ok (ID 21)

ID: 21

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm bind successful

This method confirms that the bind was successful.

1.8.2.5. Method queue.unbind (ID 50)

ID: 50

Method accepted by: Server

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

Number of parameters: 5

Label: unbind a queue from an exchange

Parameter Summary:

<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>ticket</td>
<td>access-ticket</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></td>
</tr>
<tr>
<td>4</td>
<td>routing-key</td>
<td>shortstr</td>
<td>routing key of binding</td>
</tr>
<tr>
<td>5</td>
<td>arguments</td>
<td>table</td>
<td>arguments of binding</td>
</tr>
</tbody>
</table>

This method unbinds a queue from an exchange.

Guidelines for implementers:

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

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

Ordinal: 1

Domain: access-ticket

The client provides a valid access ticket giving "active" access rights to the queue's access realm.
1.8.2.5.2. Parameter queue.unbind.queue (queue-name)

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

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

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

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

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

1.8.2.5.5. Parameter queue.unbind.arguments (table)

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

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

ID: 51
Method accepted by: Client
Synchronous: No
Number of parameters: 0
Label: confirm unbind successful
This method confirms that the unbind was successful.

1.8.2.7. Method queue.purge (ID 30)

ID: 30
Method accepted by: Server

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

**Number of parameters:** 3

**Label:** purge a queue

**Parameter Summary:**

<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>ticket</td>
<td>access-ticket</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>queue</td>
<td>queue-name</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>nowait</td>
<td>bit</td>
<td>do not send a reply method</td>
</tr>
</tbody>
</table>

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

**Guidelines for implementers:**

- A call to purge MUST result in an empty queue.
- On transacted channels the server MUST not purge messages that have already been sent to a client but not yet acknowledged.
- The server MAY implement a purge queue or log that allows system administrators to recover accidentally-purged messages. The server SHOULD NOT keep purged messages in the same storage spaces as the live messages since the volumes of purged messages may get very large.

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

**Ordinal:** 1  
**Domain:** access-ticket

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

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

**Ordinal:** 2  
**Domain:** queue-name

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

**1.8.2.7.3. Parameter queue.purge.nowait (bit)**

**Ordinal:** 3
Domain: bit
Label: do not send a reply method
If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

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

ID: 31
Method accepted by: Client
Synchronous: No
Number of parameters: 1
Label: confirms a queue purge

Parameter Summary:

<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>long</td>
<td>number of messages purged</td>
</tr>
</tbody>
</table>

This method confirms the purge of a queue.

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

Ordinal: 1
Domain: long
Label: number of messages purged
Reports the number of messages purged.

1.8.2.9. Method queue.delete (ID 40)

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

Parameter Summary:

<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>ticket</td>
<td>access-ticket</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.

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

Ordinal: 1

Domain: access-ticket

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

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

Ordinal: 2

Domain: queue-name

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

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

Ordinal: 3

Domain: bit

Label: delete only if unused

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

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

Ordinal: 4

Domain: bit
Label: delete only if empty
If set, the server will only delete the queue if it has no messages.

1.8.2.9.5. Parameter *queue.delete.nowait* (bit)

Ordinal: 5
Domain: bit
Label: do not send a reply method
If set, the server will not respond to the method. The client should not wait for a reply method. If the server could not complete the method it will raise a channel or connection exception.

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

ID: 41
Method accepted by: Client
Synchronous: No
Number of parameters: 1
Label: confirm deletion of a queue

Parameter Summary:

<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>long</td>
<td>number of messages purged</td>
</tr>
</tbody>
</table>

This method confirms the deletion of a queue.

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

Ordinal: 1
Domain: long
Label: number of messages purged
Reports the number of messages purged.

1.9. Class *basic*

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

Class Grammar:
Guidelines for implementers:

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

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

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

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

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

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

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

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

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

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

- The server MAY implement up to 10 priority levels.

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

- The server MUST deliver messages of the same priority in order irrespective of their individual persistence.
**Test scenario:** Send a set of messages with the same priority but different persistence settings to a queue. Consume and verify that messages arrive in same order as originally published.

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

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

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

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

### 1.9.1. Property and Method Summary

Class `basic` defines the following properties:

<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>destination 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>
<tr>
<td>app-id</td>
<td>shortstr</td>
<td>creating application id</td>
</tr>
<tr>
<td>cluster-id</td>
<td>shortstr</td>
<td>intra-cluster routing identifier</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</td>
<td>Y</td>
<td></td>
<td>prefetch-size</td>
<td>long</td>
<td>prefetch window in octets</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>service</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>confirm the requested qos</td>
<td>Y</td>
<td></td>
<td></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>ticket</td>
<td>access-ticket</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>nowait</td>
<td>bit</td>
<td>do not send a reply method</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>filter</td>
<td>table</td>
<td>arguments for consuming</td>
</tr>
<tr>
<td>consume- ok</td>
<td>21</td>
<td>confirm a new consumer</td>
<td>Y</td>
<td></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>nowait</td>
<td>bit</td>
<td>do not send a reply method</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></td>
<td>publish a message</td>
<td>Y</td>
<td></td>
<td>ticket</td>
<td>access-ticket</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>routing-key</td>
<td>Y</td>
<td></td>
<td>shortstr</td>
<td>Message routing key</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>mandatory</td>
<td>Y</td>
<td></td>
<td>bit</td>
<td>indicate mandatory routing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>immediate</td>
<td>Y</td>
<td></td>
<td>bit</td>
<td>request immediate delivery</td>
<td></td>
</tr>
<tr>
<td>return</td>
<td>50</td>
<td></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></td>
<td>reply-text</td>
<td>Y</td>
<td></td>
<td>reply-text</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>exchange</td>
<td>Y</td>
<td></td>
<td>exchange-name</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>routing-key</td>
<td>Y</td>
<td></td>
<td>shortstr</td>
<td>Message routing key</td>
<td></td>
</tr>
<tr>
<td>deliver</td>
<td>60</td>
<td></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>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>delivery-tag</td>
<td>Y</td>
<td></td>
<td>delivery-tag</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>redelivered</td>
<td>Y</td>
<td></td>
<td>redelivered</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>exchange</td>
<td>Y</td>
<td></td>
<td>exchange-name</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>routing-key</td>
<td>Y</td>
<td></td>
<td>shortstr</td>
<td>Message routing key</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>ID</td>
<td>Sync. Response</td>
<td>Short description</td>
<td>S</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>
</tr>
<tr>
<td>get</td>
<td>70</td>
<td>get-ok</td>
<td>direct access to a queue</td>
<td>Y</td>
<td>ticket</td>
<td>access-ticket</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>queue</td>
<td>queue-name</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>no-ack</td>
<td>no-ack</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>delivery-tag</td>
<td>delivery-tag</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>redelivered</td>
<td>redelivered</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>routing-key</td>
<td>shortstr</td>
<td>Message routing key</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>message-count</td>
<td>long</td>
<td>number of messages pending</td>
<td></td>
</tr>
<tr>
<td>get-empty</td>
<td>72</td>
<td></td>
<td>indicate no messages available</td>
<td>Y</td>
<td>cluster-id</td>
<td>shortstr</td>
<td>Cluster id</td>
<td></td>
</tr>
<tr>
<td>ack</td>
<td>80</td>
<td></td>
<td>acknowledge one or more messages</td>
<td>Y</td>
<td>delivery-tag</td>
<td>delivery-tag</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>multiple</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>delivery-tag</td>
<td>delivery-tag</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>requeue</td>
<td>bit</td>
<td>requeue the message</td>
<td></td>
</tr>
<tr>
<td>recover</td>
<td>100</td>
<td></td>
<td>redeliver unacknowledged messages</td>
<td>Y</td>
<td>requeue</td>
<td>bit</td>
<td>requeue the message</td>
<td></td>
</tr>
</tbody>
</table>

1.9.2. Properties

1.9.2.1. Property basic.content-type (shortstr)

**Domain:** shortstr

**Label:** MIME content type

1.9.2.2. Property basic.content-encoding (shortstr)

**Domain:** shortstr

**Label:** MIME content encoding

1.9.2.3. Property basic.headers (table)

**Domain:** table
Label: message header field table

1.9.2.4. Property basic.delivery-mode (octet)
Domain: octet
Label: non-persistent (1) or persistent (2)

1.9.2.5. Property basic.priority (octet)
Domain: octet
Label: message priority, 0 to 9

1.9.2.6. Property basic.correlation-id (shortstr)
Domain: shortstr
Label: application correlation identifier

1.9.2.7. Property basic.reply-to (shortstr)
Domain: shortstr
Label: destination to reply to

1.9.2.8. Property basic.expiration (shortstr)
Domain: shortstr
Label: message expiration specification

1.9.2.9. Property basic.message-id (shortstr)
Domain: shortstr
Label: application message identifier

1.9.2.10. Property basic.timestamp (timestamp)
Domain: timestamp
Label: message timestamp

1.9.2.11. Property basic.type (shortstr)
Domain: shortstr
Label: message type name
1.9.2.12. Property basic.user-id (shortstr)

Domain: shortstr
Label: creating user id

1.9.2.13. Property basic.app-id (shortstr)

Domain: shortstr
Label: creating application id

1.9.2.14. Property basic.cluster-id (shortstr)

Domain: shortstr
Label: intra-cluster routing identifier

1.9.3. Methods

1.9.3.1. Method basic.qos (ID 10)

ID: 10
Method accepted by: Server
Synchronous: Yes; expected response is from method basic.qos-ok
Number of parameters: 3
Label: specify quality of service

Parameter Summary:

<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.

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

Ordinal: 1
Domain: long

Label: prefetch window in octets

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

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

Ordinal: 2

Domain: short

Label: prefetch window in messages

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

1.9.3.1.3. Parameter basic.qos.global (bit)

Ordinal: 3

Domain: bit

Label: apply to entire connection

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

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

ID: 11

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm the requested qos

This method tells the client that the requested QoS levels could be handled by the server. The requested QoS applies to all active consumers until a new QoS is defined.
1.9.3.3. Method basic.consume (ID 20)

ID: 20

Method accepted by: Server

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

Number of parameters: 8

Label: start a queue consumer

Parameter Summary:

<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>ticket</td>
<td>access-ticket</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>nowait</td>
<td>bit</td>
<td>do not send a reply method</td>
</tr>
<tr>
<td>8</td>
<td>filter</td>
<td>table</td>
<td>arguments for consuming</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 created on, or until the client cancels them.

Guidelines for implementers:

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

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

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

Ordinal: 1

Domain: access-ticket

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

Ordinal: 2
Domain: queue-name

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

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

Ordinal: 3
Domain: consumer-tag

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

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

Ordinal: 4
Domain: no-local

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

Ordinal: 5
Domain: no-ack

1.9.3.3.6. Parameter basic.consume.exclusive (bit)

Ordinal: 6
Domain: bit
Label: request exclusive access

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

1.9.3.3.7. Parameter basic.consume.nowait (bit)

Ordinal: 7
Domain: bit
Label: do not send a reply method

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

1.9.3.3.8. Parameter basic.consume.filter (table)

Ordinal: 8
**Domain:** table

**Label:** arguments for consuming

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

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

**ID:** 21

**Method accepted by:** Client

**Synchronous:** No

**Number of parameters:** 1

**Label:** confirm a new consumer

**Parameter Summary:**

<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.

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

**Ordinal:** 1

**Domain:** consumer-tag

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

### 1.9.3.5.  Method basic.cancel (ID 30)

**ID:** 30

**Method accepted by:** Server

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

**Number of parameters:** 2

**Label:** end a queue consumer

**Parameter Summary:**

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

Guidelines for implementers:

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

  Test scenario: TODO.

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

Ordinal: 1

Domain: consumer-tag

1.9.3.5.2. Parameter basic.cancel.nowait (bit)

Ordinal: 2

Domain: bit

Label: do not send a reply method

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

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

ID: 31

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: confirm a cancelled consumer

Parameter Summary:

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

Ordinal: 1
Domain: consumer-tag

1.9.3.7. Method basic.publish (ID 40)

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

Parameter Summary:

<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>ticket</td>
<td>access-ticket</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.

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

Ordinal: 1
Domain: access-ticket

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

Ordinal: 2
Domain: exchange-name

Specifies the name of the exchange to publish to. The exchange name can be empty, meaning the default exchange. If the exchange name is specified, and that exchange does not exist, the server will raise a channel exception.
1.9.3.7.3. Parameter basic.publish.routing-key (shortstr)

Ordinal: 3

Domain: shortstr

Label: Message routing key

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

1.9.3.7.4. Parameter basic.publish.mandatory (bit)

Ordinal: 4

Domain: bit

Label: indicate mandatory routing

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

1.9.3.7.5. Parameter basic.publish.immediate (bit)

Ordinal: 5

Domain: bit

Label: request immediate delivery

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

1.9.3.8. Method basic.return (ID 50)

ID: 50

Method accepted by: Client

Synchronous: No

Number of parameters: 4

Label: return a failed message

Parameter Summary:

<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>
</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.

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

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

Ordinal: 1  
Domain: reply-code

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

Ordinal: 2  
Domain: reply-text

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

Ordinal: 3  
Domain: exchange-name

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

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

Ordinal: 4  
Domain: shortstr

Label: Message routing key

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

**1.9.3.9. Method basic.deliver (ID 60)**

ID: 60  
Method accepted by: Client

Synchronous: No  
Number of parameters: 5  
Label: notify the client of a consumer message
Parameter Summary:

<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.

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

**Ordinal:** 1

**Domain:** consumer-tag

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

**Ordinal:** 2

**Domain:** delivery-tag

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

**Ordinal:** 3

**Domain:** redelivered

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

**Ordinal:** 4

**Domain:** exchange-name
Specifies the name of the exchange that the message was originally published to.

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

**Ordinal:** 5

**Domain:** shortstr

**Label:** Message routing key

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

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

**ID:** 70

**Method accepted by:** Server

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

**Number of parameters:** 3

**Label:** direct access to a queue

**Parameter Summary:**

<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>ticket</td>
<td>access-ticket</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>queue</td>
<td>queue-name</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>no-ack</td>
<td>no-ack</td>
<td></td>
</tr>
</tbody>
</table>

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

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

**Ordinal:** 1

**Domain:** access-ticket

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

**Ordinal:** 2

**Domain:** queue-name

Specifies the name of the queue to consume from. If the queue name is null, refers to the current queue for the channel, which is the last declared queue.
1.9.3.10.3. **Parameter basic.get.no-ack (no-ack)**

Ordinal: 3
Domain: no-ack

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

ID: 71
Method accepted by: Client
Synchronous: No
Number of parameters: 5
Label: provide client with a message

Parameter Summary:

<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>long</td>
<td>number of messages pending</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.

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

Ordinal: 1
Domain: delivery-tag

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

Ordinal: 2
Domain: redelivered

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

Ordinal: 3
Domain: exchange-name
Specifies the name of the exchange that the message was originally published to. If empty, the message was published to the default exchange.

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

Ordinal: 4
Domain: shortstr
Label: Message routing key
Specifies the routing key name specified when the message was published.

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

Ordinal: 5
Domain: long
Label: number of messages pending
This field reports the number of messages pending on the queue, excluding the message being delivered. Note that this figure is indicative, not reliable, and can change arbitrarily as messages are added to the queue and removed by other clients.

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

ID: 72
Method accepted by: Client
Synchronous: No
Number of parameters: 1
Label: indicate no messages available

Parameter Summary:

<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>cluster-id</td>
<td>shortstr</td>
<td>Cluster id</td>
</tr>
</tbody>
</table>

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

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

Ordinal: 1
Domain: shortstr
1.9.3.13. Method basic.ack (ID 80)

ID: 80

Method accepted by: Server

Synchronous: No

Number of parameters: 2

Label: acknowledge one or more messages

Parameter Summary:

<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.

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

Ordinal: 1

Domain: delivery-tag

1.9.3.13.2. Parameter basic.ack.multiple (bit)

Ordinal: 2

Domain: bit

Label: acknowledge multiple messages

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

1.9.3.14. Method basic.reject (ID 90)

ID: 90

Method accepted by: Server

Synchronous: No
This method allows a client to reject a message. It can be used to interrupt and cancel large incoming messages, or return untreatable messages to their original queue.

**Guidelines for implementers:**

- The server SHOULD be capable of accepting and process the Reject method while sending message content with a Deliver or Get-Ok method. I.e. the server should read and process incoming methods while sending output frames. To cancel a partially-send content, the server sends a content body frame of size 1 (i.e. with no data except the frame-end octet).

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

  **Test scenario:** TODO.

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

  **Test scenario:** TODO.

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

**Ordinal:** 1

**Domain:** delivery-tag

### 1.9.3.14.2. Parameter `basic.reject.requeue` (bit)

**Ordinal:** 2

**Domain:** bit

**Label:** requeue the message

If this field is zero, the message will be discarded. If this bit is 1, the server will attempt to requeue the message.
1.9.3.15. **Method basic.recover (ID 100)**

ID: 100

Method accepted by: Server

Synchronous: No

Number of parameters: 1

Label: redeliver unacknowledged messages

Parameter Summary:

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

Guidelines for implementers:

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

- The server MUST raise a channel exception if this is called on a transacted channel.
  
  Test scenario: TODO.

1.9.3.15.1. **Parameter basic.recover.requeue (bit)**

Ordinal: 1

Domain: bit

Label: requeue the message

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

1.10. **Class file**

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

Class Grammar:
Guidelines for implementers:

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

### 1.10.1. Property and Method Summary

Class `file` defines the following properties:

<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>priority</td>
<td>octet</td>
<td>message priority, 0 to 9</td>
</tr>
<tr>
<td>reply-to</td>
<td>shortstr</td>
<td>destination to reply to</td>
</tr>
<tr>
<td>message-id</td>
<td>shortstr</td>
<td>application message identifier</td>
</tr>
<tr>
<td>filename</td>
<td>shortstr</td>
<td>message filename</td>
</tr>
<tr>
<td>timestamp</td>
<td>timestamp</td>
<td>message timestamp</td>
</tr>
<tr>
<td>cluster-id</td>
<td>shortstr</td>
<td>intra-cluster routing identifier</td>
</tr>
</tbody>
</table>

Class `file` 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</td>
<td>Y</td>
<td></td>
<td>prefetch-size</td>
<td>long</td>
<td>prefetch window in octets</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></td>
<td></td>
<td></td>
<td>service</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>confirm the requested qos</td>
<td>Y</td>
<td></td>
<td>[ No parameters defined for this method ]</td>
<td></td>
<td></td>
<td></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>ticket</td>
<td>access-ticket</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>queue-name</td>
<td></td>
<td></td>
<td>consumer-tag</td>
<td>consumer-tag</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>no-local</td>
<td></td>
<td></td>
<td>no-local</td>
<td>no-local</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>no-ack</td>
<td></td>
<td></td>
<td>no-ack</td>
<td>no-ack</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>exclusive</td>
<td></td>
<td></td>
<td>bit</td>
<td>request exclusive access</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>nowait</td>
<td></td>
<td></td>
<td>bit</td>
<td>do not send a reply method</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>filter</td>
<td></td>
<td></td>
<td>table</td>
<td>arguments for consuming</td>
<td></td>
</tr>
<tr>
<td>consume-ok</td>
<td>21</td>
<td>confirm a new consumer</td>
<td>Y</td>
<td></td>
<td>consumer-tag</td>
<td>consumer-tag</td>
<td></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>nowait</td>
<td></td>
<td></td>
<td>bit</td>
<td>do not send a reply method</td>
<td></td>
</tr>
<tr>
<td>cancel-ok</td>
<td>31</td>
<td>confirm a cancelled consumer</td>
<td>Y</td>
<td></td>
<td>consumer-tag</td>
<td>consumer-tag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>open</td>
<td>40</td>
<td>open-ok</td>
<td>request to start staging</td>
<td>Y</td>
<td></td>
<td>identifier</td>
<td>shortstr</td>
<td>staging identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>content-size</td>
<td></td>
<td></td>
<td>longlong</td>
<td>message content size</td>
<td></td>
</tr>
<tr>
<td>open-ok</td>
<td>41</td>
<td>stage</td>
<td>confirm staging ready</td>
<td>Y</td>
<td></td>
<td>staged-size</td>
<td>longlong</td>
<td>already staged amount</td>
</tr>
<tr>
<td>stage</td>
<td>50</td>
<td>stage message content</td>
<td>Y</td>
<td></td>
<td>[ No parameters defined for this method ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>publish</td>
<td>60</td>
<td>publish a message</td>
<td>Y</td>
<td></td>
<td>ticket</td>
<td>access-ticket</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></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></td>
<td>routing-key</td>
<td>shortstr</td>
<td></td>
<td>Message routing key</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>mandatory</td>
<td>bit</td>
<td></td>
<td>indicate mandatory routing</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>immediate</td>
<td>bit</td>
<td></td>
<td>request immediate delivery</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>identifier</td>
<td>shortstr</td>
<td></td>
<td>staging identifier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>return</td>
<td>70</td>
<td>return a failed message</td>
<td>Y</td>
<td></td>
<td>reply-code</td>
<td>reply-code</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>reply-text</td>
<td>reply-text</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>exchange</td>
<td>exchange-name</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 1.10.2. Properties

#### 1.10.2.1. Property file.content-type (shortstr)

**Domain:** shortstr  
**Label:** MIME content type

#### 1.10.2.2. Property file.content-encoding (shortstr)

**Domain:** shortstr  
**Label:** MIME content encoding

#### 1.10.2.3. Property file.headers (table)

**Domain:** table  
**Label:** message header field table

#### 1.10.2.4. Property file.priority (octet)

**Domain:** octet  
**Label:** message priority, 0 to 9
1.10.2.5. Property file.reply-to (shortstr)

Domain: shortstr
Label: destination to reply to

1.10.2.6. Property file.message-id (shortstr)

Domain: shortstr
Label: application message identifier

1.10.2.7. Property file.filename (shortstr)

Domain: shortstr
Label: message filename

1.10.2.8. Property file.timestamp (timestamp)

Domain: timestamp
Label: message timestamp

1.10.2.9. Property file.cluster-id (shortstr)

Domain: shortstr
Label: intra-cluster routing identifier

1.10.3. Methods

1.10.3.1. Method file.qos (ID 10)

ID: 10
Method accepted by: Server
Synchronous: Yes; expected response is from method file.qos-ok
Number of parameters: 3
Label: specify quality of service

Parameter Summary:

<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>
</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.

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

Ordinal: 1  
Domain: long  
Label: prefetch window in octets

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

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

Ordinal: 2  
Domain: short  
Label: prefetch window in messages

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

1.10.3.1.3. **Parameter file.qos.global (bit)**

Ordinal: 3  
Domain: bit  
Label: apply to entire connection

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

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

ID: 11
Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm the requested qos

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

1.10.3.3. Method file.consume (ID 20)

ID: 20

Method accepted by: Server

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

Number of parameters: 8

Label: start a queue consumer

Parameter Summary:

<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>ticket</td>
<td>access-ticket</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>nowait</td>
<td>bit</td>
<td>do not send a reply method</td>
</tr>
<tr>
<td>8</td>
<td>filter</td>
<td>table</td>
<td>arguments for consuming</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 created on, or until the client cancels them.

Guidelines for implementers:

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

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

Ordinal: 1
Domain: access-ticket

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

Ordinal: 2

Domain: queue-name

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

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

Ordinal: 3

Domain: consumer-tag

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

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

Ordinal: 4

Domain: no-local

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

Ordinal: 5

Domain: no-ack

1.10.3.3.6. **Parameter file.consume.exclusive (bit)**

Ordinal: 6

Domain: bit

Label: request exclusive access

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

1.10.3.3.7. **Parameter file.consume.nowait (bit)**

Ordinal: 7

Domain: bit

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

### 1.10.3.3.8. Parameter file.consume.filter (table)

**Ordinal:** 8  
**Domain:** table  
**Label:** arguments for consuming

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

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

**ID:** 21  
**Method accepted by:** Client  
**Synchronous:** No  
**Number of parameters:** 1  
**Label:** confirm a new consumer

#### Parameter Summary:

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

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

**Ordinal:** 1  
**Domain:** consumer-tag

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

### 1.10.3.5. Method file.cancel (ID 30)

**ID:** 30  
**Method accepted by:** Server  
**Synchronous:** Yes; expected response is from method file.cancel-ok  
**Number of parameters:** 2
Label: end a queue consumer

Parameter Summary:

<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>nowait</td>
<td>bit</td>
<td>do not send a reply method</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.

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

Ordinal: 1

Domain: consumer-tag

1.10.3.5.2. Parameter file.cancel.nowait (bit)

Ordinal: 2

Domain: bit

Label: do not send a reply method

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

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

ID: 31

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: confirm a cancelled consumer

Parameter Summary:

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

Ordinal: 1
Domain: consumer-tag

1.10.3.7. Method file.open (ID 40)

ID: 40
Method accepted by: Server, Client
Synchronous: Yes; expected response is from method file.open-ok
Number of parameters: 2
Label: request to start staging

Parameter Summary:

<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>identifier</td>
<td>shortstr</td>
<td>staging identifier</td>
</tr>
<tr>
<td>2</td>
<td>content-size</td>
<td>longlong</td>
<td>message content size</td>
</tr>
</tbody>
</table>

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

1.10.3.7.1. Parameter file.open.identifier (shortstr)

Ordinal: 1
Domain: shortstr
Label: staging identifier

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

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

Ordinal: 2
Domain: longlong
Label: message content size
The size of the content in octets. The recipient may use this information to allocate or check available space in advance, to avoid "disk full" errors during staging of very large messages.

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

ID: 41

Method accepted by: Server, Client

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

Number of parameters: 1

Label: confirm staging ready

Parameter Summary:

<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>staged-size</td>
<td>longlong</td>
<td>already staged amount</td>
</tr>
</tbody>
</table>

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

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

Ordinal: 1

Domain: longlong

Label: already staged amount

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

**1.10.3.9. Method file.stage (ID 50)**

ID: 50

Method accepted by: Server, Client

Synchronous: No

Number of parameters: 0

Label: stage message content

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

**1.10.3.10. Method file.publish (ID 60)**

ID: 60
Method accepted by: Server

Synchronous: No

Number of parameters: 6

Label: publish a message

Parameter Summary:

<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>ticket</td>
<td>access-ticket</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>
<tr>
<td>6</td>
<td>identifier</td>
<td>shortstr</td>
<td>staging identifier</td>
</tr>
</tbody>
</table>

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

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

Ordinal: 1

Domain: access-ticket

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

Ordinal: 2

Domain: exchange-name

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

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

Ordinal: 3

Domain: shortstr

Label: Message routing key

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

### 1.10.3.10.4. Parameter file.publish.mandatory (bit)

**Ordinal:** 4  
**Domain:** bit  
**Label:** indicate mandatory routing

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

### 1.10.3.10.5. Parameter file.publish.immediate (bit)

**Ordinal:** 5  
**Domain:** bit  
**Label:** request immediate delivery

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

### 1.10.3.10.6. Parameter file.publish.identifier (shortstr)

**Ordinal:** 6  
**Domain:** shortstr  
**Label:** staging identifier

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

### 1.10.3.11. Method file.return (ID 70)

**ID:** 70  
**Method accepted by:** Client  
**Synchronous:** No  
**Number of parameters:** 4  
**Label:** return a failed message

**Parameter Summary:**

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

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

Ordinal: 1  
Domain: reply-code

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

Ordinal: 2  
Domain: reply-text

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

Ordinal: 3  
Domain: exchange-name

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

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

Ordinal: 4  
Domain: shortstr  
Label: Message routing key

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

1.10.3.12. **Method file.deliver (ID 80)**

ID: 80  
Method accepted by: Client  
Synchronous: No  
Number of parameters: 6
**Label:** notify the client of a consumer message

**Parameter Summary:**

<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>
<tr>
<td>6</td>
<td>identifier</td>
<td>shortstr</td>
<td>staging identifier</td>
</tr>
</tbody>
</table>

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

**Guidelines for implementers:**

- The server SHOULD track the number of times a message has been delivered to clients and when a message is redelivered a certain number of times - e.g. 5 times - without being acknowledged, the server SHOULD consider the message to be unprocessable (possibly causing client applications to abort), and move the message to a dead letter queue.

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

**Ordinal:** 1

**Domain:** consumer-tag

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

**Ordinal:** 2

**Domain:** delivery-tag

**1.10.3.12.3. Parameter file.deliver.redelivered (redelivered)**

**Ordinal:** 3

**Domain:** redelivered

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

**Ordinal:** 4
**Domain:** exchange-name

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

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

- **Ordinal:** 5
- **Domain:** shortstr
- **Label:** Message routing key

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

### 1.10.3.12.6. Parameter file.deliver.identifier (shortstr)

- **Ordinal:** 6
- **Domain:** shortstr
- **Label:** staging identifier

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

### 1.10.3.13. Method file.ack (ID 90)

- **ID:** 90
- **Method accepted by:** Server
- **Synchronous:** No
- **Number of parameters:** 2
- **Label:** acknowledge one or more messages

**Parameter Summary:**

<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 method. The client can ask to confirm a single message or a set of messages up to and including a specific message.

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

- **Ordinal:** 1
**Domain:** delivery-tag

### 1.10.3.13.2. Parameter file.ack.multiple (bit)

**Ordinal:** 2

**Domain:** bit

**Label:** acknowledge multiple messages

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

### 1.10.3.14. Method file.reject (ID 100)

**ID:** 100

**Method accepted by:** Server

**Synchronous:** No

**Number of parameters:** 2

**Label:** reject an incoming message

**Parameter Summary:**

<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 return untreatable messages to their original queue. Note that file content is staged before delivery, so the client will not use this method to interrupt delivery of a large message.

**Guidelines for implementers:**

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

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

**Ordinal:** 1

**Domain:** delivery-tag
1.10.3.14.2. Parameter file.reject.requeue (bit)

Ordinal: 2
Domain: bit
Label: requeue the message

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

1.11. Class stream

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

Class Grammar:

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

Guidelines for implementers:

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

1.11.1. Property and Method Summary

Class stream defines the following properties:

<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>Name</td>
<td>Domain</td>
<td>Short Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>priority</td>
<td>octet</td>
<td>message priority, 0 to 9</td>
</tr>
<tr>
<td>timestamp</td>
<td>timestamp</td>
<td>message timestamp</td>
</tr>
</tbody>
</table>

Class **stream** 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>consume-rate</td>
<td>long</td>
<td>transfer rate in octets/second</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>Y</td>
<td></td>
<td>[ No parameters defined for this method ]</td>
<td></td>
<td></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>ticket</td>
<td>access-ticket</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>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>nowait</td>
<td>bit</td>
<td>do not send a reply method</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>filter</td>
<td>table</td>
<td>arguments for consuming</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>nowait</td>
<td>bit</td>
<td>do not send a reply method</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></td>
<td>publish a message</td>
<td>Y</td>
<td></td>
<td>ticket</td>
<td>access-ticket</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>mandatory</td>
<td>bit</td>
<td>indicate mandatory routing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>immediate</td>
<td>bit</td>
<td>request immediate delivery</td>
</tr>
<tr>
<td>return</td>
<td>50</td>
<td></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></td>
<td></td>
<td></td>
<td></td>
<td>reply-text</td>
<td>reply-text</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>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></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>deliver</td>
<td>60</td>
<td></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>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>exchange</td>
<td>exchange-name</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>
</tbody>
</table>

### 1.11.2. Properties

#### 1.11.2.1. Property stream.content-type (shortstr)

**Domain:** shortstr  
**Label:** MIME content type

#### 1.11.2.2. Property stream.content-encoding (shortstr)

**Domain:** shortstr  
**Label:** MIME content encoding

#### 1.11.2.3. Property stream.headers (table)

**Domain:** table  
**Label:** message header field table

#### 1.11.2.4. Property stream.priority (octet)

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

#### 1.11.2.5. Property stream.timestamp (timestamp)

**Domain:** timestamp  
**Label:** message timestamp
1.11.3. Methods

1.11.3.1. Method stream.qos (ID 10)

ID: 10

Method accepted by: Server

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

Number of parameters: 4

Label: specify quality of service

Parameter Summary:

<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>consume-rate</td>
<td>long</td>
<td>transfer rate in octets/second</td>
</tr>
<tr>
<td>4</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.

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

Ordinal: 1

Domain: long

Label: prefetch window in octets

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

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

Ordinal: 2

Domain: short

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

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

**Ordinal:** 3  
**Domain:** long  
**Label:** transfer rate in octets/second

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

### 1.11.3.1.4. Parameter stream.qos.global (bit)

**Ordinal:** 4  
**Domain:** bit  
**Label:** apply to entire connection

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

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

**ID:** 11  
**Method accepted by:** Client  
**Synchronous:** No  
**Number of parameters:** 0  
**Label:** confirm the requested qos

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

### 1.11.3.3. Method stream.consume (ID 20)

**ID:** 20  
**Method accepted by:** Server  
**Synchronous:** Yes; expected response is from method stream.consume-ok

**Number of parameters:** 7  
**Label:** start a queue consumer
Parameter Summary:

<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>ticket</td>
<td>access-ticket</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>exclusive</td>
<td>bit</td>
<td>request exclusive access</td>
</tr>
<tr>
<td>6</td>
<td>nowait</td>
<td>bit</td>
<td>do not send a reply method</td>
</tr>
<tr>
<td>7</td>
<td>filter</td>
<td>table</td>
<td>arguments for consuming</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 created on, or until the client cancels them.

Guidelines for implementers:

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

- Streaming applications SHOULD use different channels to select different streaming resolutions. AMQP makes no provision for filtering and/or transforming streams except on the basis of priority-based selective delivery of individual messages.

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

Ordinal: 1

Domain: access-ticket

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

Ordinal: 2

Domain: queue-name

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

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

Ordinal: 3

Domain: consumer-tag

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

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

Ordinal: 4

**Domain:** no-local

1.11.3.3.5. **Parameter stream.consume.exclusive (bit)**

Ordinal: 5

**Domain:** bit

**Label:** request exclusive access

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

1.11.3.3.6. **Parameter stream.consume.nowait (bit)**

Ordinal: 6

**Domain:** bit

**Label:** do not send a reply method

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

1.11.3.3.7. **Parameter stream.consume.filter (table)**

Ordinal: 7

**Domain:** table

**Label:** arguments for consuming

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

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

**ID:** 21

**Method accepted by:** Client

**Synchronous:** No

**Number of parameters:** 1

**Label:** confirm a new consumer
Parameter Summary:

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

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

Ordinal: 1

Domain: consumer-tag

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

1.11.3.5. Method stream.cancel (ID 30)

ID: 30

Method accepted by: Server

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

Number of parameters: 2

Label: end a queue consumer

Parameter Summary:

<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>nowait</td>
<td>bit</td>
<td>do not send a reply method</td>
</tr>
</tbody>
</table>

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

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

Ordinal: 1

Domain: consumer-tag

1.11.3.5.2. Parameter stream.cancel.nowait (bit)

Ordinal: 2
Domain: bit

Label: do not send a reply method

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

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

ID: 31

Method accepted by: Client

Synchronous: No

Number of parameters: 1

Label: confirm a cancelled consumer

Parameter Summary:

<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.

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

Ordinal: 1

Domain: consumer-tag

1.11.3.7. Method stream.publish (ID 40)

ID: 40

Method accepted by: Server

Synchronous: No

Number of parameters: 5

Label: publish a message

Parameter Summary:

<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>ticket</td>
<td>access-ticket</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>
</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 as appropriate.

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

 ordinal: 1

domain: access-ticket

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

 ordinal: 2

domain: exchange-name

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

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

 ordinal: 3

domain: shortstr

label: Message routing key

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

1.11.3.7.4. **Parameter stream.publish.mandatory (bit)**

 ordinal: 4

domain: bit

label: indicate mandatory routing

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

1.11.3.7.5. **Parameter stream.publish.immediate (bit)**

 ordinal: 5
Domain: bit

Label: request immediate delivery

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

1.11.3.8. Method stream.return (ID 50)

ID: 50

Method accepted by: Client

Synchronous: No

Number of parameters: 4

Label: return a failed message

Parameter Summary:

<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.

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

Ordinal: 1

Domain: reply-code

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

Ordinal: 2

Domain: reply-text

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

Ordinal: 3
**Domain:** exchange-name

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

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

**Ordinal:** 4

**Domain:** shortstr

**Label:** Message routing key

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

### 1.11.3.9. Method stream.deliver (ID 60)

**ID:** 60

**Method accepted by:** Client

**Synchronous:** No

**Number of parameters:** 4

**Label:** notify the client of a consumer message

**Parameter Summary:**

<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>exchange</td>
<td>exchange-name</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>queue</td>
<td>queue-name</td>
<td></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.

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

**Ordinal:** 1

**Domain:** consumer-tag

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

**Ordinal:** 2
**Domain:** delivery-tag

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

**Ordinal:** 3

**Domain:** exchange-name

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

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

**Ordinal:** 4

**Domain:** queue-name

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

### 1.12. Class tx

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

**Class Grammar:**

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

**Guidelines for implementers:**

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

### 1.12.1. Property and Method Summary

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

<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</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>[ No parameters defined for this method ]</td>
</tr>
</tbody>
</table>

AMQP Specification (XML-derived)
<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-ok</td>
<td>11</td>
<td></td>
<td>confirm transaction mode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[ No parameters defined for this method ]</td>
</tr>
<tr>
<td>commit</td>
<td>20</td>
<td>commit-ok</td>
<td>commit the current transaction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[ No parameters defined for this method ]</td>
</tr>
<tr>
<td>commit-ok</td>
<td>21</td>
<td></td>
<td>confirm a successful commit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[ No parameters defined for this method ]</td>
</tr>
<tr>
<td>rollback</td>
<td>30</td>
<td>rollback-ok</td>
<td>abandon the current transaction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[ No parameters defined for this method ]</td>
</tr>
<tr>
<td>rollback-ok</td>
<td>31</td>
<td></td>
<td>confirm successful rollback</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[ No parameters defined for this method ]</td>
</tr>
</tbody>
</table>

### 1.12.2. Methods

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

**ID:** 10

**Method accepted by:** Server

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

**Number of parameters:** 0

**Label:** select standard transaction mode

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

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

**ID:** 11

**Method accepted by:** Client

**Synchronous:** No

**Number of parameters:** 0
Label: confirm transaction mode
This method confirms to the client that the channel was successfully set to use standard transactions.

1.12.2.3. Method tx.commit (ID 20)
ID: 20
Method accepted by: Server
Synchronous: Yes; expected response is from method tx.commit-ok
Number of parameters: 0
Label: commit the current transaction
This method commits all messages published and acknowledged in the current transaction. A new transaction starts immediately after a commit.

1.12.2.4. Method tx.commit-ok (ID 21)
ID: 21
Method accepted by: Client
Synchronous: No
Number of parameters: 0
Label: confirm a successful commit
This method confirms to the client that the commit succeeded. Note that if a commit fails, the server raises a channel exception.

1.12.2.5. Method tx.rollback (ID 30)
ID: 30
Method accepted by: Server
Synchronous: Yes; expected response is from method tx.rollback-ok
Number of parameters: 0
Label: abandon the current transaction
This method abandons all messages published and acknowledged in the current transaction. A new transaction starts immediately after a rollback.

1.12.2.6. Method tx.rollback-ok (ID 31)
ID: 31
Method accepted by: Client
Synchronous: No

Number of parameters: 0

Label: confirm successful rollback

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

1.13. Class dtx

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

Class Grammar:

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

1.13.1. Property and Method Summary

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

<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>
<tr>
<td>select-ok</td>
<td>11</td>
<td></td>
<td>confirm transaction mode</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>[ No parameters defined for this method ]</td>
</tr>
<tr>
<td>start</td>
<td>20</td>
<td>start-ok</td>
<td>start a new distributed transaction</td>
<td>Y</td>
<td></td>
<td>dtx-identifier</td>
<td>shortstr</td>
<td>transaction identifier</td>
</tr>
<tr>
<td>start-ok</td>
<td>21</td>
<td></td>
<td>confirm the start of a new distributed transaction</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>[ No parameters defined for this method ]</td>
</tr>
</tbody>
</table>
1.13.2. Methods

1.13.2.1. Method dtx.select (ID 10)

ID: 10

Method accepted by: Server

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

Number of parameters: 0

Label: select standard transaction mode

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

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

ID: 11

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm transaction mode

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

1.13.2.3. Method dtx.start (ID 20)

ID: 20

Method accepted by: Server

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

Number of parameters: 1

Label: start a new distributed transaction

Parameter Summary:

<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>dtx-identifier</td>
<td>shortstr</td>
<td>transaction identifier</td>
</tr>
</tbody>
</table>

This method starts a new distributed transaction. This must be the first method on a new channel that uses the distributed transaction mode, before any methods that publish or consume messages.
1.13.2.3.1.  Parameter dtx.start.dtx-identifier (shortstr)

Ordinal: 1

Domain: shortstr

Label: transaction identifier

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

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

ID: 21

Method accepted by: Client

Synchronous: No

Number of parameters: 0

Label: confirm the start of a new distributed transaction

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

1.14.  Class tunnel

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

Class Grammar:

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

1.14.1.  Property and Method Summary

Class tunnel defines the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Domain</th>
<th>Short Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>headers</td>
<td>table</td>
<td>message header field table</td>
</tr>
<tr>
<td>proxy-name</td>
<td>shortstr</td>
<td>identity of tunnelling proxy</td>
</tr>
<tr>
<td>data-name</td>
<td>shortstr</td>
<td>name or type of message being tunnelled</td>
</tr>
<tr>
<td>durable</td>
<td>octet</td>
<td>message durability indicator</td>
</tr>
<tr>
<td>Name</td>
<td>Domain</td>
<td>Short Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>broadcast</td>
<td>octet</td>
<td>message broadcast mode</td>
</tr>
</tbody>
</table>

Class *tunnel* 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>request</td>
<td>10</td>
<td></td>
<td>sends a tunnelled method</td>
<td>Y</td>
<td></td>
<td>meta-data</td>
<td>table</td>
<td>meta data for the tunnelled block</td>
</tr>
</tbody>
</table>

### 1.14.2. Properties

**1.14.2.1. Property tunnel.headers (table)**

Domain: table

Label: message header field table

**1.14.2.2. Property tunnel.proxy-name (shortstr)**

Domain: shortstr

Label: identity of tunnelling proxy

**1.14.2.3. Property tunnel.data-name (shortstr)**

Domain: shortstr

Label: name or type of message being tunnelled

**1.14.2.4. Property tunnel.durable (octet)**

Domain: octet

Label: message durability indicator

**1.14.2.5. Property tunnel.broadcast (octet)**

Domain: octet

Label: message broadcast mode
1.14.3.  Methods

1.14.3.1.  Method tunnel.request (ID 10)

ID: 10

Method accepted by: Server

Synchronous: No

Number of parameters: 1

Label: sends a tunnelled method

Parameter Summary:

<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>meta-data</td>
<td>table</td>
<td>meta data for the tunnelled block</td>
</tr>
</tbody>
</table>

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

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

Ordinal: 1

Domain: table

Label: meta data for the tunnelled block

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

1.15.  Class message

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

Class Grammar:

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

- The server SHOULD respect the persistent property of messages and SHOULD make a best-effort to hold persistent 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 message in case of a queue overflow.
  
  **Test scenario:** Create a queue overflow situation with persistent messages and verify that messages do not get lost (presumably the server will write them to disk).

- The server MAY use the Channel.Flow method to slow or stop a message publisher when necessary.
  
  **Test scenario:** Create a queue overflow situation with non-persistent messages and verify whether the server responds with Channel Flow or not. Repeat with persistent messages.

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

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

- The server MUST implement at least 2 priority levels for messages, where priorities 0-4 and 5-9 are treated as two distinct levels.
  
  **Test scenario:** Send a number of priority 0 messages to a queue. Send one priority 9 message. Consume messages from the queue and verify that the first message received was priority 9.

- The server MAY implement up to 10 priority levels.
  
  **Test scenario:** Send a number of messages with mixed priorities to a queue, so that all priority values from 0 to 9 are exercised. A good scenario would be ten messages in low-to-high priority. Consume from queue and verify how many priority levels emerge.

- The server MUST deliver messages of the same priority in order irrespective of their individual persistence.
  
  **Test scenario:** Send a set of messages with the same priority but different persistence settings to a queue. Consume and verify that messages arrive in same order as originally published.
- The server MUST support automatic acknowledgements on messages, i.e. consumers with the no-ack field set to FALSE.

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

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

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

### 1.15.1. Property and Method Summary

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

<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>transfer</td>
<td>10</td>
<td>ok</td>
<td>[WORK IN PROGRESS] transfer a message</td>
<td>Y</td>
<td>Y</td>
<td>ticket</td>
<td>access-ticket</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>destination</td>
<td>destination</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>immediate</td>
<td>bit</td>
<td>request immediate delivery</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ttl</td>
<td>duration</td>
<td>time to live</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>priority</td>
<td>octet</td>
<td>message priority, 0 to 9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>timestamp</td>
<td>timestamp</td>
<td>message timestamp</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>delivery-mode</td>
<td>octet</td>
<td>non-persistent (1) or persistent (2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>expiration</td>
<td>timestamp</td>
<td>message expiration time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>exchange</td>
<td>exchange-name</td>
<td>originating exchange</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-id</td>
<td>shortstr</td>
<td>application message identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>correlation-id</td>
<td>shortstr</td>
<td>application correlation identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>reply-to</td>
<td>shortstr</td>
<td>destination to reply to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>content-type</td>
<td>shortstr</td>
<td>MIME content type</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>content-encoding</td>
<td>shortstr</td>
<td>MIME content encoding</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>user-id</td>
<td>shortstr</td>
<td>creating user id</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>app-id</td>
<td>shortstr</td>
<td>creating application id</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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>transaction-id</td>
<td>shortstr</td>
<td>distributed transaction id</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>security-token</td>
<td>security-token</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>application-headers</td>
<td>table</td>
<td>application specific headers table</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>body</td>
<td>content</td>
<td>message body</td>
</tr>
<tr>
<td>consume</td>
<td>20</td>
<td>ok</td>
<td>[WORK IN PROGRESS] start a queue consumer</td>
<td>Y</td>
<td></td>
<td>ticket</td>
<td>access-ticket</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>destination</td>
<td>destination</td>
<td>incoming message destination</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>filter</td>
<td>table</td>
<td>arguments for consuming</td>
</tr>
<tr>
<td>cancel</td>
<td>30</td>
<td>ok</td>
<td>[WORK IN PROGRESS] end a queue consumer</td>
<td>Y</td>
<td></td>
<td>destination</td>
<td>destination</td>
<td></td>
</tr>
<tr>
<td>get</td>
<td>40</td>
<td>ok</td>
<td>[WORK IN PROGRESS] direct access to a queue</td>
<td>Y</td>
<td></td>
<td>ticket</td>
<td>access-ticket</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>destination</td>
<td>destination</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>recover</td>
<td>50</td>
<td>ok</td>
<td>[WORK IN PROGRESS] redeliver unacknowledge d messages</td>
<td>Y</td>
<td></td>
<td>requeue</td>
<td>bit</td>
<td>requeue the message</td>
</tr>
<tr>
<td>open</td>
<td>60</td>
<td>ok</td>
<td>[WORK IN PROGRESS] create a reference to an empty message body</td>
<td>Y</td>
<td>Y</td>
<td>reference</td>
<td>reference</td>
<td></td>
</tr>
<tr>
<td>close</td>
<td>70</td>
<td>ok</td>
<td>[WORK IN PROGRESS] close a reference</td>
<td>Y</td>
<td>Y</td>
<td>reference</td>
<td>reference</td>
<td>target reference</td>
</tr>
<tr>
<td>append</td>
<td>80</td>
<td>ok</td>
<td>[WORK IN PROGRESS]</td>
<td>Y</td>
<td>Y</td>
<td>reference</td>
<td>reference</td>
<td>target reference</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>checkpoint</td>
<td>90</td>
<td>ok</td>
<td>[WORK IN PROGRESS] append to a reference</td>
<td>Y</td>
<td>Y</td>
<td>reference</td>
<td>reference</td>
<td>target reference</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>identifier</td>
<td>shortstr</td>
<td>checkpoint identifier</td>
</tr>
<tr>
<td>resume</td>
<td>100</td>
<td>offset</td>
<td>[WORK IN PROGRESS] checkpoint a message body</td>
<td>Y</td>
<td>Y</td>
<td>reference</td>
<td>reference</td>
<td>target reference</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>identifier</td>
<td>shortstr</td>
<td>checkpoint identifier</td>
</tr>
<tr>
<td>qos</td>
<td>110</td>
<td>ok</td>
<td>[WORK IN PROGRESS] 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>ok</td>
<td>500</td>
<td></td>
<td>[WORK IN PROGRESS] normal completion</td>
<td>Y</td>
<td>Y</td>
<td>[ No parameters defined for this method ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>empty</td>
<td>510</td>
<td></td>
<td>[WORK IN PROGRESS] empty queue</td>
<td>Y</td>
<td>Y</td>
<td>[ No parameters defined for this method ]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reject</td>
<td>520</td>
<td></td>
<td>[WORK IN PROGRESS] reject a message</td>
<td>Y</td>
<td>Y</td>
<td>code</td>
<td>reject-code</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>text</td>
<td>reject-text</td>
<td></td>
</tr>
<tr>
<td>offset</td>
<td>530</td>
<td></td>
<td>[WORK IN PROGRESS] return an offset</td>
<td>Y</td>
<td>Y</td>
<td>value</td>
<td>offset</td>
<td>offset into a reference body</td>
</tr>
</tbody>
</table>

1.15.2. Methods

1.15.2.1. Method message.transfer (ID 10)

ID: 10

Method accepted by: Server, Client

Synchronous: Yes; expected response is from method message.ok
Number of parameters: 22

Label: [WORK IN PROGRESS] transfer a message

Parameter Summary:

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

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

Guidelines for implementers:

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

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

**Ordinal:** 1  
**Domain:** access-ticket

1.15.2.1.2. **Parameter message.transfer.destination (destination)**

**Ordinal:** 2  
**Domain:** destination

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

1.15.2.1.3. **Parameter message.transfer.redelivered (redelivered)**

**Ordinal:** 3  
**Domain:** redelivered

1.15.2.1.4. **Parameter message.transfer.immediate (bit)**

**Ordinal:** 4  
**Domain:** bit  
**Label:** request immediate delivery

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

1.15.2.1.5. **Parameter message.transfer.ttl (duration)**

**Ordinal:** 5  
**Domain:** duration  
**Label:** time to live

If this is set to a non zero value then a message expiration time will be computed based on the current time
plus this value. Messages that live longer than their expiration time will be discarded (or dead lettered).

1.15.2.1.6. **Parameter message.transfer.priority (octet)**

Ordinal: 6
Domain: octet
Label: message priority, 0 to 9

1.15.2.1.7. **Parameter message.transfer.timestamp (timestamp)**

Ordinal: 7
Domain: timestamp
Label: message timestamp
Set on arrival by the broker.

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

Ordinal: 8
Domain: octet
Label: non-persistent (1) or persistent (2)

1.15.2.1.9. **Parameter message.transfer.expiration (timestamp)**

Ordinal: 9
Domain: timestamp
Label: message expiration time
The expiration header assigned by the broker. After receiving the message the broker sets expiration to the sum of the ttl specified in the publish method and the current time. (ttl = expiration - timestamp)

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

Ordinal: 10
Domain: exchange-name
Label: originating exchange

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

Ordinal: 11
Domain: shortstr  
Label: message routing key  

1.15.2.1.12. Parameter message.transfer.message-id (shortstr)  
Ordinal: 12  
Domain: shortstr  
Label: application message identifier  

1.15.2.1.13. Parameter message.transfer.correlation-id (shortstr)  
Ordinal: 13  
Domain: shortstr  
Label: application correlation identifier  

1.15.2.1.14. Parameter message.transfer.reply-to (shortstr)  
Ordinal: 14  
Domain: shortstr  
Label: destination to reply to  

1.15.2.1.15. Parameter message.transfer.content-type (shortstr)  
Ordinal: 15  
Domain: shortstr  
Label: MIME content type  

1.15.2.1.16. Parameter message.transfer.content-encoding (shortstr)  
Ordinal: 16  
Domain: shortstr  
Label: MIME content encoding  

1.15.2.1.17. Parameter message.transfer.user-id (shortstr)  
Ordinal: 17  
Domain: shortstr  
Label: creating user id
1.15.2.1.18. Parameter message.transfer.app-id (shortstr)

Ordinal: 18
Domain: shortstr
Label: creating application id

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

Ordinal: 19
Domain: shortstr
Label: distributed transaction id

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

Ordinal: 20
Domain: security-token

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

Ordinal: 21
Domain: table
Label: application specific headers table

1.15.2.1.22. Parameter message.transfer.body (content)

Ordinal: 22
Domain: content
Label: message body

1.15.2.2. Method message.consume (ID 20)

ID: 20
Method accepted by: Server
Synchronous: Yes; expected response is from method message.ok
Number of parameters: 7
Label: [WORK IN PROGRESS] start a queue consumer
Parameter Summary:
<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>ticket</td>
<td>access-ticket</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>queue</td>
<td>queue-name</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>destination</td>
<td>destination</td>
<td>incoming message destination</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>filter</td>
<td>table</td>
<td>arguments for consuming</td>
</tr>
</tbody>
</table>

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

Guidelines for implementers:

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

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

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

Ordinal: 1

Domain: access-ticket

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

Ordinal: 2

Domain: queue-name

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

1.15.2.2.3. Parameter message.consume.destination (destination)

Ordinal: 3

Domain: destination

Label: incoming message destination

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

1.15.2.2.4. Parameter message.consume.no-local (no-local)
Ordinal: 4
Domain: no-local

1.15.2.2.5. Parameter message.consume.no-ack (no-ack)
Ordinal: 5
Domain: no-ack

1.15.2.2.6. Parameter message.consume.exclusive (bit)
Ordinal: 6
Domain: bit
Label: request exclusive access
Request exclusive consumer access, meaning only this consumer can access the queue.

1.15.2.2.7. Parameter message.consume.filter (table)
Ordinal: 7
Domain: table
Label: arguments for consuming
A set of filters for the consume. The syntax and semantics of these filters depends on the providers implementation.

1.15.2.3. Method message.cancel (ID 30)
ID: 30
Method accepted by: Server
Synchronous: Yes; expected response is from method message.ok
Number of parameters: 1
Label: [WORK IN PROGRESS] end a queue consumer

<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>destination</td>
<td>destination</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.

### 1.15.2.3.1. Parameter `message.cancel.destination` (destination)

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

### 1.15.2.4. Method `message.get` (ID 40)

**ID:** 40  
**Method accepted by:** Server  
**Synchronous:** Yes; expected response is from method `message.ok`  
**Number of parameters:** 4  
**Label:** [WORK IN PROGRESS] direct access to a queue

**Parameter Summary:**

<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>ticket</td>
<td>access-ticket</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>queue</td>
<td>queue-name</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>destination</td>
<td>destination</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>no-ack</td>
<td>no-ack</td>
<td></td>
</tr>
</tbody>
</table>

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

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

**Ordinal:** 1  
**Domain:** access-ticket
1.15.2.4.2. Parameter message.get.queue (queue-name)

Ordinal: 2
Domain: queue-name

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

1.15.2.4.3. Parameter message.get.destination (destination)

Ordinal: 3
Domain: destination

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

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

Ordinal: 4
Domain: no-ack

1.15.2.5. Method message.recover (ID 50)

ID: 50
Method accepted by: Server
Synchronous: Yes; expected response is from method message.ok
Number of parameters: 1
Label: [WORK IN PROGRESS] redeliver unacknowledged messages

Parameter Summary:

<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>

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

Guidelines for implementers:

- The server MUST set the redelivered flag on all messages that are resent.
- The server MUST raise a channel exception if this is called on a transacted channel.
1.15.2.5.1. Parameter message.recover.requeue (bit)

Ordinal: 1
Domain: bit
Label: requeue the message

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

1.15.2.6. Method message.open (ID 60)

ID: 60
Method accepted by: Server, Client
Synchronous: Yes; expected response is from method message.ok
Number of parameters: 1
Label: [WORK IN PROGRESS] create a reference to an empty message body

Parameter Summary:

<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>reference</td>
<td>reference</td>
<td></td>
</tr>
</tbody>
</table>

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

1.15.2.6.1. Parameter message.open.reference (reference)

Ordinal: 1
Domain: reference

1.15.2.7. Method message.close (ID 70)

ID: 70
Method accepted by: Server, Client
Synchronous: Yes; expected response is from method message.ok
Number of parameters: 1

Label: [WORK IN PROGRESS] close a reference

Parameter Summary:

<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>reference</td>
<td>reference</td>
<td>target reference</td>
</tr>
</tbody>
</table>

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

Guidelines for implementers:

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

1.15.2.7.1. Parameter message.close.reference (reference)

Ordinal: 1

Domain: reference

Label: target reference

1.15.2.8. Method message.append (ID 80)

ID: 80

Method accepted by: Server, Client

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

Number of parameters: 2

Label: [WORK IN PROGRESS] append to a reference

Parameter Summary:

<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>reference</td>
<td>reference</td>
<td>target reference</td>
</tr>
<tr>
<td>2</td>
<td>bytes</td>
<td>longstr</td>
<td>data to append</td>
</tr>
</tbody>
</table>

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

1.15.2.8.1. Parameter message.append.reference (reference)

Ordinal: 1

Domain: reference
Label: target reference

1.15.2.8.2. Parameter message.append.bytes (longstr)
Ordinal: 2
Domain: longstr
Label: data to append

1.15.2.9. Method message.checkpoint (ID 90)
ID: 90
Method accepted by: Server, Client
Synchronous: Yes; expected response is from method message.ok
Number of parameters: 2
Label: [WORK IN PROGRESS] checkpoint a message body

Parameter Summary:

<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>reference</td>
<td>reference</td>
<td>target reference</td>
</tr>
<tr>
<td>2</td>
<td>identifier</td>
<td>shortstr</td>
<td>checkpoint identifier</td>
</tr>
</tbody>
</table>

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

1.15.2.9.1. Parameter message.checkpoint.reference (reference)
Ordinal: 1
Domain: reference
Label: target reference

1.15.2.9.2. Parameter message.checkpoint.identifier (shortstr)
Ordinal: 2
Domain: shortstr
Label: checkpoint identifier
This is the checkpoint identifier. This is an arbitrary string chosen by the sender. For checkpointing to work correctly the sender must use the same checkpoint identifier when resuming the message. A good choice for the checkpoint identifier would be the SHA1 hash of the message properties data (including the original filename, revised time, etc.).

**1.15.2.10. Method message.resume (ID 100)**

**ID:** 100

**Method accepted by:** Server, Client

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

**Number of parameters:** 2

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

**Parameter Summary:**

<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>reference</td>
<td>reference</td>
<td>target reference</td>
</tr>
<tr>
<td>2</td>
<td>identifier</td>
<td>shortstr</td>
<td>checkpoint identifier</td>
</tr>
</tbody>
</table>

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

**1.15.2.10.1. Parameter message.resume.reference (reference)**

**Ordinal:** 1

**Domain:** reference

**Label:** target reference

**1.15.2.10.2. Parameter message.resume.identifier (shortstr)**

**Ordinal:** 2

**Domain:** shortstr

**Label:** checkpoint identifier

**1.15.2.11. Method message.qos (ID 110)**

**ID:** 110

**Method accepted by:** Server
Synchronous: Yes; expected response is from method `message.ok`

Number of parameters: 3

Label: [WORK IN PROGRESS] specify quality of service

Parameter Summary:

<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>

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

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

Ordinal: 1

Domain: long

Label: prefetch window in octets

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

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

Ordinal: 2

Domain: short

Label: prefetch window in messages

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

Ordinal: 3
Domain: bit
Label: apply to entire connection

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

1.15.2.12. Method message.ok (ID 500)

ID: 500
Method accepted by: Server, Client
Synchronous: No
Number of parameters: 0
Label: [WORK IN PROGRESS] normal completion
[WORK IN PROGRESS] Signals the normal completion of a method.

1.15.2.13. Method message.empty (ID 510)

ID: 510
Method accepted by: Server, Client
Synchronous: No
Number of parameters: 0
Label: [WORK IN PROGRESS] empty queue
[WORK IN PROGRESS] Signals that a queue does not contain any messages.

1.15.2.14. Method message.reject (ID 520)

ID: 520
Method accepted by: Server, Client
Synchronous: No
Number of parameters: 2
Label: [WORK IN PROGRESS] reject a message

Parameter Summary:

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

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

**Ordinal:** 1

**Domain:** reject-code

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

**Ordinal:** 2

**Domain:** reject-text

### 1.15.2.15. Method message.offset (ID 530)

**ID:** 530

**Method accepted by:** Server, Client

**Synchronous:** No

**Number of parameters:** 1

**Label:** [WORK IN PROGRESS] return an offset

#### Parameter Summary:

<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>value</td>
<td>offset</td>
<td>offset into a reference body</td>
</tr>
</tbody>
</table>

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

#### 1.15.2.15.1. Parameter message.offset.value (offset)

**Ordinal:** 1

**Domain:** offset

**Label:** offset into a reference body