SMQP: Simple Message Queue Protocol

Status of this Memo

This document is an Internet-Draft and is subject to all provisions of Section 10 of RFC2026 except that the right to produce derivative works is not granted.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

All comments should be directed to the author at john@omicronsoft.com

Copyright Notice

Copyright (C) The Internet Society (2001). All Rights Reserved

Abstract

The Simple Message Queue Protocol (SMQP) provides a standard form of sharing data between those that publish data and those that desire to subscribe to data. Publish/subscribe systems decouple the two end points from knowing about one another. SMQP will provide a common protocol to be used for a variety of implementation from simple chat applications to mission critical message flows.

Table of Contents

1. Introduction ..................................................... 4
1.1 Overview of SMQP Functionality .................................. 4
1.2 Terminology ...................................................... 4
2. Architecture of SMQP .................................................. 6
2.1 Goal of the Architecture ............................................ 6
2.2 Topics and Queues .................................................. 7
2.2.1 Trash Topics ................................................... 8
2.2.2 Login Topics .................................................. 8
2.3 Server Domains .................................................... 9
2.4 Messages .......................................................... 10
3. Publisher and Subscriber Identification ............................... 11
4. The SMQP Specifications ................................................ 11
4.1 SMQP Commands .................................................... 11
4.1.1 LOGIN / PASSWORD ........................................... 12
4.1.2 COUNT .......................................................... 14
4.1.2.1 TOPIC ....................................................... 14
4.1.2.2 MESSAGE ................................................... 15
4.1.2.3 SUBSCRIBERS ............................................. 15
4.1.3 LIST ............................................................ 15
4.1.3.1 TOPIC ....................................................... 15
4.1.3.2 MESSAGE ................................................... 16
4.1.4 CREATE ........................................................ 16
4.1.5 REMOVE ........................................................ 17
4.1.5.1 TOPIC ....................................................... 17
4.1.6 SET ............................................................. 18
4.1.6.1 TOPIC ....................................................... 18
4.1.6.1.1 NAME ..................................................... 18
4.1.6.1.2 KEYWORD ............................................... 19
4.1.6.2 LOGIN ....................................................... 19
4.1.6.2.1 PASSWORD ............................................... 19
4.1.6.2.2 NAME ..................................................... 19
4.1.6.2.3 EMAIL .................................................... 20
4.1.6.2.4 ADDRESS ............................................... 20
4.1.6.2.5 PHONE ................................................... 20
4.1.6.3 NOTIFY ....................................................... 20
4.1.6.3.1 PORT ..................................................... 20
4.1.6.3.2 SUSPEND ............................................... 21
4.1.6.3.3 RESUME ............................................... 21
4.1.7 FIND ............................................................ 21
4.1.7.1 TOPIC ....................................................... 21
4.1.8 PUBLISH ........................................................ 22
4.1.8.1 MESSAGE ................................................... 22
4.1.8.2 REPLY ....................................................... 27
4.1.9 REPUBLISH ....................................................... 28
4.1.9.1 MESSAGE ................................................... 28
4.1.10 SUBSCRIBE ..................................................... 28
4.1.10.1 MESSAGE ................................................... 28
4.1.10.2 TOPIC ....................................................... 30
4.1.10.3 SUBSCRIBER ............................................. 30
4.1.10.4 SERVER ................................................... 30
4.1.10.4.1 TIMEOUT ................................................. 30
4.1.11 UNSUBSCRIBE ................................................... 31
4.1.11.1 MESSAGE ................................................... 31
4.1.11.2 TOPIC ....................................................... 32
4.1.11.3 SUBSCRIBER ............................................. 32
4.1.12 NOTIFY ........................................................ 33
4.1.12.1 MESSAGE ................................................... 33
1.0 Introduction

1.1 Overview of SMQP Functionality

This document specifies the Simple Message Queue Protocol (SMQP) for use of exchanging messages from publishers of data to subscribers of data. Many Internet applications use publish and subscribe architecture but they either use separate Internet Protocols or proprietary protocols. Proprietary protocols include those with proprietary implementations. The market leaders for publish and subscribe applications include MQSeries (R) from IBM and MSMQ (R) from Microsoft. SMQP will allow the asynchronous sending of messages from publishers of data, to subscribers of data independent from implementation, OS platform, vendor, and computer language.

Message queuing allows for a message to be pushed out to a network and reside there until at least one subscriber of that message receives it. SMQP will allow for mission critical message flow to be achieved in near real-time. Examples of publish and subscribe mechanisms include financial transactions (stock quotes and trades), mail, chat rooms, file sharing/transferring, paging, instant messaging, news forums, system alerts, database transactions, and list forums. Wherever data needs to be shared between any number of sources that provide the data and any number of recipients that want to receive that data, SMQP may be used. A single transaction protocol can be used where currently many different protocols are current being used (both open and proprietary).

1.2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].
This document used Backus-Naur Form (BNF) nomenclature [RFC822]. The augmented BNF has the following constructs:

name = definition

The name of a rule is simply the name itself (without any enclosing "<" and ">") and is separated from its definition by the equal "=" character. White space is only significant in that indentation of continuous lines is used to indicate a rule definition that spans more than one line. Certain basic rules are uppercase as SP, LWS, HT, CRLF, DIGIT, and ALPHA, etc. Angle brackets are used within definition whenever their presence will facilitate discerning the use of rule names.

"literal"
Quotation marks surround literal text. Unless otherwise, the text is case-insensitive.

rule1 | rule2

Elements separated by a bar ("|") are alternatives, e.g. "yes | no" will accept yes or no.

(rule1 | rule2)
Elements enclosed in parentheses are treated as a single element. Thus, "(elem ( foo | bar ) elem )" allows for tokens sequences "elem foo elem" and "elem bar elem".

*rule
The character "*" preceding an element indicates repetition. The full form is "<n>*<m>element" indicating at least <n> and at most <m> occurrences of element. Default values are 0 and infinity so that "*(element)" allows any number, including zero; "1*element" requires at least one; and "1*2element" allows for one or two.

[rule]
Square brackets enclose optional elements; "[foo bar]" is equivalent to "1(foo bar)".

N rule
Specific repetition: "<n>{element}" is equivalent to "<n>*<n>{element}"; that is exactly <n> occurrences of (element).
Thus 2DIGIT is a 2-digit number, and 3ALPHA is a string of three alphabetic characters.

#rule
A construct of "#" is defined, similar to "*", for defining lists of elements. The full form is "<n>#<m>element" indicating at least <n> and at most <m> elements, each separated by one or more commas (","), and OPTIONAL linear white space (LWS). This makes usual form of list very easy; a rule such as ( *LWS element *(LWS "," *LWS element ) ) can shown as 1#element. Whenever this construct is used, null elements are allowed, but do not contribute to the count of elements present. That is, "(element), (element)" is permitted, but counts as only two elements. Therefore, where at least one element is required, at least one non-null element MUST be present. Default values are 0 and infinity so that "#element" allows any number, including zero; "1#element" requires at lease one; and
"1#2element" allows one or two.

; comment
A semi-colon, set off some distance to the right of rule text, starts a comment that continues to the end of the line.

implied *LWS
The grammar described by this specification is word-based. Except where noted otherwise, linear white space (LWS) can be included between any two adjacent words (token or quoted-string), and between words and separators, without changing the interpretation of a field. At least one delimiter (LWS and/or separator) MUST exist between any two tokens, since they would otherwise be interpreted as a single token.

2. Architecture of SMQP
SMQP focuses on the distribution of messages between publishers and subscribers. SMQP is based upon client-server architecture. A client is either a publisher of messages, a subscriber of messages, or both. The server receives published messages and then brokers the messages to subscribers based upon a topic context. The server MAY also be a client to another SMQP server by publishing a message from its cluster of clients to the cluster of clients of another SMQP server. A client must authenticate itself with each server it wishes to publish or subscribe messages with. Here is an example of a publish/subscribe system.

```
[ Publisher : Stocks ]  [ Publisher : Chat ]  [ Publisher : News ]
|                       |                     |
+-----------------------+---------------------+
                  | [ Broker ]
+-----------------------+---------------------+
                   |                     |
[ Subscriber : Chat ]            [ Subscriber : News ]
```

Messages are categorized by a topic. A single message queue on the server is assigned to a single topic. Topics and queues can be considered one in the same.

The SMQP server controls access permissions of a user to publish or subscribe to a topic. A request to publish or subscribed by a client SHALL be rejected by the server.

Messages are categorized by a topic. A single message queue on the server is assigned to a single topic. Topics and queues can be considered one in the same.

The SMQP server controls access permissions of a user to publish or subscribe to a topic. A request to publish or subscribed by a client SHALL be rejected by the server.

The client, by default will listen for TCP notifications on port 721. The client, by default, will connect to the SMQP server of TCP port 720.
2.1 Goal of the Architecture

The goal of the SMTP architecture is to provide a simple to use, publish and subscribe messaging protocol. This is achieved by:

1) Messages are transaction based
2) Messages are persisted before forwarded to the subscriber
3) Asynchronous delivery of messages
4) Subscribers received a message via a two-phase commit process
5) Any number of publishers can publish messages
6) Any number of subscribers can receive messages
7) Messages MAY contain any and multiple type(s) of data based on MIME specifications.
8) Once only delivery of messages
9) Fault tolerate behavior

2.2 Topics and Queues

Topics are a hierarchal description of a message between a publisher and a subscriber. The topic is independent of the format of the message data. This allows for a topic to contain any number of message types. Topics are:

1) Hierarchical
2) Case insensitive
3) UTF-8 character set
4) Contain no spaces
5) Cannot start with a number
6) Cannot contain non-alphanumeric characters
7) Always separated by a forward slash ("/")
8) Root hierarchy begins with a forward slash

The hierarchical topics allow topics to relate to one another. Defining the topics this way makes it easier to traverse and find other topics that might be of interest to a client. The hierarchical topics are very similar to a file system directory structure. There is no root topic and all first level topics are assumed to start with the forward slash ("/"). A sample topic hierarchical MAY look something like:

```
++ chat ----+ internet
  |   |
  |   +-- intranet

/ --+ news ----+ us
  |   |
  |   +-- europe
  |       |
  |   +-- asia

++ stocks +++ quotes
  |
  ++ charts
```

Each of the end topics is considered to be a "leaf" topic. All leaf topics have parent topics, unless the topic is at the first level of the hierarchy. Topics, like chat shown above, have two child topics (internet and intranet).

There is no restriction in the architecture that limits the depth
and width of the topics. The only restrictions that MAY be put into
place would be the particular SMQP server implementation.

Topics SHALL be placed in queues. The server MAY implement queues
as necessary, but it is easy to think of queues as the same thing as
topics and could have the same structure and context as the topic.

If a particular topic does not contain permission for a particular
user, then the topic will use the permissions defined by the parent
topic. This would progress until the root is reached. Newly
created topics inherit the permissions of the parent topic.

Other characteristics of a topic/queue are:

1) Messages are processed First In-First Out (FIFO). Higher
priority messages MAY be moved forward in the queue based
upon the server’s implementation. For example, a server MAY
ignore message priority or perform some heuristics on
messages with priorities based on the publisher. This would
allow a fair weighting of message priority in the queue.

2) Topics can have keywords associated with it. These key words
MAY be used to query the server for a topic based on some
combination of these keywords.

3) Topics can be virtually represented on the SMQP server. A
virtual topic links to an actual topic. Virtual topics behave
just like actual topics. Virtual topics MAY be used for
variations of topic structure or allow an older topic to be
transitioned to a newer topic without publishers and
subscribers failing because of a re-organization of topics.
Virtual topics SHALL only map to actual topics and cannot
link to them. Any number of virtual topics can be linked to
the same actual topic.

A topic of /weather/us/ca/san/forecast MAY contain many types of
published messages including text that describes the temperature,
humidity, rainfall, and wind; as well as a JPEG image of the
satellite image of the region. Some level of standardization would
need to be done so that subscribers can find the topic path it needs
to subscribe to. A client can request the currently available
topics that are available on the server.

2.2.1. Trash Topics
Each topic contains a child topic with the name "Trash".
"Trash" is a reserved topic name (case insensitive) that MUST
NOT be used in the hierarchy of topics. The Trash topic/queue
is a repository of messages for the parent topic that contains
dead messages. Dead messages are any message that is not still
pending in the parent queue. This includes messages that were
unable to be delivered to subscribers after a certain number of
attempts. Once a message in the parent queue has been
transmitted to all subscribers or other thresholds have been
reached that the message is not needed to stay in the queue,
the message is moved to the child Trash queue. Messages in the
Trash queue will remain there until the server periodically
empties the messages in a particular topic’s trash or all the
topics trash. Messages in the "Trash" queue can be used for
auditing as well as re-submission of a message.
2.2.2. Login Topics
Each time a login account is created, a topic/queue is created for that account. These topics are used for reply messages that may be generated by a subscriber.

2.3. Server Domains
Each SMQP server SHALL resides in a domain. Each server has its own unique context that is usually the domain name of the server, or its static IP address, along with an OPTIONAL context. For example:

www.domain.com/orange

The domain name SHOULD be used for the server context due to physical changes of hardware and to keep the server’s location transparent to the clients and other SMQP servers. Multiple SMQP servers MAY reside on a single physical machine as long as each server context is unique. Each unique server SHALL have physically distinct configuration and topic structure. The configuration and topic structure MAY appear to be the same, but are not physically the same. A SMQP server can be configured to link with another SMQP server. Clients based on the same domain host name, but unique port numbers connect to multiple SMQP servers on the same host. Published messages from one SMQP domain will be forwarded to any other registered domains that have at least one subscriber for that message. Messages that are sent to the immediate SMQP server are considered "local" messages. Messages that are passed to other SMQP servers, by the local server, will then be considered "remote" messages. A message being published MAY define the depth message is allowed to be forwarded. By default, the depth is only to the local server.

Messages that are passed to other domains will have their topic prepended with the server’s domain.

Messages that are routed between servers are verified before delivery to ensure the message has not already passed to that server via another route path.

Servers connecting with another server are authenticated like any other client application connecting to a server. The server accepting the connection determines the authenticity and permissions of the adjoining server. Since the server does not contain user authentication for each of the accounts of the requesting server, it MUST assume it to be an un-trusted environment. The administrator of the server MUST determine which topics the requesting server has access to. It has to assume that any user access granted from the requesting server has permission to access the topics that it allows for the requesting server. User Access List (UAL) is also defined by a server of which remote servers an account as permission to access.
2.4. Messages

A message is a container of data, with a header that is passed from a publisher (that created the data), through the SMQP server that brokers (routes) the message to subscribed clients.

```
[ Publisher ] ----[X]-->[ Broker ]-------->[ Subscriber ]
    \                                  ^
     message                          +------------>[ Subscriber ]
```

Other than the header, the message does not concern itself with the contents of the data except that it knows the messages’ length and associated MIME type that is used by the subscriber to understand its contents. Messages MAY have multiple parts to the data section of the message. Each part MUST have an associated MIME type and length.

Messages persist with the SMQP server until some period has expired. Subscribers that are no longer logged into the SMQP server and did not unsubscribe to the topic will receive pending messages that were published since the last time the subscriber logged in, as long as the messages did not expire.

Most publish/subscribe messages are event based. When something changes in the publisher, it would publish an event-based message. For example, stock quote change, stock purchase, news item, et cetera. The other type of message is state based. State based messages persist longer than an event-based message. The only difference that separates these two types of message is the length of the messages timeout. The message timeout informs the server how long a message will wait around for a subscriber to receive it. The timeout is the life span of the message. For example, a message with a timeout of 1 minute will persist on the server for 1 minute after all the current subscribers have been informed. If a new client subscribes to that topic within that 1 minute, it will also receive that message. Clients that subscribe after the 1 minute will not receive the message nor would they know that the message ever existed. If a publisher publishes message for a chat topic, they MAY set the timeout for 48 hours, long enough time for client subscribers to get recent chat discussions. A publisher for a list or news forum MAY set the timeout to be very infinite, so any new client subscriber will always receive all the messages in that topic. While a client is subscribing to a topic, they can OPTIONALLY specify a timeout period that would limit which messages are then transmitted to them, based on server’s received date/time of the message. Publishers of state based messages SHALL be responsible for deleting those messages before publishing a new state message.

Messages are delivered to subscribers in the order that the publishing clients sent them (FIFO). However, it is possible that messages MAY be sent out of order due to network latency, server outages, resources on a topic's queue has exceeding so messages are held out until resources are balanced again, or administration of server causes later messages to slip by earlier messages.
for each message. The server does nothing with the sequence number. It SHALL be the subscriber’s responsibility to check and manage messages that MAY have been received in a different order than the sequence number indicates.

3. Publisher and Subscriber Identification
Publishers and subscribers are identified by the login account in addition to some global unique identifier (GUID). The account name, plus GUID will then allow the same account to be logged into multiple times to the same server. The server SHALL enforce unique login accounts. If a user is logged into the server more than once, each site will receive the same published messages.

4. The SMQP Specifications
Once a client connects to the SMQP server via TCP/IP connection, the server SHALL respond to the client with the SMQP identifier, version, and implementation identification; ending with <CRLF>. For example,

```
SMQP/1.0 Ready. OmicronSoft SMQP Server/1.0 (c) 2001
```

The server is REQUIRED to return this identification when a new client connects to it.

```
"SMQP" "/* version [ 1*SP status "." [ 1*SP server-ident ] ]

version = 1*DIGIT "." 1*DIGIT ; <major>.<minor>
status = ( "Ready" | "Not Available" )
```

The client MAY ignore anything after the "SMQP/1.0". The client to identify the protocol to use and the version of the protocol that the server is supporting uses this.

After the server identification, the client may interact with the server by issuing commands, along with server data, and then wait for a reply from the server.

4.1. SMQP Command
The SMQP commands define the message transaction of the user or client application. SMQP commands MUST be character strings that terminate by <CRLF>. The commands themselves are alphabetic characters terminated by <SP> if parameters follow and <CRLF> otherwise.

Various examples are provided in this section. Each command or response is prefixed to designate who is sending or receiving the information. "C:" designates that the client is issuing the command or response. "S:" designates that the server is issuing the command or response.

Many of the commands can be abbreviated to shorten the number of bytes that are transmitted and received. Abbreviated commands are OPTIONAL. Clients and server implementations SHALL NOT introduce
alternative abbreviations and MUST support the receiving of the abbreviated commands.

4.1.1.  LOGIN

"LOGIN" 1*SP username 1*SP authentication-method "/" version authentication-method is the method the client wishes to encrypt the password with the server. This may be a series of interactions with the server that may include determining which authentication method is supported by the server and/or a series of steps need as part of the selected authentication method. Example authentication methods include "CLEAR", "KERBEROS", and "MD5". Minimally, all servers SHALL implement "CLEAR" authentication, which is the client's password in the clear (not encrypted). It is up to the client to determine the method of authentication.

CLEAR authentication example:
C: LOGIN jtegen CLEAR/1.0
S: 200 OK
C: PASS jtegen mypasswd
S: 200 OK

The version for CLEAR method is meaningless but SHOULD be provided to the server for consistency.

KERBEROS authentication example (not supported by server):
C: LOGIN jtegen KERBEROS/4.0
S: 405 Not allowed

KERBEROS authentication example (supported by server):
C: LOGIN jtegen KERBEROS/4.0
S: 200-OK
S: 200 + AmFYig==
C: BAcAQU5EUkVXLkDFeFDERFCEFEFe/RETYEFE...
S: + or//EoAADZI
C: DiAF5A4gA+oIALuBKAAmw==
S: 200 OK

Client application SHALL log into the server. Upon successful authentication, the client application SHALL then be allowed to publish and subscribe to messages.

The server will OPTIONALLY respond with default values that it has currently set that the client MAY use in sending and receiving messages. The default state MAY change during the runtime period of a client. The client has the option to subscribe to state changes of the server.

Reply Codes:
200       OK
400       Bad Request
401       Unauthorized (password not correct)
404       Not found (account not found)
405       Not allowed (authentication method)
Each authentication method applies different rules to validate the client login process. Minimally, both client and server implementations SHALL include the "CLEAR" method. However, it is still up to the server’s implementation and rules that it MAY apply to its installation and login accounts. When the CLEAR method is used between the client and server, the client SHALL respond to the server with the PASS command. The PASSWORD command only applies to the CLEAR authentication-method.

```
PASSWORD = ( "PASSWORD" | "PASS" ) 1*SP account 1*SP password
```

Each login method needs to inform the server, which accounts the password verification, is meant for. The server is processing other LOGIN requests during the period of time a client is responding to their LOGIN request.

Following the return code, the server is REQUIRED to return the SMQP protocol version that it supports and an OPTIONAL host description.

The server SHALL return the REQUIRED name value attributes, and OPTIONALLY the OPTIONAL attributes. Attributes are a name, followed by a value, separated by a colon (:).

```
TOPIC = "Topic" ":" topic

REQUIRED topic of the user’s account. The client application SHOULD subscribe to this topic so it MAY receive system and reply messages.

TIMEOUT = "Timeout" ":" timeout

OPTIONAL server default message timeout. Messages that do not specify a timeout will default to the server’s timeout period. If the server does not return a TIMEOUT attribute on login, it is RECOMMENDED that the client sets a timeout period for all messages.

TIME = "Time" ":" date-time

The REQUIRED TIME is the current date/time of the server. This can be used to correlate messages published and received to and from the server. The client SHOULD assume some measure of latency in the actual time of the server, and the time it took to deliver the reply. Time is delivered as GMT.

GUID = "Guid" ":" guid

The server that describes that instance of the account login returns a global unique identifier. This allows the same login identification be logged into the server multiple times. The generation of the GUID is server implementation dependent as long it is unique for each logged in instance of the account.

Example LOGIN sequence:

```
C: LOGIN jtegen CLEAR/1.0
S: 200 OK
C: PASS jtegen rowe7kuy
```
4.1.2. COUNT

"COUNT" 1*SP resource 1*SP item

COUNT returns the quantity of a given resource. Both the <resource> and <topic> are REQUIRED fields. For all COUNT commands, a wildcard can be used for the leaf topic if the client commands the server to traverse all child topics of the parent topic, regardless of depth.

Reply codes:

200   OK, success
400   Bad request
401   Unauthorized
404   Not found (item not found)

4.1.2.1 TOPIC

"COUNT" 1*SP "TOPIC" 1*SP topic

COUNT TOPIC returns the quantity of child topics of a given parent topic.

Example COUNT TOPIC:

C: COUNT TOPIC /news
C:
S: 200-OK
S: 200 2
S:

4.1.2.2 MESSAGE

"COUNT" 1*SP ("MESSAGE" | "MESS") 1*SP topic

The COUNT command can be used to determine the number of messages currently retained by the SMQP server at the instance the request was made. This is not totally accurate, since the volume of message passing through a queue could be very large and the rate in which they are being processed MAY lag the value being returned.

Example COUNT MESSAGE sequence:

C: COUNT MESSAGE /stock/quote/ibm
C:
4.1.2.3 SUBSCRIBERS

"COUNT" 1*SP ( "SUBSCRIBERS" | "SUB" ) 1*SP topic

COUNT SUBSCRIBERS command returns the number of currently subscribed clients to a particular topic.

Example COUNT SUBSCRIBERS sequence:

C: COUNT SUB /stock/quote/ibm
C:
S: 200-OK
S: 200 12
S:

4.1.3. LIST

"LIST" 1*SP resource 1*SP item

LIST command allows a client to list various items on the server. The server controls access for what a particular login has access to. Though a particular item MAY exist on the server, the LIST command will not return its existence if the login does not have permission to access it.

Reply Codes:
200 OK, success
400 Bad request
401 Unauthorized
404 Not found (item not found)

4.1.3.1. TOPIC

"LIST" 1*SP "TOPIC" 1*SP topic

LIST TOPIC returns a list of topics underneath the provided parent topic. All child topics of the given topic will be returned by the server. Each child topic will be in its own line of the return stream, followed by a <CRLF>. The parent-topic parameter is REQUIRED.

Example LIST TOPIC sequence:

C: LIST TOPIC /
C:
S: 200-OK
S: 200-/stocks
S: 200-/info
S: 200-/weather
S:
The last returned topic displays a virtual topic that is actually linked to another topic.

4.1.3.2. MESSAGE

"LIST" 1*SP ( "MESSAGE" | "MESS" ) 1*SP topic

LIST MESSAGE returns a list of message identifiers currently in the given topic. This is only a snap shot of the current messages, since those messages MAY change in the next fraction of a second. The topic parameter is REQUIRED.

Example LIST MESSAGE sequence:

C: LIST MESSAGES /stocks/quotes/ibm
C:
S: 200-OK
S: 200 1809-1812,1845,1850-1890
S:

4.1.4. CREATE

"CREATE" 1*SP "TOPIC" 1*SP topic [ 1*SP "(" keyword "," ... ")" ]
[ 1*SP ">" 1*SP topic ]

CREATE TOPIC creates a new topic by the given topic path. If the topic already exists, the server will return a reply code that signifies this. Topics are case insensitive.

Server SHALL disallow the creation of a topic by the same path of a user’s account reply topics.

Reply codes:

200        OK, success
400        Bad request
401        Unauthorized
404        Not found (destination-topic not found)
405        Not allowed (topic link bad)
409        Conflict (topic and destination-topic the same)
414        Resource too large (too many topic for parent topic)

Example CREATE TOPIC sequence:

C: CREATE TOPIC /stocks/quotes/ibm
C:
S: 200 OK
S:

CREATE TOPIC command can also create a virtual topic by defining what the new topic SHOULD map to. The topic being mapped to MUST already exist.

Example CREATE TOPIC sequence for virtual topics:

C: CREATE TOPIC /stocks/quotes/ibm > /stocks/ibm/quote
C:
The creation of the topic can also add OPTIONAL keywords to the topic, allowing another client to query a topic based on these keywords. The topic keywords are unstructured. Keywords are OPTIONALLY added as the last parameter in the list. A comma separates each keyword. Keywords cannot contain the following characters: "/", "\\", ",", "(" and ")". Keywords are case insensitive.

Example CREATE TOPIC sequence with keywords:

C: CREATE TOPIC /stocks/ibm (IBM,STOCKS,QUOTE,BIG BLUE)
C:
S: 200 OK
S:

Example CREATE TOPIC sequence for a virtual topic, with keywords:

C: CREATE TOPIC /stocks/ibm (BIG BLUE) > /stocks/quotes/ibm
C:
S: 200 OK
S:

4.2.5. REMOVE

( "REMOVE" | "REM" ) 1*SP "TOPIC" 1*SP topic

Topics can be removed by the server, but will only be removed when the last message has been delivered to clients that MAY have subscribed to that topic. Any new, published messages to that topic will return an error as if the topic no longer exists. Only authorized clients have permission to delete a topic. The topic parameter is REQUIRED.

Reply Codes:
200 OK, success
400 Bad request
401 Unauthorized
404 Not found (topic)

Example REMOVE TOPIC sequence:
C: REMOVE TOPIC /stocks/quotes/ibm
C:
S: 200-OK
S: 200 12 messages pending
S:

4.1.6. SET

SET commands allows the client to set values of various resources. SET command allows the client to possibly modify the current value of one of these resources.

4.1.6.1. SET TOPIC

SET TOPIC family of commands allows the client to set various
properties of a topic/queue.

4.1.6.1. SET TOPIC NAME

"SET" 1*SP "TOPIC" 1*SP "NAME" 1*SP topic 1*SP new-topic

A client can set a topic’s name. Renaming a topic does not affect any pending messages in the queue. However, any new messages published to the topic will return an error to the publisher as if the topic no longer exists. The new topic would be available for publishers to send messages to. A topic cannot be renamed to itself or to a topic that already exists. Both the topic and new-topic parameters are REQUIRED.

Account topics SHALL NOT be allowed to be removed by this command.

Reply Codes:
200   OK, success
400   Bad request
401   Unauthorized
404   Not found (topic)
409   Conflict (new-topic and topic are the same)

Example SET TOPIC NAME sequence:
C: SET TOPIC NAME /stocks/quotes/ibm /stocks/ibm/quote
C:
S: 200 OK

4.1.6.1.2. SET TOPIC KEYWORD

"SET" 1*SP "TOPIC" 1*SP "KEYWORD" 1*SP topic 1*SP keywords

A client can set topics keywords. Any existing keywords for the designated topic will be replaced by the new values.

4.1.6.3. SET LOGIN

SET LOGIN command allows the client to set various properties of their user login account.

"SET" 1*SP "LOGIN" 1*SP ( "PASSWORD" | "NAME" | "EMAIL" | "ADDRESS" | "PHONE" ) ...
As with the LOGIN command, the SET LOGIN PASSWORD needs to negotiate with the server on the method of authentication. The server SHALL accept or deny the authentication method suggested by the client. In general, the client SHOULD use the same authentication method used during the LOGIN. Once the server accepts the authentication method, the client would pass the old password and then the new password.

For example:
C: SET LOGIN PASSWORD CLEAR/1.0
S: 405 Not allowed
C: SET LOGIN PASSWORD KABEROS/4.0
S: 200-OK
S: 200 + AmFYig==
C: BAcAQU5EUkVXLkDFeFDERFCEFEFee/RETYEFE...
S: + or//EoAADZI
C: DiAF5A4gA+oIALuBKAAmw==
S: AmFYig==
C: BAcAQU5eUkVXLkDF6DuRFyFEFee/RETYiFE...
S: + or//EoijArZI
C: pOAF5A4UA+oI6LuBKAimw==
S: 200 OK

4.1.6.3.2. SET LOGIN NAME

"SET" 1*SP "LOGIN" 1*SP "NAME" 1*SP name

SET LOGIN NAME allows the client to change the owner’s name of the current logged in account. This is the proper name of the account, for example, "Jayne Smith".

4.1.6.3.3. SET LOGIN EMAIL

"SET" 1*SP "LOGIN" 1*SP "EMAIL" 1*SP address

SET LOGIN EMAIL allows the client to change the owner’s e-mail account of the current logged in account.

4.1.6.3.4. SET LOGIN ADDRESS

"SET" 1*SP "LOGIN" 1*SP "ADDRESS" 1*SP address

SET LOGIN ADDRESS allows the client to change the owner’s ground mail address of the current logged in account.

4.1.6.3.5. SET LOGIN PHONE

"SET" 1*SP "LOGIN" 1*SP "PHONE" 1*SP phone-number

SET LOGIN PHONE allows the client to change the owner’s telephone number of the current logged in account.
4.1.6.5. SET NOTIFY

"SET" 1*SP "NOTIFY" 1*SP ( "PORT" | "SUSPEND" | "RESUME" )

SET NOTIFY commands allows the client to inform the server how it wants to be notified of published messages. The SET NOTIFY command affects all subscribed messages to the client.

4.1.6.5.1. SET NOTIFY PORT <port-number>

"SET" 1*SP "NOTIFY" 1*SP "PORT" 1*SP port-number

SET NOTIFY PORT informs the server that the client is now listening for publications on a different port number. The server SHALL discontinue any connections to the existing client port and re-connect to the new port number. The client SHALL begin listening to messages to the port-number prior to sending this request to the server, since the server SHALL attempt to connect to this port before replying. By default, a client begins listening on default-client-port.

Reply Codes:
200    OK, success
400    Bad request
401    Unauthorized
404    Not found (port)
408    Request timeout

4.1.6.5.2. SET NOTIFY SUSPEND

"SET" 1*SP "NOTIFY" 1*SP ( "SUSPEND" | "SUS" )

SET NOTIFY SUSPEND informs the server to suspend sending any publications that the client may have subscribed to. Messages will not be published to the client until a RESUME message is sent to the server.

4.1.6.5.3. SET NOTIFY RESUME

"SET" 1*SP "NOTIFY" 1*SP ( "RESUME" | "RES" )

SET NOTIFY RESUME informs the server to resume sending any publications that the client may have subscribed to. This is normally issued after a SUSPEND request. The server SHALL ignore the RESUME request if no previous SUSPEND request was issued by the client.

4.1.7. FIND

"FIND" 1*SP "TOPIC" 1*SP topic 1*SP keywords

FIND TOPIC will return the topic(s) specified in the REQUIRED
topic parameter. The return will provide any keywords associated with the topic, encased in "(" and ")". The topic parameter is REQUIRED. The server SHALL return a list of topics that match the search.

Reply Codes:
200  OK, success
400  Bad request
401  Unauthorized
404  Not found (topic)
405  Not allowed

Example FIND TOPIC sequence:

C: FIND TOPIC /stocks
C: S: 200-OK

This example can be used to verify an existence of a topic and return any keywords that MAY have been assigned to it.

FIND TOPIC can be used to find those topics that match a keyword search, from a given parent topic. Keyword parameter is OPTIONAL and case insensitive.

Example FIND TOPIC by keyword sequence:

C: FIND TOPIC /stocks (BLUE)
C: S: 200-OK
S: 200 /stocks/ibm (IBM,STOCKS,BIG BLUE)

Wildcards can be used on the FIND command, in combination of the keyword search. The following example will search for all child topics underneath /stocks, that contains any portion of the keyword "BLUE".

C: FIND TOPICS /stocks/* (BLUE)
C: S: 200-OK
S: 200-/stocks/ibm (IBM,STOCKS,BIG BLUE)
S: 200 /stocks/floral (FLOWERS,BLUE CARNATIONS)
S: 

4.1.8.  PUBLISH

4.1.8.1. PUBLISH MESSAGE

( "PUBLISH" | "PUB" ) 1*SP ( "MESSAGE" | "MESS" ) 1*SP topic
1*SP cmuid

The PUBLISH command publishes a message to a REQUIRED topic, by a REQUIRED client message unique identifier (cmuid). The
message comprised of two sections; the header and OPTIONAL data. Any subscriber, subscribed to that topic, will be forwarded the publishers message. Publishing a message to the SMQP server is a single-phase transaction. If the server does not respond with an affirmative reply, the publisher has to assume the message did not get published.

The published message has a list of attributes that are defined in the header of the message.

**NAME** = "Name" "::" text-UTF8

A message may have a context. This can be viewed as the subject of the message. The name does not have to be unique in a given topic. The name is free formed but shall be limited to one line of no more than 128 characters. The NAME attribute is OPTIONAL.

**PRIORITY** = ( "Priority" | "Pri" ) "::" positive-number

PRIORITY is a numeric value greater than zero that can OPTIONALLY define the priority of the message in the topic’s queue. Messages are normally processed FIFO in a topic’s queue. Messages that define a priority are organized so that lower priority numbers are sent before higher priority numbers (priority of 2 is sent before a priority of 10).

**CREATED** = "Created" "::" date-time

CREATED is the REQUIRED date/time the message was created, just prior to being streamed to the server. The date/time is REQUIRED to be based on GMT (Zulu) zone and seconds are expressed in fractions of a second. The CREATED date/time MAY be used by the subscriber to verify the ordering of messages received.

**RECEIPT** = "Receipt" "::" boolean

RECEIPT is an OPTIONAL, Boolean attribute that informs the server to return a message to the publisher that the message was successfully delivered to at least one subscriber. The
values of the attribute are either YES or NO. By default, the server SHOULD assume no receipt is needed if the attribute is not provided in the header. Recipient messages are published to the login account’s system topic/queue.

REPLY = "Reply" "::" boolean
REPLY is an OPTIONAL, Boolean attribute that informs the subscriber that the publisher is expecting some kind of reply message upon successful receipt of the published message. If the REPLY is set to YES, the subscriber SHOULD reply to the message appropriately. If the attribute is missing from the header, the subscriber MUST assume that no reply is needed. Reply messages are published to the login account’s system topic/queue.

TIMEOUT = "Timeout" "::" timeout
TIMEOUT is the OPTIONAL period of time a message will persist on the server after the current subscribers have been sent the published message. If the attribute is missing from the header, the server will default to the timeout period defined by the topic’s queue. Timeout of a negative value (<0) indicates a state message and the message will persist for an infinite period of time until the message is REMOVED.

DEPTH = "Depth" "::" number
DEPTH is an OPTIONAL, numeric value that determines how deep a published message will be routed to other SMQP servers. A depth of zero (0) will only route a message to local subscribers of the server. Depths greater than zero will allow the message to be routed to adjoining servers to the given depth, at which point, the message will no longer be routed. A depth of (-1) assumes no limit to the server depth. If the attribute is not included in the header, the server MUST assume a depth of zero, or only those subscribers on the local server.

SEQUENCE = ( "Sequence" | "Seq" ) "::" positive-number
SEQUENCE is an OPTIONAL, positive number that assists the subscriber in knowing the sequence of a series of messages from a particular publisher. If a publisher needs to ensure that published messages are received in a particular order, the publisher would increment the SEQUENCE value by one each time a new message is generated. It is the responsibility of the subscriber to cache messages that MAY have been received out of sequence. The subscriber MAY also be able to use the CREATED attribute to help determine message sequential latency at the source of the message.

Additional extended attributes can be defined in the header. These attributes are prefixed with "X-" that indicates the attribute is not a part of the SMQP specifications.

The last attribute in the header is followed by a <CRLF> before the data header begins.

MIME-VERSION = "MIME-Version" "::" major "." minor
MIME-VERSION is an OPTIONAL attribute that informs which
version of the MIME specification is being supported with the current message. The subscriber MAY use this information to help it understand the CONTENT-TYPE of the data.

CONTENT-TYPE = "Content-Type" "::" media-type
CONTENT-TYPE is a REQUIRED attribute after the header. There is a content type for each data section in the message. The content type uses the MIME specification that describes the format of the data to follow. The CONTENT-TYPE belongs to the header section of the data. It is preceded by <CRLF> that separates the header from the data section.

Example of this attribute is
Content-Type: application/x-rft
Content-Type: text/plain; charset="ISO-8859-4"

CONTENT-LENGTH = "Content-Length" "::" 1*DIGIT
The CONTENT-LENGTH is a REQUIRED attribute after the header. The content length is for each data section in the message. The content type is the number of bytes in the data to follow.

CONTENT-ENCODING = "Content-Encoding" "::" 1#content-coding
When present, its value indicates what additional content coding have been applied to the data, and thus what decoding mechanism must be applied in order to obtain the media-type referenced by the Content-Type. Content-Encoding is primarily used to allow a document to be compressed without losing the identity of its underlying media type.

Example of this attribute is
Content-Encoding: zip

ENCRYPTION = "Encryption" "::" encryption-scheme 1*SP
  encryption-scheme = token
  encryption-params = token "=" ( token | quoted-string )

Example of this attribute is
Encryption: PGP version=2.6.2,encoding=ascii

Additional extended attributes can be defined in the header of the data. These attributes are prefixed with "X-" that indicates the attribute is not a part of the SMQP specifications.

The last attribute in the data header is followed by a <CRLF> before the data begins.

The server acknowledges receipt of the published message after the reply code. The acknowledge command is followed by the CMUID that was in the header of the message, along with a server generated message unique identification (SMUID). The SMUID can be used to perform additional commands on the message that now resides on the server. The CMUID is returned so the publisher can be certain to know which response maps to which generated message.
Example PUBLISH MESSAGE sequence:

C: PUB MESSAGE /stocks/quotes/ibm A004
C: Priority: 1
C: Created: D08152001T08341245Z
C: Receipt: yes
C: Name: Stock quote for IBM
C: Reply: NO
C: Timeout: 00:01:00
C: Depth: 1
C: Sequence: 34
C:
C: Content-Type: text/plain
C: Content-Length: 5
C:
C: 24.567
C:
C: 
C:
S: 200-OK
S: 200 A004 1809
S:

A message can have multiple data sections in one message. Examples of this MAY include plain and HTML text, or weather description along with a GIF satellite image.

Example PUB MESSAGE with multiple data sections sequence:

C: PUB MESS /stocks/quotes/ibm A005
C: Priority: 1
C: Name: IBM Quote
C: Created: D08152001T08341245Z
C: Receipt: yes
C: Reply: NO
C: Timeout: 00:01:00
C: Depth: 1
C: Sequence: 34
C:
C: Content-Type: text/plain
C: Content-Length: 5
C:
C: 24.567
C:
C: Content-Type: text/html
C: Content-Length: 52
C:
C: <A HREF="http://www.stocks.com/quote=ibm">24.567</A>
C:
C:
S: 200-OK
S: 200 A005 1810
S:
4.1.8.2. PUBLISH REPLY

( "PUBLISH" | "PUB" ) 1*SP "REPLY" 1*SP topic 1*SP cmuid

A client has the option to reply to a published message without having to know who the publisher is. The command would issue a message to the topic the message was received from as well as the Client Message Unique Identification (CMUID) for the client replying to the message. This is not the same CMUID that is in the attribute list of the original message.

For example, if a subscriber received and performed the following sequence:

S: NOTIFY MESSAGE /stocks/quotes/ibm
S: Created: D08152001T08032445Z
S: Smuid: www.host.com/1835
S: Cmuid: jtegen/jtegen497549/A004
S: Content-Type: text/plain
S: Content-Length: 12
S:
S: 34.456
S: .
S:
C: 310 ACK
C:
S: 310 ACK
S:

They would then be able to send a reply back to the publisher by:

C: PUB REPLY /stocks/quotes/ibm CF0456
C: Created: D08152001T08035341Z
C: Smuid: www.host.com/1835
C: Cmuid: jtegen/jtegen497549/A004
C: Priority: 1
C: Timeout: 00:01:00
C: Content-Type: text/plain
C: Content-Length: 12
C:
C: Thank you
C: .
C:
S: 200-OK
S: 200 CF0456 2345
S:

The CMUID field additionally indicates that this new message is a reply to a previously sent message. The server would then route the reply to the originating client publisher.

At this point, a two way communication between to end points can be achieved. The original publisher can now reply to the end point that sent the reply.
REPUBLISH allows a client to republish a previously sent and completed message. If the message has already been sent to all of the current subscribers or the message timeout was reached, and it is already in the TRASH queue, the message will be recovered from the TRASH and the message will be retransmitted to all current subscribers, even if they are different from the last transmission. Deleted messages cannot be republished and the server will return an error. Both the topic and smuid are REQUIRED.

Messages that are sequenced cannot be republished.

Reply Codes:
200   OK, success
400   Bad request
401   Unauthorized
404   Not found (topic)
406   Not acceptable (message not found)

C: REPUB MESS /stocks/quotes/ibm 1809
C:
S: 200 OK
S:

4.1.10.   SUBSCRIBE

( "SUBSCRIBE" | "SUB" ) 1*SP resource 1*SP topic

SUBSCRIBE commands subscribe to a resource notification. Once subscribed, the subscriber will be notified by the server when a change to the current subscription has occurred.

4.1.10.1.   MESSAGE <topic> <since-time>

( "SUBSCRIBE" | "SUB" ) 1*SP ("MESSAGE" | "MESS" )
1*SP topic
1*SP since-time

SUBSCRIBE MESSAGE allows a client to subscribe to message for a given, REQUIRED topic. Once subscribed, any message posted to that topic will be routed to the subscriber.

The OPTIONAL, since-time parameter allows the publisher to only subscribe to message since a given GMT time. This allows the user to filter out timeless messages that MAY have built upon over time. If the subscriber sets the since parameter to only a few seconds earlier then the present time, then the number of waiting messages would be reduced. Additionally, this parameter can be used to wait to receive messages until some time is reached in the future. For example, a subscriber MAY only wish to subscribe to weather
updates until after 0400 hours GMT.

The server replies with a code and then an acknowledgement of the subscription.

Reply Codes:
200   OK, success
400   Bad request
401   Unauthorized
404   Not found (topic)
414   Resource too large (cannot subscribe to any more)

Example SUBSCRIBE MESSAGE sequence:
C: SUB MESSAGE /stocks/quotes/ibm
C:
S: 200-OK
S: 200 /stocks/quotes/ibm
S:

At this point the client will need to wait to receive a message from the SMQP server.

A client can use wild cards to subscribe to all topics under a certain leaf. This will allow them to receive new messages of publications that have yet to be published. For example.

C: SUBSCRIBE MESSAGE /stocks/quotes/*
C:
S: 200-OK
S: 200-/stocks/quotes/ibm
S: 200-/stocks/quotes/msft
S: 200 /stocks/quotes/sony
S:

The server SHALL NOT allow a subscription of all messages at the root topic as shown here:

C: SUB MESSAGE /*
C:
S: 510 Maximum quantity exceeded
S:

Additionally, the server MAY implement other constraints to limit the number of subscriptions that can be requested at one time or entirely.

See NOTIFY MESSAGE for the type of notification received by the client when subscribed to this resource.

4.1.10.2.   TOPIC <topic>

( "SUBSCRIBE" | "SUB" ) 1*SP "TOPIC" 1*SP topic

A client MAY subscribe to changes that have occurred to a topic. Changes MAY include that a topic was added, renamed,
or removed.

C: SUBSCRIBE TOPIC /stocks/quotes/ibm
C:

The SUBSCRIBE TOPIC command MAY use a wildcard in the topic
to subscribe to any changes to the topic and its children.

C: SUB TOPIC /stocks/quotes/*
C:

See NOTIFY TOPIC for the type of notifications received by
the client when subscribed to this resource.

4.1.10.3. SUBSCRIBER <topic>

( "SUBSCRIBE" | "SUB" ) 1*SP ( "SUBSCRIBER" | "SUB" )
1*SP topic

A client MAY subscribe to changes with subscribers for a
given topic.

C: SUB SUB /stocks/quotes/ibm
C:

4.1.10.4. SERVER <attribute>

( "SUBSCRIBE" | "SUB" ) 1*SP "SERVER" 1*SP attribute

A client MAY subscribe to various attributes of the server.
Changes to the servers subscribed attribute will then notify
interested clients.

4.1.10.4.1. TIMEOUT

( "SUBSCRIBE" | "SUB" ) 1*SP "SERVER" 1*SP "TIMEOUT"
Default timeout of messages by the server.

By default, all clients are automatically subscribed to
SUBSCRIBE SERVER BYE command. SERVER BYE notifies
client when the server is shutting down.

4.1.11. UNSUBSCRIBE

( "UNSUBSCRIBE" | "UNSUB" ) 1*SP resource 1*SP topic

UNSUBSCRIBE commands unsubscribe previously subscribed resources
by the client. If the client does not unsubscribe before
terminating the session with the server, the SMQP server will
retain messages until the client logs into the server again.
Pending messages, that have not expired, will be delivered to the
client.
4.1.11.1. MESSAGE <topic>

( "UNSUBSCRIBE" | "UNSUB" ) 1*SP ( "MESSAGE" | "MESS" )
    1*SP topic

A client MAY unsubscribe to messages of a topic.

Reply codes:
200   OK, success
400   Bad request
404   Not found (topic)

C: UNSUB MESSAGE /stocks/quotes/ibm
C:
S: 200-OK
S: 200 MESSAGE /stocks/quotes/ibm
S:

The client can use wild cards to unsubscribe blocks of topics. For example, to unsubscribe to all topics a client would issue a command like

C: UNSUB MESSAGE *
C:
S: 200-OK
S: 200-MESSAGE /stocks/quotes/ibm
S: 200-MESSAGE /stocks/quotes/msft
S: 200 MESSAGE /stocks/quotes/sony
S:

The wildcard can be applied to any aspect of the topic hierarchy. For example if the client subscribed to the following:

C: SUB MESSAGE /stocks/quotes/ibm
C: SUB MESSAGE /stocks/quotes/sony
C: SUB MESSAGE /weather/us/ca/san/forecast
C:
S: 200-OK
S: 200-MESSAGE /stocks/quotes/ibm
S: 200-MESSAGE /stocks/quotes/sony
S: 200 MESSAGE /weather/us/ca/san/forecast
S:

The client could later unsubscribe to just the quote subscriptions as follows:

C: UNSUB MESSAGE /stocks/*
C:
S: 200-OK
S: 200-MESSAGE /stocks/quotes/ibm
S: 200 MESSAGE /stocks/quotes/sony
S:

4.1.11.2. TOPIC <topic>
A client MAY unsubscribe to changes to a topic.

Reply codes:
200   OK, success
400   Bad request
404   Not found (topic)

C: UNSUB TOPIC /stocks/quotes/ibm
C:
S: 200-OK
S: 200 TOPIC /stocks/quotes/ibm
S:

C: UNSUB TOPIC /stocks/quotes/*/C:
S: 200-OK
S: 200-TOPIC /stocks/quotes/ibm
S: 200 TOPIC /stocks/quotes/beos
S:

C: UNSUB TOPIC *
C:
S: 200-OK
S: 200-TOPIC /stocks/quotes/ibm
S: 200-TOPIC /stocks/quotes/beos
S: 200 TOPIC /lists/news/os
S:

4.1.11.3.   SUBSCRIBERS <topic>

A client MAY unsubscribe to changes to subscribers of a topic.

Tegen                    Internet Draft                          32
Simple Message Queue Protocol (SMQP)   September 2001

Reply codes:
200   OK, success
400   Bad request
404   Not found (topic)

Unsubscribe to all subscription, regardless of resource, can be done by wildcarding the resource

C: UNSUB *
C:
S: 200-OK
S: 200-MESSAGE /stocks/quotes/ibm
S: 200-MESSAGE /stocks/quotes/sony
S: 200-TOPIC /stocks/quotes/ibm
S: 200-TOPIC /stocks/quotes/beos
S: 200 SUB /stocks/quotes/ibm
S:
4.1.12. NOTIFY

"NOTIFY" 1*SP resource

The notification of a published resource is sent to the subscriber from the SMQP server. The notification is a two-phase commit operation. While the client is not sending a command, it SHOULD be listening for notification messages.

4.1.12.1. MESSAGE <topic>

"NOTIFY" 1*SP ("MESSAGE" | "MESS") 1*SP topic

SMQP does not concern itself with the contents of the data that is carried along with the message. The sender and receiver of the message are responsible for the creation, definition, and interpretation of the message data. Examples of message data could be plain text, images (JPEG, GIF), audio files, XML, or binary streams. The receiver MUST handle a message that it is unable to interpret based on its content type gracefully. Sending a reply back to the sender would not be a friendly behavior since the publisher is not responsible for putting the data in a format that all subscribers can understand.

Data MUST be of a discrete length. Continuous stream of data, like audio broadcasts, are not supported under SMQP.

Two-phase commit

This is to ensure that the client properly receives the message and that the client knows that the server received acknowledgement of the receipt. The server will not send the next notification until it has received an acknowledgement by the client that it has received the message. The next publication notification sent to the client would occur after the server acknowledged the client’s acknowledgement.

Implementations MAY vary, but the example above would perform the following:

S: NOTIFY MESSAGE
S: ...
S:

Client saves the message (memory or disk) and sends the server an acknowledgement that it received the message.

C: 310 ACK
C:

The server would save that this message was successfully sent to the client, and will not try to resend this message again ever. The server would then tell the client that it acknowledge receipt of the message.

S: 310 ACK
S:
Now the client can act on the message by displaying it or forwarding it to a sub-system process. If the client does not receive acknowledgement from the server that the server knows the client received the message OK, it MUST assume that the server is likely to resend the message again. If it does not receive the message again in some period of time (client implementation dependent), it MAY proceed with the message that it has.

Example NOTIFY MESSAGE sequence:

S: NOTIFY MESSAGE /stocks/quotes/ibm
S: Created: D08152001T08032445Z
S: Smuid: www.host.com/1835
S: Cmuid: jtegen/jtegen3454654/A004
S: Priority: 1
S: Receipt: yes
S: Reply: NO
S: Timeout: 00:01:00
S: Depth: 1
S: Sequence: 34
S:
S: Content-Type: text/plain
S: Content-Length: 12
S:
S: 34.456
S:
S:
C: 310 ACK
C:
S: 310 ACK

The message notification contains various attributes. Attributes are named values, separated by a colon (:).

CREATED = "Created" ":" date-time
CREATED is a REQUIRED attribute. It defines the date and time, to the fraction of a second, when the client, in GMT, created the message. The client receiving the message SHOULD only use the creation time as an indication of time, since the time the publisher sent the message MAY be not an absolute value since the publisher internal clock and the latency of the network MAY impact this time as compared to the internal clock of the receiver.

SMUID = "Smuid" ":" host "/" positive-number
SMUID is the REQUIRED Server Message Unique Identifier. This is the UID generated by the server, when the publisher first published the message. This UID is always unique for a particular server and topic.

CMUID = "Cmuid" ":" account "/" guid "/" uid
CMUID is the REQUIRED Client Message Unique Identifier. This is the account, GUID, and the UID generated by the
publisher. This information is used to reply to the given message.

CONTENT-TYPE = "Content-Type"":" media-type
This is the content type of the data in the message. If there were no data associated with this message, then this attribute would not be present. A CONTENT-TYPE would exist for each data section in the message. The CONTENT-TYPE is the MIME type of the data to follow.

CONTENT-LENGTH = "Content-Length"":" positive-number
This is the content length of the data in the message. If there were not data associated with the message, then this attribute would not be present. A CONTENT-LENGTH would exist for each data section in the message. The CONTENT-LENGTH is always paired with CONTENT-TYPE. The CONTENT-LENGTH is a fixed number of bytes for the data to follow.

4.1.12.2. TOPIC <topic>

"NOTIFY" 1*SP "TOPIC" 1*SP topic

Notification that a topic has been modified. Topic notification are followed with attributes that describe the change.

ACTION = "Action"":" ( "ADDED" | "ADD" ) |

ACTION attributes describes the action done to a topic. The value for ACTION are:
ADDED
RENAME <new-topic-name>
DELETED
KEYWORDS <keyword>

S: NOTIFY TOPIC /stocks/quotes/ibm
S: ACTION: RENAME /stocks/quotes/big_blue
S:

4.1.12.3. SUBSCRIBERS <topic>

"NOTIFY" 1*SP ( "SUBSCRIBERS" | "SUB" ) 1*SP topic

Notification that a change in the number of subscribers occurred for a topic. The publisher MAY use this to stop sending updates if they know there are not any subscribers wanting their data. This can dramatically reduce the network traffic of messages.

COUNT <number>
COUNT attribute returns the current number of subscribers to the topic.
S: NOTIFY SUB /stocks/quotes/ibm
S: COUNT: 34
S:

4.1.12.4. SERVER

"NOTIFY" 1*SP ( "SERVER" | "SERV" ) 1*SP resource

4.1.12.4.1. TIMEOUT <timeout>

"NOTIFY" 1*SP ( "SERVER" | "SERV" ) 1*SP "TIMEOUT" 1*SP timeout

4.1.12.4.2. BYE <timeout>

"NOTIFY" 1*SP ( "SERVER" | "SERV" ) 1*SP "BYE" 1*SP timeout

NOTIFY BYE is a server notification to all currently connected clients. Clients do not subscribe to this message when the server MAY be shutting down. It will inform the clients that it will not be available by a certain period of time.

S: NOTIFY BYE 00:05:00
S:

This notifies the client that the server is shutting down in 5 minutes. If the server is shutting down right away, and the client really has no option but to shut down too, the client would receive either

S: NOTIFY BYE 00:00:00
S:

Or

S: NOTIFY BYE
S:

4.1.13. GET

4.1.13.1. GET MESSAGE

"GET" 1*SP ( "MESSAGE" | "MESS" ) 1*SP topic 1*SP smuid

GET MESSAGE allows a client to get a specific message that MAY still be in a queue on the server. A client can get a message even if they have already received the message.

C: GET MESSAGE /stocks/quotes/ibm 1865
C:
S: 200 OK
S: Topic: /stocks/quotes/ibm
The following are valid GET requests to the server:
C: GET MESSAGE /stocks/quotes/ibm 1865
C: GET MESSAGE /stocks/quotes/ibm 1866
C: GET MESSAGE /stocks/quotes/sony 2007
C:
C: GET MESSAGE /stocks/quotes/ibm 1865-2340
C:
C: GET MESSAGE /stocks/quotes/ibm 1845-1890,1905,1920-1930
C:
C: GET MESSAGE /stocks/quotes/ibm *

4.1.13.2. GET TOPIC

"GET" 1*SP "TOPIC" 1*SP ( "KEYWORDS" | "KEY" ) 1*SP topic

GET TOPIC KEYWORDS returns the current keywords for the given topic. The client may set the keywords by using the SET commands.

C: GET TOPIC KEYWORDS /stocks/quotes/ibm
C:
S: 200-OK
S: 200 IBM,BIG BLUE,STOCKS,COMPUTERS
S:

4.1.15. NOOP

"NOOP"

This command does not affect any parameters or previously entered commands. It specifies no action other than that the server sends an reply code.

Reply Codes:
200 OK, success
400 Bad request
501 Not implemented
"QUIT"

QUIT discontinues a session with the SMQP server. Any subscribed resources are still subscribed to.

Reply Codes:
200      OK, success
400      Bad request

C: QUIT
C:
S: 200 OK
S:

5. Implementation Considerations
This section discusses items to be considered when implementing either the client or server that supports SMQP. The protocol is independent of implementation, however there are certain capabilities that SHOULD be considered during design and development of both clients and server.

5.1. Server
1) Supply a default timeout for messages before deletion
2) Retry timeout
3) Maximum retry per message per client
4) Server SHOULD be as stateless as possible, so in case of failure, it can be restarted without any loss of information
5) Maximum file size to receive (overall and per queue), give error code
6) Send server/client limits (file size, MIME type, ...)
7) Restrict number of subscriptions in one request
8) Restrict total number of subscriptions by a user account.

5.2. Client
1) Client SHOULD be as stateless as possible
2) Client MAY wish to be multi-threaded to allow one thread to post commands and one thread to listen for server notifications.

6. Security Consideration
Security is a vital consideration for any mission critical system that shares information between distributed users. Server operations, for each topic/queue are regulated by a User Access List (UAL). UALs are controlled by the system administrator of the server and cannot be accessed through the SMQP commands. Messages cannot be published to a topic if that user does not have permission to do so. Likewise, a subscriber cannot subscribe to a topic if they do not have permission to do so.

Each topic/queue can be configured with different parameters. These parameters include:
1) User access to publish to
2) User access to subscribe from
3) User access to modify queue parameters
   a. Expiration timeout
   b. Retry timeout
   c. Maximum retries
   d. Maximum message count
   e. Maximum message size
   f. Maximum queue size (total message size)
4) User access to create, rename, or delete topics at or below the current queue/topic
5) User permission to reply to sent messages
6) User permission to access remote servers

SMQP SHALL allow unsecured and secured protocols. The secured protocol (SMQPS) SHALL use public forms of encryption like PGP.

SMQP does not enforce any type of data encryption or encoding of the data section of the message. The message header can be extended with additional name value attributes to allow the subscriber of the message to decrypt or decode the message data.

Login password authentication CAN support any number of methods. This includes from passing a clients password in the clear, to stronger authentication methods like Kerberos. The client requests a method of authentication to the server. Depending on how the server is implemented and/or configured, the server MAY reject the authentication method requested. Upon a rejection from the server, the client may try other methods until accepted by the server. Upon agreeing upon a method, the client and server would interact until the server is satisfied on the authentication of the client based on the method used.

7. Variable Definitions

The syntax of the above argument fields (using BNF notation where applicable) is given below. The "..." notation indicates that a field MAY be repeated one or more times.

<account> ::= <string> without <SP’s> or <special> characters.
<default-server-port> ::= 720
<default-client-port> ::= 721
<domain> ::= <element> | <element> "." <domain>
<dot-string> ::= <string> | <string> "." <dot-string>
<string> ::= <char> | <char> <string>
<quoted-string> ::= """ <qtext> """
<qtext> ::= "\" <x> | "\" <x> <qtext> | <q> | <q>
<char> ::= <c> | "\" <x>
<dotnum> ::= <snum> "." <snum> "." <snum> "." <snum>
<number> ::= <d> | <d> <number>
\(<\text{positive-number}\> ::= \text{<number> greater than zero}\)
\(<\text{CRLF}\> ::= \text{<CR> <LF>}\)
\(<\text{CR}\> ::= \text{the carriage return character (ASCII code 13)}\)
\(<\text{LF}\> ::= \text{the line feed character (ASCII code 10)}\)
\(<\text{SP}\> ::= \text{the space character (ASCII code 32)}\)
\(<\text{snum}\> ::= \text{one, two, or three digits representing a decimal integer value in the range 0 through 255}\)
\(<\text{a}\> ::= \text{any one of the 52 alphabetic characters A through Z in upper case and a through z in lower case}\)
\(<\text{c}\> ::= \text{any one of the 128 ASCII characters, but not any <special> or <SP>}\)
\(<\text{d}\> ::= \text{any one of the ten digits 0 through 9}\)
\(<\text{q}\> ::= \text{any one of the 128 ASCII characters except <CR>, <LF>, quote ("), or backslash (\)\)}
\(<\text{keywords}\> ::= "(" <text-UTF8>,<text-UTF8>,...")"
\(<\text{username}\> ::= \text{<string> without <SP>'s and <special> characters.}\)
\(<\text{password}\> ::= \text{<string> without <SP>'s}\)
\(<\text{x}\> ::= \text{any one of the 128 ASCII characters (no exceptions)}\)
\(<\text{special}\> ::= "<" | ">" | "(" |")" | "[" | "]" | "\" | "." | "," | ":" | ";" | ":@" | "**" | the control characters (ASCII codes 0 through 31 inclusive and 127)\)
\(<\text{ff}\> ::= \text{Fractions of a second}\)

Note that the backslash, ",", is a quote character, which is used to indicate that the next character is to be used literally (instead of its normal interpretation). For example, "Joe\,Smith" could be used to indicate a single nine character user field with comma being the fourth character of the field.

Hosts are generally known by names which are translated to addresses in each host. Note that the name elements of domains are the official names -- no use of nicknames or aliases is allowed.

Sometimes a host is not known to the translation function and communication is blocked. To bypass this barrier two numeric forms are also allowed for host "names". One form is a decimal integer prefixed by a pound sign, "#", which indicates the number is the address of the host. Another form is four small decimal integers separated by dots and enclosed by brackets, e.g., "[123.255.37.2]", which indicates a 32-bit ARPA Internet Address in four 8-bit fields.

The time stamp line and the return path line are formally defined as follows:
\(<\text{return-path-line}\> ::= "Return-Path:" <SP><reverse-path><CRLF>
The standard names for links are registered with the Network Information Center.

The standard names for protocols are registered with the Network Information Center.

<time-stamp-line> ::= "Received:" <SP> <stamp> <CRLF>
<daytime> ::= <SP> <date> <SP> <time>
<date-time> ::= D<dd><mm><yyyy>T<hh><mm><ss>Z all designated in GMT time zone
<timeout> ::= <DDDD:HH:MM:SS:FF>
<date> ::= <dd> <SP> <mon> <SP> <yy>
<time> ::= <hh> ":" <mm> ":" <ss> <SP> <zone>
<dd> ::= the one or two decimal integer day of the month in the range 1 to 31.
<yy> ::= the two decimal integer year of the century in the range 00 to 99.
<hh> ::= the two decimal integer hour of the day in the range 00 to 24.
<mm> ::= the two decimal integer minute of the hour in the range 00 to 59.
/ss> ::= the two decimal integer second of the minute in the range 00 to 59.
<ff> ::= the two decimal integer fractions of a second of the second in the range of 000 to 59.
<zone> ::= "UT" for Universal Time (the default) or other time zone designator (as in [2]).
<topic> ::= "/"<string>. Topic name are only US-ASCII printable characters, excluding "/", ",", ",-", and "+".
<guid> ::= <string>
<resource> ::= <string> command
<item> ::= <string> identifier for a <resource>
<Boolean> ::= ( "YES" | "NO" ) | ( "1" | "0" )
<smuid> ::= <number>
<cmuid> ::= <string> no <SP>’s or <special> characters

8. Authors’ Address

John Tegen
OmicronSoft
9974 Scripps Ranch Blvd., PMB 252
San Diego, CA  92131
Email : john@omicronsoft.com

APPENDIX A
Reply Codes

The three digits of the reply each have special significance. The first digit denotes whether the response is good, bad or incomplete. An unsophisticated sender will be able to determine its next action (proceed as planned, redo, retrench, etc.) by simply examining this first digit. A sender that wants to know approximately what kind of error occurred (e.g., server error, command syntax error) MAY examine the second digit, reserving the third digit for the finest gradation of information.

There are five values for the first digit of the reply code:

1yz Positive Preliminary reply

The command has been accepted, but the requested action is being held in abeyance, pending confirmation of the information in this reply. The sender SHOULD send another command specifying whether to continue or abort the action.

2yz Positive Completion reply

The requested action has been successfully completed. A new request MAY be initiated.

3yz Positive Intermediate reply

The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The sender SHOULD send another command specifying this information. This reply is used in command sequence groups.

4yz Transient Negative Completion reply

The command was not accepted and the requested action did not occur. However, the error condition is temporary and the action MAY be requested again. The sender SHOULD return to the beginning of the command sequence (if any). It is difficult to assign a meaning to "transient" when two different sites (receiver and sender) MUST agree on the interpretation. Each reply in this category might have a different time value, but the sender is encouraged to try again. A rule of thumb to determine if a reply fits into the 4yz or the 5yz category (see below) is that replies are 4yz if they can be repeated without any change in command form or in properties of the sender or receiver. (E.g., the command is repeated identically and the receiver does not put up a new implementation.)

5yz Permanent Negative Completion reply

The command was not accepted and the requested action did not occur. The sender is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user MAY want to direct the sender to reinitiate the command sequence by direct
action at some point in the future (e.g., after the spelling has been changed, or the user has altered the account status).

The second digit encodes responses in specific categories:

x0z Syntax -- These replies refer to syntax errors, syntactically correct commands that don't fit any functional category, and unimplemented or superfluous commands.

x1z Information -- These are replies to requests for information, such as status or help.

x2z Connections -- These are replies referring to the transmission channel.

x3z Unspecified as yet.

x4z Unspecified as yet.

The third digit gives a finer gradation of meaning in each category specified by the second digit. The list of replies illustrates this. Each reply text is RECOMMENDED rather than mandatory, and MAY even change according to the command with which it is associated. On the other hand, the reply codes MUST strictly follow the specifications in this section. Receiver implementations SHOULD NOT invent new codes for slightly different situations from the ones described here, but rather adapt codes already defined.

The reply text MAY be longer than a single line; in these cases the complete text MUST be marked so the sender-SMTP knows when it can stop reading the reply. This requires a special format to indicate a multiple line reply.

The format for multiline replies requires that every line, except the last, begin with the reply code, followed immediately by a hyphen, "-" (also known as minus), followed by text. The last line will begin with the reply code, followed immediately by <SP>, optionally some text, and <CRLF>.

For example:

```
123-First line
123-Second line
123-234 text beginning with numbers
123 The last line
```

In many cases the sender then simply needs to search for the reply code followed by <SP> at the beginning of a line, and ignore all preceding lines. In a few cases, there is important data for the sender in the reply "text". The sender will know these cases from the current context.

Reply codes are:

```
200 OK
310 Acknowledgement receipt
```
Message flow is the process of moving a message between queues based on its content. This MAY be also known as message routing. An example of message flow is order processing. The example steps for this sample are:

1) Order received
2) Inventory verification
3) Invoice customer
4) Inventory fulfillment
5) Shipping
6) Notification

As stated earlier in this RFC, SMQP is not concerned with the contents of a message. It is the responsibility of the subscribers to interpret and react to messages and its contents. Each one of the steps listed above involve an intervening process to perform the action, all well outside the domain of a SMPQ server. For an implementation to conduct the above scenario each step MAY have a subscriber that subscribes to a separate topic for each step. Each process along the steps MAY receive a message, act on it, and then publish a new/modified message to another topic. Each topic can be a step along the assembly line of the message flow. SMQP allows message flow due to its location transparency, guaranteed store forward design. Each step along the flow can be physically separated to allow a general fault tolerant implementation.


Glossary

ASCII
American Standard Code for Information Interchange.

command
A request for a service action sent by the sender to the receiver.

domain
The hierarchially structured global character string address of a host computer in the system.

end of data indication
A special sequence of characters that indicates the end of the data. In particular, the five characters carriage return, line feed, period, carriage return, line feed, in that order.

host
A computer in the internetwork environment on which topics or SMQP processes reside.

line
A sequence of ASCII characters ending with a <CRLF>.

publisher
A client that publishes data

reply
A reply is an acknowledgment (positive or negative) sent from receiver to sender via the transmission channel in response to a command. The general form of a reply is a completion code (including error codes) followed by a text string. The codes are for use by programs and the text is usually intended for human users.

session
The set of exchanges that occur while the transmission
channel is open.

**subscriber**
A client that subscribes to data from an unknown source.

**transaction**
The set of exchanges required for one message to be transmitted for one or more recipients.

**transmission channel**
A full-duplex communication path between a sender and a receiver for the exchange of commands, replies, and data.

**transport service**
Any reliable stream-oriented data communication services. For example, NCP, TCP, NITS.

**user**
A human being (or a process on behalf of a human being) wishing to obtain asynchronous transfer of messages.

**word**
A sequence of printing characters.

<CRLF>
The characters carriage return and line feed (in that order).

<SP>
The space character.
HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Expires: February 21, 2002