SMQ-003
Protocol Specification

Simple Message Queues Protocol Specification Guide

1 Introduction

A web based Publish-Subscribe framework

The SMQ protocol structure delivers a hybrid event-driven communication approach for information dissemination using sockets (TCP/IP) and WebSockets as a communication protocol within a Publish/Subscribe paradigm.

Message Structure (Event-Driven, Websockets, Pub/Sub)

Pub/Sub pattern operation for propagating data close to real-time creating a practical environment , while taking advantage of Web 2.0 usage by Restful web services. Messages are coordinated and integrated among software components as software applications change over time, and are transferred from one machine to another over unreliable wireless networks.

- Asynchronous Communication
- Ephemeral IDs: Client, topic, and sub-topic ID structures provide for greater decentralized Pub/Sub messaging control by the application. This allows provisioning for enhanced one to one or one to many communications.
- Message Context: SMQ fixed headers optimize processing speed and eliminate inconsistencies associated with nibbles and trickling (variable structure), which in turn alleviates server data processing complexity.

2 Acronyms and Definitions:

UID: Universally unique ID (such as a MAC address)

sock: TCP socket object

TID: Topic ID

TCP/IP: Transmission Control Protocol/Internet Protocol.

Websocket: A protocol providing full-duplex communications channels over a single TCP connection.

HTTP: Hypertext Transfer Protocol

HTTP(s): HTTP over SSL or HTTP Secure

LSP: Lua Server Pages

URL: Uniform Resource Locator - Unique address for a file that is accessible on the internet.

SMQ: Simple Message Queues Protocol

3 SMQ Protocol Overview

Protocol

- Device to broker using raw TCP/IP: The raw socket version requires a two byte packet size.
- Browser to Broker using WebSockets: The packet size is included in the WebSocket protocol and is therefore not needed in the SMQ protocol when communicating over WebSockets.

Three Byte Packet Header for Raw Sockets:

2 bytes	Packet size
1 byte	Message type
N bytes	Optional data depends on message type

One Byte Packet Header for WebSockets:

1 byte	Message type
N bytes	Optional data depends on message type

Message Types:

Mnemonic	Enumeration	Description
Invalid	0	Invalid
Init	1	Sent from server as soon as device/browser receives the initial HTTPS header.
Connect	2	Sent from device/browser to server as soon as client receives Init.
Connack	3	Server responds to Connect with OK or fail message.
Subscribe	4	Subscribe to topic by name.
Suback	5	Topic name is translated to TID.
Create	6	Create Topic ID (TID). Similar to subscribe in that a TID is reserved on the server, but the client is not subscribed to the topic. The message is typically used by publishers since a message cannot be published without a TID.
Createack	7	Topic name translated to TID
Publish	8	Publish a message to topic 'TID'
Unsubscribe	9	Unsubscribe from topic
Reserved	10	Reserved
Disconnect	11	Gracefully close the connection.

Ping	12	Enables idle connections to stay alive when communicating via a NAT router.

Pong	13	Response to ping.
Observe	14	Observe (supervise) changes for topic
Unobserve	15	Remove previously registered change notifications
Change	16	Number of subscribers for topic changed
Createsub	17	Create sub topic ID (STID).
Createsuback	18	Sub topic name translated to STID
PubFrag	19	Publish one message as a stream of chunks
Reserved	20-255	Reserved

A connection sequence for the raw socket version is as follows:

The client connects to the server using HTTP or HTTPS, where the URL is set to the entry location for the SMQ broker instance. The server then responds with Init. The sequence is similar to WebSocket upgrade, but without sending a HTTP response header. The binary Init message is sent as a response to the HTTP(S) request.

The client can send the Connect command as soon as it receives Init and the server will then respond with Connack.

Init

1 byte	Version number is 1
4 bytes	Random number created by the server. Can be used as seed value if a hash of the credentials are required.
N bytes	The server sends the client's IP address as a string. Note: the IP address may not be the same as the IP address used by the client if the client is behind a NAT.

Connect

1 byte	Version number is 2
2 bytes	Max TLS packet/frame size accepted by client. Zero means don't care. The server must fragment SMQ packets larger than this size.
1 byte	(Globally) Unique ID length
N bytes	(Globally) Unique ID (minimum 6 bytes)
1 byte	Client credential's length
N bytes	Client's credentials e.g. username: password
N bytes	Info: the client may optionally send an identification string to the server. This information can be fetched by server side code and used for identification purposes.

Connack

1 byte	Return code (see below for code values)
4 bytes	Client's unique topic ID. All clients have a unique topic ID and this ID is included in all Published messages.
N bytes	Optional human readable message describing the return code if the code is not 0x00.

Return codes:

HEX	Meaning
0x00	Connection Accepted
0x01	Connection Refused: Unacceptable Protocol Version
0x02	Connection Refused: Server Unavailable
0x03	Connection Refused: Incorrect Credentials
0x04	Connection Refused: Client Certificate Required
0x05	Connection Refused: Client Certificate Not Accepted
0x06	Connection Refused: Access Denied

Subscribe, Create, and Createsub

N bytes	Topic name i.e., "/device1/temperature"

Suback, Createack, and Createsuback

1 byte	0x00: accepted, 0x01: denied
4 bytes	Topic ID: The topic name is translated to a topic ID by the server.
N bytes	Topic name. The name provided in Subscribe

Publish

4 bytes	The topic ID (publish to this topic). The topic ID is typically created/fetched by sending message Create to the server.
4 bytes	The client's unique topic ID received in Connack (sender of message)
4 bytes	A messageID that can be used by the application (publisher/consumers) as a way to further identify the message payload data.
N bytes	data

PubFrag

Publish a fragment. This message type, which is structurally identical to Publish, makes it possible for microcontrollers to publish one message in stages. The broker assembles all fragments before publishing the message to the subscribers. The topic ID must be set to zero for fragments. The broker assumes it is the last fragment when the topic ID is set.

Unsubscribe

Unsubscribe from a topic. There is no response message to Unsubscribe.

4 bytes	The topic ID

Observe

Register for topic change notifications

4 bytes	The topic ID

Unobserve

Remove previously registered change notifications

4 bytes	The topic ID

Change

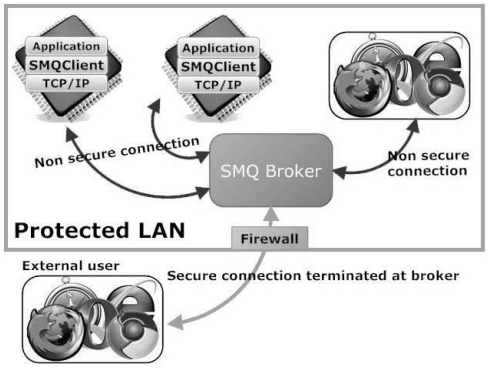
Number of subscribers for topic changed

4 bytes	The topic ID
4 bytes	The number of subscribers

Ping, Pong, Disconnect

The messages: Ping, Pong, and Disconnect (gracefully close the connection) do not include any payload data.

4 SMQ Broker Termination



SMQ Broker enables IoT SSL termination. To illustrate by example we have an external browser using secure WebSockets sending a message to non-secure devices utilizing the SMQ Client. In this example the SMQ Broker decrypts the message received from the browser and then forwards the unencrypted result to the two non-secure devices. This communication also works in reciprocal pattern where a non-secure client is able to send messages to a secure client.