MQTT V3.1 Protocol Specification

Authors:

International Business Machines Corporation (IBM)
Eurotech

Abstract

MQ Telemetry Transport (MQTT) is a lightweight broker-based publish/subscribe messaging protocol designed to be open, simple, lightweight and easy to implement. These characteristics make it ideal for use in constrained environments, for example, but not limited to:

- Where the network is expensive, has low bandwidth or is unreliable
- When run on an embedded device with limited processor or memory resources

Features of the protocol include:

- The publish/subscribe message pattern to provide one-to-many message distribution and decoupling of applications
- A messaging transport that is agnostic to the content of the payload
- The use of TCP/IP to provide basic network connectivity
- Three qualities of service for message delivery:
    - "At most once", where messages are delivered according to the best efforts of the underlying TCP/IP network. Message loss or duplication can occur. This level could be used, for example, with ambient sensor data where it does not matter if an individual reading is lost as the next one will be published soon after.
    - "At least once", where messages are assured to arrive but duplicates may occur.
    - "Exactly once", where message are assured to arrive exactly once. This level could be used, for example, with billing systems where duplicate or lost messages could lead to incorrect charges being applied.
- A small transport overhead (the fixed-length header is just 2 bytes), and protocol exchanges minimised to reduce network traffic
- A mechanism to notify interested parties to an abnormal disconnection of a client using the Last Will and Testament feature

Copyright Notice

Table of Contents

1. Introduction

This specification is split into three main sections:

- the message format that is common to all packet types,
- the specific details of each packet type,
- how the packets flow between client and server.

Information on how topic wildcards are used is provided in the appendix.

1.1. Changes

The following are the changes between MQTT V3 and MQTT V3.1:

- User name and password can now be sent with a CONNECT packet
- New return codes on CONNACK packets, for security problems
- Clarification that clients are not informed of un-authorized PUBLISH or SUBSCRIBE commands, and that the normal MQTT flow should complete even though the command has not been performed.
- Strings in MQTT now support full UTF-8, instead of just the US-ASCII subset.

The protocol version number passed with CONNECT packets, is unchanged for this revision, and remains as the "3". Existing MQTT V3 server implementations should be able to accept connections from clients that support this revision, as long as they correctly respect the "Remaining Length" field, and therefore ignore the extra security information.

2. Message format

The message header for each MQTT command message contains a fixed header. Some messages also require a variable header and a payload. The format for each part of the message header is described in the following sections:

2.1. Fixed header

The message header for each MQTT command message contains a fixed header. The table below shows the fixed header format.

bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

Byte 1

Contains the Message Type and Flags (DUP, QoS level, and RETAIN) fields.

Byte 2

(At least one byte) contains the Remaining Length field.

The fields are described in the following sections. All data values are in big-endian order: higher order bytes precede lower order bytes. A 16-bit word is presented on the wire as Most Significant Byte (MSB), followed by Least Significant Byte (LSB).

Message Type

Position: byte 1, bits 7-4.

Represented as a 4-bit unsigned value. The enumerations for this version of the protocol are shown in the table below.

Mnemonic	Enumeration	Description
Reserved	0	Reserved
CONNECT	1	Client request to connect to Server
CONNACK	2	Connect Acknowledgment
PUBLISH	3	Publish message
PUBACK	4	Publish Acknowledgment

Mnemonic	Enumeration	Description
PUBREC	5	Publish Received (assured delivery part 1)
PUBREL	6	Publish Release (assured delivery part 2)
PUBCOMP	7	Publish Complete (assured delivery part 3)
SUBSCRIBE	8	Client Subscribe request
SUBACK	9	Subscribe Acknowledgment
UNSUBSCRIBE	10	Client Unsubscribe request
UNSUBACK	11	Unsubscribe Acknowledgment
PINGREQ	12	PING Request
PINGRESP	13	PING Response
DISCONNECT	14	Client is Disconnecting
Reserved	15	Reserved

Flags

The remaining bits of byte 1 contain the fields DUP, QoS, and RETAIN. The bit positions are encoded to represent the flags as shown in the table below.

Bit position	Name	Description
3	DUP	Duplicate delivery
2-1	QoS	Quality of Service
0	RETAIN	RETAIN flag

DUP

Position: byte 1, bit 3.

This flag is set when the client or server attempts to re-deliver a PUBLISH, PUBREL, SUBSCRIBE or UNSUBSCRIBE message. This applies to messages where the value of QoS is greater than zero (0), and an acknowledgment is required. When the DUP bit is set, the variable header includes a Message ID.

The recipient should treat this flag as a hint as to whether the message may have been previously received. It should not be relied on to detect duplicates.

QoS

Position: byte 1, bits 2-1.

This flag indicates the level of assurance for delivery of a PUBLISH message. The QoS levels are shown in the table below.

QoS value	bit 2	bit 1	Description		
0	0	0	At most once	Fire and Forget	<=1
1	0	1	At least once	Acknowledged delivery	>=1
2	1	0	Exactly once	Assured delivery	=1
3	1	1	Reserved		

RETAIN

Position: byte 1, bit 0.

This flag is only used on PUBLISH messages. When a client sends a PUBLISH to a server, if the Retain flag is set (1), the server should hold on to the message after it has been delivered to the current subscribers.

When a new subscription is established on a topic, the last retained message on that topic should be sent to the subscriber with the Retain flag set. If there is no retained message, nothing is sent

This is useful where publishers send messages on a "report by exception" basis, where it might be some time between messages. This allows new subscribers to instantly receive data with the retained, or Last Known Good, value.

When a server sends a PUBLISH to a client as a result of a subscription that already existed when the original PUBLISH arrived, the Retain flag should not be set, regardless of the Retain flag of the original PUBLISH. This allows a client to distinguish messages that are being received because they were retained and those that are being received "live".

Retained messages should be kept over restarts of the server.

A server may delete a retained message if it receives a message with a zero-length payload and the Retain flag set on the same topic.

Remaining Length

Position: byte 2.

Represents the number of bytes remaining within the current message, including data in the variable header and the payload.

The variable length encoding scheme uses a single byte for messages up to 127 bytes long. Longer messages are handled as follows. Seven bits of each byte encode the Remaining Length data, and the eighth bit indicates any following bytes in the representation. Each byte encodes 128 values and a "continuation bit". For example, the number 64 decimal is encoded as a single byte, decimal value 64, hex 0x40. The number 321 decimal (= 65 + 2*128) is encoded as two bytes, least significant first. The first byte 65+128 = 193. Note that the top bit is set to indicate at least one following byte. The second byte is 2.

The protocol limits the number of bytes in the representation to a maximum of four. This allows applications to send messages of up to 268 435 455 (256 MB). The representation of this number on the wire is: 0xFF, 0xFF, 0xFF, 0x7F.

The table below shows the Remaining Length values represented by increasing numbers of bytes.

Digits	From	To
1	0 (0x00)	127 (0x7F)
2	128 (0x80, 0x01)	16 383 (0xFF, 0x7F)
3	16 384 (0x80, 0x80, 0x01)	2 097 151 (0xFF, 0xFF, 0x7F)
4	2 097 152 (0x80, 0x80, 0x80, 0x01)	268 435 455 (0xFF, 0xFF, 0xFF, 0x7F)

The algorithm for encoding a decimal number (X) into the variable length encoding scheme is as follows:

```
do
  digit = X MOD 128
  X = X DIV 128
  // if there are more digits to encode, set the top bit of this digit
  if ( X > 0 )
    digit = digit OR 0x80
  endif
  'output' digit
while ( X> 0 )
```

where `MOD` is the modulo operator (`%` in C), `DIV` is integer division (`/` in C), and `OR` is bit-wise or (`|` in C).

The algorithm for decoding the Remaining Length field is as follows:

```
multiplier = 1
value = 0
do
  digit = 'next digit from stream'
  value += (digit AND 127) * multiplier
  multiplier *= 128
while ((digit AND 128) != 0)
```

where `AND` is the bit-wise and operator (`&` in C).

When this algorithm terminates, `value` contains the Remaining Length in bytes.

Remaining Length encoding is not part of the variable header. The number of bytes used to encode the Remaining Length does not contribute to the value of the Remaining Length. The variable length "extension bytes" are part of the fixed header, not the variable header.

2.2. Variable header

Some types of MQTT command messages also contain a variable header component. It resides between the fixed header and the payload.

The variable length Remaining Length field is not part of the variable header. The bytes of the Remaining Length field do not contribute to the byte count of the Remaining Length value. This value only takes account of the variable header and the payload. See Fixed header for more information.

The format of the variable header fields are described in the following sections, in the order in which they must appear in the header:

Protocol name

The protocol name is present in the variable header of a MQTT CONNECT message. This field is a UTF-encoded string that represents the protocol name MQIsdp, capitalized as shown.

Protocol version

The protocol version is present in the variable header of a CONNECT message.

The field is an 8-bit unsigned value that represents the revision level of the protocol used by the client. The value of the Protocol version field for the current version of the protocol, 3 (0x03), is shown in the table below.

bit	7	6	5	4	3	2	1	0
	Protocol Version							
	0	0	0	0	0	0	1	1

Connect flags

The Clean session, Will, Will QoS, and Retain flags are present in the variable header of a CONNECT message.

Clean session flag

Position: bit 1 of the Connect flags byte.

If not set (0), then the server must store the subscriptions of the client after it disconnects. This includes continuing to store QoS 1 and QoS 2 messages for the subscribed topics so that they can be delivered when the client reconnects. The server must also maintain the state of in-flight messages being delivered at the point the connection is lost. This information must be kept until the client reconnects.

If set (1), then the server must discard any previously maintained information about the client and treat the connection as "clean". The server must also discard any state when the client disconnects.

Typically, a client will operate in one mode or the other and not change. The choice will depend on the application. A clean session client will not receive stale information and it must resubscribe each time it connects. A non-clean session client will not miss any QoS 1 or QoS 2 messages that were published whilst it was disconnected. QoS 0 messages are never stored, since they are delivered on a best efforts basis.

This flag was formerly known as "Clean start". It has been renamed to clarify the fact it applies to the whole session and not just to the initial connect.

A server may provide an administrative mechanism for clearing stored information about a client which can be used when it is known that a client will never reconnect.

bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved
	x	x	x	x	x	x		x

Bit 0 of this byte is not used in the current version of the protocol. It is reserved for future use.

Will flag

Position: bit 2 of the Connect flags byte.

The Will message defines that a message is published on behalf of the client by the server when either an I/O error is encountered by the server during communication with the client, or the client fails to communicate within the Keep Alive timer schedule. Sending a Will message is not triggered by the server receiving a DISCONNECT message from the client.

If the Will flag is set, the Will QoS and Will Retain fields must be present in the Connect flags byte, and the Will Topic and Will Message fields must be present in the payload.

The format of the Will flag is shown in the table below.

bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved
	x	x	x	x	x		x	x

Bit 0 of this byte is not used in the current version of the protocol. It is reserved for future use.

Will QoS

Position: bits 4 and 3 of the Connect flags byte.

A connecting client specifies the QoS level in the Will QoS field for a Will message that is sent in the event that the client is disconnected involuntarily. The Will message is defined in the payload of a CONNECT message.

If the Will flag is set, the Will QoS field is mandatory, otherwise its value is disregarded.

The value of Will QoS is 0 (0x00), 1 (0x01), or 2 (0x02). The Will QoS flag is shown in the table below.

bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved
	x	x	x			1	x	x

Bit 0 of this byte is not used in the current version of the protocol. It is reserved for future use.

Will Retain flag

Position: bit 5 of the Connect flags byte.

The Will Retain flag indicates whether the server should retain the Will message which is published by the server on behalf of the client in the event that the client is disconnected unexpectedly.

The Will Retain flag is mandatory if the Will flag is set, otherwise, it is disregarded. The format of the Will Retain flag is shown in the table below.

bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved
	x	x		x	x	1	x	x

Bit 0 of this byte is not used in the current version of the protocol. It is reserved for future use.

User name and password flags

Position: bits 6 and 7 of the Connect flags byte.

A connecting client can specify a user name and a password, and setting the flag bits signifies that a User Name, and optionally a password, are included in the payload of a CONNECT message.

If the User Name flag is set, the User Name field is mandatory, otherwise its value is disregarded. If the Password flag is set, the Password field is mandatory, otherwise its value is disregarded. It is not valid to supply a password without supplying a user

name.

bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Session	Reserved
			x	x	x	x	x	x

Bit 0 of this byte is not used in the current version of the protocol. It is reserved for future use.

Keep Alive timer

The Keep Alive timer is present in the variable header of a MQTT CONNECT message.

The Keep Alive timer, measured in seconds, defines the maximum time interval between messages received from a client. It enables the server to detect that the network connection to a client has dropped, without having to wait for the long TCP/IP timeout. The client has a responsibility to send a message within each Keep Alive time period. In the absence of a data-related message during the time period, the client sends a PINGREQ message, which the server acknowledges with a PINGRESP message.

If the server does not receive a message from the client within one and a half times the Keep Alive time period (the client is allowed "grace" of half a time period), it disconnects the client as if the client had sent a DISCONNECT message. This action does not impact any of the client's subscriptions. See DISCONNECT for more details.

If a client does not receive a PINGRESP message within a Keep Alive time period after sending a PINGREQ, it should close the TCP/IP socket connection.

The Keep Alive timer is a 16-bit value that represents the number of seconds for the time period. The actual value is application-specific, but a typical value is a few minutes. The maximum value is approximately 18 hours. A value of zero (0) means the client is not disconnected.

The format of the Keep Alive timer is shown in the table below. The ordering of the 2 bytes of the Keep Alive Timer is MSB, then LSB (big-endian).

bit	7	6	5	4	3	2	1	0
	Keep Alive MSB							
	Keep Alive LSB							

Connect return code

The connect return code is sent in the variable header of a CONNACK message.

This field defines a one byte unsigned return code. The meanings of the values, shown in the tables below, are specific to the message type. A return code of zero (0) usually

indicates success.

Enumeration	HEX	Meaning
0	0x00	Connection Accepted
1	0x01	Connection Refused: unacceptable protocol version
2	0x02	Connection Refused: identifier rejected
3	0x03	Connection Refused: server unavailable
4	0x04	Connection Refused: bad user name or password
5	0x05	Connection Refused: not authorized
6-255		Reserved for future use

bit	7	6	5	4	3	2	1	0
	Return Code							

Topic name

The topic name is present in the variable header of an MQTT PUBLISH message.

The topic name is the key that identifies the information channel to which payload data is published. Subscribers use the key to identify the information channels on which they want to receive published information.

The topic name is a UTF-encoded string. See the section on MQTT and UTF-8 for more information. Topic name has an upper length limit of 32,767 characters.

2.3. Payload

The following types of MQTT command message have a payload:

CONNECT

The payload contains one or more UTF-8 encoded strings. They specify a unqiue identifier for the client, a Will topic and message and the User Name and Password to use. All but the first are optional and their presence is determined based on flags in the variable header.

SUBSCRIBE

The payload contains a list of topic names to which the client can subscribe, and the QoS level. These strings are UTF-encoded.

SUBACK

The payload contains a list of granted QoS levels. These are the QoS levels at

which the administrators for the server have permitted the client to subscribe to a particular Topic Name. Granted QoS levels are listed in the same order as the topic names in the corresponding SUBSCRIBE message.

The payload part of a PUBLISH message contains application-specific data only. No assumptions are made about the nature or content of the data, and this part of the message is treated as a BLOB.

If you want an application to apply compression to the payload data, you need to define in the application the appropriate payload flag fields to handle the compression details. You cannot define application-specific flags in the fixed or variable headers.

2.4. Message identifiers

The message identifier is present in the variable header of the following MQTT messages: PUBLISH, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK.

The Message Identifier (Message ID) field is only present in messages where the QoS bits in the fixed header indicate QoS levels 1 or 2. See section on Quality of Service levels and flows for more information.

The Message ID is a 16-bit unsigned integer that must be unique amongst the set of "in flight" messages in a particular direction of communication. It typically increases by exactly one from one message to the next, but is not required to do so.

A client will maintain its own list of Message IDs separate to the Message IDs used by the server it is connected to. It is possible for a client to send a PUBLISH with Message ID 1 at the same time as receiving a PUBLISH with Message ID 1.

The ordering of the two bytes of the Message Identifier is MSB, then LSB (big-endian).

Do not use Message ID 0. It is reserved as an invalid Message ID.

bit	7	6	5	4	3	2	1	0
	Message Identifier MSB							
	Message Identifier LSB							

2.5. MQTT and UTF-8

UTF-8 is an efficient encoding of Unicode character-strings that optimizes the encoding of ASCII characters in support of text-based communications.

In MQTT, strings are prefixed with two bytes to denote the length, as shown in the table below.

bit	7	6	5	4	3	2	1	0
byte 1	String Length MSB							
byte 2	String Length LSB							
bytes 3 ...	Encoded Character Data							

String Length is the number of bytes of encoded string characters, not the number of characters. For example, the string OTWP is encoded in UTF-8 as shown in the table below.

bit	7	6	5	4	3	2	1	0
byte 1	Message Length MSB (0x00)							
	0	0	0	0	0	0	0	0
byte 2	Message Length LSB (0x04)							
	0	0	0	0	0	1	0	0
byte 3	'O' (0x4F)							
	0	1	0	0	1	1	1	1
byte 4	'T' (0x54)							
	0	1	0	1	0	1	0	0
byte 5	'W' (0x57)							
	0	1	0	1	0	1	1	1
byte 6	'P' (0x50)							
	0	1	0	1	0	0	0	0

The Java `writeUTF()` and `readUTF()` data stream methods use this format.

2.6. Unused bits

Any bits marked as unused should be set to zero (0).

3. Command messages

3.1. CONNECT - Client requests a connection to a server

When a TCP/IP socket connection is established from a client to a server, a protocol level session must be created using a CONNECT flow.

Fixed header

The fixed header format is shown in the table below.

bit	7	6	5	4	3	2	1	0
byte 1	Message Type (1)				DUP flag	QoS level		RETAIN
	0	0	0	1	x	x	x	x
byte 2	Remaining Length							

The DUP, QoS, and RETAIN flags are not used in the CONNECT message.

Remaining Length is the length of the variable header (12 bytes) and the length of the Payload. This can be a multibyte field.

Variable header

An example of the format of the variable header is shown in the table below.

	Description	7	6	5	4	3	2	1	0
Protocol Name									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (6)	0	0	0	0	0	1	1	0
byte 3	'M'	0	1	0	0	1	1	0	1
byte 4	'Q'	0	1	0	1	0	0	0	1
byte 5	'I'	0	1	0	0	1	0	0	1
byte 6	's'	0	1	1	1	0	0	1	1
byte 7	'd'	0	1	1	0	0	1	0	0
byte 8	'p'	0	1	1	1	0	0	0	0
Protocol Version Number									

	Description	7	6	5	4	3	2	1	0
byte 9	Version (3)	0	0	0	0	0	0	1	1
Connect Flags									
byte 10	User name flag (1) Password flag (1) Will RETAIN (0) Will QoS (01) Will flag (1) Clean Session (1)	1	1	0	0	1	1	1	x
Keep Alive timer									
byte 11	Keep Alive MSB (0)	0	0	0	0	0	0	0	0
byte 12	Keep Alive LSB (10)	0	0	0	0	1	0	1	0

User name flag

Set (1).

Password flag

Set (1).

Clean Session flag

Set (1).

Keep Alive timer

Set to 10 seconds (0x000A).

Will message

- Will flag is set (1)
- Will QoS field is 1
- Will RETAIN flag is clear (0)

Payload

The payload of the CONNECT message contains one or more UTF-8 encoded strings, based on the flags in the variable header. The strings, if present, must appear in the following order:

Client Identifier

The first UTF-encoded string. The Client Identifier (Client ID) is between 1 and 23 characters long, and uniquely identifies the client to the server. It must be unique across all clients connecting to a single server, and is the key in handling Message IDs messages with QoS levels 1 and 2. If the Client ID contains more than 23 characters, the server responds to the CONNECT message with a CONNACK return code 2: Identifier Rejected.

Will Topic

If the Will Flag is set, this is the next UTF-8 encoded string. The Will Message is published to the Will Topic. The QoS level is defined by the Will QoS field, and the RETAIN status is defined by the Will RETAIN flag in the variable header.

Will Message

If the Will Flag is set, this is the next UTF-8 encoded string. The Will Message defines the content of the message that is published to the Will Topic if the client is unexpectedly disconnected. This may be a zero-length message.

Although the Will Message is UTF-8 encoded in the CONNECT message, when it is published to the Will Topic only the bytes of the message are sent, not the first two length bytes. The message must therefore only consist of 7-bit ASCII characters.

User Name

If the User Name flag is set, this is the next UTF-encoded string. The user name identifies the name of the user who is connecting, which can be used for authentication. It is recommended that user names are kept to 12 characters or fewer, but it is not required.

Note that, for compatibility with the original MQTT V3 specification, the Remaining Length field from the fixed header takes precedence over the User Name flag. Server implementations must allow for the possibility that the User Name flag is set, but the User Name string is missing. This is valid, and connections should be allowed to continue.

Password

If the Password flag is set, this is the next UTF-encoded string. The password corresponding to the user who is connecting, which can be used for authentication. It is recommended that passwords are kept to 12 characters or fewer, but it is not required.

Note that, for compatibility with the original MQTT V3 specification, the Remaining Length field from the fixed header takes precedence over the Password flag. Server implementations must allow for the possibility that the Password flag is set, but the Password string is missing. This is valid, and connections should be allowed to continue.

Response

The server sends a CONNACK message in response to a CONNECT message from a client.

If the server does not receive a CONNECT message within a reasonable amount of time after the TCP/IP connection is established, the server should close the connection.

If the client does not receive a CONNACK message from the server within a reasonable

amount of time, the client should close the TCP/IP socket connection, and restart the session by opening a new socket to the server and issuing a CONNECT message.

In both of these scenarios, a "reasonable" amount of time depends on the type of application and the communications infrastructure.

If a client with the same Client ID is already connected to the server, the "older" client must be disconnected by the server before completing the CONNECT flow of the new client.

If the client sends an invalid CONNECT message, the server should close the connection. This includes CONNECT messages that provide invalid Protocol Name or Protocol Version Numbers. If the server can parse enough of the CONNECT message to determine that an invalid protocol has been requested, it may try to send a CONNACK containing the "Connection Refused: unacceptable protocol version" code before dropping the connection.

3.2. CONNACK - Acknowledge connection request

The CONNACK message is the message sent by the server in response to a CONNECT request from a client.

Fixed header

The fixed header format is shown in the table below.

bit	7	6	5	4	3	2	1	0
byte 1	Message type (2)				DUP flag	QoS flags		RETAIN
	0	0	1	0	x	x	x	x
byte 2	Remaining Length (2)							
	0	0	0	0	0	0	1	0

The DUP, QoS and RETAIN flags are not used in the CONNACK message.

Variable header

The variable header format is shown in the table below.

	Description	7	6	5	4	3	2	1	0
Topic Name Compression Response									
byte 1	Reserved values. Not used.	x	x	x	x	x	x	x	x
Connect Return Code									

	Description	7	6	5	4	3	2	1	0
byte 2	Return Code								

The values for the one byte unsigned Connect return code field are shown in the table below.

Enumeration	HEX	Meaning
0	0x00	Connection Accepted
1	0x01	Connection Refused: unacceptable protocol version
2	0x02	Connection Refused: identifier rejected
3	0x03	Connection Refused: server unavailable
4	0x04	Connection Refused: bad user name or password
5	0x05	Connection Refused: not authorized
6-255		Reserved for future use

Return code 2 (identifier rejected) is sent if the unique client identifier is not between 1 and 23 characters in length.

Payload

There is no payload.

3.3. PUBLISH - Publish message

A PUBLISH message is sent by a client to a server for distribution to interested subscribers. Each PUBLISH message is associated with a topic name (also known as the Subject or Channel). This is a hierarchical name space that defines a taxonomy of information sources for which subscribers can register an interest. A message that is published to a specific topic name is delivered to connected subscribers for that topic.

If a client subscribes to one or more topics, any message published to those topics are sent by the server to the client as a PUBLISH message.

Fixed header

The table below shows the fixed header format.

bit	7	6	5	4	3	2	1	0
byte 1	Message type (3)				DUP flag	QoS level		RETAIN
	0	0	1	1	0	0	1	0

bit	7	6	5	4	3	2	1	0
byte 2	Remaining Length							

QoS level

Set to 1. See QoS for more details.

DUP flag

Set to zero (0). This means that the message is being sent for the first time. See DUP for more details.

RETAIN flag

Set to zero. This means do not retain. See Retain for more details.

Remaining Length field

The length of the variable header plus the length of the payload. It can be a multibyte field.

Variable header

The variable header contains the following fields:

Topic name

A UTF-encoded string.

This must not contain Topic wildcard characters.

When received by a client that subscribed using wildcard characters, this string will be the absolute topic specified by the originating publisher and *not* the subscription string used by the client.

Message ID

Present for messages with QoS level 1 and QoS level 2. See Message identifiers for more details.

The table below shows an example variable header for a PUBLISH message.

Field	Value
Topic Name:	"a/b"
QoS level	1
Message ID:	10

The format of the variable header in this case is shown in the table below.

	Description	7	6	5	4	3	2	1	0

	Description	7	6	5	4	3	2	1	0
Topic Name									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (3)	0	0	0	0	0	0	1	1
byte 3	'a' (0x61)	0	1	1	0	0	0	0	1
byte 4	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 5	'b' (0x62)	0	1	1	0	0	0	1	0
Message Identifier									
byte 6	Message ID MSB (0)	0	0	0	0	0	0	0	0
byte 7	Message ID LSB (10)	0	0	0	0	1	0	1	0

Payload

Contains the data for publishing. The content and format of the data is application specific. The Remaining Length field in the fixed header includes both the variable header length and the payload length. As such, it is valid for a PUBLISH to contain a 0-length payload.

Response

The response to a PUBLISH message depends on the QoS level. The table below shows the expected responses.

QoS Level	Expected response
QoS 0	None
QoS 1	PUBACK
QoS 2	PUBREC

Actions

PUBLISH messages can be sent either from a publisher to the server, or from the server to a subscriber. The action of the recipient when it receives a message depends on the QoS level of the message:

QoS 0

Make the message available to any interested parties.

QoS 1

Log the message to persistent storage, make it available to any interested parties, and return a PUBACK message to the sender.

QoS 2

Log the message to persistent storage, do not make it available to interested parties yet, and return a PUBREC message to the sender.

If the server receives the message, interested parties means subscribers to the topic of the PUBLISH message. If a subscriber receives the message, interested parties means the application on the client which has subscribed to one or more topics, and is waiting for a message from the server.

See Quality of Service levels and flows for more details.

Note that if a server implementation does not authorize a PUBLISH to be made by a client, it has no way of informing that client. It must therefore make a positive acknowledgement, according to the normal QoS rules, and the client will *not* be informed that it was not authorized to publish the message.

3.4. PUBACK - Publish acknowledgment

A PUBACK message is the response to a PUBLISH message with QoS level 1. A PUBACK message is sent by a server in response to a PUBLISH message from a publishing client, and by a subscriber in response to a PUBLISH message from the server.

Fixed header

The table below shows the format of the fixed header.

bit	7	6	5	4	3	2	1	0
byte 1	Message Type (4)				DUP flag	QoS level		RETAIN
	0	1	0	0	x	x	x	x
byte 2	Remaining Length (2)							
	0	0	0	0	0	0	1	0

QoS level

Not used.

DUP flag

Not used.

RETAIN flag

Not used.

Remaining Length field

This is the length of the variable header (2 bytes). It can be a multibyte field.

Variable header

Contains the Message Identifier (Message ID) for the PUBLISH message that is being acknowledged. The table below shows the format of the variable header.

bit	7	6	5	4	3	2	1	0
byte 1	Message ID MSB							
byte 2	Message ID LSB							

Payload

There is no payload.

Actions

When the client receives the PUBACK message, it discards the original message, because it is also received (and logged) by the server.

3.5. PUBREC - Assured publish received (part 1)

A PUBREC message is the response to a PUBLISH message with QoS level 2. It is the second message of the QoS level 2 protocol flow. A PUBREC message is sent by the server in response to a PUBLISH message from a publishing client, or by a subscriber in response to a PUBLISH message from the server.

Fixed header

The table below shows the fixed header format.

bit	7	6	5	4	3	2	1	0
byte 1	Message Type (5)				DUP flag	QoS level		RETAIN
	0	1	0	1	x	x	x	x
byte 2	Remaining Length (2)							
	0	0	0	0	0	0	1	0

QoS level

Not used.

DUP flag

Not used.

RETAIN flag

Not used.

Remaining Length field

The length of the variable header (2 bytes). It can be a multibyte field.

Variable header

The variable header contains the Message ID for the acknowledged PUBLISH. The table below shows the format of the variable header.

bit	7	6	5	4	3	2	1	0
byte 1	Message ID MSB							
byte 2	Message ID LSB							

Payload

There is no payload.

Actions

When it receives a PUBREC message, the recipient sends a PUBREL message to the sender with the same Message ID as the PUBREC message.

3.6. PUBREL - Assured Publish Release (part 2)

A PUBREL message is the response either from a publisher to a PUBREC message from the server, or from the server to a PUBREC message from a subscriber. It is the third message in the QoS 2 protocol flow.

Fixed header

The table below shows the fixed header format.

bit	7	6	5	4	3	2	1	0
byte 1	Message Type (6)				DUP flag	QoS level		RETAIN
	0	1	1	0	0	0	1	x
byte 2	Remaining Length (2)							
	0	0	0	0	0	0	1	0

QoS level

PUBREL messages use QoS level 1 as an acknowledgement is expected in the form of a PUBCOMP. Retries are handled in the same way as PUBLISH messages.

DUP flag

Set to zero (0). This means that the message is being sent for the first time. See DUP for more details.

RETAIN flag

Not used.

Remaining Length field

The length of the variable header (2 bytes). It can be a multibyte field.

Variable header

The variable header contains the same Message ID as the PUBREC message that is being acknowledged. The table below shows the format of the variable header.

bit	7	6	5	4	3	2	1	0
byte 1	Message ID MSB							
byte 2	Message ID LSB							

Payload

There is no payload.

Actions

When the server receives a PUBREL message from a publisher, the server makes the original message available to interested subscribers, and sends a PUBCOMP message with the same Message ID to the publisher. When a subscriber receives a PUBREL message from the server, the subscriber makes the message available to the subscribing application and sends a PUBCOMP message to the server.

3.7. PUBCOMP - Assured publish complete (part 3)

This message is either the response from the server to a PUBREL message from a publisher, or the response from a subscriber to a PUBREL message from the server. It is the fourth and last message in the QoS 2 protocol flow.

Fixed header

The table below shows the fixed header format.

bit	7	6	5	4	3	2	1	0
byte 1	Message Type (7)				DUP flag	QoS level		RETAIN

bit	7	6	5	4	3	2	1	0
	0	1	1	1	x	x	x	x
byte 2	Remaining Length (2)							
	0	0	0	0	0	0	1	0

QoS level

Not used.

DUP flag

Not used.

RETAIN flag

Not used.

Remaining Length field

The length of the variable header (2 bytes). It can be a multibyte field.

Variable header

The variable header contains the same Message ID as the acknowledged PUBREL message.

bit	7	6	5	4	3	2	1	0
byte 1	Message ID MSB							
byte 2	Message ID LSB							

Payload

There is no payload.

Actions

When the client receives a PUBCOMP message, it discards the original message because it has been delivered, exactly once, to the server.

3.8. SUBSCRIBE - Subscribe to named topics

The SUBSCRIBE message allows a client to register an interest in one or more topic names with the server. Messages published to these topics are delivered from the server to the client as PUBLISH messages. The SUBSCRIBE message also specifies the QoS level at which the subscriber wants to receive published messages.

Fixed header

The table below shows the fixed header format.

bit	7	6	5	4	3	2	1	0
byte 1	Message Type (8)				DUP flag	QoS level		RETAIN
	1	0	0	0	0	0	1	x
byte 2	Remaining Length							

QoS level

SUBSCRIBE messages use QoS level 1 to acknowledge multiple subscription requests. The corresponding SUBACK message is identified by matching the Message ID. Retries are handled in the same way as PUBLISH messages.

DUP flag

Set to zero (0). This means that the message is being sent for the first time. See DUP for more details.

RETAIN flag

Not used.

Remaining Length field

The length of the payload. It can be a multibyte field.

Variable header

The variable header contains a Message ID because a SUBSCRIBE message has a QoS level of 1. See Message identifiers for more details.

The table below shows an example format for the variable header with a Message ID of 10.

	Description	7	6	5	4	3	2	1	0
Message Identifier									
byte 1	Message ID MSB (0)	0	0	0	0	0	0	0	0
byte 2	Message ID LSB (10)	0	0	0	0	1	0	1	0

Payload

The payload of a SUBSCRIBE message contains a list of topic names to which the client wants to subscribe, and the QoS level at which the client wants to receive the messages. The strings are UTF-encoded, and the QoS level occupies 2 bits of a single byte. The topic strings may contain special Topic wildcard characters to represent a set

of topics. These topic/QoS pairs are packed contiguously as shown in the example payload in the table below.

Topic name	"a/b"
Requested QoS	1
Topic name	"c/d"
Requested QoS	2

Topic names in a SUBSCRIBE message are not compressed.

The format of the example payload is shown in the table below.

	Description	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Topic name									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (3)	0	0	0	0	0	0	1	1
byte 3	'a' (0x61)	0	1	1	0	0	0	0	1
byte 4	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 5	'b' (0x62)	0	1	1	0	0	0	1	0
Requested QoS									
byte 6	Requested QoS (1)	x	x	x	x	x	x	0	1
Topic Name									
byte 7	Length MSB (0)	0	0	0	0	0	0	0	0
byte 8	Length LSB (3)	0	0	0	0	0	0	1	1
byte 9	'c' (0x63)	0	1	1	0	0	0	1	1
byte 10	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 11	'd' (0x64)	0	1	1	0	0	1	0	0
Requested QoS									
byte 12	Requested QoS (2)	x	x	x	x	x	x	1	0

Assuming that the requested QoS level is granted, the client receives PUBLISH messages at less than or equal to this level, depending on the QoS level of the original message from the publisher. For example, if a client has a QoS level 1 subscription to a particular topic, then a QoS level 0 PUBLISH message to that topic is delivered to the client at QoS level 0. A QoS level 2 PUBLISH message to the same topic is downgraded to QoS level 1 for delivery to the client.

A corollary to this is that subscribing to a topic at QoS level 2 is equivalent to saying "I

would like to receive messages on this topic at the QoS at which they are published".

This means a publisher is responsible for determining the maximum QoS a message can be delivered at, but a subscriber is able to downgrade the QoS to one more suitable for its usage. The QoS of a message is never upgraded.

The Requested QoS field is encoded in the byte following each UTF-encoded topic name as shown in the table below.

bit	7	6	5	4	3	2	1	0
	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	QoS level	
	x	x	x	x	x	x		

The upper 6 bits of this byte are not used in the current version of the protocol. They are reserved for future use.

A request with both QoS level bits set should be considered an invalid packet and the connection closed.

Response

When it receives a SUBSCRIBE message from a client, the server responds with a SUBACK message.

A server may start sending PUBLISH messages due to the subscription before the client receives the SUBACK message.

Note that if a server implementation does not authorize a SUBSCRIBE request to be made by a client, it has no way of informing that client. It must therefore make a positive acknowledgement with a SUBACK, and the client will *not* be informed that it was not authorized to subscribe.

A server may chose to grant a lower level of QoS than the client requested. This could happen if the server is not able to provide the higher levels of QoS. For example, if the server does not provider a reliable persistence mechanism it may chose to only grant subscriptions at QoS 0.

3.9. SUBACK - Subscription acknowledgement

A SUBACK message is sent by the server to the client to confirm receipt of a SUBSCRIBE message.

A SUBACK message contains a list of granted QoS levels. The order of granted QoS levels in the SUBACK message matches the order of the topic names in the corresponding SUBSCRIBE message.

Fixed header

The table below shows the fixed header format.

bit	7	6	5	4	3	2	1	0
byte 1	Message Type (9)				DUP flag	QoS level		RETAIN
	1	0	0	1	x	x	x	x
byte 2	Remaining Length							

QoS level

Not used.

DUP flag

Not used.

RETAIN flag

Not used.

Remaining Length field

The length of the payload. It can be a multibyte field.

Variable header

The variable header contains the Message ID for the SUBSCRIBE message that is being acknowledged. The table below shows the format of the variable header.

	7	6	5	4	3	2	1	0
byte 1	Message ID MSB							
byte 2	Message ID LSB							

Payload

The payload contains a vector of granted QoS levels. Each level corresponds to a topic name in the corresponding SUBSCRIBE message. The order of QoS levels in the SUBACK message matches the order of topic name and Requested QoS pairs in the SUBSCRIBE message. The Message ID in the variable header enables you to match SUBACK messages with the corresponding SUBSCRIBE messages.

The table below shows the Granted QoS field encoded in a byte.

bit	7	6	5	4	3	2	1	0
	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	QoS level	
	x	x	x	x	x	x		

The upper 6 bits of this byte are not used in the current version of the protocol. They are reserved for future use.

The table below shows an example payload.

Granted QoS	0
Granted QoS	2

The table below shows the format of this payload.

	Description	7	6	5	4	3	2	1	0
byte 1	Granted QoS (0)	x	x	x	x	x	x	0	0
byte 1	Granted QoS (2)	x	x	x	x	x	x	1	0

3.10. UNSUBSCRIBE - Unsubscribe from named topics

An UNSUBSCRIBE message is sent by the client to the server to unsubscribe from named topics.

Fixed header

The table below shows an example fixed header format.

bit	7	6	5	4	3	2	1	0
byte 1	Message Type (10)				DUP flag	QoS level		RETAIN
	1	0	1	0	0	0	1	x
byte 2	Remaining Length							

QoS level

UNSUBSCRIBE messages use QoS level 1 to acknowledge multiple unsubscribe requests. The corresponding UNSUBACK message is identified by the Message ID. Retries are handled in the same way as PUBLISH messages.

DUP flag

Set to zero (0). This means that the message is being sent for the first time. See DUP for more details.

RETAIN flag

Not used.

Remaining Length

This is the length of the Payload. It can be a multibyte field.

Variable header

The variable header contains a Message ID because an UNSUBSCRIBE message has a QoS level of 1. See Message identifiers for more details.

The table below shows an example format for the variable header with a Message ID of 10.

	Description	7	6	5	4	3	2	1	0
Message Identifier									
byte 1	Message ID MSB (0)	0	0	0	0	0	0	0	0
byte 2	Message ID LSB (10)	0	0	0	0	1	0	1	0

Payload

The client unsubscribes from the list of topics named in the payload. The strings are UTF-encoded and are packed contiguously. Topic names in a UNSUBSCRIBE message are not compressed. The table below shows an example payload.

Topic Name	"a/b"
Topic Name	"c/d"

The table below shows the format of this payload.

	Description	7	6	5	4	3	2	1	0
Topic Name									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (3)	0	0	0	0	0	0	1	1
byte 3	'a' (0x61)	0	1	1	0	0	0	0	1
byte 4	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 5	'b' (0x62)	0	1	1	0	0	0	1	0
Topic Name									
byte 6	Length MSB (0)	0	0	0	0	0	0	0	0
byte 7	Length LSB (3)	0	0	0	0	0	0	1	1
byte 8	'c' (0x63)	0	1	1	0	0	0	1	1
byte 9	'/' (0x2F)	0	0	1	0	1	1	1	1
byte 10	'd' (0x64)	0	1	1	0	0	1	0	0

Response

The server sends an UNSUBACK to a client in response to an UNSUBSCRIBE message.

3.11. UNSUBACK - Unsubscribe acknowledgment

The UNSUBACK message is sent by the server to the client to confirm receipt of an UNSUBSCRIBE message.

Fixed header

The table below shows the fixed header format.

bit	7	6	5	4	3	2	1	0
byte 1	Message Type (11)				DUP flag	QoS level		RETAIN
	1	0	1	1	x	x	x	x
byte 2	Remaining length (2)							
	0	0	0	0	0	0	1	0

QoS level

Not used.

DUP flag

Not used.

RETAIN flag

Not used.

Remaining Length

The length of the Variable Header (2 bytes).

Variable header

The variable header contains the Message ID for the UNSUBSCRIBE message that is being acknowledged. The table below shows the format of the variable header.

bit	7	6	5	4	3	2	1	0
byte 1	Message ID MSB							
byte 2	Message ID LSB							

Payload

There is no payload.

3.12. PINGREQ - PING request

The PINGREQ message is an "are you alive?" message that is sent from a connected client to the server.

See Keep Alive timer for more details.

Fixed header

The table below shows the fixed header format.

bit	7	6	5	4	3	2	1	0
byte 1	Message Type (12)				DUP flag	QoS level		RETAIN
	1	1	0	0	x	x	x	x
byte 2	Remaining Length (0)							
	0	0	0	0	0	0	0	0

The DUP, QoS, and RETAIN flags are not used.

Variable header

There is no variable header.

Payload

There is no payload.

Response

The response to a PINGREQ message is a PINGRESP message.

3.13. PINGRESP - PING response

A PINGRESP message is the response sent by a server to a PINGREQ message and means "yes I am alive".

See Keep Alive timer for more details.

Fixed header

The table below shows the fixed header format:

bit	7	6	5	4	3	2	1	0
byte 1	Message Type (13)				DUP flag	QoS level		RETAIN
	1	1	0	1	x	x	x	x

bit	7	6	5	4	3	2	1	0
byte 2	Remaining Length (0)							
	0	0	0	0	0	0	0	0

The DUP, QoS, and RETAIN flags are not used.

Payload

There is no payload.

Variable header

There is no variable header.

3.14. DISCONNECT - Disconnect notification

The DISCONNECT message is sent from the client to the server to indicate that it is about to close its TCP/IP connection. This allows for a clean disconnection, rather than just dropping the line.

If the client had connected with the clean session flag set, then all previously maintained information about the client will be discarded.

A server should not rely on the client to close the TCP/IP connection after receiving a DISCONNECT.

Fixed header

The fixed header format is shown in the table below.

bit	7	6	5	4	3	2	1	0
byte 1	Message Type (14)				DUP flag	QoS level		RETAIN
	1	1	1	0	x	x	x	x
byte 2	Remaining Length (0)							
	0	0	0	0	0	0	0	0

The DUP, QoS, and RETAIN flags are not used in the DISCONNECT message.

Payload

There is no payload.

Variable header

There is no variable header.

4. Flows

4.1. Quality of Service levels and flows

MQTT delivers messages according to the levels defined in a Quality of Service (QoS). The levels are described below:

QoS level 0: At most once delivery

The message is delivered according to the best efforts of the underlying TCP/IP network. A response is not expected and no retry semantics are defined in the protocol. The message arrives at the server either once or not at all.

The table below shows the QoS level 0 protocol flow.

Client	Message and direction	Server
QoS = 0	PUBLISH ---------->	**Action:** Publish message to subscribers

QoS level 1: At least once delivery

The receipt of a message by the server is acknowledged by a PUBACK message. If there is an identified failure of either the communications link or the sending device, or the acknowledgement message is not received after a specified period of time, the sender resends the message with the DUP bit set in the message header. The message arrives at the server at least once. Both SUBSCRIBE and UNSUBSCRIBE messages use QoS level 1.

A message with QoS level 1 has a Message ID in the message header.

The table below shows the QoS level 1 protocol flow.

Client	Message and direction	Server
QoS = 1 DUP = 0 Message ID = x **Action:** Store message	PUBLISH ---------->	**Actions:** • Store message • Publish message to subscribers • Delete message
Action: Discard message	PUBACK <----------	

If the client does not receive a PUBACK message (either within a time period defined in the application, or if a failure is detected and the communications session is restarted), the client may resend the PUBLISH message with the DUP flag set.

When it receives a duplicate message from the client, the server republishes the message to the subscribers, and sends another PUBACK message.

QoS level 2: Exactly once delivery

Additional protocol flows above QoS level 1 ensure that duplicate messages are not delivered to the receiving application. This is the highest level of delivery, for use when duplicate messages are not acceptable. There is an increase in network traffic, but it is usually acceptable because of the importance of the message content.

A message with QoS level 2 has a Message ID in the message header.

The table below shows the QoS level 2 protocol flow. There are two semantics available for how a PUBLISH flow should be handled by the recipient. They affect the point within the flow that the message is made available to the subscribers. The choice of semantic is implementation specific and does not affect the guarantees of a QoS level 2 flow.

Client	Message and direction	Server
QoS = 2 DUP = 0 Message ID = x **Action:** Store message	PUBLISH ---------->	**Action:** Store message *or* **Actions:** • Store message ID • Publish message to subscribers
	PUBREC <----------	Message ID = x
Message ID = x	PUBREL ---------->	**Actions:** • Publish message to subscribers • Delete message *or* **Action:** Delete message ID
Action: Discard message	PUBCOMP <----------	Message ID = x

If a failure is detected, or after a defined time period, the protocol flow is retried from the last unacknowledged protocol message; either the PUBLISH or PUBREL.

See Message delivery retry for more details. The additional protocol flows ensure that the message is delivered to subscribers once only.

Assumptions for QoS levels 1 and 2

In any network, it is possible for devices or communication links to fail. If this happens, one end of the link might not know what is happening at the other end; these are known as *in doubt* windows. In these scenarios assumptions have to be made about the reliability of the devices and networks involved in message delivery.

MQTT assumes that the client and server are generally reliable, and that the communications channel is more likely to be unreliable. If the client device fails, it is typically a catastrophic failure, rather than a transient one. The possibility of recovering data from the device is low. Some devices have non-volatile storage, for example flash ROM. The provision of more persistent storage on the client device protects the most critical data from some modes of failure.

Beyond the basic failure of the communications link, the failure mode matrix becomes complex, resulting in more scenarios than the specification for MQTT can handle.

4.2. Message delivery retry

Although TCP normally guarantees delivery of packets, there are certain scenarios where an MQTT message may not be received. In the case of MQTT messages that expect a response (QoS >0 PUBLISH, PUBREL, SUBSCRIBE, UNSUBSCRIBE), if the response is not received within a certain time period, the sender may retry delivery. The sender should set the DUP flag on the message.

The retry timeout should be a configurable option. However care must be taken to ensure message delivery does not timeout while it is still being sent. For example, sending a large message over a slow network will naturally take longer than a small message over a fast network. Repeatedly retrying a timed-out message could often make matters worse so a strategy of increasing the timeout value across multiple retries should be used.

When a client reconnects, if it is not marked clean session, both the client and server should redeliver any previous in-flight messages.

Other than this "on reconnect" retry behaviour, clients are not required to retry message delivery. Brokers, however, should retry any unacknowledged message.

4.3. Message ordering

Message ordering can be affected by a number of factors, including how many in-flight PUBLISH flows a client allows and whether the client is single- or multi-threaded. For purposes of discussion, clients are assumed to be single-threaded at the point packets

are written to and read from the network.

For an implementation to provide any guarantees regarding the ordering of messages it must ensure each stage of the message delivery flows are completed in the order they were started. For example, in a series of QoS level 2 flows, the PUBREL flows must be sent in the same order as the original PUBLISH flows:

Client	Message and direction	Server
	PUBLISH 1 ---------->	
	PUBLISH 2 ---------->	
	PUBLISH 3 ---------->	
	PUBREC 1 <----------	
	PUBREC 2 <----------	
	PUBREL 1 ---------->	
	PUBREC 3 <----------	
	PUBREL 2 ---------->	
	PUBCOMP 1 <----------	
	PUBREL 3 ---------->	
	PUBCOMP 2 <----------	
	PUBCOMP 3 <----------	

The number of in-flight messages permitted also has an effect on the type of guarantees that can be made:

- With an in-flight window of 1, each delivery flow is completed before the next one starts. This guarantees messages are delivered in the order they were submitted.
- With an in-flight window greater than 1, message ordering can only be guaranteed within the QoS level.

Appendix A - Topic wildcards

A subscription may contain special characters, which allow you to subscribe to multiple topics at once.

The topic level separator is used to introduce structure into the topic, and can therefore be specified within the topic for that purpose. The multi-level wildcard and single-level wildcard can be used for subscriptions, but they cannot be used within a topic by the publisher of a message.

Topic level separator

The forward slash (/) is used to separate each level within a topic tree and provide a hierarchical structure to the topic space. The use of the topic level separator is significant when the two wildcard characters are encountered in topics specified by subscribers.

Multi-level wildcard

The number sign (#) is a wildcard character that matches any number of levels within a topic. For example, if you subscribe to finance/stock/ibm/#, you receive messages on these topics:

```
finance/stock/ibm
finance/stock/ibm/closingprice
finance/stock/ibm/currentprice
```

The multi-level wildcard can represent zero or more levels. Therefore, *finance/#* can also match the singular *finance*, where # represents zero levels. The topic level separator is meaningless in this context, because there are no levels to separate.

The multi-level wildcard can be specified only on its own or next to the topic level separator character. Therefore, # and *finance/#* are both valid, but *finance#* is not valid. The multi-level wildcard must be the last character used within the topic tree. For example, *finance/#* is valid but *finance/#/closingprice* is not valid.

Single-level wildcard

The plus sign (+) is a wildcard character that matches only one topic level. For example, *finance/stock/+* matches *finance/stock/ibm* and *finance/stock/xyz*, but not *finance/stock/ibm/closingprice*. Also, because the single-level wildcard matches only a single level, *finance/+* does not match *finance*.

The single-level wildcard can be used at any level in the topic tree, and in conjunction with the multilevel wildcard. It must be used next to the topic level separator, except when it is specified on its own. Therefore, + and *finance/+* are both valid, but *finance+* is not valid. The single-level wildcard can be used at the end of the topic tree or within the topic tree. For example, *finance/+* and *finance/+/ibm* are both valid.

Topic semantics and usage

When you build an application, the design of the topic tree should take into account the following principles of topic name syntax and semantics:

- A topic must be at least one character long.

- Topic names are case sensitive. For example, *ACCOUNTS* and *Accounts* are two different topics.
- Topic names can include the space character. For example, *Accounts payable* is a valid topic.
- A leading "/" creates a distinct topic. For example, */finance* is different from *finance*. */finance* matches "+/+" and "/+", but not "+".
- Do not include the null character (Unicode `\x0000`) in any topic.

The following principles apply to the construction and content of a topic tree:

- The length is limited to 64k but within that there are no limits to the number of levels in a topic tree.
- There can be any number of root nodes; that is, there can be any number of topic trees.