



# MQTT Version 5.0

## Candidate OASIS Standard 02

11 February 2019

### Specification URIs

#### This version:

<https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.docx> (Authoritative)  
<https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.html>  
<https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.pdf>

#### Previous version:

<http://docs.oasis-open.org/mqtt/mqtt/v5.0/cos01/mqtt-v5.0-cos01.docx> (Authoritative)  
<http://docs.oasis-open.org/mqtt/mqtt/v5.0/cos01/mqtt-v5.0-cos01.html>  
<http://docs.oasis-open.org/mqtt/mqtt/v5.0/cos01/mqtt-v5.0-cos01.pdf>

#### Latest version:

<https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.docx> (Authoritative)  
<https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>  
<https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf>

#### Technical Committee:

OASIS Message Queuing Telemetry Transport (MQTT) TC

#### Chairs:

Richard Coppen ([coppen@uk.ibm.com](mailto:coppen@uk.ibm.com)), IBM

#### Editors:

Andrew Banks ([andrew\\_banks@uk.ibm.com](mailto:andrew_banks@uk.ibm.com)), IBM  
Ed Briggs ([edbriggs@microsoft.com](mailto:edbriggs@microsoft.com)), Microsoft  
Ken Borgendale ([kwb@us.ibm.com](mailto:kwb@us.ibm.com)), IBM  
Rahul Gupta ([rahul.gupta@us.ibm.com](mailto:rahul.gupta@us.ibm.com)), IBM

#### Related work:

This specification replaces or supersedes:

- *MQTT Version 3.1.1*. Edited by Andrew Banks and Rahul Gupta. 29 October 2014. OASIS Standard. <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>.

This specification is related to:

- *MQTT and the NIST Cybersecurity Framework Version 1.0*. Edited by Geoff Brown and Louis-Philippe Lamoureux. Latest version: <http://docs.oasis-open.org/mqtt/mqtt-nist-cybersecurity/v1.0/mqtt-nist-cybersecurity-v1.0.html>.

#### Abstract:

MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium.

The protocol runs over TCP/IP, or over other network protocols that provide ordered, lossless, bi-directional connections. Its features include:

- Use of the publish/subscribe message pattern which provides one-to-many message distribution and decoupling of applications.
- A messaging transport that is agnostic to the content of the payload.
- Three qualities of service for message delivery:
  - "At most once", where messages are delivered according to the best efforts of the operating environment. Message loss 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 can occur.
  - "Exactly once", where messages 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 and protocol exchanges minimized to reduce network traffic.
- A mechanism to notify interested parties when an abnormal disconnection occurs.

#### Status:

This document was last revised or approved by the OASIS Message Queuing Telemetry Transport (MQTT) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at [https://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=mqtt#technical](https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt#technical).

TC members should send comments on this specification to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "[Send A Comment](#)" button on the TC's web page at <https://www.oasis-open.org/committees/mqtt/>.

This specification is provided under the [Non-Assertion](#) Mode of the [OASIS IPR Policy](#), the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (<https://www.oasis-open.org/committees/mqtt/ipr.php>).

Note that any machine-readable content ([Computer Language Definitions](#)) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

#### Citation format:

When referencing this specification the following citation format should be used:

##### [mqtt-v5.0]

*MQTT Version 5.0*. Edited by Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. 11 February 2019. Candidate OASIS Standard 02. <https://docs.oasis-open.org/mqtt/mqtt/v5.0/cos02/mqtt-v5.0-cos02.html>. Latest version: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>.

---

## Notices

Copyright © OASIS Open 2019. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full [Policy](#) may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of [OASIS](#), the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see <https://www.oasis-open.org/policies-guidelines/trademark> for above guidance.

---

# Table of Contents

1	Introduction .....	11
1.0	Intellectual property rights policy .....	11
1.1	Organization of the MQTT specification .....	11
1.2	Terminology .....	11
1.3	Normative references .....	13
1.4	Non-normative references .....	13
1.5	Data representation .....	16
1.5.1	Bits .....	16
1.5.2	Two Byte Integer .....	16
1.5.3	Four Byte Integer .....	16
1.5.4	UTF-8 Encoded String .....	16
1.5.5	Variable Byte Integer .....	18
1.5.6	Binary Data .....	19
1.5.7	UTF-8 String Pair .....	19
1.6	Security .....	19
1.7	Editing convention .....	20
1.8	Change history .....	20
1.8.1	MQTT v3.1.1 .....	20
1.8.2	MQTT v5.0 .....	20
2	MQTT Control Packet format .....	21
2.1	Structure of an MQTT Control Packet .....	21
2.1.1	Fixed Header .....	21
2.1.2	MQTT Control Packet type .....	21
2.1.3	Flags .....	22
2.1.4	Remaining Length .....	23
2.2	Variable Header .....	23
2.2.1	Packet Identifier .....	23
2.2.2	Properties .....	25
2.2.2.1	Property Length .....	25
2.2.2.2	Property .....	25
2.3	Payload .....	26
2.4	Reason Code .....	27
3	MQTT Control Packets .....	30
3.1	CONNECT – Connection Request .....	30
3.1.1	CONNECT Fixed Header .....	30
3.1.2	CONNECT Variable Header .....	30
3.1.2.1	Protocol Name .....	30
3.1.2.2	Protocol Version .....	31
3.1.2.3	Connect Flags .....	31
3.1.2.4	Clean Start .....	32
3.1.2.5	Will Flag .....	32
3.1.2.6	Will QoS .....	33
3.1.2.7	Will Retain .....	33
3.1.2.8	User Name Flag .....	33

3.1.2.9 Password Flag .....	33
3.1.2.10 Keep Alive.....	34
3.1.2.11 CONNECT Properties.....	34
3.1.2.11.1 Property Length.....	34
3.1.2.11.2 Session Expiry Interval.....	35
3.1.2.11.3 Receive Maximum.....	36
3.1.2.11.4 Maximum Packet Size.....	36
3.1.2.11.5 Topic Alias Maximum .....	37
3.1.2.11.6 Request Response Information.....	37
3.1.2.11.7 Request Problem Information.....	37
3.1.2.11.8 User Property .....	38
3.1.2.11.9 Authentication Method.....	38
3.1.2.11.10 Authentication Data .....	38
3.1.2.12 Variable Header non-normative example.....	39
3.1.3 CONNECT Payload.....	40
3.1.3.1 Client Identifier (ClientID) .....	40
3.1.3.2 Will Properties.....	40
3.1.3.2.1 Property Length.....	41
3.1.3.2.2 Will Delay Interval .....	41
3.1.3.2.3 Payload Format Indicator .....	41
3.1.3.2.4 Message Expiry Interval.....	41
3.1.3.2.5 Content Type.....	42
3.1.3.2.6 Response Topic .....	42
3.1.3.2.7 Correlation Data .....	42
3.1.3.2.8 User Property .....	42
3.1.3.3 Will Topic .....	43
3.1.3.4 Will Payload .....	43
3.1.3.5 User Name.....	43
3.1.3.6 Password .....	43
3.1.4 CONNECT Actions .....	43
3.2 CONNACK – Connect acknowledgement .....	44
3.2.1 CONNACK Fixed Header .....	45
3.2.2 CONNACK Variable Header .....	45
3.2.2.1 Connect Acknowledge Flags.....	45
3.2.2.1.1 Session Present.....	45
3.2.2.2 Connect Reason Code.....	46
3.2.2.3 CONNACK Properties.....	47
3.2.2.3.1 Property Length.....	47
3.2.2.3.2 Session Expiry Interval.....	47
3.2.2.3.3 Receive Maximum.....	48
3.2.2.3.4 Maximum QoS .....	48
3.2.2.3.5 Retain Available .....	49
3.2.2.3.6 Maximum Packet Size.....	49
3.2.2.3.7 Assigned Client Identifier .....	49
3.2.2.3.8 Topic Alias Maximum .....	50
3.2.2.3.9 Reason String .....	50
3.2.2.3.10 User Property .....	50
3.2.2.3.11 Wildcard Subscription Available .....	50
3.2.2.3.12 Subscription Identifiers Available .....	51

3.2.2.3.13 Shared Subscription Available .....	51
3.2.2.3.14 Server Keep Alive .....	51
3.2.2.3.15 Response Information .....	52
3.2.2.3.16 Server Reference .....	52
3.2.2.3.17 Authentication Method.....	52
3.2.2.3.18 Authentication Data .....	52
3.2.3 CONNACK Payload .....	53
3.3 PUBLISH – Publish message .....	53
3.3.1 PUBLISH Fixed Header .....	53
3.3.1.1 DUP .....	53
3.3.1.2 QoS.....	54
3.3.1.3 RETAIN.....	54
3.3.1.4 Remaining Length.....	55
3.3.2 PUBLISH Variable Header .....	55
3.3.2.1 Topic Name .....	55
3.3.2.2 Packet Identifier .....	56
3.3.2.3 PUBLISH Properties .....	56
3.3.2.3.1 Property Length.....	56
3.3.2.3.2 Payload Format Indicator .....	56
3.3.2.3.3 Message Expiry Interval` .....	56
3.3.2.3.4 Topic Alias.....	57
3.3.2.3.5 Response Topic .....	58
3.3.2.3.6 Correlation Data .....	58
3.3.2.3.7 User Property .....	58
3.3.2.3.8 Subscription Identifier.....	59
3.3.2.3.9 Content Type.....	59
3.3.3 PUBLISH Payload .....	60
3.3.4 PUBLISH Actions .....	60
3.4 PUBACK – Publish acknowledgement .....	62
3.4.1 PUBACK Fixed Header .....	62
3.4.2 PUBACK Variable Header.....	63
3.4.2.1 PUBACK Reason Code .....	63
3.4.2.2 PUBACK Properties.....	64
3.4.2.2.1 Property Length.....	64
3.4.2.2.2 Reason String .....	64
3.4.2.2.3 User Property .....	64
3.4.3 PUBACK Payload.....	64
3.4.4 PUBACK Actions .....	65
3.5 PUBREC – Publish received (QoS 2 delivery part 1) .....	65
3.5.1 PUBREC Fixed Header.....	65
3.5.2 PUBREC Variable Header .....	65
3.5.2.1 PUBREC Reason Code .....	65
3.5.2.2 PUBREC Properties.....	66
3.5.2.2.1 Property Length.....	66
3.5.2.2.2 Reason String .....	66
3.5.2.2.3 User Property .....	67
3.5.3 PUBREC Payload .....	67
3.5.4 PUBREC Actions.....	67

3.6 PUBREL – Publish release (QoS 2 delivery part 2)	67
3.6.1 PUBREL Fixed Header	67
3.6.2 PUBREL Variable Header	67
3.6.2.1 PUBREL Reason Code	68
3.6.2.2 PUBREL Properties	68
3.6.2.2.1 Property Length	68
3.6.2.2.2 Reason String	68
3.6.2.2.3 User Property	68
3.6.3 PUBREL Payload	69
3.6.4 PUBREL Actions	69
3.7 PUBCOMP – Publish complete (QoS 2 delivery part 3)	69
3.7.1 PUBCOMP Fixed Header	69
3.7.2 PUBCOMP Variable Header	69
3.7.2.1 PUBCOMP Reason Code	70
3.7.2.2 PUBCOMP Properties	70
3.7.2.2.1 Property Length	70
3.7.2.2.2 Reason String	70
3.7.2.2.3 User Property	70
3.7.3 PUBCOMP Payload	71
3.7.4 PUBCOMP Actions	71
3.8 SUBSCRIBE - Subscribe request	71
3.8.1 SUBSCRIBE Fixed Header	71
3.8.2 SUBSCRIBE Variable Header	71
3.8.2.1 SUBSCRIBE Properties	72
3.8.2.1.1 Property Length	72
3.8.2.1.2 Subscription Identifier	72
3.8.2.1.3 User Property	72
3.8.3 SUBSCRIBE Payload	72
3.8.3.1 Subscription Options	73
3.8.4 SUBSCRIBE Actions	75
3.9 SUBACK – Subscribe acknowledgement	77
3.9.1 SUBACK Fixed Header	77
3.9.2 SUBACK Variable Header	77
3.9.2.1 SUBACK Properties	77
3.9.2.1.1 Property Length	77
3.9.2.1.2 Reason String	78
3.9.2.1.3 User Property	78
3.9.3 SUBACK Payload	78
3.10 UNSUBSCRIBE – Unsubscribe request	79
3.10.1 UNSUBSCRIBE Fixed Header	79
3.10.2 UNSUBSCRIBE Variable Header	80
3.10.2.1 UNSUBSCRIBE Properties	80
3.10.2.1.1 Property Length	80
3.10.2.1.2 User Property	80
3.10.3 UNSUBSCRIBE Payload	80
3.10.4 UNSUBSCRIBE Actions	81
3.11 UNSUBACK – Unsubscribe acknowledgement	81

3.11.1 UNSUBACK Fixed Header .....	81
3.11.2 UNSUBACK Variable Header .....	82
3.11.2.1 UNSUBACK Properties.....	82
3.11.2.1.1 Property Length.....	82
3.11.2.1.2 Reason String .....	82
3.11.2.1.3 User Property .....	82
3.11.3 UNSUBACK Payload .....	83
3.12 PINGREQ – PING request .....	83
3.12.1 PINGREQ Fixed Header .....	84
3.12.2 PINGREQ Variable Header.....	84
3.12.3 PINGREQ Payload.....	84
3.12.4 PINGREQ Actions .....	84
3.13 PINGRESP – PING response .....	84
3.13.1 PINGRESP Fixed Header .....	84
3.13.2 PINGRESP Variable Header.....	85
3.13.3 PINGRESP Payload.....	85
3.13.4 PINGRESP Actions .....	85
3.14 DISCONNECT – Disconnect notification.....	85
3.14.1 DISCONNECT Fixed Header .....	85
3.14.2 DISCONNECT Variable Header.....	85
3.14.2.1 Disconnect Reason Code .....	86
3.14.2.2 DISCONNECT Properties.....	88
3.14.2.2.1 Property Length.....	88
3.14.2.2.2 Session Expiry Interval.....	88
3.14.2.2.3 Reason String .....	88
3.14.2.2.4 User Property .....	88
3.14.2.2.5 Server Reference .....	88
3.14.3 DISCONNECT Payload.....	89
3.14.4 DISCONNECT Actions.....	89
3.15 AUTH – Authentication exchange .....	89
3.15.1 AUTH Fixed Header .....	90
3.15.2 AUTH Variable Header.....	90
3.15.2.1 Authenticate Reason Code .....	90
3.15.2.2 AUTH Properties.....	90
3.15.2.2.1 Property Length.....	90
3.15.2.2.2 Authentication Method.....	91
3.15.2.2.3 Authentication Data .....	91
3.15.2.2.4 Reason String .....	91
3.15.2.2.5 User Property .....	91
3.15.3 AUTH Payload.....	91
3.15.4 AUTH Actions.....	91
4 Operational behavior .....	92
4.1 Session State.....	92
4.1.1 Storing Session State.....	92
4.1.2 Session State non-normative examples.....	93
4.2 Network Connections.....	93
4.3 Quality of Service levels and protocol flows .....	93



4.3.1 QoS 0: At most once delivery .....	94
4.3.2 QoS 1: At least once delivery .....	94
4.3.3 QoS 2: Exactly once delivery .....	95
4.4 Message delivery retry .....	96
4.5 Message receipt .....	97
4.6 Message ordering .....	97
4.7 Topic Names and Topic Filters .....	98
4.7.1 Topic wildcards .....	98
4.7.1.1 Topic level separator .....	98
4.7.1.2 Multi-level wildcard .....	98
4.7.1.3 Single-level wildcard .....	99
4.7.2 Topics beginning with \$ .....	99
4.7.3 Topic semantic and usage .....	100
4.8 Subscriptions .....	101
4.8.1 Non-shared Subscriptions .....	101
4.8.2 Shared Subscriptions .....	101
4.9 Flow Control .....	103
4.10 Request / Response .....	104
4.10.1 Basic Request Response (non-normative) .....	104
4.10.2 Determining a Response Topic value (non-normative) .....	105
4.11 Server redirection .....	106
4.12 Enhanced authentication .....	107
4.12.1 Re-authentication .....	108
4.13 Handling errors .....	109
4.13.1 Malformed Packet and Protocol Errors .....	109
4.13.2 Other errors .....	110
5 Security (non-normative) .....	111
5.1 Introduction .....	111
5.2 MQTT solutions: security and certification .....	111
5.3 Lightweight cryptography and constrained devices .....	112
5.4 Implementation notes .....	112
5.4.1 Authentication of Clients by the Server .....	112
5.4.2 Authorization of Clients by the Server .....	112
5.4.3 Authentication of the Server by the Client .....	113
5.4.4 Integrity of Application Messages and MQTT Control Packets .....	113
5.4.5 Privacy of Application Messages and MQTT Control Packets .....	113
5.4.6 Non-repudiation of message transmission .....	114
5.4.7 Detecting compromise of Clients and Servers .....	114
5.4.8 Detecting abnormal behaviors .....	114
5.4.9 Handling of Disallowed Unicode code points .....	115
5.4.9.1 Considerations for the use of Disallowed Unicode code points .....	115
5.4.9.2 Interactions between Publishers and Subscribers .....	115
5.4.9.3 Remedies .....	116
5.4.10 Other security considerations .....	116
5.4.11 Use of SOCKS .....	116
5.4.12 Security profiles .....	117

5.4.12.1 Clear communication profile.....	117
5.4.12.2 Secured network communication profile .....	117
5.4.12.3 Secured transport profile.....	117
5.4.12.4 Industry specific security profiles .....	117
6 Using WebSocket as a network transport .....	118
6.1 IANA considerations .....	118
7 Conformance .....	119
7.1 Conformance clauses .....	119
7.1.1 MQTT Server conformance clause .....	119
7.1.2 MQTT Client conformance clause.....	119
Appendix A. Acknowledgments .....	120
Appendix B. Mandatory normative statement (non-normative) .....	121
Appendix C. Summary of new features in MQTT v5.0 (non-normative) .....	136

---

# 1 Introduction

## 1.0 Intellectual property rights policy

This specification is provided under the [Non-Assertion](#) Mode of the [OASIS IPR Policy](#), the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (<https://www.oasis-open.org/committees/mqtt/ipr.php>).

## 1.1 Organization of the MQTT specification

The specification is split into seven chapters:

- [Chapter 1 - Introduction](#)
- [Chapter 2 - MQTT Control Packet format](#)
- [Chapter 3 - MQTT Control Packets](#)
- [Chapter 4 - Operational behavior](#)
- [Chapter 5 - Security](#)
- [Chapter 6 - Using WebSocket as a network transport](#)
- [Chapter 7 - Conformance Targets](#)

## 1.2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in IETF RFC 2119 [\[RFC2119\]](#), except where they appear in text that is marked as non-normative.

### **Network Connection:**

**A construct provided by the underlying transport protocol that is being used by MQTT.**

- It connects the Client to the Server.
- It provides the means to send an ordered, lossless, stream of bytes in both directions.

Refer to [section 4.2](#) Network Connection for non-normative examples.

### **Application Message:**

The data carried by the MQTT protocol across the network for the application. When an Application Message is transported by MQTT it contains payload data, a Quality of Service (QoS), a collection of Properties, and a Topic Name.

### **Client:**

A program or device that uses MQTT. A Client:

- opens the Network Connection to the Server
- publishes Application Messages that other Clients might be interested in.
- subscribes to request Application Messages that it is interested in receiving.
- unsubscribes to remove a request for Application Messages.
- closes the Network Connection to the Server.

## **Server:**

A program or device that acts as an intermediary between Clients which publish Application Messages and Clients which have made Subscriptions. A Server:

- accepts Network Connections from Clients.
- accepts Application Messages published by Clients.
- processes Subscribe and Unsubscribe requests from Clients.
- forwards Application Messages that match Client Subscriptions.
- closes the Network Connection from the Client.

## **Session:**

A stateful interaction between a Client and a Server. Some Sessions last only as long as the Network Connection, others can span multiple consecutive Network Connections between a Client and a Server.

## **Subscription:**

A Subscription comprises a Topic Filter and a maximum QoS. A Subscription is associated with a single Session. A Session can contain more than one Subscription. Each Subscription within a Session has a different Topic Filter.

## **Shared Subscription:**

A Shared Subscription comprises a Topic Filter and a maximum QoS. A Shared Subscription can be associated with more than one Session to allow a wider range of message exchange patterns. An Application Message that matches a Shared Subscription is only sent to the Client associated with one of these Sessions. A Session can subscribe to more than one Shared Subscription and can contain both Shared Subscriptions and Subscriptions which are not shared.

## **Wildcard Subscription:**

A Wildcard Subscription is a Subscription with a Topic Filter containing one or more wildcard characters. This allows the subscription to match more than one Topic Name. Refer to [section 4.7](#) for a description of wildcard characters in a Topic Filter.

## **Topic Name:**

The label attached to an Application Message which is matched against the Subscriptions known to the Server.

## **Topic Filter:**

An expression contained in a Subscription to indicate an interest in one or more topics. A Topic Filter can include wildcard characters.

## **MQTT Control Packet:**

A packet of information that is sent across the Network Connection. The MQTT specification defines fifteen different types of MQTT Control Packet, for example the PUBLISH packet is used to convey Application Messages.

## **Malformed Packet:**

A control packet that cannot be parsed according to this specification. Refer to [section 4.13](#) for information about error handling.

## **Protocol Error:**

An error that is detected after the packet has been parsed and found to contain data that is not allowed by the protocol or is inconsistent with the state of the Client or Server. Refer to [section 4.13](#) for information about error handling.

## **Will Message:**

An Application Message which is published by the Server after the Network Connection is closed in cases where the Network Connection is not closed normally. Refer to [section 3.1.2.5](#) for information about Will Messages.

## **Disallowed Unicode code point:**

The set of Unicode Control Codes and Unicode Noncharacters which should not be included in a UTF-8 Encoded String. Refer to [section 1.5.4](#) for more information about the Disallowed Unicode code points.

# **1.3 Normative references**

## **[RFC2119]**

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>

## **[RFC3629]**

Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>

## **[RFC6455]**

Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <http://www.rfc-editor.org/info/rfc6455>

## **[Unicode]**

The Unicode Consortium. The Unicode Standard, <http://www.unicode.org/versions/latest/>

# **1.4 Non-normative references**

## **[RFC0793]**

Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>

## **[RFC5246]**

Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008,

132 <http://www.rfc-editor.org/info/rfc5246>  
133  
134 **[AES]**  
135 Advanced Encryption Standard (AES) (FIPS PUB 197).  
136 <https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf>  
137  
138 **[CHACHA20]**  
139 ChaCha20 and Poly1305 for IETF Protocols  
140 <https://tools.ietf.org/html/rfc7539>  
141  
142 **[FIPS1402]**  
143 Security Requirements for Cryptographic Modules (FIPS PUB 140-2)  
144 <https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402.pdf>  
145  
146 **[IEEE 802.1AR]**  
147 IEEE Standard for Local and metropolitan area networks - Secure Device Identity  
148 <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>  
149  
150 **[ISO29192]**  
151 ISO/IEC 29192-1:2012 Information technology -- Security techniques -- Lightweight cryptography -- Part  
152 1: General  
153 <https://www.iso.org/standard/56425.html>  
154  
155 **[MQTT NIST]**  
156 MQTT supplemental publication, MQTT and the NIST Framework for Improving Critical Infrastructure  
157 Cybersecurity  
158 <http://docs.oasis-open.org/mqtt/mqtt-nist-cybersecurity/v1.0/mqtt-nist-cybersecurity-v1.0.html>  
159  
160 **[MQTTV311]**  
161 MQTT V3.1.1 Protocol Specification  
162 <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>  
163  
164 **[ISO20922]**  
165 MQTT V3.1.1 ISO Standard (ISO/IEC 20922:2016)  
166 <https://www.iso.org/standard/69466.html>  
167  
168 **[NISTCSF]**  
169 Improving Critical Infrastructure Cybersecurity Executive Order 13636  
170 <https://www.nist.gov/sites/default/files/documents/itl/preliminary-cybersecurity-framework.pdf>  
171  
172 **[NIST7628]**  
173 NISTIR 7628 Guidelines for Smart Grid Cyber Security Catalogue  
174 [https://www.nist.gov/sites/default/files/documents/smartgrid/nistir-7628\\_total.pdf](https://www.nist.gov/sites/default/files/documents/smartgrid/nistir-7628_total.pdf)

175  
176 **[NSAB]**  
177 NSA Suite B Cryptography  
178 [http://www.nsa.gov/ia/programs/suiteb\\_cryptography/](http://www.nsa.gov/ia/programs/suiteb_cryptography/)  
179  
180 **[PCIDSS]**  
181 PCI-DSS Payment Card Industry Data Security Standard  
182 [https://www.pcisecuritystandards.org/pci\\_security/](https://www.pcisecuritystandards.org/pci_security/)  
183  
184 **[RFC1928]**  
185 Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5",  
186 RFC 1928, DOI 10.17487/RFC1928, March 1996,  
187 <http://www.rfc-editor.org/info/rfc1928>  
188  
189 **[RFC4511]**  
190 Sermersheim, J., Ed., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511,  
191 DOI 10.17487/RFC4511, June 2006,  
192 <http://www.rfc-editor.org/info/rfc4511>  
193  
194 **[RFC5280]**  
195 Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key  
196 Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280,  
197 DOI 10.17487/RFC5280, May 2008,  
198 <http://www.rfc-editor.org/info/rfc5280>  
199  
200 **[RFC6066]**  
201 Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066,  
202 DOI 10.17487/RFC6066, January 2011,  
203 <http://www.rfc-editor.org/info/rfc6066>  
204  
205 **[RFC6749]**  
206 Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October  
207 2012,  
208 <http://www.rfc-editor.org/info/rfc6749>  
209  
210 **[RFC6960]**  
211 Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public  
212 Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June  
213 2013,  
214 <http://www.rfc-editor.org/info/rfc6960>  
215  
216 **[SARBANES]**  
217 Sarbanes-Oxley Act of 2002.  
218 <http://www.gpo.gov/fdsys/pkg/PLAW-107publ204/html/PLAW-107publ204.htm>

## **[USEUPRIVSH]**

U.S.-EU Privacy Shield Framework

<https://www.privacyshield.gov>

## **[RFC3986]**

Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005,

<http://www.rfc-editor.org/info/rfc3986>

## **[RFC1035]**

Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987,

<http://www.rfc-editor.org/info/rfc1035>

## **[RFC2782]**

Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000,

<http://www.rfc-editor.org/info/rfc2782>

# **1.5 Data representation**

## **1.5.1 Bits**

Bits in a byte are labelled 7 to 0. Bit number 7 is the most significant bit, the least significant bit is assigned bit number 0.

## **1.5.2 Two Byte Integer**

Two Byte Integer data values are 16-bit unsigned integers in big-endian order: the high order byte precedes the lower order byte. This means that a 16-bit word is presented on the network as Most Significant Byte (MSB), followed by Least Significant Byte (LSB).

## **1.5.3 Four Byte Integer**

Four Byte Integer data values are 32-bit unsigned integers in big-endian order: the high order byte precedes the successively lower order bytes. This means that a 32-bit word is presented on the network as Most Significant Byte (MSB), followed by the next most Significant Byte (MSB), followed by the next most Significant Byte (MSB), followed by Least Significant Byte (LSB).

## **1.5.4 UTF-8 Encoded String**

Text fields within the MQTT Control Packets described later are encoded as UTF-8 strings. UTF-8 [\[RFC3629\]](#) is an efficient encoding of Unicode [\[Unicode\]](#) characters that optimizes the encoding of ASCII characters in support of text-based communications.



Each of these strings is prefixed with a Two Byte Integer length field that gives the number of bytes in a UTF-8 encoded string itself, as illustrated in [Figure 1.1 Structure of UTF-8 Encoded Strings](#) below. Consequently, the maximum size of a UTF-8 Encoded String is 65,535 bytes.

Unless stated otherwise all UTF-8 encoded strings can have any length in the range 0 to 65,535 bytes.

Figure 1-1 Structure of UTF-8 Encoded Strings

Bit	7	6	5	4	3	2	1	0
byte 1	String length MSB							
byte 2	String length LSB							
byte 3 ....	UTF-8 encoded character data, if length > 0.							

The character data in a UTF-8 Encoded String MUST be well-formed UTF-8 as defined by the Unicode specification [\[Unicode\]](#) and restated in RFC 3629 [\[RFC3629\]](#). In particular, the character data MUST NOT include encodings of code points between U+D800 and U+DFFF [\[MQTT-1.5.4-1\]](#). If the Client or Server receives an MQTT Control Packet containing ill-formed UTF-8 it is a Malformed Packet. Refer to [section 4.13](#) for information about handling errors.

A UTF-8 Encoded String MUST NOT include an encoding of the null character U+0000. [\[MQTT-1.5.4-2\]](#). If a receiver (Server or Client) receives an MQTT Control Packet containing U+0000 it is a Malformed Packet. Refer to [section 4.13](#) for information about handling errors.

The data SHOULD NOT include encodings of the Unicode [\[Unicode\]](#) code points listed below. If a receiver (Server or Client) receives an MQTT Control Packet containing any of them it MAY treat it as a Malformed Packet. These are the Disallowed Unicode code points. Refer to [section 5.4.9](#) for more information about handling Disallowed Unicode code points.

- U+0001..U+001F control characters
- U+007F..U+009F control characters
- Code points defined in the Unicode specification [\[Unicode\]](#) to be non-characters (for example U+0FFFF)

A UTF-8 encoded sequence 0xEF 0xBB 0xBF is always interpreted as U+FEFF ("ZERO WIDTH NO-BREAK SPACE") wherever it appears in a string and MUST NOT be skipped over or stripped off by a packet receiver [\[MQTT-1.5.4-3\]](#).

#### Non-normative example

For example, the string A𑀧 which is LATIN CAPITAL Letter A followed by the code point U+2A6D4 (which represents a CJK IDEOGRAPH EXTENSION B character) is encoded as follows:

Figure 1-2 UTF-8 Encoded String non-normative example

Bit	7	6	5	4	3	2	1	0
byte 1	String Length MSB (0x00)							
	0	0	0	0	0	0	0	0

byte 2	String Length LSB (0x05)							
	0	0	0	0	0	1	0	1
byte 3	'A' (0x41)							
	0	1	0	0	0	0	0	1
byte 4	(0xF0)							
	1	1	1	1	0	0	0	0
byte 5	(0xAA)							
	1	0	1	0	1	0	1	0
byte 6	(0x9B)							
	1	0	0	1	1	0	1	1
byte 7	(0x94)							
	1	0	0	1	0	1	0	0

### 1.5.5 Variable Byte Integer

The Variable Byte Integer is encoded using an encoding scheme which uses a single byte for values up to 127. Larger values are handled as follows. The least significant seven bits of each byte encode the data, and the most significant bit is used to indicate whether there are bytes following in the representation. Thus, each byte encodes 128 values and a "continuation bit". The maximum number of bytes in the Variable Byte Integer field is four. The encoded value MUST use the minimum number of bytes necessary to represent the value [MQTT-1.5.5-1]. This is shown in Table 1-1 Size of Variable Byte Integer.

Table 1-1 Size of Variable Byte Integer

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)

#### Non-normative comment

The algorithm for encoding a non-negative integer (X) into the Variable Byte Integer encoding scheme is as follows:

```

do
  encodedByte = X MOD 128
  X = X DIV 128
  // if there are more data to encode, set the top bit of this byte
  if (X > 0)

```

```

318         encodedByte = encodedByte OR 128
319     endif
320     'output' encodedByte
321 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).

### Non-normative comment

The algorithm for decoding a Variable Byte Integer type is as follows:

```

329 multiplier = 1
330 value = 0
331 do
332     encodedByte = 'next byte from stream'
333     value += (encodedByte AND 127) * multiplier
334     if (multiplier > 128*128*128)
335         throw Error(Malformed Variable Byte Integer)
336     multiplier *= 128
337 while ((encodedByte AND 128) != 0)

```

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

When this algorithm terminates, value contains the Variable Byte Integer value.

## 1.5.6 Binary Data

Binary Data is represented by a Two Byte Integer length which indicates the number of data bytes, followed by that number of bytes. Thus, the length of Binary Data is limited to the range of 0 to 65,535 Bytes.

## 1.5.7 UTF-8 String Pair

A UTF-8 String Pair consists of two UTF-8 Encoded Strings. This data type is used to hold name-value pairs. The first string serves as the name, and the second string contains the value.

Both strings **MUST** comply with the requirements for UTF-8 Encoded Strings [MQTT-1.5.7-1]. If a receiver (Client or Server) receives a string pair which does not meet these requirements it is a Malformed Packet. Refer to [section 4.13](#) for information about handling errors.

## 1.6 Security

MQTT Client and Server implementations SHOULD offer Authentication, Authorization and secure communication options, such as those discussed in Chapter 5. Applications concerned with critical infrastructure, personally identifiable information, or other personal or sensitive information are strongly advised to use these security capabilities.

## 1.7 Editing convention

Text highlighted in **Yellow** within this specification identifies conformance statements. Each conformance statement has been assigned a reference in the format **[MQTT-x.x.x-y]** where **x.x.x** is the section number and **y** is a statement counter within the section.

## 1.8 Change history

### 1.8.1 MQTT v3.1.1

MQTT v3.1.1 was the first OASIS standard version of MQTT **[MQTTV311]**.

MQTT v3.1.1 is also standardized as ISO/IEC 20922:2016 **[ISO20922]**.

### 1.8.2 MQTT v5.0

MQTT v5.0 adds a significant number of new features to MQTT while keeping much of the core in place. The major functional objectives are:

- Enhancements for scalability and large scale systems
- Improved error reporting
- Formalize common patterns including capability discovery and request response
- Extensibility mechanisms including user properties
- Performance improvements and support for small clients

Refer to [Appendix C](#) for a summary of changes in MQTT v5.0.

## 2 MQTT Control Packet format

### 2.1 Structure of an MQTT Control Packet

The MQTT protocol operates by exchanging a series of MQTT Control Packets in a defined way. This section describes the format of these packets.

An MQTT Control Packet consists of up to three parts, always in the following order as shown below.

Figure 2-1 Structure of an MQTT Control Packet

Fixed Header, present in all MQTT Control Packets
Variable Header, present in some MQTT Control Packets
Payload, present in some MQTT Control Packets

#### 2.1.1 Fixed Header

Each MQTT Control Packet contains a Fixed Header as shown below.

Figure 2-2 Fixed Header format

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2...	Remaining Length							

#### 2.1.2 MQTT Control Packet type

**Position:** byte 1, bits 7-4.

Represented as a 4-bit unsigned value, the values are shown below.

Table 2-1 MQTT Control Packet types

Name	Value	Direction of flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Connection request
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment (QoS 1)

PUBREC	5	Client to Server or Server to Client	Publish received (QoS 2 delivery part 1)
PUBREL	6	Client to Server or Server to Client	Publish release (QoS 2 delivery part 2)
PUBCOMP	7	Client to Server or Server to Client	Publish complete (QoS 2 delivery part 3)
SUBSCRIBE	8	Client to Server	Subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment
UNSUBSCRIBE	10	Client to Server	Unsubscribe request
UNSUBACK	11	Server to Client	Unsubscribe acknowledgment
PINGREQ	12	Client to Server	PING request
PINGRESP	13	Server to Client	PING response
DISCONNECT	14	Client to Server or Server to Client	Disconnect notification
AUTH	15	Client to Server or Server to Client	Authentication exchange

402

### 403 2.1.3 Flags

404 The remaining bits [3-0] of byte 1 in the Fixed Header contain flags specific to each MQTT Control Packet  
405 type as shown below. Where a flag bit is marked as "Reserved", it is reserved for future use and MUST  
406 be set to the value listed [MQTT-2.1.3-1]. If invalid flags are received it is a Malformed Packet. Refer to  
407 section 4.13 for details about handling errors.

408

409 Table 2-2 Flag Bits

MQTT Control Packet	Fixed Header flags	Bit 3	Bit 2	Bit 1	Bit 0
CONNECT	Reserved	0	0	0	0
CONNACK	Reserved	0	0	0	0
PUBLISH	Used in MQTT v5.0	DUP	QoS		RETAIN
PUBACK	Reserved	0	0	0	0
PUBREC	Reserved	0	0	0	0
PUBREL	Reserved	0	0	1	0
PUBCOMP	Reserved	0	0	0	0
SUBSCRIBE	Reserved	0	0	1	0
SUBACK	Reserved	0	0	0	0
UNSUBSCRIBE	Reserved	0	0	1	0

UNSUBACK	Reserved	0	0	0	0
PINGREQ	Reserved	0	0	0	0
PINGRESP	Reserved	0	0	0	0
DISCONNECT	Reserved	0	0	0	0
AUTH	Reserved	0	0	0	0

DUP = Duplicate delivery of a PUBLISH packet

QoS = PUBLISH Quality of Service

RETAIN = PUBLISH retained message flag

Refer to [section 3.3.1](#) for a description of the DUP, QoS, and RETAIN flags in the PUBLISH packet.

## 2.1.4 Remaining Length

**Position:** starts at byte 2.

The Remaining Length is a Variable Byte Integer that represents the number of bytes remaining within the current Control Packet, including data in the Variable Header and the Payload. The Remaining Length does not include the bytes used to encode the Remaining Length. The packet size is the total number of bytes in an MQTT Control Packet, this is equal to the length of the Fixed Header plus the Remaining Length.

## 2.2 Variable Header

Some types of MQTT Control Packet contain a Variable Header component. It resides between the Fixed Header and the Payload. The content of the Variable Header varies depending on the packet type. The Packet Identifier field of Variable Header is common in several packet types.

### 2.2.1 Packet Identifier

The Variable Header component of many of the MQTT Control Packet types includes a Two Byte Integer Packet Identifier field. These MQTT Control Packets are PUBLISH (where QoS > 0), PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK.

MQTT Control Packets that require a Packet Identifier are shown below:

*Table 2-3 MQTT Control Packets that contain a Packet Identifier*

MQTT Control Packet	Packet Identifier field
CONNECT	NO
CONNACK	NO
PUBLISH	YES (If QoS > 0)

PUBACK	YES
PUBREC	YES
PUBREL	YES
PUBCOMP	YES
SUBSCRIBE	YES
SUBACK	YES
UNSUBSCRIBE	YES
UNSUBACK	YES
PINGREQ	NO
PINGRESP	NO
DISCONNECT	NO
AUTH	NO

A PUBLISH packet MUST NOT contain a Packet Identifier if its QoS value is set to 0 [MQTT-2.2.1-2].

Each time a Client sends a new SUBSCRIBE, UNSUBSCRIBE, or PUBLISH (where QoS > 0) MQTT Control Packet it MUST assign it a non-zero Packet Identifier that is currently unused [MQTT-2.2.1-3].

Each time a Server sends a new PUBLISH (with QoS > 0) MQTT Control Packet it MUST assign it a non-zero Packet Identifier that is currently unused [MQTT-2.2.1-4].

The Packet Identifier becomes available for reuse after the sender has processed the corresponding acknowledgement packet, defined as follows. In the case of a QoS 1 PUBLISH, this is the corresponding PUBACK; in the case of QoS 2 PUBLISH it is PUBCOMP or a PUBREC with a Reason Code of 128 or greater. For SUBSCRIBE or UNSUBSCRIBE it is the corresponding SUBACK or UNSUBACK.

Packet Identifiers used with PUBLISH, SUBSCRIBE and UNSUBSCRIBE packets form a single, unified set of identifiers separately for the Client and the Server in a Session. A Packet Identifier cannot be used by more than one command at any time.

A PUBACK, PUBREC, PUBREL, or PUBCOMP packet MUST contain the same Packet Identifier as the PUBLISH packet that was originally sent [MQTT-2.2.1-5]. A SUBACK and UNSUBACK MUST contain the Packet Identifier that was used in the corresponding SUBSCRIBE and UNSUBSCRIBE packet respectively [MQTT-2.2.1-6].

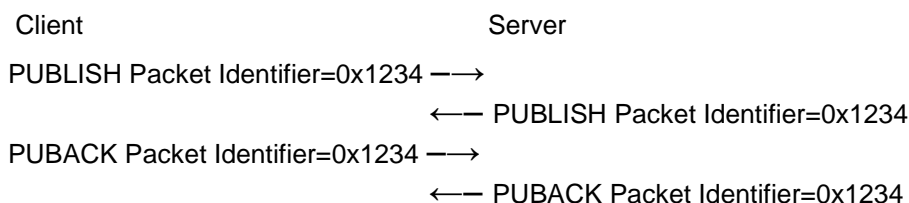
The Client and Server assign Packet Identifiers independently of each other. As a result, Client-Server pairs can participate in concurrent message exchanges using the same Packet Identifiers.

#### Non-normative comment

It is possible for a Client to send a PUBLISH packet with Packet Identifier 0x1234 and then receive a different PUBLISH packet with Packet Identifier 0x1234 from its Server before it receives a PUBACK for the PUBLISH packet that it sent.



468  
469  
470  
471  
472  
473  
474  
475



## 476 2.2.2 Properties

477 The last field in the Variable Header of the CONNECT, CONNACK, PUBLISH, PUBACK, PUBREC,  
478 PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK, DISCONNECT, and  
479 AUTH packet is a set of Properties. In the CONNECT packet there is also an optional set of Properties in  
480 the Will Properties field with the Payload.

481  
482 The set of Properties is composed of a Property Length followed by the Properties.  
483

### 484 2.2.2.1 Property Length

485 The Property Length is encoded as a Variable Byte Integer. The Property Length does not include the  
486 bytes used to encode itself, but includes the length of the Properties. **If there are no properties, this MUST**  
487 **be indicated by including a Property Length of zero** [MQTT-2.2.2-1].  
488

### 489 2.2.2.2 Property

490 A Property consists of an Identifier which defines its usage and data type, followed by a value. The  
491 Identifier is encoded as a Variable Byte Integer. A Control Packet which contains an Identifier which is not  
492 valid for its packet type, or contains a value not of the specified data type, is a Malformed Packet. If  
493 received, use a CONNACK or DISCONNECT packet with Reason Code 0x81 (Malformed Packet) as  
494 described in [section 4.13](#) Handling errors. There is no significance in the order of Properties with different  
495 Identifiers.

496

497 *Table 2-4 - Properties*

Identifier		Name (usage)	Type	Packet / Will Properties
Dec	Hex			
1	0x01	Payload Format Indicator	Byte	PUBLISH, Will Properties
2	0x02	Message Expiry Interval	Four Byte Integer	PUBLISH, Will Properties
3	0x03	Content Type	UTF-8 Encoded String	PUBLISH, Will Properties
8	0x08	Response Topic	UTF-8 Encoded String	PUBLISH, Will Properties
9	0x09	Correlation Data	Binary Data	PUBLISH, Will Properties
11	0x0B	Subscription Identifier	Variable Byte Integer	PUBLISH, SUBSCRIBE
17	0x11	Session Expiry Interval	Four Byte Integer	CONNECT, CONNACK, DISCONNECT

18	0x12	Assigned Client Identifier	UTF-8 Encoded String	CONNACK
19	0x13	Server Keep Alive	Two Byte Integer	CONNACK
21	0x15	Authentication Method	UTF-8 Encoded String	CONNECT, CONNACK, AUTH
22	0x16	Authentication Data	Binary Data	CONNECT, CONNACK, AUTH
23	0x17	Request Problem Information	Byte	CONNECT
24	0x18	Will Delay Interval	Four Byte Integer	Will Properties
25	0x19	Request Response Information	Byte	CONNECT
26	0x1A	Response Information	UTF-8 Encoded String	CONNACK
28	0x1C	Server Reference	UTF-8 Encoded String	CONNACK, DISCONNECT
31	0x1F	Reason String	UTF-8 Encoded String	CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK, DISCONNECT, AUTH
33	0x21	Receive Maximum	Two Byte Integer	CONNECT, CONNACK
34	0x22	Topic Alias Maximum	Two Byte Integer	CONNECT, CONNACK
35	0x23	Topic Alias	Two Byte Integer	PUBLISH
36	0x24	Maximum QoS	Byte	CONNACK
37	0x25	Retain Available	Byte	CONNACK
38	0x26	User Property	UTF-8 String Pair	CONNECT, CONNACK, PUBLISH, Will Properties, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK, DISCONNECT, AUTH
39	0x27	Maximum Packet Size	Four Byte Integer	CONNECT, CONNACK
40	0x28	Wildcard Subscription Available	Byte	CONNACK
41	0x29	Subscription Identifier Available	Byte	CONNACK
42	0x2A	Shared Subscription Available	Byte	CONNACK

#### Non-normative comment

Although the Property Identifier is defined as a Variable Byte Integer, in this version of the specification all of the Property Identifiers are one byte long.

## 2.3 Payload

Some MQTT Control Packets contain a Payload as the final part of the packet. In the PUBLISH packet this is the Application Message

507 Table 2-5 - MQTT Control Packets that contain a Payload

MQTT Control Packet	Payload
CONNECT	Required
CONNACK	None
PUBLISH	Optional
PUBACK	None
PUBREC	None
PUBREL	None
PUBCOMP	None
SUBSCRIBE	Required
SUBACK	Required
UNSUBSCRIBE	Required
UNSUBACK	Required
PINGREQ	None
PINGRESP	None
DISCONNECT	None
AUTH	None

508

## 509 2.4 Reason Code

510 A Reason Code is a one byte unsigned value that indicates the result of an operation. Reason Codes less  
511 than 0x80 indicate successful completion of an operation. The normal Reason Code for success is 0.  
512 Reason Code values of 0x80 or greater indicate failure.

513

514 The CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, DISCONNECT and AUTH Control Packets  
515 have a single Reason Code as part of the Variable Header. The SUBACK and UNSUBACK packets  
516 contain a list of one or more Reason Codes in the Payload.

517

518 The Reason Codes share a common set of values as shown below.

519

520 Table 2-6 - Reason Codes

Reason Code		Name	Packets
Decimal	Hex		
0	0x00	Success	CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, UNSUBACK, AUTH
0	0x00	Normal disconnection	DISCONNECT

0	0x00	Granted QoS 0	SUBACK
1	0x01	Granted QoS 1	SUBACK
2	0x02	Granted QoS 2	SUBACK
4	0x04	Disconnect with Will Message	DISCONNECT
16	0x10	No matching subscribers	PUBACK, PUBREC
17	0x11	No subscription existed	UNSUBACK
24	0x18	Continue authentication	AUTH
25	0x19	Re-authenticate	AUTH
128	0x80	Unspecified error	CONNACK, PUBACK, PUBREC, SUBACK, UNSUBACK, DISCONNECT
129	0x81	Malformed Packet	CONNACK, DISCONNECT
130	0x82	Protocol Error	CONNACK, DISCONNECT
131	0x83	Implementation specific error	CONNACK, PUBACK, PUBREC, SUBACK, UNSUBACK, DISCONNECT
132	0x84	Unsupported Protocol Version	CONNACK
133	0x85	Client Identifier not valid	CONNACK
134	0x86	Bad User Name or Password	CONNACK
135	0x87	Not authorized	CONNACK, PUBACK, PUBREC, SUBACK, UNSUBACK, DISCONNECT
136	0x88	Server unavailable	CONNACK
137	0x89	Server busy	CONNACK, DISCONNECT
138	0x8A	Banned	CONNACK
139	0x8B	Server shutting down	DISCONNECT
140	0x8C	Bad authentication method	CONNACK, DISCONNECT
141	0x8D	Keep Alive timeout	DISCONNECT
142	0x8E	Session taken over	DISCONNECT
143	0x8F	Topic Filter invalid	SUBACK, UNSUBACK, DISCONNECT
144	0x90	Topic Name invalid	CONNACK, PUBACK, PUBREC, DISCONNECT
145	0x91	Packet Identifier in use	PUBACK, PUBREC, SUBACK, UNSUBACK
146	0x92	Packet Identifier not found	PUBREL, PUBCOMP
147	0x93	Receive Maximum exceeded	DISCONNECT
148	0x94	Topic Alias invalid	DISCONNECT
149	0x95	Packet too large	CONNACK, DISCONNECT
150	0x96	Message rate too high	DISCONNECT

151	0x97	Quota exceeded	CONNACK, PUBACK, PUBREC, SUBACK, DISCONNECT
152	0x98	Administrative action	DISCONNECT
153	0x99	Payload format invalid	CONNACK, PUBACK, PUBREC, DISCONNECT
154	0x9A	Retain not supported	CONNACK, DISCONNECT
155	0x9B	QoS not supported	CONNACK, DISCONNECT
156	0x9C	Use another server	CONNACK, DISCONNECT
157	0x9D	Server moved	CONNACK, DISCONNECT
158	0x9E	Shared Subscriptions not supported	SUBACK, DISCONNECT
159	0x9F	Connection rate exceeded	CONNACK, DISCONNECT
160	0xA0	Maximum connect time	DISCONNECT
161	0xA1	Subscription Identifiers not supported	SUBACK, DISCONNECT
162	0xA2	Wildcard Subscriptions not supported	SUBACK, DISCONNECT

#### Non-normative comment

For Reason Code 0x91 (Packet identifier in use), the response to this is either to try to fix the state, or to reset the Session state by connecting using Clean Start set to 1, or to decide if the Client or Server implementations are defective.

## 3 MQTT Control Packets

### 3.1 CONNECT – Connection Request

After a Network Connection is established by a Client to a Server, the first packet sent from the Client to the Server MUST be a CONNECT packet [MQTT-3.1.0-1].

A Client can only send the CONNECT packet once over a Network Connection. The Server MUST process a second CONNECT packet sent from a Client as a Protocol Error and close the Network Connection [MQTT-3.1.0-2]. Refer to section 4.13 for information about handling errors.

The Payload contains one or more encoded fields. They specify a unique Client identifier for the Client, a Will Topic, Will Payload, User Name and Password. All but the Client identifier can be omitted and their presence is determined based on flags in the Variable Header.

#### 3.1.1 CONNECT Fixed Header

Figure 3-1 - CONNECT packet Fixed Header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (1)				Reserved			
	0	0	0	1	0	0	0	0
byte 2...	Remaining Length							

#### Remaining Length field

This is the length of the Variable Header plus the length of the Payload. It is encoded as a Variable Byte Integer.

#### 3.1.2 CONNECT Variable Header

The Variable Header for the CONNECT Packet contains the following fields in this order: Protocol Name, Protocol Level, Connect Flags, Keep Alive, and Properties. The rules for encoding Properties are described in section 2.2.2.

##### 3.1.2.1 Protocol Name

Figure 3-2 - Protocol Name bytes

	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 (4)	0	0	0	0	0	1	0	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	'T'	0	1	0	1	0	1	0	0
byte 6	'T'	0	1	0	1	0	1	0	0

The Protocol Name is a UTF-8 Encoded String that represents the protocol name “MQTT”, capitalized as shown. The string, its offset and length will not be changed by future versions of the MQTT specification.

A Server which support multiple protocols uses the Protocol Name to determine whether the data is MQTT. The protocol name MUST be the UTF-8 String "MQTT". If the Server does not want to accept the CONNECT, and wishes to reveal that it is an MQTT Server it MAY send a CONNACK packet with Reason Code of 0x84 (Unsupported Protocol Version), and then it MUST close the Network Connection [MQTT-3.1.2-1].

#### Non-normative comment

Packet inspectors, such as firewalls, could use the Protocol Name to identify MQTT traffic.

### 3.1.2.2 Protocol Version

Figure 3-3 - Protocol Version byte

	Description	7	6	5	4	3	2	1	0
Protocol Level									
byte 7	Version(5)	0	0	0	0	0	1	0	1

The one byte unsigned value that represents the revision level of the protocol used by the Client. The value of the Protocol Version field for version 5.0 of the protocol is 5 (0x05).

A Server which supports multiple versions of the MQTT protocol uses the Protocol Version to determine which version of MQTT the Client is using. If the Protocol Version is not 5 and the Server does not want to accept the CONNECT packet, the Server MAY send a CONNACK packet with Reason Code 0x84 (Unsupported Protocol Version) and then MUST close the Network Connection [MQTT-3.1.2-2].

### 3.1.2.3 Connect Flags

The Connect Flags byte contains several parameters specifying the behavior of the MQTT connection. It also indicates the presence or absence of fields in the Payload.

Figure 3-4 - Connect Flag bits

Bit	7	6	5	4	3	2	1	0
	User Name Flag	Password Flag	Will Retain	Will QoS		Will Flag	Clean Start	Reserved
byte 8	X	X	X	X	X	X	X	0

The Server MUST validate that the reserved flag in the CONNECT packet is set to 0 [MQTT-3.1.2-3]. If the reserved flag is not 0 it is a Malformed Packet. Refer to section 4.13 for information about handling errors.

### 3.1.2.4 Clean Start

**Position:** bit 1 of the Connect Flags byte.

This bit specifies whether the Connection starts a new Session or is a continuation of an existing Session. Refer to section 4.1 for a definition of the Session State.

If a CONNECT packet is received with Clean Start is set to 1, the Client and Server MUST discard any existing Session and start a new Session [MQTT-3.1.2-4]. Consequently, the Session Present flag in CONNACK is always set to 0 if Clean Start is set to 1.

If a CONNECT packet is received with Clean Start set to 0 and there is a Session associated with the Client Identifier, the Server MUST resume communications with the Client based on state from the existing Session [MQTT-3.1.2-5]. If a CONNECT packet is received with Clean Start set to 0 and there is no Session associated with the Client Identifier, the Server MUST create a new Session [MQTT-3.1.2-6].

### 3.1.2.5 Will Flag

**Position:** bit 2 of the Connect Flags.

If the Will Flag is set to 1 this indicates that a Will Message MUST be stored on the Server and associated with the Session [MQTT-3.1.2-7]. The Will Message consists of the Will Properties, Will Topic, and Will Payload fields in the CONNECT Payload. The Will Message MUST be published after the Network Connection is subsequently closed and either the Will Delay Interval has elapsed or the Session ends, unless the Will Message has been deleted by the Server on receipt of a DISCONNECT packet with Reason Code 0x00 (Normal disconnection) or a new Network Connection for the ClientID is opened before the Will Delay Interval has elapsed [MQTT-3.1.2-8].

Situations in which the Will Message is published include, but are not limited to:

- An I/O error or network failure detected by the Server.
- The Client fails to communicate within the Keep Alive time.
- The Client closes the Network Connection without first sending a DISCONNECT packet with a Reason Code 0x00 (Normal disconnection).
- The Server closes the Network Connection without first receiving a DISCONNECT packet with a Reason Code 0x00 (Normal disconnection).

If the Will Flag is set to 1, the Will Properties, Will Topic, and Will Payload fields MUST be present in the Payload [MQTT-3.1.2-9]. The Will Message MUST be removed from the stored Session State in the Server once it has been published or the Server has received a DISCONNECT packet with a Reason Code of 0x00 (Normal disconnection) from the Client [MQTT-3.1.2-10].

The Server SHOULD publish Will Messages promptly after the Network Connection is closed and the Will Delay Interval has passed, or when the Session ends, whichever occurs first. In the case of a Server shutdown or failure, the Server MAY defer publication of Will Messages until a subsequent restart. If this happens, there might be a delay between the time the Server experienced failure and when the Will Message is published.



Refer to [section 3.1.3.2](#) for information about the Will Delay Interval.

#### Non-normative comment

The Client can arrange for the Will Message to notify that Session Expiry has occurred by setting the Will Delay Interval to be longer than the Session Expiry Interval and sending DISCONNECT with Reason Code 0x04 (Disconnect with Will Message).

### 3.1.2.6 Will QoS

**Position:** bits 4 and 3 of the Connect Flags.

These two bits specify the QoS level to be used when publishing the Will Message.

If the Will Flag is set to 0, then the Will QoS MUST be set to 0 (0x00) [MQTT-3.1.2-11].

If the Will Flag is set to 1, the value of Will QoS can be 0 (0x00), 1 (0x01), or 2 (0x02) [MQTT-3.1.2-12]. A value of 3 (0x03) is a Malformed Packet. Refer to [section 4.13](#) for information about handling errors.

### 3.1.2.7 Will Retain

**Position:** bit 5 of the Connect Flags.

This bit specifies if the Will Message is to be retained when it is published.

If the Will Flag is set to 0, then Will Retain MUST be set to 0 [MQTT-3.1.2-13]. If the Will Flag is set to 1 and Will Retain is set to 0, the Server MUST publish the Will Message as a non-retained message [MQTT-3.1.2-14]. If the Will Flag is set to 1 and Will Retain is set to 1, the Server MUST publish the Will Message as a retained message [MQTT-3.1.2-15].

### 3.1.2.8 User Name Flag

**Position:** bit 7 of the Connect Flags.

If the User Name Flag is set to 0, a User Name MUST NOT be present in the Payload [MQTT-3.1.2-16]. If the User Name Flag is set to 1, a User Name MUST be present in the Payload [MQTT-3.1.2-17].

### 3.1.2.9 Password Flag

**Position:** bit 6 of the Connect Flags.

If the Password Flag is set to 0, a Password MUST NOT be present in the Payload [MQTT-3.1.2-18]. If the Password Flag is set to 1, a Password MUST be present in the Payload [MQTT-3.1.2-19].

#### Non-normative comment

This version of the protocol allows the sending of a Password with no User Name, where MQTT v3.1.1 did not. This reflects the common use of Password for credentials other than a password.

### 3.1.2.10 Keep Alive

Figure 3-5 - Keep Alive bytes

Bit	7	6	5	4	3	2	1	0
byte 9	Keep Alive MSB							
byte 10	Keep Alive LSB							

The Keep Alive is a Two Byte Integer which is a time interval measured in seconds. It is the maximum time interval that is permitted to elapse between the point at which the Client finishes transmitting one MQTT Control Packet and the point it starts sending the next. It is the responsibility of the Client to ensure that the interval between MQTT Control Packets being sent does not exceed the Keep Alive value. If Keep Alive is non-zero and in the absence of sending any other MQTT Control Packets, the Client MUST send a PINGREQ packet [MQTT-3.1.2-20].

If the Server returns a Server Keep Alive on the CONNACK packet, the Client MUST use that value instead of the value it sent as the Keep Alive [MQTT-3.1.2-21].

The Client can send PINGREQ at any time, irrespective of the Keep Alive value, and check for a corresponding PINGRESP to determine that the network and the Server are available.

If the Keep Alive value is non-zero and the Server does not receive an MQTT Control Packet from the Client within one and a half times the Keep Alive time period, it MUST close the Network Connection to the Client as if the network had failed [MQTT-3.1.2-22].

If a Client does not receive a PINGRESP packet within a reasonable amount of time after it has sent a PINGREQ, it SHOULD close the Network Connection to the Server.

A Keep Alive value of 0 has the effect of turning off the Keep Alive mechanism. If Keep Alive is 0 the Client is not obliged to send MQTT Control Packets on any particular schedule.

**Non-normative comment**

The Server may have other reasons to disconnect the Client, for instance because it is shutting down. Setting Keep Alive does not guarantee that the Client will remain connected.

**Non-normative comment**

The actual value of the Keep Alive is application specific; typically, this is a few minutes. The maximum value of 65,535 is 18 hours 12 minutes and 15 seconds.

### 3.1.2.11 CONNECT Properties

#### 3.1.2.11.1 Property Length

The length of the Properties in the CONNECT packet Variable Header encoded as a Variable Byte Integer.

### 3.1.2.11.2 Session Expiry Interval

**17 (0x11) Byte**, Identifier of the Session Expiry Interval.

Followed by the Four Byte Integer representing the Session Expiry Interval in seconds. It is a Protocol Error to include the Session Expiry Interval more than once.

If the Session Expiry Interval is absent the value 0 is used. If it is set to 0, or is absent, the Session ends when the Network Connection is closed.

If the Session Expiry Interval is 0xFFFFFFFF (UINT\_MAX), the Session does not expire.

The Client and Server MUST store the Session State after the Network Connection is closed if the Session Expiry Interval is greater than 0 [MQTT-3.1.2-23].

#### **Non-normative comment**

The clock in the Client or Server may not be running for part of the time interval, for instance because the Client or Server are not running. This might cause the deletion of the state to be delayed.

Refer to [section 4.1](#) for more information about Sessions. Refer to [section 4.1.1](#) for details and limitations of stored state.

When the Session expires the Client and Server need not process the deletion of state atomically.

#### **Non-normative comment**

Setting Clean Start to 1 and a Session Expiry Interval of 0, is equivalent to setting CleanSession to 1 in the MQTT Specification Version 3.1.1. Setting Clean Start to 0 and no Session Expiry Interval, is equivalent to setting CleanSession to 0 in the MQTT Specification Version 3.1.1.

#### **Non-normative comment**

A Client that only wants to process messages while connected will set the Clean Start to 1 and set the Session Expiry Interval to 0. It will not receive Application Messages published before it connected and has to subscribe afresh to any topics that it is interested in each time it connects.

#### **Non-normative comment**

A Client might be connecting to a Server using a network that provides intermittent connectivity. This Client can use a short Session Expiry Interval so that it can reconnect when the network is available again and continue reliable message delivery. If the Client does not reconnect, allowing the Session to expire, then Application Messages will be lost.

#### **Non-normative comment**

When a Client connects with a long Session Expiry Interval, it is requesting that the Server maintain its MQTT session state after it disconnects for an extended period. Clients should only connect with a long Session Expiry Interval if they intend to reconnect to the Server at some later point in time. When a Client has determined that it has no further use for the Session it should disconnect with a Session Expiry Interval set to 0.

**Non-normative comment**

The Client should always use the Session Present flag in the CONNACK to determine whether the Server has a Session State for this Client.

**Non-normative comment**

The Client can avoid implementing its own Session expiry and instead rely on the Session Present flag returned from the Server to determine if the Session had expired. If the Client does implement its own Session expiry, it needs to store the time at which the Session State will be deleted as part of its Session State.

### 3.1.2.11.3 Receive Maximum

**33 (0x21) Byte**, Identifier of the Receive Maximum.

Followed by the Two Byte Integer representing the Receive Maximum value. It is a Protocol Error to include the Receive Maximum value more than once or for it to have the value 0.

The Client uses this value to limit the number of QoS 1 and QoS 2 publications that it is willing to process concurrently. There is no mechanism to limit the QoS 0 publications that the Server might try to send.

The value of Receive Maximum applies only to the current Network Connection. If the Receive Maximum value is absent then its value defaults to 65,535.

Refer to [section 4.9](#) Flow Control for details of how the Receive Maximum is used.

### 3.1.2.11.4 Maximum Packet Size

**39 (0x27) Byte**, Identifier of the Maximum Packet Size.

Followed by a Four Byte Integer representing the Maximum Packet Size the Client is willing to accept. If the Maximum Packet Size is not present, no limit on the packet size is imposed beyond the limitations in the protocol as a result of the remaining length encoding and the protocol header sizes.

It is a Protocol Error to include the Maximum Packet Size more than once, or for the value to be set to zero.

**Non-normative comment**

It is the responsibility of the application to select a suitable Maximum Packet Size value if it chooses to restrict the Maximum Packet Size.

The packet size is the total number of bytes in an MQTT Control Packet, as defined in [section 2.1.4](#). The Client uses the Maximum Packet Size to inform the Server that it will not process packets exceeding this limit.

**The Server MUST NOT send packets exceeding Maximum Packet Size to the Client [MQTT-3.1.2-24].** If a Client receives a packet whose size exceeds this limit, this is a Protocol Error, the Client uses DISCONNECT with Reason Code 0x95 (Packet too large), as described in [section 4.13](#).

Where a Packet is too large to send, the Server **MUST** discard it without sending it and then behave as if it had completed sending that Application Message [MQTT-3.1.2-25].

In the case of a Shared Subscription where the message is too large to send to one or more of the Clients but other Clients can receive it, the Server can choose either discard the message without sending the message to any of the Clients, or to send the message to one of the Clients that can receive it.

#### **Non-normative comment**

Where a packet is discarded without being sent, the Server could place the discarded packet on a 'dead letter queue' or perform other diagnostic action. Such actions are outside the scope of this specification.

### **3.1.2.11.5 Topic Alias Maximum**

**34 (0x22) Byte**, Identifier of the Topic Alias Maximum.

Followed by the Two Byte Integer representing the Topic Alias Maximum value. It is a Protocol Error to include the Topic Alias Maximum value more than once. If the Topic Alias Maximum property is absent, the default value is 0.

This value indicates the highest value that the Client will accept as a Topic Alias sent by the Server. The Client uses this value to limit the number of Topic Aliases that it is willing to hold on this Connection. **The Server MUST NOT send a Topic Alias in a PUBLISH packet to the Client greater than Topic Alias Maximum [MQTT-3.1.2-26].** A value of 0 indicates that the Client does not accept any Topic Aliases on this connection. If Topic Alias Maximum is absent or zero, the Server **MUST NOT** send any Topic Aliases to the Client [MQTT-3.1.2-27].

### **3.1.2.11.6 Request Response Information**

**25 (0x19) Byte**, Identifier of the Request Response Information.

Followed by a Byte with a value of either 0 or 1. It is Protocol Error to include the Request Response Information more than once, or to have a value other than 0 or 1. If the Request Response Information is absent, the value of 0 is used.

The Client uses this value to request the Server to return Response Information in the CONNACK. **A value of 0 indicates that the Server MUST NOT return Response Information [MQTT-3.1.2-28].** If the value is 1 the Server **MAY** return Response Information in the CONNACK packet.

#### **Non-normative comment**

The Server can choose not to include Response Information in the CONNACK, even if the Client requested it.

Refer to [section 4.10](#) for more information about Request / Response.

### **3.1.2.11.7 Request Problem Information**

**23 (0x17) Byte**, Identifier of the Request Problem Information.

Followed by a Byte with a value of either 0 or 1. It is a Protocol Error to include Request Problem Information more than once, or to have a value other than 0 or 1. If the Request Problem Information is absent, the value of 1 is used.

The Client uses this value to indicate whether the Reason String or User Properties are sent in the case of failures.

If the value of Request Problem Information is 0, the Server MAY return a Reason String or User Properties on a CONNACK or DISCONNECT packet, but MUST NOT send a Reason String or User Properties on any packet other than PUBLISH, CONNACK, or DISCONNECT [MQTT-3.1.2-29]. If the value is 0 and the Client receives a Reason String or User Properties in a packet other than PUBLISH, CONNACK, or DISCONNECT, it uses a DISCONNECT packet with Reason Code 0x82 (Protocol Error) as described in [section 4.13](#) Handling errors.

If this value is 1, the Server MAY return a Reason String or User Properties on any packet where it is allowed.

### 3.1.2.11.8 User Property

**38 (0x26) Byte**, Identifier of the User Property.

Followed by a UTF-8 String Pair.

The User Property is allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to appear more than once.

#### Non-normative comment

User Properties on the CONNECT packet can be used to send connection related properties from the Client to the Server. The meaning of these properties is not defined by this specification.

### 3.1.2.11.9 Authentication Method

**21 (0x15) Byte**, Identifier of the Authentication Method.

Followed by a UTF-8 Encoded String containing the name of the authentication method used for extended authentication. It is a Protocol Error to include Authentication Method more than once.

If Authentication Method is absent, extended authentication is not performed. Refer to [section 4.12](#).

If a Client sets an Authentication Method in the CONNECT, the Client MUST NOT send any packets other than AUTH or DISCONNECT packets until it has received a CONNACK packet [MQTT-3.1.2-30].

### 3.1.2.11.10 Authentication Data

**22 (0x16) Byte**, Identifier of the Authentication Data.

Followed by Binary Data containing authentication data. It is a Protocol Error to include Authentication Data if there is no Authentication Method. It is a Protocol Error to include Authentication Data more than once.

888 The contents of this data are defined by the authentication method. Refer to [section 4.12](#) for more  
889 information about extended authentication.

890

### 891 3.1.2.12 Variable Header non-normative example

892 *Figure 3-6 - Variable Header example*

	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 (4)	0	0	0	0	0	1	0	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	'T'	0	1	0	1	0	1	0	0
byte 6	'T'	0	1	0	1	0	1	0	0
Protocol Version									
	Description	7	6	5	4	3	2	1	0
byte 7	Version (5)	0	0	0	0	0	1	0	1
Connect Flags									
byte 8	User Name Flag (1)	1	1	0	0	1	1	1	0
	Password Flag (1)								
	Will Retain (0)								
	Will QoS (01)								
	Will Flag (1)								
	Clean Start(1)								
	Reserved (0)								
Keep Alive									
byte 9	Keep Alive MSB (0)	0	0	0	0	0	0	0	0
byte 10	Keep Alive LSB (10)	0	0	0	0	1	0	1	0
Properties									
byte 11	Length (5)	0	0	0	0	0	1	0	1
byte 12	Session Expiry Interval identifier (17)	0	0	0	1	0	0	0	1
byte 13	Session Expiry Interval (10)	0	0	0	0	0	0	0	0
byte 14		0	0	0	0	0	0	0	0

byte 15		0	0	0	0	0	0	0	0
byte 16		0	0	0	0	1	0	1	0

### 3.1.3 CONNECT Payload

The Payload of the CONNECT packet contains one or more length-prefixed fields, whose presence is determined by the flags in the Variable Header. These fields, if present, MUST appear in the order Client Identifier, Will Properties, Will Topic, Will Payload, User Name, Password [MQTT-3.1.3-1].

#### 3.1.3.1 Client Identifier (ClientID)

The Client Identifier (ClientID) identifies the Client to the Server. Each Client connecting to the Server has a unique ClientID. The ClientID MUST be used by Clients and by Servers to identify state that they hold relating to this MQTT Session between the Client and the Server [MQTT-3.1.3-2]. Refer to section 4.1 for more information about Session State.

The ClientID MUST be present and is the first field in the CONNECT packet Payload [MQTT-3.1.3-3].

The ClientID MUST be a UTF-8 Encoded String as defined in section 1.5.4 [MQTT-3.1.3-4].

The Server MUST allow ClientID's which are between 1 and 23 UTF-8 encoded bytes in length, and that contain only the characters

"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ" [MQTT-3.1.3-5].

The Server MAY allow ClientID's that contain more than 23 encoded bytes. The Server MAY allow ClientID's that contain characters not included in the list given above.

A Server MAY allow a Client to supply a ClientID that has a length of zero bytes, however if it does so the Server MUST treat this as a special case and assign a unique ClientID to that Client [MQTT-3.1.3-6]. It MUST then process the CONNECT packet as if the Client had provided that unique ClientID, and MUST return the Assigned Client Identifier in the CONNACK packet [MQTT-3.1.3-7].

If the Server rejects the ClientID it MAY respond to the CONNECT packet with a CONNACK using Reason Code 0x85 (Client Identifier not valid) as described in section 4.13 Handling errors, and then it MUST close the Network Connection [MQTT-3.1.3-8].

#### Non-normative comment

A Client implementation could provide a convenience method to generate a random ClientID. Clients using this method should take care to avoid creating long-lived orphaned Sessions.

#### 3.1.3.2 Will Properties

If the Will Flag is set to 1, the Will Properties is the next field in the Payload. The Will Properties field defines the Application Message properties to be sent with the Will Message when it is published, and properties which define when to publish the Will Message. The Will Properties consists of a Property Length and the Properties.



### 3.1.3.2.1 Property Length

The length of the Properties in the Will Properties encoded as a Variable Byte Integer.

### 3.1.3.2.2 Will Delay Interval

**24 (0x18) Byte**, Identifier of the Will Delay Interval.

Followed by the Four Byte Integer representing the Will Delay Interval in seconds. It is a Protocol Error to include the Will Delay Interval more than once. If the Will Delay Interval is absent, the default value is 0 and there is no delay before the Will Message is published.

The Server delays publishing the Client's Will Message until the Will Delay Interval has passed or the Session ends, whichever happens first. **If a new Network Connection to this Session is made before the Will Delay Interval has passed, the Server MUST NOT send the Will Message [MQTT-3.1.3-9].**

#### Non-normative comment

One use of this is to avoid publishing Will Messages if there is a temporary network disconnection and the Client succeeds in reconnecting and continuing its Session before the Will Message is published.

#### Non-normative comment

If a Network Connection uses a Client Identifier of an existing Network Connection to the Server, the Will Message for the exiting connection is sent unless the new connection specifies Clean Start of 0 and the Will Delay is greater than zero. If the Will Delay is 0 the Will Message is sent at the close of the existing Network Connection, and if Clean Start is 1 the Will Message is sent because the Session ends.

### 3.1.3.2.3 Payload Format Indicator

**1 (0x01) Byte**, Identifier of the Payload Format Indicator.

Followed by the value of the Payload Format Indicator, either of:

- 0 (0x00) Byte Indicates that the Will Message is unspecified bytes, which is equivalent to not sending a Payload Format Indicator.
- 1 (0x01) Byte Indicates that the Will Message is UTF-8 Encoded Character Data. The UTF-8 data in the Payload MUST be well-formed UTF-8 as defined by the Unicode specification [Unicode] and restated in RFC 3629 [RFC3629].

It is a Protocol Error to include the Payload Format Indicator more than once. The Server MAY validate that the Will Message is of the format indicated, and if it is not send a CONNACK with the Reason Code of 0x99 (Payload format invalid) as described in section 4.13.

### 3.1.3.2.4 Message Expiry Interval

**2 (0x02) Byte**, Identifier of the Message Expiry Interval.

Followed by the Four Byte Integer representing the Message Expiry Interval. It is a Protocol Error to include the Message Expiry Interval more than once.

978 If present, the Four Byte value is the lifetime of the Will Message in seconds and is sent as the  
979 Publication Expiry Interval when the Server publishes the Will Message.

980

981 If absent, no Message Expiry Interval is sent when the Server publishes the Will Message.

982

### 983 3.1.3.2.5 Content Type

984 **3 (0x03)** Identifier of the Content Type.

985 Followed by a UTF-8 Encoded String describing the content of the Will Message. It is a Protocol Error to  
986 include the Content Type more than once. The value of the Content Type is defined by the sending and  
987 receiving application.

988

### 989 3.1.3.2.6 Response Topic

990 **8 (0x08) Byte**, Identifier of the Response Topic.

991 Followed by a UTF-8 Encoded String which is used as the Topic Name for a response message. It is a  
992 Protocol Error to include the Response Topic more than once. The presence of a Response Topic  
993 identifies the Will Message as a Request.

994

995 Refer to [section 4.10](#) for more information about Request / Response.

996

### 997 3.1.3.2.7 Correlation Data

998 **9 (0x09) Byte**, Identifier of the Correlation Data.

999 Followed by Binary Data. The Correlation Data is used by the sender of the Request Message to identify  
1000 which request the Response Message is for when it is received. It is a Protocol Error to include  
1001 Correlation Data more than once. If the Correlation Data is not present, the Requester does not require  
1002 any correlation data.

1003

1004 The value of the Correlation Data only has meaning to the sender of the Request Message and receiver  
1005 of the Response Message.

1006

1007 Refer to [section 4.10](#) for more information about Request / Response

1008

### 1009 3.1.3.2.8 User Property

1010 **38 (0x26) Byte**, Identifier of the User Property.

1011 Followed by a UTF-8 String Pair. The User Property is allowed to appear multiple times to represent  
1012 multiple name, value pairs. The same name is allowed to appear more than once.

1013

1014 **The Server MUST maintain the order of User Properties when publishing the Will Message [MQTT-3.1.3-10].**

1016

#### 1017 Non-normative comment

1018 This property is intended to provide a means of transferring application layer name-value tags  
1019 whose meaning and interpretation are known only by the application programs responsible for  
1020 sending and receiving them.

### 3.1.3.3 Will Topic

If the Will Flag is set to 1, the Will Topic is the next field in the Payload. The Will Topic MUST be a UTF-8 Encoded String as defined in section 1.5.4 [MQTT-3.1.3-11].

### 3.1.3.4 Will Payload

If the Will Flag is set to 1 the Will Payload is the next field in the Payload. The Will Payload defines the Application Message Payload that is to be published to the Will Topic as described in section 3.1.2.5. This field consists of Binary Data.

### 3.1.3.5 User Name

If the User Name Flag is set to 1, the User Name is the next field in the Payload. The User Name MUST be a UTF-8 Encoded String as defined in section 1.5.4 [MQTT-3.1.3-12]. It can be used by the Server for authentication and authorization.

### 3.1.3.6 Password

If the Password Flag is set to 1, the Password is the next field in the Payload. The Password field is Binary Data. Although this field is called Password, it can be used to carry any credential information.

## 3.1.4 CONNECT Actions

Note that a Server MAY support multiple protocols (including other versions of the MQTT protocol) on the same TCP port or other network endpoint. If the Server determines that the protocol is MQTT v5.0 then it validates the connection attempt as follows.

1. If the Server does not receive a CONNECT packet within a reasonable amount of time after the Network Connection is established, the Server SHOULD close the Network Connection.
2. The Server MUST validate that the CONNECT packet matches the format described in section 3.1 and close the Network Connection if it does not match [MQTT-3.1.4-1]. The Server MAY send a CONNACK with a Reason Code of 0x80 or greater as described in section 4.13 before closing the Network Connection.
3. The Server MAY check that the contents of the CONNECT packet meet any further restrictions and SHOULD perform authentication and authorization checks. If any of these checks fail, it MUST close the Network Connection [MQTT-3.1.4-2]. Before closing the Network Connection, it MAY send an appropriate CONNACK response with a Reason Code of 0x80 or greater as described in section 3.2 and section 4.13.

If validation is successful, the Server performs the following steps.

1. If the ClientID represents a Client already connected to the Server, the Server sends a DISCONNECT packet to the existing Client with Reason Code of 0x8E (Session taken over) as described in section 4.13 and MUST close the Network Connection of the existing Client [MQTT-3.1.4-3]. If the existing Client has a Will Message, that Will Message is published as described in section 3.1.2.5.

**Non-normative comment**

If the Will Delay Interval of the existing Network Connection is 0 and there is a Will Message, it will be sent because the Network Connection is closed. If the Session Expiry Interval of the existing Network Connection is 0, or the new Network Connection has Clean Start set to 1 then if the existing Network Connection has a Will Message it will be sent because the original Session is ended on the takeover.

2. The Server MUST perform the processing of Clean Start that is described in [section 3.1.2.4 \[MQTT-3.1.4-4\]](#).
3. The Server MUST acknowledge the CONNECT packet with a CONNACK packet containing a 0x00 (Success) Reason Code [\[MQTT-3.1.4-5\]](#).

**Non-normative comment**

It is recommended that authentication and authorization checks be performed if the Server is being used to process any form of business critical data. If these checks succeed, the Server responds by sending CONNACK with a 0x00 (Success) Reason Code. If they fail, it is suggested that the Server does not send a CONNACK at all, as this could alert a potential attacker to the presence of the MQTT Server and encourage such an attacker to launch a denial of service or password-guessing attack.

4. Start message delivery and Keep Alive monitoring.

Clients are allowed to send further MQTT Control Packets immediately after sending a CONNECT packet; Clients need not wait for a CONNACK packet to arrive from the Server. If the Server rejects the CONNECT, it MUST NOT process any data sent by the Client after the CONNECT packet except AUTH packets [\[MQTT-3.1.4-6\]](#).

**Non-normative comment**

Clients typically wait for a CONNACK packet, However, if the Client exploits its freedom to send MQTT Control Packets before it receives a CONNACK, it might simplify the Client implementation as it does not have to police the connected state. The Client accepts that any data that it sends before it receives a CONNACK packet from the Server will not be processed if the Server rejects the connection.

**Non-normative comment**

Clients that send MQTT Control Packets before they receive CONNACK will be unaware of the Server constraints and whether any existing Session is being used.

**Non-normative comment**

The Server can limit reading from the Network Connection or close the Network Connection if the Client sends too much data before authentication is complete. This is suggested as a way of avoiding denial of service attacks.

## 3.2 CONNACK – Connect acknowledgement

The CONNACK packet is the packet sent by the Server in response to a CONNECT packet received from a Client. The Server MUST send a CONNACK with a 0x00 (Success) Reason Code before sending any

Packet other than AUTH [MQTT-3.2.0-1]. The Server MUST NOT send more than one CONNACK in a Network Connection [MQTT-3.2.0-2].

If the Client does not receive a CONNACK packet from the Server within a reasonable amount of time, the Client SHOULD close the Network Connection. A "reasonable" amount of time depends on the type of application and the communications infrastructure.

### 3.2.1 CONNACK Fixed Header

The Fixed Header format is illustrated in Figure 3-7.

Figure 3-7 – CONNACK packet Fixed Header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet Type (2)				Reserved			
	0	0	1	0	0	0	0	0
byte 2	Remaining Length							

#### Remaining Length field

This is the length of the Variable Header encoded as a Variable Byte Integer.

### 3.2.2 CONNACK Variable Header

The Variable Header of the CONNACK Packet contains the following fields in the order: Connect Acknowledge Flags, Connect Reason Code, and Properties. The rules for encoding Properties are described in [section 2.2.2](#).

#### 3.2.2.1 Connect Acknowledge Flags

Byte 1 is the "Connect Acknowledge Flags". Bits 7-1 are reserved and MUST be set to 0 [MQTT-3.2.2-1].

Bit 0 is the Session Present Flag.

##### 3.2.2.1.1 Session Present

Position: bit 0 of the Connect Acknowledge Flags.

The Session Present flag informs the Client whether the Server is using Session State from a previous connection for this ClientID. This allows the Client and Server to have a consistent view of the Session State.

If the Server accepts a connection with Clean Start set to 1, the Server MUST set Session Present to 0 in the CONNACK packet in addition to setting a 0x00 (Success) Reason Code in the CONNACK packet [MQTT-3.2.2-2].

If the Server accepts a connection with Clean Start set to 0 and the Server has Session State for the ClientID, it MUST set Session Present to 1 in the CONNACK packet, otherwise it MUST set Session

Present to 0 in the CONNACK packet. In both cases it MUST set a 0x00 (Success) Reason Code in the CONNACK packet [MQTT-3.2.2-3].

If the value of Session Present received by the Client from the Server is not as expected, the Client proceeds as follows:

- If the Client does not have Session State and receives Session Present set to 1 it MUST close the Network Connection [MQTT-3.2.2-4]. If it wishes to restart with a new Session the Client can reconnect using Clean Start set to 1.
- If the Client does have Session State and receives Session Present set to 0 it MUST discard its Session State if it continues with the Network Connection [MQTT-3.2.2-5].

If a Server sends a CONNACK packet containing a non-zero Reason Code it MUST set Session Present to 0 [MQTT-3.2.2-6].

### 3.2.2.2 Connect Reason Code

Byte 2 in the Variable Header is the Connect Reason Code.

The values the Connect Reason Code are shown below. If a well formed CONNECT packet is received by the Server, but the Server is unable to complete the Connection the Server MAY send a CONNACK packet containing the appropriate Connect Reason code from this table. If a Server sends a CONNACK packet containing a Reason code of 128 or greater it MUST then close the Network Connection [MQTT-3.2.2-7].

Table 3-1 - Connect Reason Code values

Value	Hex	Reason Code name	Description
0	0x00	Success	The Connection is accepted.
128	0x80	Unspecified error	The Server does not wish to reveal the reason for the failure, or none of the other Reason Codes apply.
129	0x81	Malformed Packet	Data within the CONNECT packet could not be correctly parsed.
130	0x82	Protocol Error	Data in the CONNECT packet does not conform to this specification.
131	0x83	Implementation specific error	The CONNECT is valid but is not accepted by this Server.
132	0x84	Unsupported Protocol Version	The Server does not support the version of the MQTT protocol requested by the Client.
133	0x85	Client Identifier not valid	The Client Identifier is a valid string but is not allowed by the Server.
134	0x86	Bad User Name or Password	The Server does not accept the User Name or Password specified by the Client
135	0x87	Not authorized	The Client is not authorized to connect.
136	0x88	Server unavailable	The MQTT Server is not available.

137	0x89	Server busy	The Server is busy. Try again later.
138	0x8A	Banned	This Client has been banned by administrative action. Contact the server administrator.
140	0x8C	Bad authentication method	The authentication method is not supported or does not match the authentication method currently in use.
144	0x90	Topic Name invalid	The Will Topic Name is not malformed, but is not accepted by this Server.
149	0x95	Packet too large	The CONNECT packet exceeded the maximum permissible size.
151	0x97	Quota exceeded	An implementation or administrative imposed limit has been exceeded.
153	0x99	Payload format invalid	The Will Payload does not match the specified Payload Format Indicator.
154	0x9A	Retain not supported	The Server does not support retained messages, and Will Retain was set to 1.
155	0x9B	QoS not supported	The Server does not support the QoS set in Will QoS.
156	0x9C	Use another server	The Client should temporarily use another server.
157	0x9D	Server moved	The Client should permanently use another server.
159	0x9F	Connection rate exceeded	The connection rate limit has been exceeded.

The Server sending the CONNACK packet MUST use one of the Connect Reason Code values T-3.2.2-8].

#### Non-normative comment

Reason Code 0x80 (Unspecified error) may be used where the Server knows the reason for the failure but does not wish to reveal it to the Client, or when none of the other Reason Code values applies.

The Server may choose to close the Network Connection without sending a CONNACK to enhance security in the case where an error is found on the CONNECT. For instance, when on a public network and the connection has not been authorized it might be unwise to indicate that this is an MQTT Server.

### 3.2.2.3 CONNACK Properties

#### 3.2.2.3.1 Property Length

This is the length of the Properties in the CONNACK packet Variable Header encoded as a Variable Byte Integer.

#### 3.2.2.3.2 Session Expiry Interval

**17 (0x11) Byte**, Identifier of the Session Expiry Interval.



Followed by the Four Byte Integer representing the Session Expiry Interval in seconds. It is a Protocol Error to include the Session Expiry Interval more than once.

If the Session Expiry Interval is absent the value in the CONNECT Packet used. The server uses this property to inform the Client that it is using a value other than that sent by the Client in the CONNACK. Refer to section 3.1.2.11.2 for a description of the use of Session Expiry Interval.

### 3.2.2.3.3 Receive Maximum

**33 (0x21) Byte**, Identifier of the Receive Maximum.

Followed by the Two Byte Integer representing the Receive Maximum value. It is a Protocol Error to include the Receive Maximum value more than once or for it to have the value 0.

The Server uses this value to limit the number of QoS 1 and QoS 2 publications that it is willing to process concurrently for the Client. It does not provide a mechanism to limit the QoS 0 publications that the Client might try to send.

If the Receive Maximum value is absent, then its value defaults to 65,535.

Refer to [section 4.9](#) Flow Control for details of how the Receive Maximum is used.

### 3.2.2.3.4 Maximum QoS

**36 (0x24) Byte**, Identifier of the Maximum QoS.

Followed by a Byte with a value of either 0 or 1. It is a Protocol Error to include Maximum QoS more than once, or to have a value other than 0 or 1. If the Maximum QoS is absent, the Client uses a Maximum QoS of 2.

If a Server does not support QoS 1 or QoS 2 PUBLISH packets it MUST send a Maximum QoS in the CONNACK packet specifying the highest QoS it supports [MQTT-3.2.2-9]. A Server that does not support QoS 1 or QoS 2 PUBLISH packets MUST still accept SUBSCRIBE packets containing a Requested QoS of 0, 1 or 2 [MQTT-3.2.2-10].

If a Client receives a Maximum QoS from a Server, it MUST NOT send PUBLISH packets at a QoS level exceeding the Maximum QoS level specified [MQTT-3.2.2-11]. It is a Protocol Error if the Server receives a PUBLISH packet with a QoS greater than the Maximum QoS it specified. In this case use DISCONNECT with Reason Code 0x9B (QoS not supported) as described in [section 4.13](#) Handling errors.

If a Server receives a CONNECT packet containing a Will QoS that exceeds its capabilities, it MUST reject the connection. It SHOULD use a CONNACK packet with Reason Code 0x9B (QoS not supported) as described in [section 4.13](#) Handling errors, and MUST close the Network Connection [MQTT-3.2.2-12].

#### Non-normative comment

A Client does not need to support QoS 1 or QoS 2 PUBLISH packets. If this is the case, the Client simply restricts the maximum QoS field in any SUBSCRIBE commands it sends to a value it can support.



### 3.2.2.3.5 Retain Available

**37 (0x25) Byte**, Identifier of Retain Available.

Followed by a Byte field. If present, this byte declares whether the Server supports retained messages. A value of 0 means that retained messages are not supported. A value of 1 means retained messages are supported. If not present, then retained messages are supported. It is a Protocol Error to include Retain Available more than once or to use a value other than 0 or 1.

If a Server receives a CONNECT packet containing a Will Message with the Will Retain set to 1, and it does not support retained messages, the Server MUST reject the connection request. It SHOULD send CONNACK with Reason Code 0x9A (Retain not supported) and then it MUST close the Network Connection [MQTT-3.2.2-13].

A Client receiving Retain Available set to 0 from the Server MUST NOT send a PUBLISH packet with the RETAIN flag set to 1 [MQTT-3.2.2-14]. If the Server receives such a packet, this is a Protocol Error. The Server SHOULD send a DISCONNECT with Reason Code of 0x9A (Retain not supported) as described in section 4.13.

### 3.2.2.3.6 Maximum Packet Size

**39 (0x27) Byte**, Identifier of the Maximum Packet Size.

Followed by a Four Byte Integer representing the Maximum Packet Size the Server is willing to accept. If the Maximum Packet Size is not present, there is no limit on the packet size imposed beyond the limitations in the protocol as a result of the remaining length encoding and the protocol header sizes.

It is a Protocol Error to include the Maximum Packet Size more than once, or for the value to be set to zero.

The packet size is the total number of bytes in an MQTT Control Packet, as defined in section 2.1.4. The Server uses the Maximum Packet Size to inform the Client that it will not process packets whose size exceeds this limit.

The Client MUST NOT send packets exceeding Maximum Packet Size to the Server [MQTT-3.2.2-15]. If a Server receives a packet whose size exceeds this limit, this is a Protocol Error, the Server uses DISCONNECT with Reason Code 0x95 (Packet too large), as described in section 4.13.

### 3.2.2.3.7 Assigned Client Identifier

**18 (0x12) Byte**, Identifier of the Assigned Client Identifier.

Followed by the UTF-8 string which is the Assigned Client Identifier. It is a Protocol Error to include the Assigned Client Identifier more than once.

The Client Identifier which was assigned by the Server because a zero length Client Identifier was found in the CONNECT packet.

If the Client connects using a zero length Client Identifier, the Server MUST respond with a CONNACK containing an Assigned Client Identifier. The Assigned Client Identifier MUST be a new Client Identifier not used by any other Session currently in the Server [MQTT-3.2.2-16].

### 3.2.2.3.8 Topic Alias Maximum

**34 (0x22) Byte**, Identifier of the Topic Alias Maximum.

Followed by the Two Byte Integer representing the Topic Alias Maximum value. It is a Protocol Error to include the Topic Alias Maximum value more than once. If the Topic Alias Maximum property is absent, the default value is 0.

This value indicates the highest value that the Server will accept as a Topic Alias sent by the Client. The Server uses this value to limit the number of Topic Aliases that it is willing to hold on this Connection. **The Client MUST NOT send a Topic Alias in a PUBLISH packet to the Server greater than this value [MQTT-3.2.2-17].** A value of 0 indicates that the Server does not accept any Topic Aliases on this connection. **If Topic Alias Maximum is absent or 0, the Client MUST NOT send any Topic Aliases on to the Server [MQTT-3.2.2-18].**

### 3.2.2.3.9 Reason String

**31 (0x1F) Byte** Identifier of the Reason String.

Followed by the UTF-8 Encoded String representing the reason associated with this response. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the Client.

The Server uses this value to give additional information to the Client. **The Server MUST NOT send this property if it would increase the size of the CONNACK packet beyond the Maximum Packet Size specified by the Client [MQTT-3.2.2-19].** It is a Protocol Error to include the Reason String more than once.

#### Non-normative comment

Proper uses for the reason string in the Client would include using this information in an exception thrown by the Client code, or writing this string to a log.

### 3.2.2.3.10 User Property

**38 (0x26) Byte**, Identifier of User Property.

Followed by a UTF-8 String Pair. This property can be used to provide additional information to the Client including diagnostic information. **The Server MUST NOT send this property if it would increase the size of the CONNACK packet beyond the Maximum Packet Size specified by the Client [MQTT-3.2.2-20].** The User Property is allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to appear more than once.

The content and meaning of this property is not defined by this specification. The receiver of a CONNACK containing this property MAY ignore it.

### 3.2.2.3.11 Wildcard Subscription Available

**40 (0x28) Byte**, Identifier of Wildcard Subscription Available.

Followed by a Byte field. If present, this byte declares whether the Server supports Wildcard Subscriptions. A value of 0 means that Wildcard Subscriptions are not supported. A value of 1 means Wildcard Subscriptions are supported. If not present, then Wildcard Subscriptions are supported. It is a Protocol Error to include the Wildcard Subscription Available more than once or to send a value other than 0 or 1.

If the Server receives a SUBSCRIBE packet containing a Wildcard Subscription and it does not support Wildcard Subscriptions, this is a Protocol Error. The Server uses DISCONNECT with Reason Code 0xA2 (Wildcard Subscriptions not supported) as described in [section 4.13](#).

If a Server supports Wildcard Subscriptions, it can still reject a particular subscribe request containing a Wildcard Subscription. In this case the Server MAY send a SUBACK Control Packet with a Reason Code 0xA2 (Wildcard Subscriptions not supported).

#### 3.2.2.3.12 Subscription Identifiers Available

**41 (0x29) Byte**, Identifier of Subscription Identifier Available.

Followed by a Byte field. If present, this byte declares whether the Server supports Subscription Identifiers. A value of 0 means that Subscription Identifiers are not supported. A value of 1 means Subscription Identifiers are supported. If not present, then Subscription Identifiers are supported. It is a Protocol Error to include the Subscription Identifier Available more than once, or to send a value other than 0 or 1.

If the Server receives a SUBSCRIBE packet containing Subscription Identifier and it does not support Subscription Identifiers, this is a Protocol Error. The Server uses DISCONNECT with Reason Code of 0xA1 (Subscription Identifiers not supported) as described in [section 4.13](#).

#### 3.2.2.3.13 Shared Subscription Available

**42 (0x2A) Byte**, Identifier of Shared Subscription Available.

Followed by a Byte field. If present, this byte declares whether the Server supports Shared Subscriptions. A value of 0 means that Shared Subscriptions are not supported. A value of 1 means Shared Subscriptions are supported. If not present, then Shared Subscriptions are supported. It is a Protocol Error to include the Shared Subscription Available more than once or to send a value other than 0 or 1.

If the Server receives a SUBSCRIBE packet containing Shared Subscriptions and it does not support Shared Subscriptions, this is a Protocol Error. The Server uses DISCONNECT with Reason Code 0x9E (Shared Subscriptions not supported) as described in [section 4.13](#).

#### 3.2.2.3.14 Server Keep Alive

**19 (0x13) Byte**, Identifier of the Server Keep Alive.

Followed by a Two Byte Integer with the Keep Alive time assigned by the Server. If the Server sends a Server Keep Alive on the CONNACK packet, the Client MUST use this value instead of the Keep Alive value the Client sent on CONNECT [MQTT-3.2.2-21]. If the Server does not send the Server Keep Alive, the Server MUST use the Keep Alive value set by the Client on CONNECT [MQTT-3.2.2-22]. It is a Protocol Error to include the Server Keep Alive more than once.

##### Non-normative comment

The primary use of the Server Keep Alive is for the Server to inform the Client that it will disconnect the Client for inactivity sooner than the Keep Alive specified by the Client.

### 3.2.2.3.15 Response Information

**26 (0x1A) Byte**, Identifier of the Response Information.

Followed by a UTF-8 Encoded String which is used as the basis for creating a Response Topic. The way in which the Client creates a Response Topic from the Response Information is not defined by this specification. It is a Protocol Error to include the Response Information more than once.

If the Client sends a Request Response Information with a value 1, it is OPTIONAL for the Server to send the Response Information in the CONNACK.

#### Non-normative comment

A common use of this is to pass a globally unique portion of the topic tree which is reserved for this Client for at least the lifetime of its Session. This often cannot just be a random name as both the requesting Client and the responding Client need to be authorized to use it. It is normal to use this as the root of a topic tree for a particular Client. For the Server to return this information, it normally needs to be correctly configured. Using this mechanism allows this configuration to be done once in the Server rather than in each Client.

Refer to [section 4.10](#) for more information about Request / Response.

### 3.2.2.3.16 Server Reference

**28 (0x1C) Byte**, Identifier of the Server Reference.

Followed by a UTF-8 Encoded String which can be used by the Client to identify another Server to use. It is a Protocol Error to include the Server Reference more than once.

The Server uses a Server Reference in either a CONNACK or DISCONNECT packet with Reason code of 0x9C (Use another server) or Reason Code 0x9D (Server moved) as described in [section 4.13](#).

Refer to [section 4.11](#) Server redirection for information about how Server Reference is used.

### 3.2.2.3.17 Authentication Method

**21 (0x15) Byte**, Identifier of the Authentication Method.

Followed by a UTF-8 Encoded String containing the name of the authentication method. It is a Protocol Error to include the Authentication Method more than once. Refer to [section 4.12](#) for more information about extended authentication.

### 3.2.2.3.18 Authentication Data

**22 (0x16) Byte**, Identifier of the Authentication Data.

Followed by Binary Data containing authentication data. The contents of this data are defined by the authentication method and the state of already exchanged authentication data. It is a Protocol Error to include the Authentication Data more than once. Refer to [section 4.12](#) for more information about extended authentication.

### 3.2.3 CONNACK Payload

The CONNACK packet has no Payload.

## 3.3 PUBLISH – Publish message

A PUBLISH packet is sent from a Client to a Server or from a Server to a Client to transport an Application Message.

### 3.3.1 PUBLISH Fixed Header

Figure 3-8 – PUBLISH packet Fixed Header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (3)				DUP flag	QoS level		RETAIN
	0	0	1	1	X	X	X	X
byte 2...	Remaining Length							

#### 3.3.1.1 DUP

**Position:** byte 1, bit 3.

If the DUP flag is set to 0, it indicates that this is the first occasion that the Client or Server has attempted to send this PUBLISH packet. If the DUP flag is set to 1, it indicates that this might be re-delivery of an earlier attempt to send the packet.

The DUP flag MUST be set to 1 by the Client or Server when it attempts to re-deliver a PUBLISH packet [MQTT-3.3.1-1]. The DUP flag MUST be set to 0 for all QoS 0 messages [MQTT-3.3.1-2].

The value of the DUP flag from an incoming PUBLISH packet is not propagated when the PUBLISH packet is sent to subscribers by the Server. The DUP flag in the outgoing PUBLISH packet is set independently to the incoming PUBLISH packet, its value MUST be determined solely by whether the outgoing PUBLISH packet is a retransmission [MQTT-3.3.1-3].

#### Non-normative comment

The receiver of an MQTT Control Packet that contains the DUP flag set to 1 cannot assume that it has seen an earlier copy of this packet.

#### Non-normative comment

It is important to note that the DUP flag refers to the MQTT Control Packet itself and not to the Application Message that it contains. When using QoS 1, it is possible for a Client to receive a PUBLISH packet with DUP flag set to 0 that contains a repetition of an Application Message that it received earlier, but with a different Packet Identifier. Section 2.2.1 provides more information about Packet Identifiers.

### 3.3.1.2 QoS

**Position:** byte 1, bits 2-1.

This field indicates the level of assurance for delivery of an Application Message. The QoS levels are shown below.

Table 3-2 - QoS definitions

QoS value	Bit 2	bit 1	Description
0	0	0	At most once delivery
1	0	1	At least once delivery
2	1	0	Exactly once delivery
-	1	1	Reserved – must not be used

If the Server included a Maximum QoS in its CONNACK response to a Client and it receives a PUBLISH packet with a QoS greater than this, then it uses DISCONNECT with Reason Code 0x9B (QoS not supported) as described in [section 4.13](#) Handling errors.

A PUBLISH Packet MUST NOT have both QoS bits set to 1 [MQTT-3.3.1-4]. If a Server or Client receives a PUBLISH packet which has both QoS bits set to 1 it is a Malformed Packet. Use DISCONNECT with Reason Code 0x81 (Malformed Packet) as described in [section 4.13](#).

### 3.3.1.3 RETAIN

**Position:** byte 1, bit 0.

If the RETAIN flag is set to 1 in a PUBLISH packet sent by a Client to a Server, the Server MUST replace any existing retained message for this topic and store the Application Message [MQTT-3.3.1-5], so that it can be delivered to future subscribers whose subscriptions match its Topic Name. If the Payload contains zero bytes it is processed normally by the Server but any retained message with the same topic name MUST be removed and any future subscribers for the topic will not receive a retained message [MQTT-3.3.1-6]. A retained message with a Payload containing zero bytes MUST NOT be stored as a retained message on the Server [MQTT-3.3.1-7].

If the RETAIN flag is 0 in a PUBLISH packet sent by a Client to a Server, the Server MUST NOT store the message as a retained message and MUST NOT remove or replace any existing retained message [MQTT-3.3.1-8].

If the Server included Retain Available in its CONNACK response to a Client with its value set to 0 and it receives a PUBLISH packet with the RETAIN flag is set to 1, then it uses the DISCONNECT Reason Code of 0x9A (Retain not supported) as described in [section 4.13](#).

When a new Non-shared Subscription is made, the last retained message, if any, on each matching topic name is sent to the Client as directed by the Retain Handling Subscription Option. These messages are sent with the RETAIN flag set to 1. Which retained messages are sent is controlled by the Retain Handling Subscription Option. At the time of the Subscription:

- If Retain Handling is set to 0 the Server MUST send the retained messages matching the Topic Filter of the subscription to the Client [MQTT-3.3.1-9].
- If Retain Handling is set to 1 then if the subscription did not already exist, the Server MUST send all retained message matching the Topic Filter of the subscription to the Client, and if the subscription did exist the Server MUST NOT send the retained messages. [MQTT-3.3.1-10].
- If Retain Handling is set to 2, the Server MUST NOT send the retained messages [MQTT-3.3.1-11].

Refer to [section 3.8.3.1](#) for a definition of the Subscription Options.

If the Server receives a PUBLISH packet with the RETAIN flag set to 1, and QoS 0 it SHOULD store the new QoS 0 message as the new retained message for that topic, but MAY choose to discard it at any time. If this happens there will be no retained message for that topic.

If the current retained message for a Topic expires, it is discarded and there will be no retained message for that topic.

The setting of the RETAIN flag in an Application Message forwarded by the Server from an established connection is controlled by the Retain As Published subscription option. Refer to [section 3.8.3.1](#) for a definition of the Subscription Options.

- If the value of Retain As Published subscription option is set to 0, the Server MUST set the RETAIN flag to 0 when forwarding an Application Message regardless of how the RETAIN flag was set in the received PUBLISH packet [MQTT-3.3.1-12].
- If the value of Retain As Published subscription option is set to 1, the Server MUST set the RETAIN flag equal to the RETAIN flag in the received PUBLISH packet [MQTT-3.3.1-13].

#### **Non-normative comment**

Retained messages are useful where publishers send state messages on an irregular basis. A new non-shared subscriber will receive the most recent state.

### **3.3.1.4 Remaining Length**

This is the length of Variable Header plus the length of the Payload, encoded as a Variable Byte Integer.

## **3.3.2 PUBLISH Variable Header**

The Variable Header of the PUBLISH Packet contains the following fields in the order: Topic Name, Packet Identifier, and Properties. The rules for encoding Properties are described in [section 2.2.2](#).

### **3.3.2.1 Topic Name**

The Topic Name identifies the information channel to which Payload data is published.

The Topic Name MUST be present as the first field in the PUBLISH packet Variable Header. It MUST be a UTF-8 Encoded String as defined in [section 1.5.4](#) [MQTT-3.3.2-1].



The Topic Name in the PUBLISH packet MUST NOT contain wildcard characters [MQTT-3.3.2-2].

The Topic Name in a PUBLISH packet sent by a Server to a subscribing Client MUST match the Subscription's Topic Filter according to the matching process defined in section 4.7 [MQTT-3.3.2-3]. However, as the Server is permitted to map the Topic Name to another name, it might not be the same as the Topic Name in the original PUBLISH packet.

To reduce the size of the PUBLISH packet the sender can use a Topic Alias. The Topic Alias is described in section 3.3.2.3.4. It is a Protocol Error if the Topic Name is zero length and there is no Topic Alias.

### 3.3.2.2 Packet Identifier

The Packet Identifier field is only present in PUBLISH packets where the QoS level is 1 or 2. Section 2.2.1 provides more information about Packet Identifiers.

### 3.3.2.3 PUBLISH Properties

#### 3.3.2.3.1 Property Length

The length of the Properties in the PUBLISH packet Variable Header encoded as a Variable Byte Integer.

#### 3.3.2.3.2 Payload Format Indicator

**1 (0x01) Byte**, Identifier of the Payload Format Indicator.

Followed by the value of the Payload Format Indicator, either of:

- 0 (0x00) Byte Indicates that the Payload is unspecified bytes, which is equivalent to not sending a Payload Format Indicator.
- 1 (0x01) Byte Indicates that the Payload is UTF-8 Encoded Character Data. The UTF-8 data in the Payload MUST be well-formed UTF-8 as defined by the Unicode specification [Unicode] and restated in RFC 3629 [RFC3629].

A Server MUST send the Payload Format Indicator unaltered to all subscribers receiving the Application Message [MQTT-3.3.2-4]. The receiver MAY validate that the Payload is of the format indicated, and if it is not send a PUBACK, PUBREC, or DISCONNECT with Reason Code of 0x99 (Payload format invalid) as described in section 4.13. Refer to section 5.4.9 for information about security issues in validating the payload format.

#### 3.3.2.3.3 Message Expiry Interval

**2 (0x02) Byte**, Identifier of the Message Expiry Interval.

Followed by the Four Byte Integer representing the Message Expiry Interval.

If present, the Four Byte value is the lifetime of the Application Message in seconds. If the Message Expiry Interval has passed and the Server has not managed to start onward delivery to a matching subscriber, then it MUST delete the copy of the message for that subscriber [MQTT-3.3.2-5].

If absent, the Application Message does not expire.



The PUBLISH packet sent to a Client by the Server MUST contain a Message Expiry Interval set to the received value minus the time that the Application Message has been waiting in the Server [MQTT-3.3.2-6]. Refer to section 4.1 for details and limitations of stored state.

#### 3.3.2.3.4 Topic Alias

**35 (0x23) Byte**, Identifier of the Topic Alias.

Followed by the Two Byte integer representing the Topic Alias value. It is a Protocol Error to include the Topic Alias value more than once.

A Topic Alias is an integer value that is used to identify the Topic instead of using the Topic Name. This reduces the size of the PUBLISH packet, and is useful when the Topic Names are long and the same Topic Names are used repetitively within a Network Connection.

The sender decides whether to use a Topic Alias and chooses the value. It sets a Topic Alias mapping by including a non-zero length Topic Name and a Topic Alias in the PUBLISH packet. The receiver processes the PUBLISH as normal but also sets the specified Topic Alias mapping to this Topic Name.

If a Topic Alias mapping has been set at the receiver, a sender can send a PUBLISH packet that contains that Topic Alias and a zero length Topic Name. The receiver then treats the incoming PUBLISH as if it had contained the Topic Name of the Topic Alias.

A sender can modify the Topic Alias mapping by sending another PUBLISH in the same Network Connection with the same Topic Alias value and a different non-zero length Topic Name.

Topic Alias mappings exist only within a Network Connection and last only for the lifetime of that Network Connection. A receiver MUST NOT carry forward any Topic Alias mappings from one Network Connection to another [MQTT-3.3.2-7].

A Topic Alias of 0 is not permitted. A sender MUST NOT send a PUBLISH packet containing a Topic Alias which has the value 0 [MQTT-3.3.2-8].

A Client MUST NOT send a PUBLISH packet with a Topic Alias greater than the Topic Alias Maximum value returned by the Server in the CONNACK packet [MQTT-3.3.2-9]. A Client MUST accept all Topic Alias values greater than 0 and less than or equal to the Topic Alias Maximum value that it sent in the CONNECT packet [MQTT-3.3.2-10].

A Server MUST NOT send a PUBLISH packet with a Topic Alias greater than the Topic Alias Maximum value sent by the Client in the CONNECT packet [MQTT-3.3.2-11]. A Server MUST accept all Topic Alias values greater than 0 and less than or equal to the Topic Alias Maximum value that it returned in the CONNACK packet [MQTT-3.3.2-12].

The Topic Alias mappings used by the Client and Server are independent from each other. Thus, when a Client sends a PUBLISH containing a Topic Alias value of 1 to a Server and the Server sends a PUBLISH with a Topic Alias value of 1 to that Client they will in general be referring to different Topics.

### 3.3.2.3.5 Response Topic

**8 (0x08) Byte**, Identifier of the Response Topic.

Followed by a UTF-8 Encoded String which is used as the Topic Name for a response message. The Response Topic MUST be a UTF-8 Encoded String as defined in [section 1.5.4 \[MQTT-3.3.2-13\]](#). The Response Topic MUST NOT contain wildcard characters [\[MQTT-3.3.2-14\]](#). It is a Protocol Error to include the Response Topic more than once. The presence of a Response Topic identifies the Message as a Request.

Refer to [section 4.10](#) for more information about Request / Response.

The Server MUST send the Response Topic unaltered to all subscribers receiving the Application Message [\[MQTT-3.3.2-15\]](#).

**Non-normative comment:**

The receiver of an Application Message with a Response Topic sends a response by using the Response Topic as the Topic Name of a PUBLISH. If the Request Message contains a Correlation Data, the receiver of the Request Message should also include this Correlation Data as a property in the PUBLISH packet of the Response Message.

### 3.3.2.3.6 Correlation Data

**9 (0x09) Byte**, Identifier of the Correlation Data.

Followed by Binary Data. The Correlation Data is used by the sender of the Request Message to identify which request the Response Message is for when it is received. It is a Protocol Error to include Correlation Data more than once. If the Correlation Data is not present, the Requester does not require any correlation data.

The Server MUST send the Correlation Data unaltered to all subscribers receiving the Application Message [\[MQTT-3.3.2-16\]](#). The value of the Correlation Data only has meaning to the sender of the Request Message and receiver of the Response Message.

**Non-normative comment**

The receiver of an Application Message which contains both a Response Topic and a Correlation Data sends a response by using the Response Topic as the Topic Name of a PUBLISH. The Client should also send the Correlation Data unaltered as part of the PUBLISH of the responses.

**Non-normative comment**

If the Correlation Data contains information which can cause application failures if modified by the Client responding to the request, it should be encrypted and/or hashed to allow any alteration to be detected.

Refer to [section 4.10](#) for more information about Request / Response

### 3.3.2.3.7 User Property

**38 (0x26) Byte**, Identifier of the User Property.

Followed by a UTF-8 String Pair. The User Property is allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to appear more than once.

The Server MUST send all User Properties unaltered in a PUBLISH packet when forwarding the Application Message to a Client [MQTT-3.3.2-17]. The Server MUST maintain the order of User Properties when forwarding the Application Message [MQTT-3.3.2-18].

#### Non-normative comment

This property is intended to provide a means of transferring application layer name-value tags whose meaning and interpretation are known only by the application programs responsible for sending and receiving them.

### 3.3.2.3.8 Subscription Identifier

**11 (0x0B)**, Identifier of the Subscription Identifier.

Followed by a Variable Byte Integer representing the identifier of the subscription.

The Subscription Identifier can have the value of 1 to 268,435,455. It is a Protocol Error if the Subscription Identifier has a value of 0. Multiple Subscription Identifiers will be included if the publication is the result of a match to more than one subscription, in this case their order is not significant.

### 3.3.2.3.9 Content Type

**3 (0x03)** Identifier of the Content Type.

Followed by a UTF-8 Encoded String describing the content of the Application Message. The Content Type MUST be a UTF-8 Encoded String as defined in section 1.5.4 [MQTT-3.3.2-19].

It is a Protocol Error to include the Content Type more than once. The value of the Content Type is defined by the sending and receiving application.

A Server MUST send the Content Type unaltered to all subscribers receiving the Application Message [MQTT-3.3.2-20].

#### Non-normative comment

The UTF-8 Encoded String may use a MIME content type string to describe the contents of the Application message. However, since the sending and receiving applications are responsible for the definition and interpretation of the string, MQTT performs no validation of the string except to insure it is a valid UTF-8 Encoded String.

#### Non-normative example

Figure 3-9 shows an example of a PUBLISH packet with the Topic Name set to "a/b", the Packet Identifier set to 10, and having no properties.

Figure 3-9 - PUBLISH packet Variable Header non-normative example

	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
Packet Identifier									
byte 6	Packet Identifier MSB (0)	0	0	0	0	0	0	0	0
byte 7	Packet Identifier LSB (10)	0	0	0	0	1	0	1	0
Property Length									
byte 8	No Properties	0	0	0	0	0	0	0	0

1708

### 1709 3.3.3 PUBLISH Payload

1710 The Payload contains the Application Message that is being published. The content and format of the  
 1711 data is application specific. The length of the Payload can be calculated by subtracting the length of the  
 1712 Variable Header from the Remaining Length field that is in the Fixed Header. It is valid for a PUBLISH  
 1713 packet to contain a zero length Payload.

1714

### 1715 3.3.4 PUBLISH Actions

1716 The receiver of a PUBLISH Packet MUST respond with the packet as determined by the QoS in the  
 1717 PUBLISH Packet [MQTT-3.3.4-1].

1718

1719 Table 3-3 Expected PUBLISH packet response

QoS Level	Expected Response
QoS 0	None
QoS 1	PUBACK packet
QoS 2	PUBREC packet

1720

1721 The Client uses a PUBLISH packet to send an Application Message to the Server, for distribution to  
 1722 Clients with matching subscriptions.

1723

1724 The Server uses a PUBLISH packet to send an Application Message to each Client which has a matching  
 1725 subscription. The PUBLISH packet includes the Subscription Identifier carried in the SUBSCRIBE packet,  
 1726 if there was one.

1727

1728 When Clients make subscriptions with Topic Filters that include wildcards, it is possible for a Client's  
 1729 subscriptions to overlap so that a published message might match multiple filters. In this case the Server  
 1730 MUST deliver the message to the Client respecting the maximum QoS of all the matching subscriptions  
 1731 [MQTT-3.3.4-2]. In addition, the Server MAY deliver further copies of the message, one for each  
 1732 additional matching subscription and respecting the subscription's QoS in each case.

1733

1734 If a Client receives an unsolicited Application Message (not resulting from a subscription) which has a  
 1735 QoS greater than Maximum QoS, it uses a DISCONNECT packet with Reason Code 0x9B (QoS not  
 1736 supported ) as described in section 4.13 Handling errors.

1737

If the Client specified a Subscription Identifier for any of the overlapping subscriptions the Server MUST send those Subscription Identifiers in the message which is published as the result of the subscriptions [MQTT-3.3.4-3]. If the Server sends a single copy of the message it MUST include in the PUBLISH packet the Subscription Identifiers for all matching subscriptions which have a Subscription Identifiers, their order is not significant [MQTT-3.3.4-4]. If the Server sends multiple PUBLISH packets it MUST send, in each of them, the Subscription Identifier of the matching subscription if it has a Subscription Identifier [MQTT-3.3.4-5].

It is possible that the Client made several subscriptions which match a publication and that it used the same identifier for more than one of them. In this case the PUBLISH packet will carry multiple identical Subscription Identifiers.

It is a Protocol Error for a PUBLISH packet to contain any Subscription Identifier other than those received in SUBSCRIBE packet which caused it to flow. A PUBLISH packet sent from a Client to a Server MUST NOT contain a Subscription Identifier [MQTT-3.3.4-6].

If the subscription was shared, then only the Subscription Identifiers that were present in the SUBSCRIBE packet from the Client which is receiving the message are returned in the PUBLISH packet.

The action of the recipient when it receives a PUBLISH packet depends on the QoS level as described in section 4.3.

If the PUBLISH packet contains a Topic Alias, the receiver processes it as follows:

- 1) A Topic Alias value of 0 or greater than the Maximum Topic Alias is a Protocol Error, the receiver uses DISCONNECT with Reason Code of 0x94 (Topic Alias invalid) as described in section 4.13.
- 2) If the receiver has already established a mapping for the Topic Alias, then
  - a) If the packet has a zero length Topic Name, the receiver processes it using the Topic Name that corresponds to the Topic Alias
  - b) If the packet contains a non-zero length Topic Name, the receiver processes the packet using that Topic Name and updates its mapping for the Topic Alias to the Topic Name from the incoming packet
- 3) If the receiver does not already have a mapping for this Topic Alias
  - a) If the packet has a zero length Topic Name field it is a Protocol Error and the receiver uses DISCONNECT with Reason Code of 0x82 (Protocol Error) as described in section 4.13.
  - b) If the packet contains a Topic Name with a non-zero length, the receiver processes the packet using that Topic Name and sets its mappings for the Topic Alias to Topic Name from the incoming packet.

#### **Non-normative Comment**

If the Server distributes Application Messages to Clients at different protocol levels (such as MQTT V3.1.1) which do not support properties or other features provided by this specification, some information in the Application Message can be lost, and applications which depend on this information might not work correctly.

The Client MUST NOT send more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets for which it has not received PUBACK, PUBCOMP, or PUBREC with a Reason Code of 128 or greater from the Server [MQTT-3.3.4-7]. If it receives more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets where it has not sent a PUBACK or PUBCOMP in response, the Server uses a DISCONNECT packet

with Reason Code 0x93 (Receive Maximum exceeded) as described in [section 4.13](#) Handling errors. Refer to [section 4.9](#) for more information about flow control.

The Client MUST NOT delay the sending of any packets other than PUBLISH packets due to having sent Receive Maximum PUBLISH packets without receiving acknowledgements for them [MQTT-3.3.4-8]. The value of Receive Maximum applies only to the current Network Connection.

**Non-normative comment**

The Client might choose to send fewer than Receive Maximum messages to the Server without receiving acknowledgement, even if it has more than this number of messages available to send.

**Non-normative comment**

The Client might choose to suspend the sending of QoS 0 PUBLISH packets when it suspends the sending of QoS 1 and QoS 2 PUBLISH packets.

**Non-normative comment**

If the Client sends QoS 1 or QoS 2 PUBLISH packets before it has received a CONNACK packet, it risks being disconnected because it has sent more than Receive Maximum publications.

The Server MUST NOT send more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets for which it has not received PUBACK, PUBCOMP, or PUBREC with a Reason Code of 128 or greater from the Client [MQTT-3.3.4-9]. If it receives more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets where it has not sent a PUBACK or PUBCOMP in response, the Client uses DISCONNECT with Reason Code 0x93 (Receive Maximum exceeded) as described in [section 4.13](#) Handling errors. Refer to [section 4.9](#) for more information about flow control.

The Server MUST NOT delay the sending of any packets other than PUBLISH packets due to having sent Receive Maximum PUBLISH packets without receiving acknowledgements for them [MQTT-3.3.4-10].

**Non-normative comment**

The Server might choose to send fewer than Receive Maximum messages to the Client without receiving acknowledgement, even if it has more than this number of messages available to send.

**Non-normative comment**

The Server might choose to suspend the sending of QoS 0 PUBLISH packets when it suspends the sending of QoS 1 and QoS 2 PUBLISH packets.

## 3.4 PUBACK – Publish acknowledgement

A PUBACK packet is the response to a PUBLISH packet with QoS 1.

### 3.4.1 PUBACK Fixed Header

Figure 3-10 - PUBACK packet Fixed Header

Bit	7	6	5	4	3	2	1	0
-----	---	---	---	---	---	---	---	---

byte 1	MQTT Control Packet type (4)				Reserved			
	0	1	0	0	0	0	0	0
byte 2	Remaining Length							

### Remaining Length field

This is the length of the Variable Header, encoded as a Variable Byte Integer.

## 3.4.2 PUBACK Variable Header

The Variable Header of the PUBACK Packet contains the following fields in the order: Packet Identifier from the PUBLISH packet that is being acknowledged, PUBACK Reason Code, Property Length, and the Properties. The rules for encoding Properties are described in [section 2.2.2](#).

Figure 3-11 – PUBACK packet Variable Header

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							
byte 3	PUBACK Reason Code							
byte 4	Property Length							

### 3.4.2.1 PUBACK Reason Code

Byte 3 in the Variable Header is the PUBACK Reason Code. If the Remaining Length is 2, then there is no Reason Code and the value of 0x00 (Success) is used.

Table 3-4 - PUBACK Reason Codes

Value	Hex	Reason Code name	Description
0	0x00	Success	The message is accepted. Publication of the QoS 1 message proceeds.
16	0x10	No matching subscribers	The message is accepted but there are no subscribers. This is sent only by the Server. If the Server knows that there are no matching subscribers, it MAY use this Reason Code instead of 0x00 (Success).
128	0x80	Unspecified error	The receiver does not accept the publish but either does not want to reveal the reason, or it does not match one of the other values.
131	0x83	Implementation specific error	The PUBLISH is valid but the receiver is not willing to accept it.
135	0x87	Not authorized	The PUBLISH is not authorized.



144	0x90	Topic Name invalid	The Topic Name is not malformed, but is not accepted by this Client or Server.
145	0x91	Packet identifier in use	The Packet Identifier is already in use. This might indicate a mismatch in the Session State between the Client and Server.
151	0x97	Quota exceeded	An implementation or administrative imposed limit has been exceeded.
153	0x99	Payload format invalid	The payload format does not match the specified Payload Format Indicator.

1847

1848 The Client or Server sending the PUBACK packet MUST use one of the PUBACK Reason Codes [MQTT-  
 1849 3.4.2-1]. The Reason Code and Property Length can be omitted if the Reason Code is 0x00 (Success)  
 1850 and there are no Properties. In this case the PUBACK has a Remaining Length of 2.

1851

## 1852 3.4.2.2 PUBACK Properties

### 1853 3.4.2.2.1 Property Length

1854 The length of the Properties in the PUBACK packet Variable Header encoded as a Variable Byte Integer.  
 1855 If the Remaining Length is less than 4 there is no Property Length and the value of 0 is used.

1856

### 1857 3.4.2.2.2 Reason String

1858 **31 (0x1F) Byte**, Identifier of the Reason String.

1859 Followed by the UTF-8 Encoded String representing the reason associated with this response. This  
 1860 Reason String is a human readable string designed for diagnostics and is not intended to be parsed by  
 1861 the receiver.

1862

1863 The sender uses this value to give additional information to the receiver. The sender MUST NOT send  
 1864 this property if it would increase the size of the PUBACK packet beyond the Maximum Packet Size  
 1865 specified by the receiver [MQTT-3.4.2-2]. It is a Protocol Error to include the Reason String more than  
 1866 once.

1867

### 1868 3.4.2.2.3 User Property

1869 **38 (0x26) Byte**, Identifier of the User Property.

1870 Followed by UTF-8 String Pair. This property can be used to provide additional diagnostic or other  
 1871 information. The sender MUST NOT send this property if it would increase the size of the PUBACK  
 1872 packet beyond the Maximum Packet Size specified by the receiver [MQTT-3.4.2-3]. The User Property is  
 1873 allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to  
 1874 appear more than once.

1875

## 1876 3.4.3 PUBACK Payload

1877 The PUBACK packet has no Payload.

1878



### 3.4.4 PUBACK Actions

This is described in [section 4.3.2](#).

## 3.5 PUBREC – Publish received (QoS 2 delivery part 1)

A PUBREC packet is the response to a PUBLISH packet with QoS 2. It is the second packet of the QoS 2 protocol exchange.

### 3.5.1 PUBREC Fixed Header

Figure 3-12 - PUBREC packet Fixed Header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (5)				Reserved			
	0	1	0	1	0	0	0	0
byte 2	Remaining Length							

#### Remaining Length field

This is the length of the Variable Header, encoded as a Variable Byte Integer.

### 3.5.2 PUBREC Variable Header

The Variable Header of the PUBREC Packet consists of the following fields in the order: the Packet Identifier from the PUBLISH packet that is being acknowledged, PUBREC Reason Code, and Properties. The rules for encoding Properties are described in [section 2.2.2](#).

Figure 3-13 - PUBREC packet Variable Header

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							
byte 3	PUBREC Reason Code							
byte 4	Property Length							

#### 3.5.2.1 PUBREC Reason Code

Byte 3 in the Variable Header is the PUBREC Reason Code. If the Remaining Length is 2, then the Publish Reason Code has the value 0x00 (Success).

Table 3-5 – PUBREC Reason Codes

Value	Hex	Reason Code name	Description
-------	-----	------------------	-------------

0	0x00	Success	The message is accepted. Publication of the QoS 2 message proceeds.
16	0x10	No matching subscribers.	The message is accepted but there are no subscribers. This is sent only by the Server. If the Server knows that there are no matching subscribers, it MAY use this Reason Code instead of 0x00 (Success).
128	0x80	Unspecified error	The receiver does not accept the publish but either does not want to reveal the reason, or it does not match one of the other values.
131	0x83	Implementation specific error	The PUBLISH is valid but the receiver is not willing to accept it.
135	0x87	Not authorized	The PUBLISH is not authorized.
144	0x90	Topic Name invalid	The Topic Name is not malformed, but is not accepted by this Client or Server.
145	0x91	Packet Identifier in use	The Packet Identifier is already in use. This might indicate a mismatch in the Session State between the Client and Server.
151	0x97	Quota exceeded	An implementation or administrative imposed limit has been exceeded.
153	0x99	Payload format invalid	The payload format does not match the one specified in the Payload Format Indicator.

The Client or Server sending the PUBREC packet MUST use one of the PUBREC Reason Code values. [MQTT-3.5.2-1]. The Reason Code and Property Length can be omitted if the Reason Code is 0x00 (Success) and there are no Properties. In this case the PUBREC has a Remaining Length of 2.

## 3.5.2.2 PUBREC Properties

### 3.5.2.2.1 Property Length

The length of the Properties in the PUBREC packet Variable Header encoded as a Variable Byte Integer. If the Remaining Length is less than 4 there is no Property Length and the value of 0 is used.

### 3.5.2.2.2 Reason String

**31 (0x1F) Byte**, Identifier of the Reason String.

Followed by the UTF-8 Encoded String representing the reason associated with this response. This Reason String is human readable, designed for diagnostics and SHOULD NOT be parsed by the receiver.

The sender uses this value to give additional information to the receiver. The sender MUST NOT send this property if it would increase the size of the PUBREC packet beyond the Maximum Packet Size specified by the receiver [MQTT-3.5.2-2]. It is a Protocol Error to include the Reason String more than once.

### 3.5.2.2.3 User Property

**38 (0x26) Byte**, Identifier of the User Property.

Followed by UTF-8 String Pair. This property can be used to provide additional diagnostic or other information. The sender **MUST NOT** send this property if it would increase the size of the PUBREC packet beyond the Maximum Packet Size specified by the receiver [MQTT-3.5.2-3]. The User Property is allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to appear more than once.

### 3.5.3 PUBREC Payload

The PUBREC packet has no Payload.

### 3.5.4 PUBREC Actions

This is described in [section 4.3.3](#).

## 3.6 PUBREL – Publish release (QoS 2 delivery part 2)

A PUBREL packet is the response to a PUBREC packet. It is the third packet of the QoS 2 protocol exchange.

### 3.6.1 PUBREL Fixed Header

Figure 3-14 – PUBREL packet Fixed Header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (6)				Reserved			
	0	1	1	0	0	0	1	0
byte 2	Remaining Length							

Bits 3,2,1 and 0 of the Fixed Header in the PUBREL packet are reserved and **MUST** be set to 0,0,1 and 0 respectively. The Server **MUST** treat any other value as malformed and close the Network Connection [MQTT-3.6.1-1].

#### Remaining Length field

This is the length of the Variable Header, encoded as a Variable Byte Integer.

### 3.6.2 PUBREL Variable Header

The Variable Header of the PUBREL Packet contains the following fields in the order: the Packet Identifier from the PUBREC packet that is being acknowledged, PUBREL Reason Code, and Properties. The rules for encoding Properties are described in [section 2.2.2](#).

Figure 3-15 – PUBREL packet Variable Header

Bit	7	6	5	4	3	2	1	0
-----	---	---	---	---	---	---	---	---

byte 1	Packet Identifier MSB
byte 2	Packet Identifier LSB
byte 3	PUBREL Reason Code
byte 4	Property Length

1958

### 1959 3.6.2.1 PUBREL Reason Code

1960 Byte 3 in the Variable Header is the PUBREL Reason Code. If the Remaining Length is 2, the value of  
1961 0x00 (Success) is used.

1962

1963 Table 3-6 - PUBREL Reason Codes

Value	Hex	Reason Code name	Description
0	0x00	Success	Message released.
146	0x92	Packet Identifier not found	The Packet Identifier is not known. This is not an error during recovery, but at other times indicates a mismatch between the Session State on the Client and Server.

1964

1965 The Client or Server sending the PUBREL packet MUST use one of the PUBREL Reason Code values  
1966 [MQTT-3.6.2-1]. The Reason Code and Property Length can be omitted if the Reason Code is 0x00  
1967 (Success) and there are no Properties. In this case the PUBREL has a Remaining Length of 2.

1968

### 1969 3.6.2.2 PUBREL Properties

#### 1970 3.6.2.2.1 Property Length

1971 The length of the Properties in the PUBREL packet Variable Header encoded as a Variable Byte Integer.  
1972 If the Remaining Length is less than 4 there is no Property Length and the value of 0 is used.

1973

#### 1974 3.6.2.2.2 Reason String

1975 **31 (0x1F) Byte**, Identifier of the Reason String.

1976 Followed by the UTF-8 Encoded String representing the reason associated with this response. This  
1977 Reason String is human readable, designed for diagnostics and SHOULD NOT be parsed by the  
1978 receiver.

1979

1980 The sender uses this value to give additional information to the receiver. The sender MUST NOT send  
1981 this Property if it would increase the size of the PUBREL packet beyond the Maximum Packet Size  
1982 specified by the receiver [MQTT-3.6.2-2]. It is a Protocol Error to include the Reason String more than  
1983 once.

1984

#### 1985 3.6.2.2.3 User Property

1986 **38 (0x26) Byte**, Identifier of the User Property.

Followed by UTF-8 String Pair. This property can be used to provide additional diagnostic or other information for the PUBREL. The sender MUST NOT send this property if it would increase the size of the PUBREL packet beyond the Maximum Packet Size specified by the receiver [MQTT-3.6.2-3]. The User Property is allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to appear more than once.

### 3.6.3 PUBREL Payload

The PUBREL packet has no Payload.

### 3.6.4 PUBREL Actions

This is described in [section 4.3.3](#).

## 3.7 PUBCOMP – Publish complete (QoS 2 delivery part 3)

The PUBCOMP packet is the response to a PUBREL packet. It is the fourth and final packet of the QoS 2 protocol exchange.

### 3.7.1 PUBCOMP Fixed Header

Figure 3-16 – PUBCOMP packet Fixed Header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control packet type (7)				Reserved			
	0	1	1	1	0	0	0	0
byte 2	Remaining Length							

#### Remaining Length field

This is the length of the Variable Header, encoded as a Variable Byte Integer.

### 3.7.2 PUBCOMP Variable Header

The Variable Header of the PUBCOMP Packet contains the following fields in the order: Packet Identifier from the PUBREL packet that is being acknowledged, PUBCOMP Reason Code, and Properties. The rules for encoding Properties are described in [section 2.2.2](#).

Figure 3-17 - PUBCOMP packet Variable Header

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							
byte 3	PUBCOMP Reason Code							

byte 4	Property Length
--------	-----------------

### 3.7.2.1 PUBCOMP Reason Code

Byte 3 in the Variable Header is the PUBCOMP Reason Code. If the Remaining Length is 2, then the value 0x00 (Success) is used.

Table 3-7 – PUBCOMP Reason Codes

Value	Hex	Reason Code name	Description
0	0x00	Success	Packet Identifier released. Publication of QoS 2 message is complete.
146	0x92	Packet Identifier not found	The Packet Identifier is not known. This is not an error during recovery, but at other times indicates a mismatch between the Session State on the Client and Server.

The Client or Server sending the PUBCOMP packet MUST use one of the PUBCOMP Reason Code values [MQTT-3.7.2-1]. The Reason Code and Property Length can be omitted if the Reason Code is 0x00 (Success) and there are no Properties. In this case the PUBCOMP has a Remaining Length of 2.

### 3.7.2.2 PUBCOMP Properties

#### 3.7.2.2.1 Property Length

The length of the Properties in the PUBCOMP packet Variable Header encoded as a Variable Byte Integer. If the Remaining Length is less than 4 there is no Property Length and the value of 0 is used.

#### 3.7.2.2.2 Reason String

**31 (0x1F) Byte**, Identifier of the Reason String.

Followed by the UTF-8 Encoded String representing the reason associated with this response. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the receiver.

The sender uses this value to give additional information to the receiver. The sender MUST NOT send this Property if it would increase the size of the PUBCOMP packet beyond the Maximum Packet Size specified by the receiver [MQTT-3.7.2-2]. It is a Protocol Error to include the Reason String more than once.

#### 3.7.2.2.3 User Property

**38 (0x26) Byte**, Identifier of the User Property.

Followed by UTF-8 String Pair. This property can be used to provide additional diagnostic or other information. The sender MUST NOT send this property if it would increase the size of the PUBCOMP packet beyond the Maximum Packet Size specified by the receiver [MQTT-3.7.2-3]. The User Property is allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to appear more than once.

### 3.7.3 PUBCOMP Payload

The PUBCOMP packet has no Payload.

### 3.7.4 PUBCOMP Actions

This is described in [section 4.3.3](#).

## 3.8 SUBSCRIBE - Subscribe request

The SUBSCRIBE packet is sent from the Client to the Server to create one or more Subscriptions. Each Subscription registers a Client's interest in one or more Topics. The Server sends PUBLISH packets to the Client to forward Application Messages that were published to Topics that match these Subscriptions. The SUBSCRIBE packet also specifies (for each Subscription) the maximum QoS with which the Server can send Application Messages to the Client.

### 3.8.1 SUBSCRIBE Fixed Header

Figure 3-18 SUBSCRIBE packet Fixed Header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (8)				Reserved			
	1	0	0	0	0	0	1	0
byte 2	Remaining Length							

Bits 3,2,1 and 0 of the Fixed Header of the SUBSCRIBE packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network Connection [MQTT-3.8.1-1].

### Remaining Length field

This is the length of Variable Header plus the length of the Payload, encoded as a Variable Byte Integer.

### 3.8.2 SUBSCRIBE Variable Header

The Variable Header of the SUBSCRIBE Packet contains the following fields in the order: Packet Identifier, and Properties. [Section 2.2.1](#) provides more information about Packet Identifiers. The rules for encoding Properties are described in [section 2.2.2](#).

#### Non-normative example

Figure 3-19 shows an example of a SUBSCRIBE variable header with a Packet Identifier of 10 and no properties.

Figure 3-19 – SUBSCRIBE Variable Header example

	Description	7	6	5	4	3	2	1	0
Packet Identifier									
byte 1	Packet Identifier MSB (0)	0	0	0	0	0	0	0	0
byte 2	Packet Identifier LSB (10)	0	0	0	0	1	0	1	0
byte 3	Property Length (0)	0	0	0	0	0	0	0	0

2083

## 2084 3.8.2.1 SUBSCRIBE Properties

### 2085 3.8.2.1.1 Property Length

2086 The length of Properties in the SUBSCRIBE packet Variable Header encoded as a Variable Byte Integer.  
2087

### 2088 3.8.2.1.2 Subscription Identifier

2089 **11 (0x0B) Byte**, Identifier of the Subscription Identifier.

2090 Followed by a Variable Byte Integer representing the identifier of the subscription. The Subscription  
2091 Identifier can have the value of 1 to 268,435,455. It is a Protocol Error if the Subscription Identifier has a  
2092 value of 0. It is a Protocol Error to include the Subscription Identifier more than once.

2093

2094 The Subscription Identifier is associated with any subscription created or modified as the result of this  
2095 SUBSCRIBE packet. If there is a Subscription Identifier, it is stored with the subscription. If this property is  
2096 not specified, then the absence of a Subscription Identifier is stored with the subscription.

2097

2098 Refer to [section 3.8.3.1](#) for more information about the handling of Subscription Identifiers.

2099

### 2100 3.8.2.1.3 User Property

2101 **38 (0x26) Byte**, Identifier of the User Property.

2102 Followed by a UTF-8 String Pair.

2103

2104 The User Property is allowed to appear multiple times to represent multiple name, value pairs. The same  
2105 name is allowed to appear more than once.

2106

#### 2107 Non-normative comment

2108 User Properties on the SUBSCRIBE packet can be used to send subscription related properties  
2109 from the Client to the Server. The meaning of these properties is not defined by this specification.

2110

## 2111 3.8.3 SUBSCRIBE Payload

2112 The Payload of a SUBSCRIBE packet contains a list of Topic Filters indicating the Topics to which the  
2113 Client wants to subscribe. **The Topic Filters MUST be a UTF-8 Encoded String [MQTT-3.8.3-1].** Each  
2114 Topic Filter is followed by a Subscription Options byte.

2115



2116 The Payload **MUST** contain at least one Topic Filter and Subscription Options pair [MQTT-3.8.3-2]. A  
2117 SUBSCRIBE packet with no Payload is a Protocol Error. Refer to [section 4.13](#) for information about  
2118 handling errors.

2119

### 2120 3.8.3.1 Subscription Options

2121 Bits 0 and 1 of the Subscription Options represent Maximum QoS field. This gives the maximum QoS  
2122 level at which the Server can send Application Messages to the Client. It is a Protocol Error if the  
2123 Maximum QoS field has the value 3.

2124

2125 Bit 2 of the Subscription Options represents the No Local option. If the value is 1, Application Messages  
2126 **MUST NOT** be forwarded to a connection with a ClientID equal to the ClientID of the publishing  
2127 connection [MQTT-3.8.3-3]. It is a Protocol Error to set the No Local bit to 1 on a Shared Subscription  
2128 [MQTT-3.8.3-4].

2129

2130 Bit 3 of the Subscription Options represents the Retain As Published option. If 1, Application Messages  
2131 forwarded using this subscription keep the RETAIN flag they were published with. If 0, Application  
2132 Messages forwarded using this subscription have the RETAIN flag set to 0. Retained messages sent  
2133 when the subscription is established have the RETAIN flag set to 1.

2134

2135 Bits 4 and 5 of the Subscription Options represent the Retain Handling option. This option specifies  
2136 whether retained messages are sent when the subscription is established. This does not affect the  
2137 sending of retained messages at any point after the subscribe. If there are no retained messages  
2138 matching the Topic Filter, all of these values act the same. The values are:

2139 0 = Send retained messages at the time of the subscribe

2140 1 = Send retained messages at subscribe only if the subscription does not currently exist

2141 2 = Do not send retained messages at the time of the subscribe

2142 It is a Protocol Error to send a Retain Handling value of 3.

2143

2144 Bits 6 and 7 of the Subscription Options byte are reserved for future use. The Server **MUST** treat a  
2145 SUBSCRIBE packet as malformed if any of Reserved bits in the Payload are non-zero [MQTT-3.8.3-5].

2146

#### 2147 **Non-normative comment**

2148 The No Local and Retain As Published subscription options can be used to implement bridging  
2149 where the Client is sending the message on to another Server.

2150

#### 2151 **Non-normative comment**

2152 Not sending retained messages for an existing subscription is useful when a reconnect is done  
2153 and the Client is not certain whether the subscriptions were completed in the previous connection  
2154 to the Session.

2155

#### 2156 **Non-normative comment**

2157 Not sending stored retained messages because of a new subscription is useful where a Client  
2158 wishes to receive change notifications and does not need to know the initial state.

2159

#### 2160 **Non-normative comment**

2161 For a Server that indicates it does not support retained messages, all valid values of Retain As  
 2162 Published and Retain Handling give the same result which is to not send any retained messages  
 2163 at subscribe and to set the RETAIN flag to 0 for all messages.

2164

2165 Figure 3-20– SUBSCRIBE packet Payload format

Description	7	6	5	4	3	2	1	0
Topic Filter								
byte 1	Length MSB							
byte 2	Length LSB							
bytes 3..N	Topic Filter							
Subscription Options								
	Reserved		Retain Handling		RAP	NL	QoS	
byte N+1	0	0	X	X	X	X	X	X

2166 RAP means Retain as Published.

2167 NL means No Local.

2168

2169 **Non-normative example**

2170 Figure 3.21 show the SUBSCRIBE Payload example with two Topic Filters. The first is “a/b” with  
 2171 QoS 1, and the second is “c/d” with QoS 2.

2172

2173 Figure 3-21 - Payload byte format non-normative example

	Description	7	6	5	4	3	2	1	0
Topic Filter									
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
Subscription Options									
byte 6	Subscription Options (1)	0	0	0	0	0	0	0	1
Topic Filter									
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

Subscription Options									
byte 12	Subscription Options (2)	0	0	0	0	0	0	1	0

2174

### 2175 3.8.4 SUBSCRIBE Actions

2176 When the Server receives a SUBSCRIBE packet from a Client, the Server MUST respond with a  
 2177 SUBACK packet [MQTT-3.8.4-1]. The SUBACK packet MUST have the same Packet Identifier as the  
 2178 SUBSCRIBE packet that it is acknowledging [MQTT-3.8.4-2].

2179

2180 The Server is permitted to start sending PUBLISH packets matching the Subscription before the Server  
 2181 sends the SUBACK packet.

2182

2183 If a Server receives a SUBSCRIBE packet containing a Topic Filter that is identical to a Non-shared  
 2184 Subscription's Topic Filter for the current Session, then it MUST replace that existing Subscription with a  
 2185 new Subscription [MQTT-3.8.4-3]. The Topic Filter in the new Subscription will be identical to that in the  
 2186 previous Subscription, although its Subscription Options could be different. If the Retain Handling option  
 2187 is 0, any existing retained messages matching the Topic Filter MUST be re-sent, but Application  
 2188 Messages MUST NOT be lost due to replacing the Subscription [MQTT-3.8.4-4].

2189

2190 If a Server receives a Non-shared Topic Filter that is not identical to any Topic Filter for the current  
 2191 Session, a new Non-shared Subscription is created. If the Retain Handling option is not 2, all matching  
 2192 retained messages are sent to the Client.

2193

2194 If a Server receives a Topic Filter that is identical to the Topic Filter for a Shared Subscription that already  
 2195 exists on the Server, the Session is added as a subscriber to that Shared Subscription. No retained  
 2196 messages are sent.

2197

2198 If a Server receives a Shared Subscription Topic Filter that is not identical to any existing Shared  
 2199 Subscription's Topic Filter, a new Shared Subscription is created. The Session is added as a subscriber  
 2200 to that Shared Subscription. No retained messages are sent.

2201

2202 Refer to [section 4.8](#) for more details on Shared Subscriptions.

2203

2204 If a Server receives a SUBSCRIBE packet that contains multiple Topic Filters it MUST handle that packet  
 2205 as if it had received a sequence of multiple SUBSCRIBE packets, except that it combines their responses  
 2206 into a single SUBACK response [MQTT-3.8.4-5].

2207

2208 The SUBACK packet sent by the Server to the Client MUST contain a Reason Code for each Topic  
 2209 Filter/Subscription Option pair [MQTT-3.8.4-6]. This Reason Code MUST either show the maximum QoS  
 2210 that was granted for that Subscription or indicate that the subscription failed [MQTT-3.8.4-7]. The Server  
 2211 might grant a lower Maximum QoS than the subscriber requested. The QoS of Application Messages sent  
 2212 in response to a Subscription MUST be the minimum of the QoS of the originally published message and  
 2213 the Maximum QoS granted by the Server [MQTT-3.8.4-8]. The server is permitted to send duplicate  
 2214 copies of a message to a subscriber in the case where the original message was published with QoS 1  
 2215 and the maximum QoS granted was QoS 0.

2216

#### 2217 Non-normative comment

2218 If a subscribing Client has been granted maximum QoS 1 for a particular Topic Filter, then a

2219 QoS 0 Application Message matching the filter is delivered to the Client at QoS 0. This means  
2220 that at most one copy of the message is received by the Client. On the other hand, a QoS 2  
2221 Message published to the same topic is downgraded by the Server to QoS 1 for delivery to the  
2222 Client, so that Client might receive duplicate copies of the Message.  
2223

2224 **Non-normative comment**

2225 If the subscribing Client has been granted maximum QoS 0, then an Application Message  
2226 originally published as QoS 2 might get lost on the hop to the Client, but the Server should never  
2227 send a duplicate of that Message. A QoS 1 Message published to the same topic might either get  
2228 lost or duplicated on its transmission to that Client.  
2229

2230 **Non-normative comment**

2231 Subscribing to a Topic Filter at QoS 2 is equivalent to saying "I would like to receive Messages  
2232 matching this filter at the QoS with which they were published". This means a publisher is  
2233 responsible for determining the maximum QoS a Message can be delivered at, but a subscriber is  
2234 able to require that the Server downgrades the QoS to one more suitable for its usage.  
2235

2236 The Subscription Identifiers are part of the Session State in the Server and are returned to the Client  
2237 receiving a matching PUBLISH packet. They are removed from the Server's Session State when the  
2238 Server receives an UNSUBSCRIBE packet, when the Server receives a SUBSCRIBE packet from the  
2239 Client for the same Topic Filter but with a different Subscription Identifier or with no Subscription Identifier,  
2240 or when the Server sends Session Present 0 in a CONNACK packet.  
2241

2242 The Subscription Identifiers do not form part of the Client's Session State in the Client. In a useful  
2243 implementation, a Client will associate the Subscription Identifiers with other Client side state, this state is  
2244 typically removed when the Client unsubscribes, when the Client subscribes for the same Topic Filter with  
2245 a different identifier or no identifier, or when the Client receives Session Present 0 in a CONNACK  
2246 packet.  
2247

2248 The Server need not use the same set of Subscription Identifiers in the retransmitted PUBLISH packet.  
2249 The Client can remake a Subscription by sending a SUBSCRIBE packet containing a Topic Filter that is  
2250 identical to the Topic Filter of an existing Subscription in the current Session. If the Client remade a  
2251 subscription after the initial transmission of a PUBLISH packet and used a different Subscription Identifier,  
2252 then the Server is allowed to use the identifiers from the first transmission in any  
2253 retransmission. Alternatively, the Server is allowed to use the new identifiers during a retransmission. The  
2254 Server is not allowed to revert to the old identifier after it has sent a PUBLISH packet containing the new  
2255 one.  
2256

2257 **Non-normative comment**

2258 Usage scenarios, for illustration of Subscription Identifiers.

- 2259 • The Client implementation indicates via its programming interface that a publication matched  
2260 more than one subscription. The Client implementation generates a new identifier each time  
2261 a subscription is made. If the returned publication carries more than one Subscription  
2262 Identifier, then the publication matched more than one subscription.  
2263
- 2264 • The Client implementation allows the subscriber to direct messages to a callback associated  
2265 with the subscription. The Client implementation generates an identifier which uniquely maps  
2266 the identifier to the callback. When a publication is received it uses the Subscription Identifier  
2267 to determine which callback is driven.  
2268
- 2269 • The Client implementation returns the topic string used to make the subscription to the  
2270 application when it delivers the published message. To achieve this the Client generates an  
2271 identifier which uniquely identifies the Topic Filter. When a publication is received the

2272 Client implementation uses the identifiers to look up the original Topic Filters and return them  
 2273 to the Client application.  
 2274  
 2275 • A gateway forwards publications received from a Server to Clients that have made  
 2276 subscriptions to the gateway. The gateway implementation maintains a map of each unique  
 2277 Topic Filter it receives to the set of ClientID, Subscription Identifier pairs that it also  
 2278 received. It generates a unique identifier for each Topic Filter that it forwards to the Server.  
 2279 When a publication is received, the gateway uses the Subscription Identifiers it received from  
 2280 the Server to look up the Client Identifier, Subscription Identifier pairs associated with them. It  
 2281 adds these to the PUBLISH packets it sends to the Clients. If the upstream Server sent  
 2282 multiple PUBLISH packets because the message matched multiple subscriptions, then this  
 2283 behavior is mirrored to the Clients.  
 2284

2285 **3.9 SUBACK – Subscribe acknowledgement**

2286 A SUBACK packet is sent by the Server to the Client to confirm receipt and processing of a SUBSCRIBE  
 2287 packet.  
 2288  
 2289 A SUBACK packet contains a list of Reason Codes, that specify the maximum QoS level that was  
 2290 granted or the error which was found for each Subscription that was requested by the SUBSCRIBE.  
 2291

2292 **3.9.1 SUBACK Fixed Header**

2293 *Figure 3-22 - SUBACK Packet Fixed Header*

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (9)				Reserved			
	1	0	0	1	0	0	0	0
byte 2	Remaining Length							

2294  
 2295 **Remaining Length field**  
 2296 This is the length of Variable Header plus the length of the Payload, encoded as a Variable Byte Integer.  
 2297

2298 **3.9.2 SUBACK Variable Header**

2299 The Variable Header of the SUBACK Packet contains the following fields in the order: the Packet  
 2300 Identifier from the SUBSCRIBE Packet that is being acknowledged, and Properties.  
 2301

2302 **3.9.2.1 SUBACK Properties**

2303 **3.9.2.1.1 Property Length**

2304 The length of Properties in the SUBACK packet Variable Header encoded as a Variable Byte Integer  
 2305

### 3.9.2.1.2 Reason String

**31 (0x1F) Byte**, Identifier of the Reason String.

Followed by the UTF-8 Encoded String representing the reason associated with this response. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the Client.

The Server uses this value to give additional information to the Client. The Server MUST NOT send this Property if it would increase the size of the SUBACK packet beyond the Maximum Packet Size specified by the Client [MQTT-3.9.2-1]. It is a Protocol Error to include the Reason String more than once.

### 3.9.2.1.3 User Property

**38 (0x26) Byte**, Identifier of the User Property.

Followed by UTF-8 String Pair. This property can be used to provide additional diagnostic or other information. The Server MUST NOT send this property if it would increase the size of the SUBACK packet beyond the Maximum Packet Size specified by Client [MQTT-3.9.2-2]. The User Property is allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to appear more than once.

Figure 3-23 SUBACK packet Variable Header

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							

## 3.9.3 SUBACK Payload

The Payload contains a list of Reason Codes. Each Reason Code corresponds to a Topic Filter in the SUBSCRIBE packet being acknowledged. The order of Reason Codes in the SUBACK packet MUST match the order of Topic Filters in the SUBSCRIBE packet [MQTT-3.9.3-1].

Table 3-8 - Subscribe Reason Codes

Value	Hex	Reason Code name	Description
0	0x00	Granted QoS 0	The subscription is accepted and the maximum QoS sent will be QoS 0. This might be a lower QoS than was requested.
1	0x01	Granted QoS 1	The subscription is accepted and the maximum QoS sent will be QoS 1. This might be a lower QoS than was requested.
2	0x02	Granted QoS 2	The subscription is accepted and any received QoS will be sent to this subscription.
128	0x80	Unspecified error	The subscription is not accepted and the Server either does not wish to reveal the reason or none of the other Reason Codes apply.
131	0x83	Implementation specific error	The SUBSCRIBE is valid but the Server does not accept it.

135	0x87	Not authorized	The Client is not authorized to make this subscription.
143	0x8F	Topic Filter invalid	The Topic Filter is correctly formed but is not allowed for this Client.
145	0x91	Packet Identifier in use	The specified Packet Identifier is already in use.
151	0x97	Quota exceeded	An implementation or administrative imposed limit has been exceeded.
158	0x9E	Shared Subscriptions not supported	The Server does not support Shared Subscriptions for this Client.
161	0xA1	Subscription Identifiers not supported	The Server does not support Subscription Identifiers; the subscription is not accepted.
162	0xA2	Wildcard Subscriptions not supported	The Server does not support Wildcard Subscriptions; the subscription is not accepted.

The Server sending a SUBACK packet MUST use one of the Subscribe Reason Codes for each Topic Filter received [MQTT-3.9.3-2].

#### Non-normative comment

There is always one Reason Code for each Topic Filter in the corresponding SUBSCRIBE packet. If the Reason Code is not specific to a Topic Filters (such as 0x91 (Packet Identifier in use)) it is set for each Topic Filter.

## 3.10 UNSUBSCRIBE – Unsubscribe request

An UNSUBSCRIBE packet is sent by the Client to the Server, to unsubscribe from topics.

### 3.10.1 UNSUBSCRIBE Fixed Header

Figure 3.28 – UNSUBSCRIBE packet Fixed Header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (10)				Reserved			
	1	0	1	0	0	0	1	0
byte 2	Remaining Length							

Bits 3,2,1 and 0 of the Fixed Header of the UNSUBSCRIBE packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network Connection [MQTT-3.10.1-1].

#### Remaining Length field

This is the length of Variable Header (2 bytes) plus the length of the Payload, encoded as a Variable Byte Integer.

### 3.10.2 UNSUBSCRIBE Variable Header

The Variable Header of the UNSUBSCRIBE Packet contains the following fields in the order: Packet Identifier, and Properties. [Section 2.2.1](#) provides more information about Packet Identifiers. The rules for encoding Properties are described in [section 2.2.2](#).

#### 3.10.2.1 UNSUBSCRIBE Properties

##### 3.10.2.1.1 Property Length

The length of Properties in the UNSUBSCRIBE packet Variable Header encoded as a Variable Byte Integer.

##### 3.10.2.1.2 User Property

**38 (0x26) Byte**, Identifier of the User Property.

Followed by a UTF-8 String Pair.

The User Property is allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to appear more than once.

**Non-normative comment**

User Properties on the UNSUBSCRIBE packet can be used to send subscription related properties from the Client to the Server. The meaning of these properties is not defined by this specification.

### 3.10.3 UNSUBSCRIBE Payload

The Payload for the UNSUBSCRIBE packet contains the list of Topic Filters that the Client wishes to unsubscribe from. [The Topic Filters in an UNSUBSCRIBE packet MUST be UTF-8 Encoded Strings \[MQTT-3.10.3-1\]](#) as defined in [section 1.5.4](#), packed contiguously.

[The Payload of an UNSUBSCRIBE packet MUST contain at least one Topic Filter \[MQTT-3.10.3-2\]](#). An UNSUBSCRIBE packet with no Payload is a Protocol Error. Refer to [section 4.13](#) for information about handling errors.

**Non-normative example**

Figure 3.30 shows the Payload for an UNSUBSCRIBE packet with two Topic Filters “a/b” and “c/d”.

Figure 3.30 - Payload byte format non-normative example

	Description	7	6	5	4	3	2	1	0
Topic Filter									
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 Filter									
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

2390

### 2391 3.10.4 UNSUBSCRIBE Actions

2392 The Topic Filters (whether they contain wildcards or not) supplied in an UNSUBSCRIBE packet MUST be  
 2393 compared character-by-character with the current set of Topic Filters held by the Server for the Client. If  
 2394 any filter matches exactly then its owning Subscription MUST be deleted [MQTT-3.10.4-1], otherwise no  
 2395 additional processing occurs.

2396  
 2397 When a Server receives UNSUBSCRIBE :

- 2398 • It MUST stop adding any new messages which match the Topic Filters, for delivery to the Client  
 2399 [MQTT-3.10.4-2].
- 2400 • It MUST complete the delivery of any QoS 1 or QoS 2 messages which match the Topic Filters  
 2401 and it has started to send to the Client [MQTT-3.10.4-3].
- 2402 • It MAY continue to deliver any existing messages buffered for delivery to the Client.

2403  
 2404 The Server MUST respond to an UNSUBSCRIBE request by sending an UNSUBACK packet [MQTT-  
 2405 3.10.4-4]. The UNSUBACK packet MUST have the same Packet Identifier as the UNSUBSCRIBE packet.  
 2406 Even where no Topic Subscriptions are deleted, the Server MUST respond with an UNSUBACK [MQTT-  
 2407 3.10.4-5].

2408  
 2409 If a Server receives an UNSUBSCRIBE packet that contains multiple Topic Filters, it MUST process that  
 2410 packet as if it had received a sequence of multiple UNSUBSCRIBE packets, except that it sends just one  
 2411 UNSUBACK response [MQTT-3.10.4-6].

2412  
 2413 If a Topic Filter represents a Shared Subscription, this Session is detached from the Shared Subscription.  
 2414 If this Session was the only Session that the Shared Subscription was associated with, the Shared  
 2415 Subscription is deleted. Refer to [section 4.8.2](#) for a description of Shared Subscription handling.

2416

## 2417 3.11 UNSUBACK – Unsubscribe acknowledgement

2418 The UNSUBACK packet is sent by the Server to the Client to confirm receipt of an UNSUBSCRIBE  
 2419 packet.

2420

### 2421 3.11.1 UNSUBACK Fixed Header

2422 *Figure 3.31 – UNSUBACK packet Fixed Header*

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (11)				Reserved			
	1	0	1	1	0	0	0	0
byte 2	Remaining Length							

## Remaining Length field

This is the length of the Variable Header plus the length of the Payload, encoded as a Variable Byte Integer.

## 3.11.2 UNSUBACK Variable Header

The Variable Header of the UNSUBACK Packet the following fields in the order: the Packet Identifier from the UNSUBSCRIBE Packet that is being acknowledged, and Properties. The rules for encoding Properties are described in [section 2.2.2](#).

Figure 3.32 – UNSUBACK packet Variable Header

Bit	7	6	5	4	3	2	1	0
byte 1	Packet Identifier MSB							
byte 2	Packet Identifier LSB							

## 3.11.2.1 UNSUBACK Properties

### 3.11.2.1.1 Property Length

The length of the Properties in the UNSUBACK packet Variable Header encoded as a Variable Byte Integer.

### 3.11.2.1.2 Reason String

**31 (0x1F) Byte**, Identifier of the Reason String.

Followed by the UTF-8 Encoded String representing the reason associated with this response. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the Client.

The Server uses this value to give additional information to the Client. **The Server MUST NOT send this Property if it would increase the size of the UNSUBACK packet beyond the Maximum Packet Size specified by the Client [MQTT-3.11.2-1].** It is a Protocol Error to include the Reason String more than once.

### 3.11.2.1.3 User Property

**38 (0x26) Byte**, Identifier of the User Property.

Followed by UTF-8 String Pair. This property can be used to provide additional diagnostic or other information. The Server MUST NOT send this property if it would increase the size of the UNSUBACK packet beyond the Maximum Packet Size specified by the Client [MQTT-3.11.2-2]. The User Property is allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to appear more than once.

### 3.11.3 UNSUBACK Payload

The Payload contains a list of Reason Codes. Each Reason Code corresponds to a Topic Filter in the UNSUBSCRIBE packet being acknowledged. The order of Reason Codes in the UNSUBACK packet MUST match the order of Topic Filters in the UNSUBSCRIBE packet [MQTT-3.11.3-1].

The values for the one byte unsigned Unsubscribe Reason Codes are shown below. The Server sending an UNSUBACK packet MUST use one of the Unsubscribe Reason Code values for each Topic Filter received [MQTT-3.11.3-2].

Table 3-9 - Unsubscribe Reason Codes

Value	Hex	Reason Code name	Description
0	0x00	Success	The subscription is deleted.
17	0x11	No subscription existed	No matching Topic Filter is being used by the Client.
128	0x80	Unspecified error	The unsubscribe could not be completed and the Server either does not wish to reveal the reason or none of the other Reason Codes apply.
131	0x83	Implementation specific error	The UNSUBSCRIBE is valid but the Server does not accept it.
135	0x87	Not authorized	The Client is not authorized to unsubscribe.
143	0x8F	Topic Filter invalid	The Topic Filter is correctly formed but is not allowed for this Client.
145	0x91	Packet Identifier in use	The specified Packet Identifier is already in use.

#### Non-normative comment

There is always one Reason Code for each Topic Filter in the corresponding UNSUBSCRIBE packet. If the Reason Code is not specific to a Topic Filters (such as 0x91 (Packet Identifier in use)) it is set for each Topic Filter.

### 3.12 PINGREQ – PING request

The PINGREQ packet is sent from a Client to the Server. It can be used to:

- Indicate to the Server that the Client is alive in the absence of any other MQTT Control Packets being sent from the Client to the Server.
- Request that the Server responds to confirm that it is alive.
- Exercise the network to indicate that the Network Connection is active.

This packet is used in Keep Alive processing. Refer to [section 3.1.2.10](#) for more details.

### 3.12.1 PINGREQ Fixed Header

Figure 3.33 – PINGREQ packet Fixed Header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (12)				Reserved			
	1	1	0	0	0	0	0	0
byte 2	Remaining Length (0)							
	0	0	0	0	0	0	0	0

### 3.12.2 PINGREQ Variable Header

The PINGREQ packet has no Variable Header.

### 3.12.3 PINGREQ Payload

The PINGREQ packet has no Payload.

### 3.12.4 PINGREQ Actions

The Server MUST send a PINGRESP packet in response to a PINGREQ packet [MQTT-3.12.4-1].

## 3.13 PINGRESP – PING response

A PINGRESP Packet is sent by the Server to the Client in response to a PINGREQ packet. It indicates that the Server is alive.

This packet is used in Keep Alive processing. Refer to [section 3.1.2.10](#) for more details.

### 3.13.1 PINGRESP Fixed Header

Figure 3.34 – PINGRESP packet Fixed Header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (13)				Reserved			
	1	1	0	1	0	0	0	0
byte 2	Remaining Length (0)							
	0	0	0	0	0	0	0	0

### 3.13.2 PINGRESP Variable Header

The PINGRESP packet has no Variable Header.

### 3.13.3 PINGRESP Payload

The PINGRESP packet has no Payload.

### 3.13.4 PINGRESP Actions

The Client takes no action on receiving this packet

## 3.14 DISCONNECT – Disconnect notification

The DISCONNECT packet is the final MQTT Control Packet sent from the Client or the Server. It indicates the reason why the Network Connection is being closed. The Client or Server MAY send a DISCONNECT packet before closing the Network Connection. If the Network Connection is closed without the Client first sending a DISCONNECT packet with Reason Code 0x00 (Normal disconnection) and the Connection has a Will Message, the Will Message is published. Refer to [section 3.1.2.5](#) for further details.

A Server MUST NOT send a DISCONNECT until after it has sent a CONNACK with Reason Code of less than 0x80 [MQTT-3.14.0-1].

### 3.14.1 DISCONNECT Fixed Header

Figure 3.35 – DISCONNECT packet Fixed Header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (14)				Reserved			
	1	1	1	0	0	0	0	0
byte 2	Remaining Length							

The Client or Server MUST validate that reserved bits are set to 0. If they are not zero it sends a DISCONNECT packet with a Reason code of 0x81 (Malformed Packet) as described in [section 4.13](#) [MQTT-3.14.1-1].

#### Remaining Length field

This is the length of the Variable Header encoded as a Variable Byte Integer.

### 3.14.2 DISCONNECT Variable Header

The Variable Header of the DISCONNECT Packet contains the following fields in the order: Disconnect Reason Code, and Properties. The rules for encoding Properties are described in [section 2.2.2](#).

### 3.14.2.1 Disconnect Reason Code

Byte 1 in the Variable Header is the Disconnect Reason Code. If the Remaining Length is less than 1 the value of 0x00 (Normal disconnection) is used.

The values for the one byte unsigned Disconnect Reason Code field are shown below.

Table 3-10 – Disconnect Reason Code values

Value	Hex	Reason Code name	Sent by	Description
0	0x00	Normal disconnection	Client or Server	Close the connection normally. Do not send the Will Message.
4	0x04	Disconnect with Will Message	Client	The Client wishes to disconnect but requires that the Server also publishes its Will Message.
128	0x80	Unspecified error	Client or Server	The Connection is closed but the sender either does not wish to reveal the reason, or none of the other Reason Codes apply.
129	0x81	Malformed Packet	Client or Server	The received packet does not conform to this specification.
130	0x82	Protocol Error	Client or Server	An unexpected or out of order packet was received.
131	0x83	Implementation specific error	Client or Server	The packet received is valid but cannot be processed by this implementation.
135	0x87	Not authorized	Server	The request is not authorized.
137	0x89	Server busy	Server	The Server is busy and cannot continue processing requests from this Client.
139	0x8B	Server shutting down	Server	The Server is shutting down.
141	0x8D	Keep Alive timeout	Server	The Connection is closed because no packet has been received for 1.5 times the Keepalive time.
142	0x8E	Session taken over	Server	Another Connection using the same ClientID has connected causing this Connection to be closed.
143	0x8F	Topic Filter invalid	Server	The Topic Filter is correctly formed, but is not accepted by this Sever.
144	0x90	Topic Name invalid	Client or Server	The Topic Name is correctly formed, but is not accepted by this Client or Server.
147	0x93	Receive Maximum exceeded	Client or Server	The Client or Server has received more than Receive Maximum publication for which it has not sent PUBACK or PUBCOMP.
148	0x94	Topic Alias invalid	Client or Server	The Client or Server has received a PUBLISH packet containing a Topic Alias which is greater than the Maximum Topic Alias it sent in the CONNECT or CONNACK packet.

149	0x95	Packet too large	Client or Server	The packet size is greater than Maximum Packet Size for this Client or Server.
150	0x96	Message rate too high	Client or Server	The received data rate is too high.
151	0x97	Quota exceeded	Client or Server	An implementation or administrative imposed limit has been exceeded.
152	0x98	Administrative action	Client or Server	The Connection is closed due to an administrative action.
153	0x99	Payload format invalid	Client or Server	The payload format does not match the one specified by the Payload Format Indicator.
154	0x9A	Retain not supported	Server	The Server has does not support retained messages.
155	0x9B	QoS not supported	Server	The Client specified a QoS greater than the QoS specified in a Maximum QoS in the CONNACK.
156	0x9C	Use another server	Server	The Client should temporarily change its Server.
157	0x9D	Server moved	Server	The Server is moved and the Client should permanently change its server location.
158	0x9E	Shared Subscriptions not supported	Server	The Server does not support Shared Subscriptions.
159	0x9F	Connection rate exceeded	Server	This connection is closed because the connection rate is too high.
160	0xA0	Maximum connect time	Server	The maximum connection time authorized for this connection has been exceeded.
161	0xA1	Subscription Identifiers not supported	Server	The Server does not support Subscription Identifiers; the subscription is not accepted.
162	0xA2	Wildcard Subscriptions not supported	Server	The Server does not support Wildcard Subscriptions; the subscription is not accepted.

2545

2546 The Client or Server sending the DISCONNECT packet MUST use one of the DISCONNECT Reason  
2547 Code values [MQTT-3.14.2-1]. The Reason Code and Property Length can be omitted if the Reason  
2548 Code is 0x00 (Normal disconnect) and there are no Properties. In this case the DISCONNECT has a  
2549 Remaining Length of 0.

2550

2551 **Non-normative comment**

2552 The DISCONNECT packet is used to indicate the reason for a disconnect for cases where there  
2553 is no acknowledge packet (such as a QoS 0 publish) or when the Client or Server is unable to  
2554 continue processing the Connection.

2555

2556 **Non-normative comment**

2557 The information can be used by the Client to decide whether to retry the connection, and how  
2558 long it should wait before retrying the connection.

2559

## 2560 3.14.2.2 DISCONNECT Properties

### 2561 3.14.2.2.1 Property Length

2562 The length of Properties in the DISCONNECT packet Variable Header encoded as a Variable Byte  
2563 Integer. If the Remaining Length is less than 2, a value of 0 is used.

2564

### 2565 3.14.2.2.2 Session Expiry Interval

2566 **17 (0x11) Byte**, Identifier of the Session Expiry Interval.

2567 Followed by the Four Byte Integer representing the Session Expiry Interval in seconds. It is a Protocol  
2568 Error to include the Session Expiry Interval more than once.

2569

2570 If the Session Expiry Interval is absent, the Session Expiry Interval in the CONNECT packet is used.

2571

2572 The Session Expiry Interval MUST NOT be sent on a DISCONNECT by the Server [MQTT-3.14.2-2].

2573

2574 If the Session Expiry Interval in the CONNECT packet was zero, then it is a Protocol Error to set a non-  
2575 zero Session Expiry Interval in the DISCONNECT packet sent by the Client. If such a non-zero Session  
2576 Expiry Interval is received by the Server, it does not treat it as a valid DISCONNECT packet. The Server  
2577 uses DISCONNECT with Reason Code 0x82 (Protocol Error) as described in [section 4.13](#).

2578

### 2579 3.14.2.2.3 Reason String

2580 **31 (0x1F) Byte**, Identifier of the Reason String.

2581 Followed by the UTF-8 Encoded String representing the reason for the disconnect. This Reason String is  
2582 human readable, designed for diagnostics and SHOULD NOT be parsed by the receiver.

2583

2584 The sender MUST NOT send this Property if it would increase the size of the DISCONNECT packet  
2585 beyond the Maximum Packet Size specified by the receiver [MQTT-3.14.2-3]. It is a Protocol Error to  
2586 include the Reason String more than once.

2587

### 2588 3.14.2.2.4 User Property

2589 **38 (0x26) Byte**, Identifier of the User Property.

2590 Followed by UTF-8 String Pair. This property may be used to provide additional diagnostic or other  
2591 information. The sender MUST NOT send this property if it would increase the size of the DISCONNECT  
2592 packet beyond the Maximum Packet Size specified by the receiver [MQTT-3.14.2-4]. The User Property is  
2593 allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to  
2594 appear more than once.

2595

### 2596 3.14.2.2.5 Server Reference

2597 **28 (0x1C) Byte**, Identifier of the Server Reference.

2598 Followed by a UTF-8 Encoded String which can be used by the Client to identify another Server to use. It  
2599 is a Protocol Error to include the Server Reference more than once.



The Server sends DISCONNECT including a Server Reference and Reason Code 0x9C (Use another server) or 0x9D (Server moved) as described in [section 4.13](#).

Refer to [section 4.11](#) Server Redirection for information about how Server Reference is used.

Figure 3-24 DISCONNECT packet Variable Header non-normative example

	Description	7	6	5	4	3	2	1	0
Disconnect Reason Code									
byte 1		0	0	0	0	0	0	0	0
Properties									
byte 2	Length (5)	0	0	0	0	0	1	1	1
byte 3	Session Expiry Interval identifier (17)	0	0	0	1	0	0	0	1
byte 4	Session Expiry Interval (0)	0	0	0	0	0	0	0	0
byte 5		0	0	0	0	0	0	0	0
byte 6		0	0	0	0	0	0	0	0
byte 7		0	0	0	0	0	0	0	0

### 3.14.3 DISCONNECT Payload

The DISCONNECT packet has no Payload.

### 3.14.4 DISCONNECT Actions

After sending a DISCONNECT packet the sender:

- MUST NOT send any more MQTT Control Packets on that Network Connection [MQTT-3.14.4-1].
- MUST close the Network Connection [MQTT-3.14.4-2].

On receipt of DISCONNECT with a Reason Code of 0x00 (Success) the Server:

- MUST discard any Will Message associated with the current Connection without publishing it [MQTT-3.14.4-3], as described in [section 3.1.2.5](#).

On receipt of DISCONNECT, the receiver:

- SHOULD close the Network Connection.

## 3.15 AUTH – Authentication exchange

An AUTH packet is sent from Client to Server or Server to Client as part of an extended authentication exchange, such as challenge / response authentication. It is a Protocol Error for the Client or Server to send an AUTH packet if the CONNECT packet did not contain the same Authentication Method.

### 3.15.1 AUTH Fixed Header

Figure 3.35 – AUTH packet Fixed Header

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type (15)				Reserved			
	1	1	1	1	0	0	0	0
byte 2	Remaining Length							

Bits 3,2,1 and 0 of the Fixed Header of the AUTH packet are reserved and MUST all be set to 0. The Client or Server MUST treat any other value as malformed and close the Network Connection [MQTT-3.15.1-1].

#### Remaining Length field

This is the length of the Variable Header encoded as a Variable Byte Integer.

### 3.15.2 AUTH Variable Header

The Variable Header of the AUTH Packet contains the following fields in the order: Authenticate Reason Code, and Properties. The rules for encoding Properties are described in [section 2.2.2](#).

#### 3.15.2.1 Authenticate Reason Code

Byte 0 in the Variable Header is the Authenticate Reason Code. The values for the one byte unsigned Authenticate Reason Code field are shown below. The sender of the AUTH Packet MUST use one of the Authenticate Reason Codes [MQTT-3.15.2-1].

Table 3-11 Authenticate Reason Codes

Value	Hex	Reason Code name	Sent by	Description
0	0x00	Success	Server	Authentication is successful
24	0x18	Continue authentication	Client or Server	Continue the authentication with another step
25	0x19	Re-authenticate	Client	Initiate a re-authentication

The Reason Code and Property Length can be omitted if the Reason Code is 0x00 (Success) and there are no Properties. In this case the AUTH has a Remaining Length of 0.

#### 3.15.2.2 AUTH Properties

##### 3.15.2.2.1 Property Length

The length of Properties in the AUTH packet Variable Header encoded as a Variable Byte Integer.

2655 **3.15.2.2.2 Authentication Method**

2656 **21 (0x15) Byte**, Identifier of the Authentication Method.

2657 Followed by a UTF-8 Encoded String containing the name of the authentication method. It is a Protocol  
2658 Error to omit the Authentication Method or to include it more than once. Refer to [section 4.12](#) for more  
2659 information about extended authentication.

2660

2661 **3.15.2.2.3 Authentication Data**

2662 **22 (0x16) Byte**, Identifier of the Authentication Data.

2663 Followed by Binary Data containing authentication data. It is a Protocol Error to include Authentication  
2664 Data more than once. The contents of this data are defined by the authentication method. Refer to  
2665 [section 4.12](#) for more information about extended authentication.

2666

2667 **3.15.2.2.4 Reason String**

2668 **31 (0x1F) Byte**, Identifier of the Reason String.

2669 Followed by the UTF-8 Encoded String representing the reason for the disconnect. This Reason String is  
2670 human readable, designed for diagnostics and SHOULD NOT be parsed by the receiver.

2671

2672 The sender MUST NOT send this property if it would increase the size of the AUTH packet beyond the  
2673 Maximum Packet Size specified by the receiver [MQTT-3.15.2-2]. It is a Protocol Error to include the  
2674 Reason String more than once.

2675

2676 **3.15.2.2.5 User Property**

2677 **38 (0x26) Byte**, Identifier of the User Property.

2678 Followed by UTF-8 String Pair. This property may be used to provide additional diagnostic or other  
2679 information. The sender MUST NOT send this property if it would increase the size of the AUTH packet  
2680 beyond the Maximum Packet Size specified by the receiver [MQTT-3.15.2-3]. The User Property is  
2681 allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to  
2682 appear more than once.

2683

2684 **3.15.3 AUTH Payload**

2685 The AUTH packet has no Payload.

2686

2687 **3.15.4 AUTH Actions**

2688 Refer to [section 4.12](#) for more information about extended authentication.

---

## 4 Operational behavior

### 4.1 Session State

In order to implement QoS 1 and QoS 2 protocol flows the Client and Server need to associate state with the Client Identifier, this is referred to as the Session State. The Server also stores the subscriptions as part of the Session State.

The session can continue across a sequence of Network Connections. It lasts as long as the latest Network Connection plus the Session Expiry Interval.

The Session State in the Client consists of:

- QoS 1 and QoS 2 messages which have been sent to the Server, but have not been completely acknowledged.
- QoS 2 messages which have been received from the Server, but have not been completely acknowledged.

The Session State in the Server consists of:

- The existence of a Session, even if the rest of the Session State is empty.
- The Clients subscriptions, including any Subscription Identifiers.
- QoS 1 and QoS 2 messages which have been sent to the Client, but have not been completely acknowledged.
- QoS 1 and QoS 2 messages pending transmission to the Client and OPTIONALLY QoS 0 messages pending transmission to the Client.
- QoS 2 messages which have been received from the Client, but have not been completely acknowledged. The Will Message and the Will Delay Interval
- If the Session is currently not connected, the time at which the Session will end and Session State will be discarded.

Retained messages do not form part of the Session State in the Server, they are not deleted as a result of a Session ending.

#### 4.1.1 Storing Session State

The Client and Server MUST NOT discard the Session State while the Network Connection is open [MQTT-4.1.0-1]. The Server MUST discard the Session State when the Network Connection is closed and the Session Expiry Interval has passed [MQTT-4.1.0-2].

##### Non-normative comment

The storage capabilities of Client and Server implementations will of course have limits in terms of capacity and may be subject to administrative policies. Stored Session State can be discarded as a result of an administrator action, including an automated response to defined conditions. This has the effect of terminating the Session. These actions might be prompted by resource constraints or for other operational reasons. It is possible that hardware or software failures may result in loss or corruption of Session State stored by the Client or Server. It is prudent to evaluate the storage capabilities of the Client and Server to ensure that they are sufficient.

### 4.1.2 Session State non-normative examples

For example, an electricity meter reading solution might use QoS 1 messages to protect the readings against loss over the network. The solution developer might have determined that the power supply is sufficiently reliable that, in this case, the data in the Client and Server can be stored in volatile memory without too much risk of its loss.

Conversely a parking meter payment application provider might decide that the payment messages should never be lost due to a network or Client failure. Thus, they require that all data be written to non-volatile memory before it is transmitted across the network.

## 4.2 Network Connections

The MQTT protocol requires an underlying transport that provides an ordered, lossless, stream of bytes from the Client to Server and Server to Client. This specification does not require the support of any specific transport protocol. A Client or Server MAY support any of the transport protocols listed here, or any other transport protocol that meets the requirements of this [section](#).

A Client or Server MUST support the use of one or more underlying transport protocols that provide an ordered, lossless, stream of bytes from the Client to Server and Server to Client [\[MQTT-4.2-1\]](#).

#### Non-normative comment

TCP/IP as defined in [\[RFC0793\]](#) can be used for MQTT v5.0. The following transport protocols are also suitable:

- TLS [\[RFC5246\]](#)
- WebSocket [\[RFC6455\]](#)

#### Non-normative comment

TCP ports 8883 and 1883 are registered with IANA for MQTT TLS and non-TLS communication respectively.

#### Non-normative comment

Connectionless network transports such as User Datagram Protocol (UDP) are not suitable on their own because they might lose or reorder data.

## 4.3 Quality of Service levels and protocol flows

MQTT delivers Application Messages according to the Quality of Service (QoS) levels defined in the following sections. The delivery protocol is symmetric, in the description below the Client and Server can each take the role of either sender or receiver. The delivery protocol is concerned solely with the delivery of an application message from a single sender to a single receiver. When the Server is delivering an Application Message to more than one Client, each Client is treated independently. The QoS level used to deliver an Application Message outbound to the Client could differ from that of the inbound Application Message.

### 4.3.1 QoS 0: At most once delivery

The message is delivered according to the capabilities of the underlying network. No response is sent by the receiver and no retry is performed by the sender. The message arrives at the receiver either once or not at all.

In the QoS 0 delivery protocol, the sender

- **MUST send a PUBLISH packet with QoS 0 and DUP flag set to 0 [MQTT-4.3.1-1].**

In the QoS 0 delivery protocol, the receiver

- Accepts ownership of the message when it receives the PUBLISH packet.

Figure 4.1 – QoS 0 protocol flow diagram, non-normative example

Sender Action	Control Packet	Receiver Action
PUBLISH QoS 0, DUP=0		
	----->	
		Deliver Application Message to appropriate onward recipient(s)

### 4.3.2 QoS 1: At least once delivery

This Quality of Service level ensures that the message arrives at the receiver at least once. A QoS 1 PUBLISH packet has a Packet Identifier in its Variable Header and is acknowledged by a PUBACK packet. [Section 2.2.1](#) provides more information about Packet Identifiers.

In the QoS 1 delivery protocol, the sender

- **MUST assign an unused Packet Identifier each time it has a new Application Message to publish [MQTT-4.3.2-1].**
- **MUST send a PUBLISH packet containing this Packet Identifier with QoS 1 and DUP flag set to 0 [MQTT-4.3.2-2].**
- **MUST treat the PUBLISH packet as “unacknowledged” until it has received the corresponding PUBACK packet from the receiver.** Refer to [section 4.4](#) for a discussion of unacknowledged messages **[MQTT-4.3.2-3].**

The Packet Identifier becomes available for reuse once the sender has received the PUBACK packet.

Note that a sender is permitted to send further PUBLISH packets with different Packet Identifiers while it is waiting to receive acknowledgements.

In the QoS 1 delivery protocol, the receiver

- **MUST respond with a PUBACK packet containing the Packet Identifier from the incoming PUBLISH packet, having accepted ownership of the Application Message [MQTT-4.3.2-4].**

- After it has sent a PUBACK packet the receiver MUST treat any incoming PUBLISH packet that contains the same Packet Identifier as being a new Application Message, irrespective of the setting of its DUP flag [MQTT-4.3.2-5].

Figure 4.2 – QoS 1 protocol flow diagram, non-normative example

Sender Action	MQTT Control Packet	Receiver action
Store message		
Send PUBLISH QoS 1, DUP=0, <Packet Identifier>	----->	
		Initiate onward delivery of the Application Message <sup>1</sup>
	<-----	Send PUBACK <Packet Identifier>
Discard message		

<sup>1</sup> The receiver does not need to complete delivery of the Application Message before sending the PUBACK. When its original sender receives the PUBACK packet, ownership of the Application Message is transferred to the receiver.

### 4.3.3 QoS 2: Exactly once delivery

This is the highest Quality of Service level, for use when neither loss nor duplication of messages are acceptable. There is an increased overhead associated with QoS 2.

A QoS 2 message has a Packet Identifier in its Variable Header. Section 2.2.1 provides more information about Packet Identifiers. The receiver of a QoS 2 PUBLISH packet acknowledges receipt with a two-step acknowledgement process.

In the QoS 2 delivery protocol, the sender:

- MUST assign an unused Packet Identifier when it has a new Application Message to publish [MQTT-4.3.3-1].
- MUST send a PUBLISH packet containing this Packet Identifier with QoS 2 and DUP flag set to 0 [MQTT-4.3.3-2].
- MUST treat the PUBLISH packet as “unacknowledged” until it has received the corresponding PUBREC packet from the receiver [MQTT-4.3.3-3]. Refer to section 4.4 for a discussion of unacknowledged messages.
- MUST send a PUBREL packet when it receives a PUBREC packet from the receiver with a Reason Code value less than 0x80. This PUBREL packet MUST contain the same Packet Identifier as the original PUBLISH packet [MQTT-4.3.3-4].
- MUST treat the PUBREL packet as “unacknowledged” until it has received the corresponding PUBCOMP packet from the receiver [MQTT-4.3.3-5].
- MUST NOT re-send the PUBLISH once it has sent the corresponding PUBREL packet [MQTT-4.3.3-6].
- MUST NOT apply Message expiry if a PUBLISH packet has been sent [MQTT-4.3.3-7].

The Packet Identifier becomes available for reuse once the sender has received the PUBCOMP packet or a PUBREC with a Reason Code of 0x80 or greater.

Note that a sender is permitted to send further PUBLISH packets with different Packet Identifiers while it is waiting to receive acknowledgements, subject to flow control as described in [section 4.9](#).

In the QoS 2 delivery protocol, the receiver:

- MUST respond with a PUBREC containing the Packet Identifier from the incoming PUBLISH packet, having accepted ownership of the Application Message [\[MQTT-4.3.3-8\]](#).
- If it has sent a PUBREC with a Reason Code of 0x80 or greater, the receiver MUST treat any subsequent PUBLISH packet that contains that Packet Identifier as being a new Application Message [\[MQTT-4.3.3-9\]](#).
- Until it has received the corresponding PUBREL packet, the receiver MUST acknowledge any subsequent PUBLISH packet with the same Packet Identifier by sending a PUBREC. It MUST NOT cause duplicate messages to be delivered to any onward recipients in this case [\[MQTT-4.3.3-10\]](#).
- MUST respond to a PUBREL packet by sending a PUBCOMP packet containing the same Packet Identifier as the PUBREL [\[MQTT-4.3.3-11\]](#).
- After it has sent a PUBCOMP, the receiver MUST treat any subsequent PUBLISH packet that contains that Packet Identifier as being a new Application Message [\[MQTT-4.3.3-12\]](#).
- MUST continue the QoS 2 acknowledgement sequence even if it has applied message expiry [\[MQTT-4.3.3-13\]](#).

## 4.4 Message delivery retry

When a Client reconnects with Clean Start set to 0 and a session is present, both the Client and Server MUST resend any unacknowledged PUBLISH packets (where QoS > 0) and PUBREL packets using their original Packet Identifiers. This is the only circumstance where a Client or Server is REQUIRED to resend messages. Clients and Servers MUST NOT resend messages at any other time [\[MQTT-4.4.0-1\]](#).

If PUBACK or PUBREC is received containing a Reason Code of 0x80 or greater the corresponding PUBLISH packet is treated as acknowledged, and MUST NOT be retransmitted [\[MQTT-4.4.0-2\]](#).

Figure 4.3 – QoS 2 protocol flow diagram, non-normative example

Sender Action	MQTT Control Packet	Receiver Action
Store message		
PUBLISH QoS 2, DUP=0 <Packet Identifier>		
	----->	
		Store <Packet Identifier> then Initiate onward delivery of the Application Message <sup>1</sup>
		PUBREC <Packet Identifier><Reason Code>



	<-----	
Discard message, Store PUBREC received <Packet Identifier>		
PUBREL <Packet Identifier>		
	----->	
		Discard <Packet Identifier>
		Send PUBCOMP <Packet Identifier>
	<-----	
Discard stored state		

<sup>1</sup> The receiver does not need to complete delivery of the Application Message before sending the PUBREC or PUBCOMP. When its original sender receives the PUBREC packet, ownership of the Application Message is transferred to the receiver. However, the receiver needs to perform all checks for conditions which might result in a forwarding failure (e.g. quota exceeded, authorization, etc.) before accepting ownership. The receiver indicates success or failure using the appropriate Reason Code in the PUBREC.

## 4.5 Message receipt

When a Server takes ownership of an incoming Application Message it MUST add it to the Session State for those Clients that have matching Subscriptions [MQTT-4.5.0-1]. Matching rules are defined in section 4.7.

Under normal circumstances Clients receive messages in response to Subscriptions they have created. A Client could also receive messages that do not match any of its explicit Subscriptions. This can happen if the Server automatically assigned a subscription to the Client. A Client could also receive messages while an UNSUBSCRIBE operation is in progress. The Client MUST acknowledge any Publish packet it receives according to the applicable QoS rules regardless of whether it elects to process the Application Message that it contains [MQTT-4.5.0-2].

## 4.6 Message ordering

The following these rules apply to the Client when implementing the protocol flows defined in section 4.3.

- When the Client re-sends any PUBLISH packets, it MUST re-send them in the order in which the original PUBLISH packets were sent (this applies to QoS 1 and QoS 2 messages) [MQTT-4.6.0-1]
- The Client MUST send PUBACK packets in the order in which the corresponding PUBLISH packets were received (QoS 1 messages) [MQTT-4.6.0-2]
- The Client MUST send PUBREC packets in the order in which the corresponding PUBLISH packets were received (QoS 2 messages) [MQTT-4.6.0-3]
- The Client MUST send PUBREL packets in the order in which the corresponding PUBREC packets were received (QoS 2 messages) [MQTT-4.6.0-4]

An Ordered Topic is a Topic where the Client can be certain that the Application Messages in that Topic from the same Client and at the same QoS are received in the order they were published. When a Server processes a message that has been published to an Ordered Topic, it MUST send PUBLISH packets to consumers (for the same Topic and QoS) in the order that they were received from any given Client [MQTT-4.6.0-5]. This is addition to the rules listed above.

By default, a Server MUST treat every Topic as an Ordered Topic when it is forwarding messages on Non-shared Subscriptions. [MQTT-4.6.0-6]. A Server MAY provide an administrative or other mechanism to allow one or more Topics to not be treated as an Ordered Topic.

#### Non-normative comment

The rules listed above ensure that when a stream of messages is published and subscribed to an Ordered Topic with QoS 1, the final copy of each message received by the subscribers will be in the order that they were published. If the message is re-sent the duplicate message can be received after one of the earlier messages is received. For example, a publisher might send messages in the order 1,2,3,4 but the subscriber might receive them in the order 1,2,3,2,3,4 if there is a network disconnection after message 3 has been sent.

If both Client and Server set Receive Maximum to 1, they make sure that no more than one message is "in-flight" at any one time. In this case no QoS 1 message will be received after any later one even on re-connection. For example a subscriber might receive them in the order 1,2,3,3,4 but not 1,2,3,2,3,4. Refer to section 4.9 Flow Control for details of how the Receive Maximum is used.

## 4.7 Topic Names and Topic Filters

### 4.7.1 Topic wildcards

The topic level separator is used to introduce structure into the Topic Name. If present, it divides the Topic Name into multiple "topic levels".

A subscription's Topic Filter can contain special wildcard characters, which allow a Client to subscribe to multiple topics at once.

The wildcard characters can be used in Topic Filters, but MUST NOT be used within a Topic Name [MQTT-4.7.0-1].

#### 4.7.1.1 Topic level separator

The forward slash ('/' U+002F) is used to separate each level within a topic tree and provide a hierarchical structure to the Topic Names. The use of the topic level separator is significant when either of the two wildcard characters is encountered in Topic Filters specified by subscribing Clients. Topic level separators can appear anywhere in a Topic Filter or Topic Name. Adjacent Topic level separators indicate a zero-length topic level.

#### 4.7.1.2 Multi-level wildcard

The number sign ('#' U+0023) is a wildcard character that matches any number of levels within a topic. The multi-level wildcard represents the parent and any number of child levels. The multi-level wildcard character MUST be specified either on its own or following a topic level separator. In either case it MUST be the last character specified in the Topic Filter [MQTT-4.7.1-1].

**Non-normative comment**

For example, if a Client subscribes to “sport/tennis/player1/#”, it would receive messages published using these Topic Names:

- “sport/tennis/player1”
- “sport/tennis/player1/ranking
- “sport/tennis/player1/score/wimbledon”

**Non-normative comment**

- “sport/#” also matches the singular “sport”, since # includes the parent level.
- “#” is valid and will receive every Application Message
- “sport/tennis/#” is valid
- “sport/tennis#” is not valid
- “sport/tennis##/ranking” is not valid

### 4.7.1.3 Single-level wildcard

The plus sign (‘+’ U+002B) is a wildcard character that matches only one topic level.

The single-level wildcard can be used at any level in the Topic Filter, including first and last levels. Where it is used, it MUST occupy an entire level of the filter [MQTT-4.7.1-2]. It can be used at more than one level in the Topic Filter and can be used in conjunction with the multi-level wildcard.

**Non-normative comment**

For example, “sport/tennis/+” matches “sport/tennis/player1” and “sport/tennis/player2”, but not “sport/tennis/player1/ranking”. Also, because the single-level wildcard matches only a single level, “sport/+” does not match “sport” but it does match “sport/”.

- “+” is valid
- “+/tennis/#” is valid
- “sport+” is not valid
- “sport/+/player1” is valid
- “/finance” matches “+/+” and “/+”, but not “+”

### 4.7.2 Topics beginning with \$

The Server MUST NOT match Topic Filters starting with a wildcard character (# or +) with Topic Names beginning with a \$ character [MQTT-4.7.2-1]. The Server SHOULD prevent Clients from using such Topic Names to exchange messages with other Clients. Server implementations MAY use Topic Names that start with a leading \$ character for other purposes.

**Non-normative comment**

- \$SYS/ has been widely adopted as a prefix to topics that contain Server-specific information or control APIs
- Applications cannot use a topic with a leading \$ character for their own purposes

**Non-normative comment**

- A subscription to “#” will not receive any messages published to a topic beginning with a \$
- A subscription to “+/monitor/Clients” will not receive any messages published to “\$SYS/monitor/Clients”
- A subscription to “\$SYS/#” will receive messages published to topics beginning with “\$SYS/”
- A subscription to “\$SYS/monitor/+” will receive messages published to “\$SYS/monitor/Clients”
- For a Client to receive messages from topics that begin with \$SYS/ and from topics that don't begin with a \$, it has to subscribe to both “#” and “\$SYS/#”

### 4.7.3 Topic semantic and usage

The following rules apply to Topic Names and Topic Filters:

- All Topic Names and Topic Filters MUST be at least one character long [MQTT-4.7.3-1]
- Topic Names and Topic Filters are case sensitive
- Topic Names and Topic Filters can include the space character
- A leading or trailing ‘/’ creates a distinct Topic Name or Topic Filter
- A Topic Name or Topic Filter consisting only of the ‘/’ character is valid
- Topic Names and Topic Filters MUST NOT include the null character (Unicode U+0000) [Unicode] [MQTT-4.7.3-2]
- Topic Names and Topic Filters are UTF-8 Encoded Strings; they MUST NOT encode to more than 65,535 bytes [MQTT-4.7.3-3]. Refer to section 1.5.4.

There is no limit to the number of levels in a Topic Name or Topic Filter, other than that imposed by the overall length of a UTF-8 Encoded String.

When it performs subscription matching the Server MUST NOT perform any normalization of Topic Names or Topic Filters, or any modification or substitution of unrecognized characters [MQTT-4.7.3-4]. Each non-wildcarded level in the Topic Filter has to match the corresponding level in the Topic Name character for character for the match to succeed.

**Non-normative comment**

The UTF-8 encoding rules mean that the comparison of Topic Filter and Topic Name could be performed either by comparing the encoded UTF-8 bytes, or by comparing decoded Unicode characters

**Non-normative comment**

- “ACCOUNTS” and “Accounts” are two different Topic Names
- “Accounts payable” is a valid Topic Name
- “/finance” is different from “finance”

An Application Message is sent to each Client Subscription whose Topic Filter matches the Topic Name attached to an Application Message. The topic resource MAY be either predefined in the Server by an administrator or it MAY be dynamically created by the Server when it receives the first subscription or an Application Message with that Topic Name. The Server MAY also use a security component to authorize particular actions on the topic resource for a given Client.

3043

## 3044 4.8 Subscriptions

3045 MQTT provides two kinds of Subscription, Shared and Non-shared.

3046

### 3047 **Non-normative comment**

3048 In earlier versions of MQTT all Subscriptions are Non-shared.

3049

### 3050 4.8.1 Non-shared Subscriptions

3051 A Non-shared Subscription is associated only with the MQTT Session that created it. Each Subscription  
3052 includes a Topic Filter, indicating the topic(s) for which messages are to be delivered on that Session,  
3053 and Subscription Options. The Server is responsible for collecting messages that match the filter and  
3054 transmitting them on the Session's MQTT connection if and when that connection is active.

3055

3056 A Session cannot have more than one Non-shared Subscription with the same Topic Filter, so the Topic  
3057 Filter can be used as a key to identify the subscription within that Session.

3058

3059 If there are multiple Clients, each with its own Non-shared Subscription to the same Topic, each Client  
3060 gets its own copy of the Application Messages that are published on that Topic. This means that the  
3061 Non-shared Subscriptions cannot be used to load-balance Application Messages across multiple  
3062 consuming Clients as in such cases every message is delivered to every subscribing Client.

3063

### 3064 4.8.2 Shared Subscriptions

3065 A Shared Subscription can be associated with multiple subscribing MQTT Sessions. Like a Non-shared  
3066 Subscription, it has a Topic Filter and Subscription Options; however, a publication that matches its Topic  
3067 Filter is only sent to one of its subscribing Sessions. Shared Subscriptions are useful where several  
3068 consuming Clients share the processing of the publications in parallel.

3069

3070 A Shared Subscription is identified using a special style of Topic Filter. The format of this filter is:

3071

3072 `$share/{ShareName}/{filter}`

- 3073 • `$share` is a literal string that marks the Topic Filter as being a Shared Subscription Topic Filter.
- 3074 • `{ShareName}` is a character string that does not include `/`, `+` or `#`
- 3075 • `{filter}` The remainder of the string has the same syntax and semantics as a Topic Filter in a non-  
3076 shared subscription. Refer to [section 4.7](#).

3077

3078 A Shared Subscription's Topic Filter MUST start with `$share/` and MUST contain a ShareName that is at  
3079 least one character long [\[MQTT-4.8.2-1\]](#). The ShareName MUST NOT contain the characters `/`, `+` or  
3080 `#`, but MUST be followed by a `/` character. This `/` character MUST be followed by a Topic Filter  
3081 [\[MQTT-4.8.2-2\]](#) as described in [section 4.7](#).

3082

### 3083 **Non-normative comment**

3084 Shared Subscriptions are defined at the scope of the MQTT Server, rather than of a Session. A  
3085 ShareName is included in the Shared Subscription's Topic Filter so that there can be more than  
3086 one Shared Subscription on a Server that has the same `{filter}` component. Typically, applications  
3087 use the ShareName to represent the group of subscribing Sessions that are sharing the

subscription.

Examples:

- Shared subscriptions "\$share/consumer1/sport/tennis/+" and "\$share/consumer2/sport/tennis/+" are distinct shared subscriptions and so can be associated with different groups of Sessions. Both of them match the same topics as a non-shared subscription to sport/tennis/+ .

If a message were to be published that matches sport/tennis/+ then a copy would be sent to exactly one of the Sessions subscribed to \$share/consumer1/sport/tennis/+ , a separate copy of the message would be sent to exactly one of the Sessions subscribed to \$share/consumer2/sport/tennis/+ and further copies would be sent to any Clients with non-shared subscriptions to sport/tennis/+

- Shared subscription "\$share/consumer1//finance" matches the same topics as a non-shared subscription to /finance.

Note that "\$share/consumer1//finance" and "\$share/consumer1/sport/tennis/+" are distinct shared subscriptions, even though they have the same ShareName. While they might be related in some way, no specific relationship between them is implied by them having the same ShareName.

A Shared Subscription is created by using a Shared Subscription Topic Filter in a SUBSCRIBE request. So long as only one Session subscribes to a particular Shared Subscription, the shared subscription behaves like a non-shared subscription, except that:

- The \$share and {ShareName} portions of the Topic Filter are not taken into account when matching against publications.
- No Retained Messages are sent to the Session when it first subscribes. It will be sent other matching messages as they are published.

Once a Shared Subscription exists, it is possible for other Sessions to subscribe with the same Shared Subscription Topic Filter. The new Session is associated with the Shared Subscription as an additional subscriber. Retained messages are not sent to this new subscriber. Each subsequent Application Message that matches the Shared Subscription is now sent to one and only one of the Sessions that are subscribed to the Shared Subscription.

A Session can explicitly detach itself from a Shared Subscription by sending an UNSUBSCRIBE Packet that contains the full Shared Subscription Topic Filter. Sessions are also detached from the Shared Subscription when they terminate.

A Shared Subscription lasts for as long as it is associated with at least one Session (i.e. a Session that has issued a successful SUBSCRIBE request to its Topic Filter and that has not completed a corresponding UNSUBSCRIBE). A Shared Subscription survives when the Session that originally created it unsubscribes, unless there are no other Sessions left when this happens. A Shared Subscription ends, and any undelivered messages associated with it are deleted, when there are no longer any Sessions subscribed to it.

Notes on Shared Subscriptions

- If there's more than one Session subscribed to the Shared Subscription, the Server implementation is free to choose, on a message by message basis, which Session to use and what criteria it uses to

make this selection.

- Different subscribing Clients are permitted to ask for different Requested QoS levels in their SUBSCRIBE packets. The Server decides which Maximum QoS to grant to each Client, and it is permitted to grant different Maximum QoS levels to different subscribers. When sending an Application Message to a Client, the Server MUST respect the granted QoS for the Client's subscription [MQTT-4.8.2-3], in the same that it does when sending a message to a -Subscriber.
- If the Server is in the process of sending a QoS 2 message to its chosen subscribing Client and the connection to the Client breaks before delivery is complete, the Server MUST complete the delivery of the message to that Client when it reconnects [MQTT-4.8.2-4] as described in section 4.3.3. If the Client's Session terminates before the Client reconnects, the Server MUST NOT send the Application Message to any other subscribed Client [MQTT-4.8.2-5].
- If the Server is in the process of sending a QoS 1 message to its chosen subscribing Client and the connection to that Client breaks before the Server has received an acknowledgement from the Client, the Server MAY wait for the Client to reconnect and retransmit the message to that Client. If the Client's Session terminates before the Client reconnects, the Server SHOULD send the Application Message to another Client that is subscribed to the same Shared Subscription. It MAY attempt to send the message to another Client as soon as it loses its connection to the first Client.
- If a Client responds with a PUBACK or PUBREC containing a Reason Code of 0x80 or greater to a PUBLISH packet from the Server, the Server MUST discard the Application Message and not attempt to send it to any other Subscriber [MQTT-4.8.2-6].
- A Client is permitted to submit a second SUBSCRIBE request to a Shared Subscription on a Session that's already subscribed to that Shared Subscription. For example, it might do this to change the Requested QoS for its subscription or because it was uncertain that the previous subscribe completed before the previous connection was closed. This does not increase the number of times that the Session is associated with the Shared Subscription, so the Session will leave the Shared Subscription on its first UNSUBSCRIBE.
- Each Shared Subscription is independent from any other. It is possible to have two Shared Subscriptions with overlapping filters. In such cases a message that matches both Shared Subscriptions will be processed separately by both of them. If a Client has a Shared Subscription and a Non-shared Subscription and a message matches both of them, the Client will receive a copy of the message by virtue of it having the Non-shared Subscription. A second copy of the message will be delivered to one of the subscribers to the Shared Subscription, and this could result in a second copy being sent to this Client.

## 4.9 Flow Control

Clients and Servers control the number of unacknowledged PUBLISH packets they receive by using a Receive Maximum value as described in section 3.1.2.11.4 and section 3.2.2.3.2. The Receive Maximum establishes a send quota which is used to limit the number of PUBLISH QoS > 0 packets which can be sent without receiving an PUBACK (for QoS 1) or PUBCOMP (for QoS 2). The PUBACK and PUBCOMP replenish the quota in the manner described below.

The Client or Server MUST set its initial send quota to a non-zero value not exceeding the Receive Maximum [MQTT-4.9.0-1].

Each time the Client or Server sends a PUBLISH packet at QoS > 0, it decrements the send quota. If the send quota reaches zero, the Client or Server MUST NOT send any more PUBLISH packets with QoS > 0 [MQTT-4.9.0-2]. It MAY continue to send PUBLISH packets with QoS 0, or it MAY choose to suspend



3193 sending these as well. The Client and Server MUST continue to process and respond to all other MQTT  
3194 Control Packets even if the quota is zero [MQTT-4.9.0-3].

3195  
3196 The send quota is incremented by 1:

- 3197 • Each time a PUBACK or PUBCOMP packet is received, regardless of whether the PUBACK or  
3198 PUBCOMP carried an error code.
- 3199 • Each time a PUBREC packet is received with a Return Code of 0x80 or greater.

3200  
3201 The send quota is not incremented if it is already equal to the initial send quota. The attempt to increment  
3202 above the initial send quota might be caused by the re-transmission of a PUBREL packet after a new  
3203 Network Connection is established.

3204  
3205 Refer to [section 3.3.4](#) for a description of how Clients and Servers react if they are sent more PUBLISH  
3206 packets than the Receive Maximum allows.

3207  
3208 The send quota and Receive Maximum value are not preserved across Network Connections, and are re-  
3209 initialized with each new Network Connection as described above. They are not part of the session state.

## 3211 4.10 Request / Response

3212 Some applications or standards might wish to run a Request/Response interaction over MQTT. This  
3213 version of MQTT includes three properties that can be used for this purpose:

- 3214 • Response Topic, described in [section 3.3.2.3.5](#)
- 3215 • Correlation Data, described in [section 3.3.2.3.6](#)
- 3216 • Request Response Information, described in [section 3.1.2.11.7](#)
- 3217 • Response Information, described in [section 3.2.2.3.14](#)

3218 The following non-normative sections describe how these properties can be used.

3219  
3220 A Client sends a Request Message by publishing an Application Message which has a Response Topic  
3221 set as described in [section 3.3.2.3.5](#). The Request can include a Correlation Data property as described  
3222 in [section 3.3.2.3.6](#).

### 3224 4.10.1 Basic Request Response (non-normative)

3225 Request/Response interaction proceeds as follows:

- 3226 1. An MQTT Client (the Requester) publishes a Request Message to a topic. A Request Message  
3227 is an Application Message with a Response Topic.
- 3228 2. Another MQTT Client (the Responder) has subscribed to a Topic Filter which matches the Topic  
3229 Name used when the Request Message was published. As a result, it receives the Request  
3230 Message. There could be multiple Responders subscribed to this Topic Name or there could be  
3231 none.
- 3232 3. The Responder takes the appropriate action based on the Request Message, and then publishes  
3233 a Response Message to the Topic Name in the Response Topic property that was carried in the  
3234 Request Message.
- 3235 4. In typical usage the Requester has subscribed to the Response Topic and thereby receives the  
3236 Response Message. However, some other Client might be subscribed to the Response Topic in  
3237 which case the Response Message will also be received and processed by that Client. As with  
3238 the Request Message, the topic on which the Response Message is sent could be subscribed to  
3239 by multiple Clients, or by none.



3240  
3241 If the Request Message contains a Correlation Data property, the Responder copies this property into the  
3242 Response Message and this is used by the receiver of the Response Message to associate the  
3243 Response Message with the original request. The Response Message does not include a Response  
3244 Topic property.  
3245  
3246 The MQTT Server forwards the Response Topic and Correlation Data Property in the Request Message  
3247 and the Correlation Data in the Response Message. The Server treats the Request Message and the  
3248 Response Message like any other Application Message.  
3249  
3250 The Requester normally subscribes to the Response Topic before publishing a Request Message. If there  
3251 are no subscribers to the Response Topic when the Response Message is sent, the Response Message  
3252 will not be delivered to any Client.  
3253  
3254 The Request Message and Response Message can be of any QoS, and the Responder can be using a  
3255 Session with a non-zero Session Expiry Interval. It is common to send Request Messages at QoS 0 and  
3256 only when the Responder is expected to be connected. However, this is not necessary.  
3257  
3258 The Responder can use a Shared Subscription to allow for a pool of responding Clients. Note however  
3259 that when using Shared Subscriptions that the order of message delivery is not guaranteed between  
3260 multiple Clients.  
3261  
3262 It is the responsibility of the Requester to make sure it has the necessary authority to publish to the  
3263 request topic, and to subscribe to the Topic Name that it sets in the Response Topic property. It is the  
3264 responsibility of the Responder to make sure it has the authority to subscribe to the request topic and  
3265 publish to the Response Topic. While topic authorization is outside of this specification, it is  
3266 recommended that Servers implement such authorization.  
3267

#### 3268 **4.10.2 Determining a Response Topic value (non-normative)**

3269 Requesters can determine a Topic Name to use as their Response Topic in any manner they choose  
3270 including via local configuration. To avoid clashes between different Requesters, it is desirable that the  
3271 Response Topic used by a Requester Client be unique to that Client. As the Requester and Responder  
3272 commonly need to be authorized to these topics, it can be an authorization challenge to use a random  
3273 Topic Name.  
3274  
3275 To help with this problem, this specification defines a property in the CONNACK packet called Response  
3276 Information. The Server can use this property to guide the Client in its choice for the Response Topic to  
3277 use. This mechanism is optional for both the Client and the Server. At connect time, the Client requests  
3278 that the Server send a Response Information by setting the Request Response Information property in  
3279 the CONNECT packet. This causes the Server to insert a Response Information property (a UTF-8  
3280 Encoded String) sent in the CONNACK packet.  
3281  
3282 This specification does not define the contents of the Response Information but it could be used to pass a  
3283 globally unique portion of the topic tree which is reserved for that Client for at least the lifetime of its  
3284 Session. Using this mechanism allows this configuration to be done once in the Server rather than in  
3285 each Client.  
3286  
3287 Refer to [section 3.1.2.11.7](#) for the definition of the Response Information.

3288

## 3289 4.11 Server redirection

3290 A Server can request that the Client uses another Server by sending CONNACK or DISCONNECT with  
3291 Reason Codes 0x9C (Use another server), or 0x9D (Server moved) as described in [section 4.13](#). When  
3292 sending one of these Reason Codes, the Server MAY also include a Server Reference property to  
3293 indicate the location of the Server or Servers the Client SHOULD use.

3294

3295 The Reason Code 0x9C (Use another server) specifies that the Client SHOULD temporarily switch to  
3296 using another Server. The other Server is either already known to the Client, or is specified using a  
3297 Server Reference.

3298

3299 The Reason Code 0x9D (Server moved) specifies that the Client SHOULD permanently switch to using  
3300 another Server. The other Server is either already known to the Client, or is specified using a Server  
3301 Reference.

3302

3303 The Server Reference is a UTF-8 Encoded String. The value of this string is a space separated list of  
3304 references. The format of references is not specified here.

3305

### 3306 **Non-normative comment**

3307 It is recommended that each reference consists of a name optionally followed by a colon and a  
3308 port number. If the name contains a colon the name string can be enclosed within square  
3309 brackets (“[” and “]”). A name enclosed by square brackets cannot contain the right square  
3310 bracket (“]”) character. This is used to represent an IPv6 literal address which uses colon  
3311 separators. This is a simplified version of an URI authority as described in [\[RFC3986\]](#).

3312

### 3313 **Non-normative comment**

3314 The name within a Server Reference commonly represents a host name, DNS name [\[RFC1035\]](#),  
3315 SRV name [\[RFC2782\]](#), or literal IP address. The value following the colon separator is commonly  
3316 a port number in decimal. This is not needed where the port information comes from the name  
3317 resolution (such as with SRV) or is defaulted.

3318

### 3319 **Non-normative comment**

3320 If multiple references are given, the expectation is that that Client will choose one of them.

3321

### 3322 **Non-normative comment**

3323 Examples of the Server Reference are:

3324 `myserver.xyz.org`

3325 `myserver.xyz.org:8883`

3326 `10.10.151.22:8883 [fe80::9610:3eff:fe1c]:1883`

3327

3328 The Server is allowed to not ever send a Server Reference, and the Client is allowed to ignore a Server  
3329 Reference. This feature can be used to allow for load balancing, Server relocation, and Client  
3330 provisioning to a Server.

3331

## 4.12 Enhanced authentication

The MQTT CONNECT packet supports basic authentication of a Network Connection using the User Name and Password fields. While these fields are named for a simple password authentication, they can be used to carry other forms of authentication such as passing a token as the Password.

Enhanced authentication extends this basic authentication to include challenge / response style authentication. It might involve the exchange of AUTH packets between the Client and the Server after the CONNECT and before the CONNACK packets.

To begin an enhanced authentication, the Client includes an Authentication Method in the CONNECT packet. This specifies the authentication method to use. If the Server does not support the Authentication Method supplied by the Client, it MAY send a CONNACK with a Reason Code of 0x8C (Bad authentication method) or 0x87 (Not Authorized) as described in [section 4.13](#) and MUST close the Network Connection [MQTT-4.12.0-1].

The Authentication Method is an agreement between the Client and Server about the meaning of the data sent in the Authentication Data and any of the other fields in CONNECT, and the exchanges and processing needed by the Client and Server to complete the authentication.

### Non-normative comment

The Authentication Method is commonly a SASL mechanism, and using such a registered name aids interchange. However, the Authentication Method is not constrained to using registered SASL mechanisms.

If the Authentication Method selected by the Client specifies that the Client sends data first, the Client SHOULD include an Authentication Data property in the CONNECT packet. This property can be used to provide data as specified by the Authentication Method. The contents of the Authentication Data are defined by the authentication method.

If the Server requires additional information to complete the authentication, it can send an AUTH packet to the Client. This packet MUST contain a Reason Code of 0x18 (Continue authentication) [MQTT-4.12.0-2]. If the authentication method requires the Server to send authentication data to the Client, it is sent in the Authentication Data.

The Client responds to an AUTH packet from the Server by sending a further AUTH packet. This packet MUST contain a Reason Code of 0x18 (Continue authentication) [MQTT-4.12.0-3]. If the authentication method requires the Client to send authentication data for the Server, it is sent in the Authentication Data.

The Client and Server exchange AUTH packets as needed until the Server accepts the authentication by sending a CONNACK with a Reason Code of 0. If the acceptance of the authentication requires data to be sent to the Client, it is sent in the Authentication Data.

The Client can close the connection at any point in this process. It MAY send a DISCONNECT packet before doing so. The Server can reject the authentication at any point in this process. It MAY send a CONNACK with a Reason Code of 0x80 or above as described in [section 4.13](#), and MUST close the Network Connection [MQTT-4.12.0-4].

If the initial CONNECT packet included an Authentication Method property then all AUTH packets, and any successful CONNACK packet MUST include an Authentication Method Property with the same value as in the CONNECT packet [MQTT-4.12.0-5].

The implementation of enhanced authentication is OPTIONAL for both Clients and Servers. If the Client does not include an Authentication Method in the CONNECT, the Server MUST NOT send an AUTH packet, and it MUST NOT send an Authentication Method in the CONNACK packet [MQTT-4.12.0-6]. If the Client does not include an Authentication Method in the CONNECT, the Client MUST NOT send an AUTH packet to the Server [MQTT-4.12.0-7].

If the Client does not include an Authentication Method in the CONNECT packet, the Server SHOULD authenticate using some or all of the information in the CONNECT packet, TLS session, and Network Connection.

#### Non-normative example showing a SCRAM challenge

- Client to Server: CONNECT Authentication Method="SCRAM-SHA-1" Authentication Data=client-first-data
- Server to Client: AUTH rc=0x18 Authentication Method="SCRAM-SHA-1" Authentication Data=server-first-data
- Client to Server AUTH rc=0x18 Authentication Method="SCRAM-SHA-1" Authentication Data=client-final-data
- Server to Client CONNACK rc=0 Authentication Method="SCRAM-SHA-1" Authentication Data=server-final-data

#### Non-normative example showing a Kerberos challenge

- Client to Server CONNECT Authentication Method="GS2-KRB5"
- Server to Client AUTH rc=0x18 Authentication Method="GS2-KRB5"
- Client to Server AUTH rc=0x18 Authentication Method="GS2-KRB5" Authentication Data=initial context token
- Server to Client AUTH rc=0x18 Authentication Method="GS2-KRB5" Authentication Data=reply context token
- Client to Server AUTH rc=0x18 Authentication Method="GS2-KRB5"
- Server to Client CONNACK rc=0 Authentication Method="GS2-KRB5" Authentication Data=outcome of authentication

### 4.12.1 Re-authentication

If the Client supplied an Authentication Method in the CONNECT packet it can initiate a re-authentication at any time after receiving a CONNACK. It does this by sending an AUTH packet with a Reason Code of 0x19 (Re-authentication). The Client MUST set the Authentication Method to the same value as the Authentication Method originally used to authenticate the Network Connection [MQTT-4.12.1-1]. If the authentication method requires Client data first, this AUTH packet contains the first piece of authentication data as the Authentication Data.

The Server responds to this re-authentication request by sending an AUTH packet to the Client with a Reason Code of 0x00 (Success) to indicate that the re-authentication is complete, or a Reason Code of 0x18 (Continue authentication) to indicate that more authentication data is needed. The Client can respond with additional authentication data by sending an AUTH packet with a Reason Code of 0x18 (Continue authentication). This flow continues as with the original authentication until the re-authentication is complete or the re-authentication fails.

If the re-authentication fails, the Client or Server SHOULD send DISCONNECT with an appropriate Reason Code as described in [section 4.13](#), and MUST close the Network Connection [MQTT-4.12.1-2].

During this re-authentication sequence, the flow of other packets between the Client and Server can continue using the previous authentication.

#### Non-normative comment

The Server might limit the scope of the changes the Client can attempt in a re-authentication by rejecting the re-authentication. For instance, if the Server does not allow the User Name to be changed it can fail any re-authentication attempt which changes the User Name.

## 4.13 Handling errors

### 4.13.1 Malformed Packet and Protocol Errors

Definitions of Malformed Packet and Protocol Errors are contained in [section 1.2](#) Terminology, some but not all, of these error cases are noted throughout the specification. The rigor with which a Client or Server checks an MQTT Control Packet it has received will be a compromise between:

- The size of the Client or Server implementation.
- The capabilities that the implementation supports.
- The degree to which the receiver trusts the sender to send correct MQTT Control Packets.
- The degree to which the receiver trusts the network to deliver MQTT Control Packets correctly.
- The consequences of continuing to process a packet that is incorrect.

If the sender is compliant with this specification it will not send Malformed Packets or cause Protocol Errors. However, if a Client sends MQTT Control Packets before it receives CONNACK, it might cause a Protocol Error because it made an incorrect assumption about the Server capabilities. Refer to [section 3.1.4](#) CONNECT Actions.

The Reason Codes used for Malformed Packet and Protocol Errors are:

- 0x81 Malformed Packet
- 0x82 Protocol Error
- 0x93 Receive Maximum exceeded
- 0x95 Packet too large
- 0x9A Retain not supported
- 0x9B QoS not supported
- 0x9E Shared Subscriptions not supported
- 0xA1 Subscription Identifiers not supported
- 0xA2 Wildcard Subscriptions not supported

When a Client detects a Malformed Packet or Protocol Error, and a Reason Code is given in the specification, it SHOULD close the Network Connection. In the case of an error in a AUTH packet it MAY send a DISCONNECT packet containing the reason code, before closing the Network Connection. In the case of an error in any other packet it SHOULD send a DISCONNECT packet containing the reason code before closing the Network Connection. Use Reason Code 0x81 (Malformed Packet) or 0x82 (Protocol Error) unless a more specific Reason Code has been defined in [section 3.14.2.1 Disconnect Reason Code](#).

3475 When a Server detects a Malformed Packet or Protocol Error, and a Reason Code is given in the  
3476 specification, it MUST close the Network Connection [MQTT-4.13.1-1]. In the case of an error in a  
3477 CONNECT packet it MAY send a CONNACK packet containing the Reason Code, before closing the  
3478 Network Connection. In the case of an error in any other packet it SHOULD send a DISCONNECT packet  
3479 containing the Reason Code before closing the Network Connection. Use Reason Code 0x81 (Malformed  
3480 Packet) or 0x82 (Protocol Error) unless a more specific Reason Code has been defined in [section 3.2.2.2](#)  
3481 - Connect Reason Code or in section 3.14.2.1 – Disconnect Reason Code. There are no consequences  
3482 for other Sessions.

3483

3484 If either the Server or Client omits to check some feature of an MQTT Control Packet, it might fail to  
3485 detect an error, consequently it might allow data to be damaged.

3486

#### 3487 4.13.2 Other errors

3488 Errors other than Malformed Packet and Protocol Errors cannot be anticipated by the sender because the  
3489 receiver might have constraints which it has not communicated to the sender. A receiving Client or Server  
3490 might encounter a transient error, such as a shortage of memory, that prevents successful processing of  
3491 an individual MQTT Control Packet.

3492

3493 Acknowledgment packets PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK with a  
3494 Reason Code of 0x80 or greater indicate that the received packet, identified by a Packet Identifier, was in  
3495 error. There are no consequences for other Sessions or other Packets flowing on the same Session.

3496

3497 The CONNACK and DISCONNECT packets allow a Reason Code of 0x80 or greater to indicate that the  
3498 Network Connection will be closed. If a Reason Code of 0x80 or greater is specified, then the Network  
3499 Connection MUST be closed whether or not the CONNACK or DISCONNECT is sent [MQTT-4.13.2-1].  
3500 Sending of one of these Reason Codes does not have consequence for any other Session.

3501

3502 If the Control Packet contains multiple errors the receiver of the Packet can validate the Packet in any  
3503 order and take the appropriate action for any of the errors found.

3504

3505 Refer to [section 5.4.9](#) for information about handling Disallowed Unicode code points.



---

## 5 Security (non-normative)

### 5.1 Introduction

MQTT is a transport protocol specification for message transmission, allowing implementers a choice of network, privacy, authentication and authorization technologies. Since the exact security technologies chosen will be context specific, it is the implementer's responsibility to include the appropriate features as part of their design.

MQTT Implementations will likely need to keep pace with an evolving security landscape.

This Chapter provides general implementation guidance so as not to restrict choices available and is therefore non-normative. This should not detract from its importance.

It is strongly recommended that Server implementations that offer TLS [\[RFC5246\]](#) should use TCP port 8883 (IANA service name: secure-mqtt).

There are a number of threats that solution providers should consider. For example:

- Devices could be compromised
- Data at rest in Clients and Servers might be accessible
- Protocol behaviors could have side effects (e.g. "timing attacks")
- Denial of Service (DoS) attacks
- Communications could be intercepted, altered, re-routed or disclosed
- Injection of spoofed MQTT Control Packets

MQTT solutions are often deployed in hostile communication environments. In such cases, implementations will often need to provide mechanisms for:

- Authentication of users and devices
- Authorization of access to Server resources
- Integrity of MQTT Control Packets and application data contained therein
- Privacy of MQTT Control Packets and application data contained therein

In addition to technical security issues there could also be geographic (e.g. U.S.-EU Privacy Shield Framework [\[USEUPRIVSH\]](#)), industry specific (e.g. PCI DSS [\[PCIDSS\]](#)) and regulatory considerations (e.g. Sarbanes-Oxley [\[SARBANES\]](#)).

### 5.2 MQTT solutions: security and certification

An implementation might want to provide conformance with specific industry security standards such as NIST Cyber Security Framework [\[NISTCSF\]](#), PCI-DSS [\[PCIDSS\]](#), FIPS-140-2 [\[FIPS1402\]](#) and NSA Suite B [\[NSAB\]](#).

Guidance on using MQTT within the NIST Cyber Security Framework [\[NISTCSF\]](#) can be found in the MQTT supplemental publication, MQTT and the NIST Framework for Improving Critical Infrastructure Cybersecurity [\[MQTTNIST\]](#). The use of industry proven, independently verified and certified technologies will help meet compliance requirements.

3549

## 3550 **5.3 Lightweight cryptography and constrained devices**

3551 Advanced Encryption Standard [\[AES\]](#) is the most widely adopted encryption algorithm. There is hardware  
3552 support for AES in many processors, but not commonly for embedded processors. The encryption  
3553 algorithm ChaCha20 [\[CHACHA20\]](#) encrypts and decrypts much faster in software, but is not as widely  
3554 available as AES.

3555

3556 ISO 29192 [\[ISO29192\]](#) makes recommendations for cryptographic primitives specifically tuned to perform  
3557 on constrained “low end” devices.

3558

## 3559 **5.4 Implementation notes**

3560 There are many security concerns to consider when implementing or using MQTT. The following section  
3561 should not be considered a “check list”.

3562

3563 An implementation might want to achieve some, or all, of the following:

3564

### 3565 **5.4.1 Authentication of Clients by the Server**

3566 The CONNECT packet contains User Name and Password fields. Implementations can choose how to  
3567 make use of the content of these fields. They may provide their own authentication mechanism, use an  
3568 external authentication system such as LDAP [\[RFC4511\]](#) or OAuth [\[RFC6749\]](#) tokens, or leverage  
3569 operating system authentication mechanisms.

3570

3571 MQTT v5.0 provides an enhanced authentication mechanism as described in [section 4.12](#). Using this  
3572 requires support for it in both the Client and Server.

3573

3574 Implementations passing authentication data in clear text, obfuscating such data elements or requiring no  
3575 authentication data should be aware this can give rise to Man-in-the-Middle and replay attacks. [Section](#)  
3576 [5.4.5](#) introduces approaches to ensure data privacy.

3577

3578 A Virtual Private Network (VPN) between the Clients and Servers can provide confidence that data is only  
3579 being received from authorized Clients.

3580

3581 Where TLS [\[RFC5246\]](#) is used, TLS Certificates sent from the Client can be used by the Server to  
3582 authenticate the Client.

3583

3584 An implementation might allow for authentication where the credentials are sent in an Application  
3585 Message from the Client to the Server.

3586

### 3587 **5.4.2 Authorization of Clients by the Server**

3588 If a Client has been successfully authenticated, a Server implementation should check that it is authorized  
3589 before accepting its connection.

3590



Authorization may be based on information provided by the Client such as User Name, the hostname/IP address of the Client, or the outcome of authentication mechanisms.

In particular, the implementation should check that the Client is authorized to use the Client Identifier as this gives access to the MQTT Session State (described in [section 4.1](#)). This authorization check is to protect against the case where one Client, accidentally or maliciously, provides a Client Identifier that is already being used by some other Client.

An implementation should provide access controls that take place after CONNECT to restrict the Clients ability to publish to particular Topics or to subscribe using particular Topic Filters. An implementation should consider limiting access to Topic Filters that have broad scope, such as the # Topic Filter.

### 5.4.3 Authentication of the Server by the Client

The MQTT protocol is not trust symmetrical. When using basic authentication, there is no mechanism for the Client to authenticate the Server. Some forms of extended authentication do allow for mutual authentication.

Where TLS [\[RFC5246\]](#) is used, TLS Certificates sent from the Server can be used by the Client to authenticate the Server. Implementations providing MQTT service for multiple hostnames from a single IP address should be aware of the Server Name Indication extension to TLS defined in section 3 of [\[RFC6066\]](#). This allows a Client to tell the Server the hostname of the Server it is trying to connect to.

An implementation might allow for authentication where the credentials are sent in an Application Message from the Server to the Client. MQTT v5.0 provides an enhanced authentication mechanism as described in [section 4.12.](#), which can be used to Authenticate the Server to the Client. Using this requires support for it in both the Client and Server.

A VPN between Clients and Servers can provide confidence that Clients are connecting to the intended Server.

### 5.4.4 Integrity of Application Messages and MQTT Control Packets

Applications can independently include hash values in their Application Messages. This can provide integrity of the contents of Publish packets across the network and at rest.

TLS [\[RFC5246\]](#) provides hash algorithms to verify the integrity of data sent over the network.

The use of VPNs to connect Clients and Servers can provide integrity of data across the section of the network covered by a VPN.

### 5.4.5 Privacy of Application Messages and MQTT Control Packets

TLS [\[RFC5246\]](#) can provide encryption of data sent over the network. There are valid TLS cipher suites that include a NULL encryption algorithm that does not encrypt data. To ensure privacy Clients and Servers should avoid these cipher suites.

An application might independently encrypt the contents of its Application Messages. This could provide privacy of the Application Message both over the network and at rest. This would not provide privacy for other Properties of the Application Message such as Topic Name.

Client and Server implementations can provide encrypted storage for data at rest such as Application Messages stored as part of a Session.

The use of VPNs to connect Clients and Servers can provide privacy of data across the section of the network covered by a VPN.

#### 5.4.6 Non-repudiation of message transmission

Application designers might need to consider appropriate strategies to achieve end to end non-repudiation.

#### 5.4.7 Detecting compromise of Clients and Servers

Client and Server implementations using TLS [\[RFC5246\]](#) should provide capabilities to ensure that any TLS certificates provided when initiating a TLS connection are associated with the hostname of the Client connecting or Server being connected to.

Client and Server implementations using TLS can choose to provide capabilities to check Certificate Revocation Lists (CRLs [\[RFC5280\]](#)) and Online Certificate Status Protocol (OCSP) [\[RFC6960\]](#) to prevent revoked certificates from being used.

Physical deployments might combine tamper-proof hardware with the transmission of specific data in Application Messages. For example, a meter might have an embedded GPS to ensure it is not used in an unauthorized location. [\[IEEE8021AR\]](#) is a standard for implementing mechanisms to authenticate a device's identity using a cryptographically bound identifier.

#### 5.4.8 Detecting abnormal behaviors

Server implementations might monitor Client behavior to detect potential security incidents. For example:

- Repeated connection attempts
- Repeated authentication attempts
- Abnormal termination of connections
- Topic scanning (attempts to send or subscribe to many topics)
- Sending undeliverable messages (no subscribers to the topics)
- Clients that connect but do not send data

Server implementations might close the Network Connection of Clients that breach its security rules.

Server implementations detecting unwelcome behavior might implement a dynamic block list based on identifiers such as IP address or Client Identifier.

3677 Deployments might use network-level controls (where available) to implement rate limiting or blocking  
3678 based on IP address or other information.

3679

## 3680 **5.4.9 Handling of Disallowed Unicode code points**

3681 **Section** 1.5.4 describes the Disallowed Unicode code points, which should not be included in a UTF-8  
3682 Encoded String. A Client or Server implementation can choose whether to validate that these code points  
3683 are not used in UTF-8 Encoded Strings such as the Topic Name or Properties.

3684

3685 If the Server does not validate the code points in a UTF-8 Encoded String but a subscribing Client does,  
3686 then a second Client might be able to cause the subscribing Client to close the Network Connection by  
3687 publishing on a Topic Name or using Properties that contain a Disallowed Unicode code point. This  
3688 section recommends some steps that can be taken to prevent this problem.

3689

3690 A similar problem can occur when the Client validates that the payload matches the Payload Format  
3691 Indicator and the Server does not. The considerations and remedies for this are similar to those for  
3692 handling Disallowed Unicode code points.

3693

### 3694 **5.4.9.1 Considerations for the use of Disallowed Unicode code points**

3695 An implementation would normally choose to validate UTF-8 Encoded strings, checking that the  
3696 Disallowed Unicode code points are not used. This avoids implementation difficulties such as the use of  
3697 libraries that are sensitive to these code points, it also protects applications from having to process them.

3698

3699 Validating that these code points are not used removes some security exposures. There are possible  
3700 security exploits which use control characters in log files to mask entries in the logs or confuse the tools  
3701 which process log files. The Unicode Noncharacters are commonly used as special markers and allowing  
3702 them into UTF-8 Encoded Strings could permit such exploits.

3703

### 3704 **5.4.9.2 Interactions between Publishers and Subscribers**

3705 The publisher of an Application Message normally expects that the Servers will forward the message to  
3706 subscribers, and that these subscribers are capable of processing the messages.

3707 These are some conditions under which a publishing Client can cause the subscribing Client to close the  
3708 Network Connection. Consider a situation where:

- 3709 • A Client publishes an Application Message using a Topic Name containing one of the Disallowed  
3710 Unicode code points.
- 3711 • The publishing Client library allows the Disallowed Unicode code point to be used in a Topic  
3712 Name rather than rejecting it.
- 3713 • The publishing Client is authorized to send the publication.
- 3714 • A subscribing Client is authorized to use a Topic Filter which matches the Topic Name. Note that  
3715 the Disallowed Unicode code point might occur in a part of the Topic Name matching a wildcard  
3716 character in the Topic Filter.
- 3717 • The Server forwards the message to the matching subscriber rather than disconnecting the  
3718 publisher.
- 3719 • In this case the subscribing Client might:
  - 3720 ○ Close the Network Connection because it does not allow the use of Disallowed Unicode  
3721 code points, possibly sending a DISCONNECT before doing so. For QoS 1 and QoS 2  
3722 messages this might cause the Server to send the message again, causing the Client to  
3723 close the Network Connection again.

- 3724                   ○ Reject the Application Message by sending a Reason Code greater than or equal to 0x80  
3725                   in a PUBACK (QoS 1) or PUBREC (QoS 2).  
3726                   ○ Accept the Application Message but fail to process it because it contains one of the  
3727                   Disallowed Unicode code points.  
3728                   ○ Successfully process the Application Message.

3729

3730   The potential for the Client to close the Network Connection might go unnoticed until a publisher uses one  
3731   of the Disallowed Unicode code points.

3732

### 3733   5.4.9.3 Remedies

3734   If there is a possibility that a Disallowed Unicode code point could be included in a Topic Name or other  
3735   Properties delivered to a Client, the solution owner can adopt one of the following suggestions:

- 3736           1) Change the Server implementation to one that rejects UTF-8 Encoded Strings containing a  
3737           Disallowed Unicode code point either by sending a Reason Code greater than or equal to 0x80 or  
3738           closing the Network Connection.  
3739           2) Change the Client library used by the subscribers to one that tolerates the use of Disallowed  
3740           Code points. The client can either process or discard messages with UTF-8 Encoded Strings that  
3741           contain Disallowed Unicode code points so long as it continues the protocol.

3742

### 3743   5.4.10 Other security considerations

3744   If Client or Server TLS certificates are lost or it is considered that they might be compromised they should  
3745   be revoked (utilizing CRLs [\[RFC5280\]](#) and/or OSCP [\[RFC6960\]](#)).

3746

3747   Client or Server authentication credentials, such as User Name and Password, that are lost or considered  
3748   compromised should be revoked and/or reissued.

3749

3750   In the case of long lasting connections:

- 3751           • Client and Server implementations using TLS [\[RFC5246\]](#) should allow for session renegotiation to  
3752           establish new cryptographic parameters (replace session keys, change cipher suites, change  
3753           authentication credentials).  
3754           • Servers may close the Network Connection of Clients and require them to re-authenticate with new  
3755           credentials.  
3756           • Servers may require their Client to reauthenticate periodically using the mechanism described in  
3757           [section 4.12.1](#).

3758

3759   Constrained devices and Clients on constrained networks can make use of TLS [\[RFC5246\]](#) session  
3760   resumption, in order to reduce the costs of reconnecting TLS [\[RFC5246\]](#) sessions.

3761

3762   Clients connected to a Server have a transitive trust relationship with other Clients connected to the same  
3763   Server and who have authority to publish data on the same topics.

3764

### 3765   5.4.11 Use of SOCKS

3766   Implementations of Clients should be aware that some environments will require the use of SOCKSv5  
3767   [\[RFC1928\]](#) proxies to make outbound Network Connections. Some MQTT implementations could make

3768 use of alternative secured tunnels (e.g. SSH) through the use of SOCKS. Where implementations choose  
3769 to use SOCKS, they should support both anonymous and User Name, Password authenticating SOCKS  
3770 proxies. In the latter case, implementations should be aware that SOCKS authentication might occur in  
3771 plain-text and so should avoid using the same credentials for connection to a MQTT Server.  
3772

## 3773 **5.4.12 Security profiles**

3774 Implementers and solution designers might wish to consider security as a set of profiles which can be  
3775 applied to the MQTT protocol. An example of a layered security hierarchy is presented below.  
3776

### 3777 **5.4.12.1 Clear communication profile**

3778 When using the clear communication profile, the MQTT protocol runs over an open network with no  
3779 additional secure communication mechanisms in place.  
3780

### 3781 **5.4.12.2 Secured network communication profile**

3782 When using the secured network communication profile, the MQTT protocol runs over a physical or virtual  
3783 network which has security controls e.g., VPNs or physically secure network.  
3784

### 3785 **5.4.12.3 Secured transport profile**

3786 When using the secured transport profile, the MQTT protocol runs over a physical or virtual network and  
3787 using TLS [\[RFC5246\]](#) which provides authentication, integrity and privacy.  
3788

3789 TLS [\[RFC5246\]](#) Client authentication can be used in addition to – or in place of – MQTT Client  
3790 authentication as provided by the User Name and Password fields.  
3791

### 3792 **5.4.12.4 Industry specific security profiles**

3793 It is anticipated that the MQTT protocol will be designed into industry specific application profiles, each  
3794 defining a threat model and the specific security mechanisms to be used to address these threats.  
3795 Recommendations for specific security mechanisms will often be taken from existing works including:  
3796

3797 [\[NISTCSF\]](#) NIST Cyber Security Framework  
3798 [\[NIST7628\]](#) NISTIR 7628 Guidelines for Smart Grid Cyber Security  
3799 [\[FIPS1402\]](#) Security Requirements for Cryptographic Modules (FIPS PUB 140-2)  
3800 [\[PCIDSS\]](#) PCI-DSS Payment Card Industry Data Security Standard  
3801 [\[NSAB\]](#) NSA Suite B Cryptography  
3802

## 6 Using WebSocket as a network transport

If MQTT is transported over a WebSocket [RFC6455] connection, the following conditions apply:

- MQTT Control Packets MUST be sent in WebSocket binary data frames. If any other type of data frame is received the recipient MUST close the Network Connection [MQTT-6.0.0-1].
- A single WebSocket data frame can contain multiple or partial MQTT Control Packets. The receiver MUST NOT assume that MQTT Control Packets are aligned on WebSocket frame boundaries [MQTT-6.0.0-2].
- The Client MUST include “mqtt” in the list of WebSocket Sub Protocols it offers [MQTT-6.0.0-3].
- The WebSocket Subprotocol name selected and returned by the Server MUST be “mqtt” [MQTT-6.0.0-4].
- The WebSocket URI used to connect the Client and Server has no impact on the MQTT protocol.

### 6.1 IANA considerations

This specification requests IANA to modify the registration of the WebSocket MQTT sub-protocol under the “WebSocket Subprotocol Name” registry with the following data:

Figure 6.6-1 - IANA WebSocket Identifier

Subprotocol Identifier	mqtt
Subprotocol Common Name	mqtt
Subprotocol Definition	<a href="http://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html">http://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html</a>

---

## 7 Conformance

The MQTT specification defines conformance for MQTT Client implementations and MQTT Server implementations. An MQTT implementation can conform as both an MQTT Client and an MQTT Server.

### 7.1 Conformance clauses

#### 7.1.1 MQTT Server conformance clause

Refer to [Server](#) in the Terminology section for a definition of Server.

An MQTT Server conforms to this specification only if it satisfies all the statements below:

1. The format of all MQTT Control Packets that the Server sends matches the format described in [Chapter 2](#) and [Chapter 3](#).
2. It follows the Topic matching rules described in [section 4.7](#) and the Subscription rules in [section 4.8](#).
3. It satisfies the MUST level requirements in the following chapters that are identified except for those that only apply to the Client:
  - [Chapter 1 - Introduction](#)
  - [Chapter 2 - MQTT Control Packet format](#)
  - [Chapter 3 - MQTT Control Packets](#)
  - [Chapter 4 - Operational behavior](#)
  - [Chapter 6 - Using WebSocket as a network transport](#)
4. It does not require the use of any extensions defined outside of the specification in order to interoperate with any other conformant implementation.

#### 7.1.2 MQTT Client conformance clause

Refer to [Client](#) in the Terminology section for a definition of Client.

An MQTT Client conforms to this specification only if it satisfies all the statements below:

1. The format of all MQTT Control Packets that the Client sends matches the format described in [Chapter 2](#) and [Chapter 3](#).
2. It satisfies the MUST level requirements in the following chapters that are identified except for those that only apply to the Server:
  - [Chapter 1 - Introduction](#)
  - [Chapter 2 - MQTT Control Packet format](#)
  - [Chapter 3 - MQTT Control Packets](#)
  - [Chapter 4 - Operational behavior](#)
  - [Chapter 6 - Using WebSocket as a network transport](#)
3. It does not require the use of any extensions defined outside of the specification in order to interoperate with any other conformant implementation.



---

## Appendix A. Acknowledgments

The TC owes special thanks to Dr. Andy Stanford-Clark and Arlen Nipper as the original inventors of the MQTT protocol and for their continued support with the standardization process.

The TC wishes to thank Brian Raymor (formerly of Microsoft) for his work as co-chairman of the MQTT TC during much of the development of the version 5.0 standard.

The following individuals were members of the OASIS Technical Committee during the creation of this standard and their contributions are gratefully acknowledged:

### Participants:

- Senthil Nathan Balasubramaniam (Infiswift)
- Dr. Andrew Banks, editor (IBM)
- Ken Borgendale, editor (IBM)
- Ed Briggs, editor (Microsoft)
- Raphael Cohn (Individual)
- Richard Coppen, chairman (IBM)
- William Cox (Individual)
- Ian Craggs , secretary (IBM)
- Konstantin Dotchkoff (Microsoft)
- Derek Fu (IBM)
- Rahul Gupta, editor (IBM)
- Stefan Hagen (Individual)
- David Horton (Solace Systems)
- Alex Kritikos (Software AG, Inc.)
- Jonathan Levell (IBM)
- Shawn McAllister (Solace Systems)
- William McLane (TIBCO Software Inc.)
- Peter Niblett (IBM)
- Dominik Obermaier (dc-square GmbH)
- Nicholas O'Leary (IBM)
- Brian Raymor (Microsoft)
- Andrew Schofield (IBM)
- Tobias Sommer (Cumulocity)
- Joe Speed (IBM)
- Dr Andy Stanford-Clark (IBM)
- Allan Stockdill-Mander (IBM)
- Stehan Vaillant (Cumulocity)

For a list of those who contributed to earlier versions of MQTT refer to Appendix A in the MQTT v3.1.1 specification **[MQTTV311]**.



## Appendix B. Mandatory normative statement (non-normative)

This Appendix is non-normative and is provided as a convenient summary of the numbered conformance statements found in the main body of this document. Refer to [Chapter 7](#) for a definitive list of conformance requirements.

Normative Statement Number	Normative Statement
[MQTT-1.5.4-1]	The character data in a UTF-8 Encoded String MUST be well-formed UTF-8 as defined by the Unicode specification [Unicode] and restated in RFC 3629 [RFC3629]. In particular, the character data MUST NOT include encodings of code points between U+D800 and U+DFFF.
[MQTT-1.5.4-2]	A UTF-8 Encoded String MUST NOT include an encoding of the null character U+0000.
[MQTT-1.5.4-3]	A UTF-8 encoded sequence 0xEF 0xBB 0xBF is always interpreted as U+FEFF ("ZERO WIDTH NO-BREAK SPACE") wherever it appears in a string and MUST NOT be skipped over or stripped off by a packet receiver.
[MQTT-1.5.5-1]	The encoded value MUST use the minimum number of bytes necessary to represent the value.
[MQTT-1.5.7-1]	Both strings MUST comply with the requirements for UTF-8 Encoded Strings.
[MQTT-2.1.3-1]	Where a flag bit is marked as "Reserved" it is reserved for future use and MUST be set to the value listed.
[MQTT-2.2.1-2]	A PUBLISH packet MUST NOT contain a Packet Identifier if its QoS value is set to 0.
[MQTT-2.2.1-3]	Each time a Client sends a new SUBSCRIBE, UNSUBSCRIBE, or PUBLISH (where QoS > 0) MQTT Control Packet it MUST assign it a non-zero Packet Identifier that is currently unused.
[MQTT-2.2.1-4]	Each time a Server sends a new PUBLISH (with QoS > 0) MQTT Control Packet it MUST assign it a non zero Packet Identifier that is currently unused.
[MQTT-2.2.1-5]	A PUBACK, PUBREC, PUBREL, or PUBCOMP packet MUST contain the same Packet Identifier as the PUBLISH packet that was originally sent.
[MQTT-2.2.1-6]	A SUBACK and UNSUBACK MUST contain the Packet Identifier that was used in the corresponding SUBSCRIBE and UNSUBSCRIBE packet respectively.
[MQTT-2.2.2-1]	If there are no properties, this MUST be indicated by including a Property Length of zero.
[MQTT-3.1.0-1]	After a Network Connection is established by a Client to a Server, the first packet sent from the Client to the Server MUST be a CONNECT packet.

[MQTT-3.1.0-2]	The Server MUST process a second CONNECT packet sent from a Client as a Protocol Error and close the Network Connection.
[MQTT-3.1.2-1]	The protocol name MUST be the UTF-8 String "MQTT". If the Server does not want to accept the CONNECT, and wishes to reveal that it is an MQTT Server it MAY send a CONNACK packet with Reason Code of 0x84 (Unsupported Protocol Version), and then it MUST close the Network Connection.
[MQTT-3.1.2-2]	If the Protocol Version is not 5 and the Server does not want to accept the CONNECT packet, the Server MAY send a CONNACK packet with Reason Code 0x84 (Unsupported Protocol Version) and then MUST close the Network Connection
[MQTT-3.1.2-3]	The Server MUST validate that the reserved flag in the CONNECT packet is set to 0.
[MQTT-3.1.2-4]	If a CONNECT packet is received with Clean Start is set to 1, the Client and Server MUST discard any existing Session and start a new Session.
[MQTT-3.1.2-5]	If a CONNECT packet is received with Clean Start set to 0 and there is a Session associated with the Client Identifier, the Server MUST resume communications with the Client based on state from the existing Session.
[MQTT-3.1.2-6]	If a CONNECT packet is received with Clean Start set to 0 and there is no Session associated with the Client Identifier, the Server MUST create a new Session.
[MQTT-3.1.2-7]	If the Will Flag is set to 1 this indicates that, a Will Message MUST be stored on the Server and associated with the Session.
[MQTT-3.1.2-8]	The Will Message MUST be published after the Network Connection is subsequently closed and either the Will Delay Interval has elapsed or the Session ends, unless the Will Message has been deleted by the Server on receipt of a DISCONNECT packet with Reason Code 0x00 (Normal disconnection) or a new Network Connection for the ClientID is opened before the Will Delay Interval has elapsed.
[MQTT-3.1.2-9]	If the Will Flag is set to 1, the Will QoS and Will Retain fields in the Connect Flags will be used by the Server, and the Will Properties, Will Topic and Will Message fields MUST be present in the Payload.
[MQTT-3.1.2-10]	The Will Message MUST be removed from the stored Session State in the Server once it has been published or the Server has received a DISCONNECT packet with a Reason Code of 0x00 (Normal disconnection) from the Client.
[MQTT-3.1.2-11]	If the Will Flag is set to 0, then the Will QoS MUST be set to 0 (0x00).
[MQTT-3.1.2-12]	If the Will Flag is set to 1, the value of Will QoS can be 0 (0x00), 1 (0x01), or 2 (0x02).
[MQTT-3.1.2-13]	If the Will Flag is set to 0, then Will Retain MUST be set to 0.
[MQTT-3.1.2-14]	If the Will Flag is set to 1 and Will Retain is set to 0, the Server MUST publish the Will Message as a non-retained message.
[MQTT-3.1.2-15]	If the Will Flag is set to 1 and Will Retain is set to 1, the Server MUST publish the Will Message as a retained message.
[MQTT-3.1.2-16]	If the User Name Flag is set to 0, a User Name MUST NOT be present in the Payload.

[MQTT-3.1.2-17]	If the User Name Flag is set to 1, a User Name MUST be present in the Payload.
[MQTT-3.1.2-18]	If the Password Flag is set to 0, a Password MUST NOT be present in the Payload.
[MQTT-3.1.2-19]	If the Password Flag is set to 1, a Password MUST be present in the Payload.
[MQTT-3.1.2-20]	If Keep Alive is non-zero and in the absence of sending any other MQTT Control Packets, the Client MUST send a PINGREQ packet.
[MQTT-3.1.2-21]	If the Server returns a Server Keep Alive on the CONNACK packet, the Client MUST use that value instead of the value it sent as the Keep Alive.
[MQTT-3.1.2-22]	If the Keep Alive value is non-zero and the Server does not receive an MQTT Control Packet from the Client within one and a half times the Keep Alive time period, it MUST close the Network Connection to the Client as if the network had failed.
[MQTT-3.1.2-23]	The Client and Server MUST store the Session State after the Network Connection is closed if the Session Expiry Interval is greater than 0.
[MQTT-3.1.2-24]	The Server MUST NOT send packets exceeding Maximum Packet Size to the Client.
[MQTT-3.1.2-25]	Where a Packet is too large to send, the Server MUST discard it without sending it and then behave as if it had completed sending that Application Message.
[MQTT-3.1.2-26]	The Server MUST NOT send a Topic Alias in a PUBLISH packet to the Client greater than Topic Alias Maximum.
[MQTT-3.1.2-27]	If Topic Alias Maximum is absent or zero, the Server MUST NOT send any Topic Aliases to the.
[MQTT-3.1.2-28]	A value of 0 indicates that the Server MUST NOT return Response Information.
[MQTT-3.1.2-29]	If the value of Request Problem Information is 0, the Server MAY return a Reason String or User Properties on a CONNACK or DISCONNECT packet, but MUST NOT send a Reason String or User Properties on any packet other than PUBLISH, CONNACK, or DISCONNECT.
[MQTT-3.1.2-30]	If a Client sets an Authentication Method in the CONNECT, the Client MUST NOT send any packets other than AUTH or DISCONNECT packets until it has received a CONNACK packet.
[MQTT-3.1.3-1]	The Payload of the CONNECT packet contains one or more length-prefixed fields, whose presence is determined by the flags in the Variable Header. These fields, if present, MUST appear in the order Client Identifier, Will Topic, Will Message, User Name, Password.
[MQTT-3.1.3-2]	The ClientID MUST be used by Clients and by Servers to identify state that they hold relating to this MQTT Session between the Client and the Server.
[MQTT-3.1.3-3]	The ClientID MUST be present and is the first field in the CONNECT packet Payload.
[MQTT-3.1.3-4]	The ClientID MUST be a UTF-8 Encoded String.

[MQTT-3.1.3-5]	The Server MUST allow ClientID's which are between 1 and 23 UTF-8 encoded bytes in length, and that contain only the characters "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ".
[MQTT-3.1.3-6]	A Server MAY allow a Client to supply a ClientID that has a length of zero bytes, however if it does so the Server MUST treat this as a special case and assign a unique ClientID to that Client.
[MQTT-3.1.3-7]	It MUST then process the CONNECT packet as if the Client had provided that unique ClientID, and MUST return the Assigned Client Identifier in the CONNACK packet.
[MQTT-3.1.3-8]	If the Server rejects the ClientID it MAY respond to the CONNECT packet with a CONNACK using Reason Code 0x85 (Client Identifier not valid) as described in section 4.13 Handling errors, and then it MUST close the Network Connection.
[MQTT-3.1.3-9]	If a new Network Connection to this Session is made before the Will Delay Interval has passed, the Server MUST NOT send the Will Message.
[MQTT-3.1.3-10]	The Server MUST maintain the order of User Properties when forwarding the Application Message.
[MQTT-3.1.3-11]	The Will Topic MUST be a UTF-8 Encoded String.
[MQTT-3.1.3-12]	If the User Name Flag is set to 1, the User Name is the next field in the Payload. The User Name MUST be a UTF-8 Encoded String.
[MQTT-3.1.4-1]	The Server MUST validate that the CONNECT packet matches the format described in section 3.1 and close the Network Connection if it does not match.
[MQTT-3.1.4-2]	The Server MAY check that the contents of the CONNECT packet meet any further restrictions and SHOULD perform authentication and authorization checks. If any of these checks fail, it MUST close the Network Connection.
[MQTT-3.1.4-3]	If the ClientID represents a Client already connected to the Server, the Server sends a DISCONNECT packet to the existing Client with Reason Code of 0x8E (Session taken over) as described in section 4.13 and MUST close the Network Connection of the existing Client.
[MQTT-3.1.4-4]	The Server MUST perform the processing of Clean Start.
[MQTT-3.1.4-5]	The Server MUST acknowledge the CONNECT packet with a CONNACK packet containing a 0x00 (Success) Reason Code.
[MQTT-3.1.4-6]	If the Server rejects the CONNECT, it MUST NOT process any data sent by the Client after the CONNECT packet except AUTH packets.
[MQTT-3.2.0-1]	The Server MUST send a CONNACK with a 0x00 (Success) Reason Code before sending any Packet other than AUTH.
[MQTT-3.2.0-2]	The Server MUST NOT send more than one CONNACK in a Network Connection.
[MQTT-3.2.2-1]	Byte 1 is the "Connect Acknowledge Flags". Bits 7-1 are reserved and MUST be set to 0.

[MQTT-3.2.2-2]	If the Server accepts a connection with Clean Start set to 1, the Server MUST set Session Present to 0 in the CONNACK packet in addition to setting a 0x00 (Success) Reason Code in the CONNACK packet.
[MQTT-3.2.2-3]	If the Server accepts a connection with Clean Start set to 0 and the Server has Session State for the ClientID, it MUST set Session Present to 1 in the CONNACK packet, otherwise it MUST set Session Present to 0 in the CONNACK packet. In both cases it MUST set a 0x00 (Success) Reason Code in the CONNACK packet.
[MQTT-3.2.2-4]	If the Client does not have Session State and receives Session Present set to 1 it MUST close the Network Connection.
[MQTT-3.2.2-5]	If the Client does have Session State and receives Session Present set to 0 it MUST discard its Session State if it continues with the Network Connection.
[MQTT-3.2.2-6]	If a Server sends a CONNACK packet containing a non-zero Reason Code it MUST set Session Present to 0.
[MQTT-3.2.2-7]	If a Server sends a CONNACK packet containing a Reason code of 0x80 or greater it MUST then close the Network Connection.
[MQTT-3.2.2-8]	The Server sending the CONNACK packet MUST use one of the Connect Reason Code values.
[MQTT-3.2.2-9]	If a Server does not support QoS 1 or QoS 2 PUBLISH packets it MUST send a Maximum QoS in the CONNACK packet specifying the highest QoS it supports.
[MQTT-3.2.2-10]	A Server that does not support QoS 1 or QoS 2 PUBLISH packets MUST still accept SUBSCRIBE packets containing a Requested QoS of 0, 1 or 2.
[MQTT-3.2.2-11]	If a Client receives a Maximum QoS from a Server, it MUST NOT send PUBLISH packets at a QoS level exceeding the Maximum QoS level specified.
[MQTT-3.2.2-12]	If a Server receives a CONNECT packet containing a Will QoS that exceeds its capabilities, it MUST reject the connection. It SHOULD use a CONNACK packet with Reason Code 0x9B (QoS not supported) as described in section 4.13 Handling errors, and MUST close the Network Connection.
[MQTT-3.2.2-13]	If a Server receives a CONNECT packet containing a Will Message with the Will Retain 1, and it does not support retained messages, the Server MUST reject the connection request. It SHOULD send CONNACK with Reason Code 0x9A (Retain not supported) and then it MUST close the Network Connection.
[MQTT-3.2.2-14]	A Client receiving Retain Available set to 0 from the Server MUST NOT send a PUBLISH packet with the RETAIN flag set to 1.
[MQTT-3.2.2-15]	The Client MUST NOT send packets exceeding Maximum Packet Size to the Server.
[MQTT-3.2.2-16]	If the Client connects using a zero length Client Identifier, the Server MUST respond with a CONNACK containing an Assigned Client Identifier. The Assigned Client Identifier MUST be a new Client Identifier not used by any other Session currently in the Server.
[MQTT-3.2.2-17]	The Client MUST NOT send a Topic Alias in a PUBLISH packet to the Server greater than this value.
[MQTT-3.2.2-18]	Topic Alias Maximum is absent, the Client MUST NOT send any Topic Aliases on to the Server.

[MQTT-3.2.2-19]	The Server MUST NOT send this property if it would increase the size of the CONNACK packet beyond the Maximum Packet Size specified by the Client.
[MQTT-3.2.2-20]	The Server MUST NOT send this property if it would increase the size of the CONNACK packet beyond the Maximum Packet Size specified by the Client.
[MQTT-3.2.2-21]	If the Server sends a Server Keep Alive on the CONNACK packet, the Client MUST use this value instead of the Keep Alive value the Client sent on CONNECT.
[MQTT-3.2.2-22]	If the Server does not send the Server Keep Alive, the Server MUST use the Keep Alive value set by the Client on CONNECT.
[MQTT-3.3.1-1]	The DUP flag MUST be set to 1 by the Client or Server when it attempts to re-deliver a PUBLISH packet.
[MQTT-3.3.1-2]	The DUP flag MUST be set to 0 for all QoS 0 messages.
[MQTT-3.3.1-3]	The DUP flag in the outgoing PUBLISH packet is set independently to the incoming PUBLISH packet, its value MUST be determined solely by whether the outgoing PUBLISH packet is a retransmission.
[MQTT-3.3.1-4]	A PUBLISH Packet MUST NOT have both QoS bits set to 1.
[MQTT-3.3.1-5]	If the RETAIN flag is set to 1 in a PUBLISH packet sent by a Client to a Server, the Server MUST replace any existing retained message for this topic and store the Application Message.
[MQTT-3.3.1-6]	If the Payload contains zero bytes it is processed normally by the Server but any retained message with the same topic name MUST be removed and any future subscribers for the topic will not receive a retained message.
[MQTT-3.3.1-7]	A retained message with a Payload containing zero bytes MUST NOT be stored as a retained message on the Server.
[MQTT-3.3.1-8]	If the RETAIN flag is 0 in a PUBLISH packet sent by a Client to a Server, the Server MUST NOT store the message as a retained message and MUST NOT remove or replace any existing retained message.
[MQTT-3.3.1-9]	If Retain Handling is set to 0 the Server MUST send the retained messages matching the Topic Filter of the subscription to the Client.
[MQTT-3.3.1-10]	If Retain Handling is set to 1 then if the subscription did already exist, the Server MUST send all retained message matching the Topic Filter of the subscription to the Client, and if the subscription did not exist, the Server MUST NOT send the retained messages.
[MQTT-3.3.1-11]	If Retain Handling is set to 2, the Server MUST NOT send the retained
[MQTT-3.3.1-12]	If the value of Retain As Published subscription option is set to 0, the Server MUST set the RETAIN flag to 0 when forwarding an Application Message regardless of how the RETAIN flag was set in the received PUBLISH packet.
[MQTT-3.3.1-13]	If the value of Retain As Published subscription option is set to 1, the Server MUST set the RETAIN flag equal to the RETAIN flag in the received PUBLISH packet.
[MQTT-3.3.2-1]	The Topic Name MUST be present as the first field in the PUBLISH packet Variable Header. It MUST be a UTF-8 Encoded String.

[MQTT-3.3.2-2]	The Topic Name in the PUBLISH packet MUST NOT contain wildcard characters.
[MQTT-3.3.2-3]	The Topic Name in a PUBLISH packet sent by a Server to a subscribing Client MUST match the Subscription's Topic Filter.
[MQTT-3.3.2-4]	A Server MUST send the Payload Format Indicator unaltered to all subscribers receiving the message.
[MQTT-3.3.2-5]	If the Message Expiry Interval has passed and the Server has not managed to start onward delivery to a matching subscriber, then it MUST delete the copy of the message for that subscriber.
[MQTT-3.3.2-6]	The PUBLISH packet sent to a Client by the Server MUST contain a Message Expiry Interval set to the received value minus the time that the message has been waiting in the Server.
[MQTT-3.3.2-7]	A receiver MUST NOT carry forward any Topic Alias mappings from one Network Connection to another.
[MQTT-3.3.2-8]	A sender MUST NOT send a PUBLISH packet containing a Topic Alias which has the value 0.
[MQTT-3.3.2-9]	A Client MUST NOT send a PUBLISH packet with a Topic Alias greater than the Topic Alias Maximum value returned by the Server in the CONNACK packet.
[MQTT-3.3.2-10]	A Client MUST accept all Topic Alias values greater than 0 and less than or equal to the Topic Alias Maximum value that it sent in the CONNECT packet.
[MQTT-3.3.2-11]	A Server MUST NOT send a PUBLISH packet with a Topic Alias greater than the Topic Alias Maximum value sent by the Client in the CONNECT packet.
[MQTT-3.3.2-12]	A Server MUST accept all Topic Alias values greater than 0 and less than or equal to the Topic Alias Maximum value that it returned in the CONNACK packet.
[MQTT-3.3.2-13]	The Response Topic MUST be a UTF-8 Encoded String.
[MQTT-3.3.2-14]	The Response Topic MUST NOT contain wildcard characters.
[MQTT-3.3.2-15]	The Server MUST send the Response Topic unaltered to all subscribers receiving the Application Message.
[MQTT-3.3.2-16]	The Server MUST send the Correlation Data unaltered to all subscribers receiving the Application Message.
[MQTT-3.3.2-17]	The Server MUST send all User Properties unaltered in a PUBLISH packet when forwarding the Application Message to a Client.
[MQTT-3.3.2-18]	The Server MUST maintain the order of User Properties when forwarding the Application Message.
[MQTT-3.3.2-19]	The Content Type MUST be a UTF-8 Encoded String.
[MQTT-3.3.2-20]	A Server MUST send the Content Type unaltered to all subscribers receiving the Application Message.

[MQTT-3.3.4-1]	The receiver of a PUBLISH Packet MUST respond with the packet as determined by the QoS in the PUBLISH Packet.
[MQTT-3.3.4-2]	In this case the Server MUST deliver the message to the Client respecting the maximum QoS of all the matching subscriptions.
[MQTT-3.3.4-3]	If the Client specified a Subscription Identifier for any of the overlapping subscriptions the Server MUST send those Subscription Identifiers in the message which is published as the result of the subscriptions.
[MQTT-3.3.4-4]	If the Server sends a single copy of the message it MUST include in the PUBLISH packet the Subscription Identifiers for all matching subscriptions which have a Subscription Identifiers, their order is not significant.
[MQTT-3.3.4-5]	If the Server sends multiple PUBLISH packets it MUST send, in each of them, the Subscription Identifier of the matching subscription if it has a Subscription Identifier.
[MQTT-3.3.4-6]	A PUBLISH packet sent from a Client to a Server MUST NOT contain a Subscription Identifier.
[MQTT-3.3.4-7]	The Client MUST NOT send more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets for which it has not received PUBACK, PUBCOMP, or PUBREC with a Reason Code of 128 or greater from the Server.
[MQTT-3.3.4-8]	The Client MUST NOT delay the sending of any packets other than PUBLISH packets due to having sent Receive Maximum PUBLISH packets without receiving acknowledgements for them.
[MQTT-3.3.4-9]	The Server MUST NOT send more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets for which it has not received PUBACK, PUBCOMP, or PUBREC with a Reason Code of 128 or greater from the Client.
[MQTT-3.3.4-10]	The Server MUST NOT delay the sending of any packets other than PUBLISH packets due to having sent Receive Maximum PUBLISH packets without receiving acknowledgements for them.
[MQTT-3.4.2-1]	The Client or Server sending the PUBACK packet MUST use one of the PUBACK Reason Codes.
[MQTT-3.4.2-2]	The sender MUST NOT send this property if it would increase the size of the PUBACK packet beyond the Maximum Packet Size specified by the receiver.
[MQTT-3.4.2-3]	The sender MUST NOT send this property if it would increase the size of the PUBACK packet beyond the Maximum Packet Size specified by the receiver.
[MQTT-3.5.2-1]	The Client or Server sending the PUBREC packet MUST use one of the PUBREC Reason Codes.
[MQTT-3.5.2-2]	The sender MUST NOT send this property if it would increase the size of the PUBREC packet beyond the Maximum Packet Size specified by the receiver.
[MQTT-3.5.2-3]	The sender MUST NOT send this property if it would increase the size of the PUBREC packet beyond the Maximum Packet Size specified by the receiver.
[MQTT-3.6.1-1]	Bits 3,2,1 and 0 of the Fixed Header in the PUBREL packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network Connection.



[MQTT-3.6.2-1]	The Client or Server sending the PUBREL packet MUST use one of the PUBREL Reason Codes.
[MQTT-3.6.2-2]	The sender MUST NOT send this Property if it would increase the size of the PUBREL packet beyond the Maximum Packet Size specified by the receiver.
[MQTT-3.6.2-3]	The sender MUST NOT send this property if it would increase the size of the PUBREL packet beyond the Maximum Packet Size specified by the receiver.
[MQTT-3.7.2-1]	The Client or Server sending the PUBCOMP packets MUST use one of the PUBCOMP Reason Codes.
[MQTT-3.7.2-2]	The sender MUST NOT use this Property if it would increase the size of the PUBCOMP packet beyond the Maximum Packet Size specified by the receiver.
[MQTT-3.7.2-3]	The sender MUST NOT send this property if it would increase the size of the PUBCOMP packet beyond the Maximum Packet Size specified by receiver.
[MQTT-3.8.1-1]	Bits 3,2,1 and 0 of the Fixed Header of the SUBSCRIBE packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network Connection
[MQTT-3.8.3-1]	The Topic Filters MUST be a UTF-8 Encoded String.
[MQTT-3.8.3-2]	The Payload MUST contain at least one Topic Filter and Subscription Options pair.
[MQTT-3.8.3-3]	Bit 2 of the Subscription Options represents the No Local option. If the value is 1, Application Messages MUST NOT be forwarded to a connection with a ClientID equal to the ClientID of the publishing connection.
[MQTT-3.8.3-4]	It is a Protocol Error to set the No Local bit to 1 on a Shared Subscription.
[MQTT-3.8.3-5]	The Server MUST treat a SUBSCRIBE packet as malformed if any of Reserved bits in the Payload are non-zero.
[MQTT-3.8.4-1]	When the Server receives a SUBSCRIBE packet from a Client, the Server MUST respond with a SUBACK packet.
[MQTT-3.8.4-2]	The SUBACK packet MUST have the same Packet Identifier as the SUBSCRIBE packet that it is acknowledging.
[MQTT-3.8.4-3]	If a Server receives a SUBSCRIBE packet containing a Topic Filter that is identical to a Non-shared Subscription's Topic Filter for the current Session then it MUST replace that existing Subscription with a new Subscription.
[MQTT-3.8.4-4]	If the Retain Handling option is 0, any existing retained messages matching the Topic Filter MUST be re-sent, but Application Messages MUST NOT be lost due to replacing the Subscription.
[MQTT-3.8.4-5]	If a Server receives a SUBSCRIBE packet that contains multiple Topic Filters it MUST handle that packet as if it had received a sequence of multiple SUBSCRIBE packets, except that it combines their responses into a single SUBACK response.
[MQTT-3.8.4-6]	The SUBACK packet sent by the Server to the Client MUST contain a Reason Code for each Topic Filter/Subscription Option pair.

[MQTT-3.8.4-7]	This Reason Code MUST either show the maximum QoS that was granted for that Subscription or indicate that the subscription failed.
[MQTT-3.8.4-8]	The QoS of Payload Messages sent in response to a Subscription MUST be the minimum of the QoS of the originally published message and the Maximum QoS granted by the Server.
[MQTT-3.9.2-1]	The Server MUST NOT send this Property if it would increase the size of the SUBACK packet beyond the Maximum Packet Size specified by the Client.
[MQTT-3.9.2-2]	The Server MUST NOT send this property if it would increase the size of the SUBACK packet beyond the Maximum Packet Size specified by the Client.
[MQTT-3.9.3-1]	The order of Reason Codes in the SUBACK packet MUST match the order of Topic Filters in the SUBSCRIBE packet.
[MQTT-3.9.3-2]	The Server sending the SUBACK packet MUST send one of the Subscribe Reason Code values for each Topic Filter received.
[MQTT-3.10.1-1]	Bits 3,2,1 and 0 of the Fixed Header of the UNSUBSCRIBE packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network Connection
[MQTT-3.10.3-1]	The Topic Filters in an UNSUBSCRIBE packet MUST be UTF-8 Encoded Strings.
[MQTT-3.10.3-2]	The Payload of an UNSUBSCRIBE packet MUST contain at least one Topic Filter.
[MQTT-3.10.4-1]	The Topic Filters (whether they contain wildcards or not) supplied in an UNSUBSCRIBE packet MUST be compared character-by-character with the current set of Topic Filters held by the Server for the Client. If any filter matches exactly then its owning Subscription MUST be deleted.
[MQTT-3.10.4-2]	When a Server receives UNSUBSCRIBE It MUST stop adding any new messages which match the Topic Filters, for delivery to the Client.
[MQTT-3.10.4-3]	When a Server receives UNSUBSCRIBE It MUST complete the delivery of any QoS 1 or QoS 2 messages which match the Topic Filters and it has started to send to the Client.
[MQTT-3.10.4-4]	The Server MUST respond to an UNSUBSCRIBE request by sending an UNSUBACK packet.
[MQTT-3.10.4-5]	The UNSUBACK packet MUST have the same Packet Identifier as the UNSUBSCRIBE packet. Even where no Topic Subscriptions are deleted, the Server MUST respond with an UNSUBACK.
[MQTT-3.10.4-6]	If a Server receives an UNSUBSCRIBE packet that contains multiple Topic Filters, it MUST process that packet as if it had received a sequence of multiple UNSUBSCRIBE packets, except that it sends just one UNSUBACK response.
[MQTT-3.11.2-1]	The Server MUST NOT send this Property if it would increase the size of the UNSUBACK packet beyond the Maximum Packet Size specified by the Client.
[MQTT-3.11.2-2]	The Server MUST NOT send this property if it would increase the size of the UNSUBACK packet beyond the Maximum Packet Size specified by the receiver.
[MQTT-3.11.3-1]	The order of Reason Codes in the UNSUBACK packet MUST match the order of Topic Filters in the UNSUBSCRIBE packet.

[MQTT-3.11.3-2]	The Server sending the UNSUBACK packet MUST use one of the UNSUBSCRIBE Reason Code values for each Topic Filter received.
[MQTT-3.12.4-1]	The Server MUST send a PINGRESP packet in response to a PINGREQ packet.
[MQTT-3.14.0-1]	A Server MUST NOT send a DISCONNECT until after it has sent a CONNACK with Reason Code of less than 0x80.
[MQTT-3.14.1-1]	The Client or Server MUST validate that reserved bits are set to 0. If they are not zero it sends a DISCONNECT packet with a Reason code of 0x81 (Malformed Packet).
[MQTT-3.14.2-1]	The Client or Server sending the DISCONNECT packet MUST use one of the DISCONNECT Reason Codes.
[MQTT-3.14.2-2]	The Session Expiry Interval MUST NOT be sent on a DISCONNECT by the Server.
[MQTT-3.14.2-3]	The sender MUST NOT use this Property if it would increase the size of the DISCONNECT packet beyond the Maximum Packet Size specified by the receiver.
[MQTT-3.14.2-4]	The sender MUST NOT send this property if it would increase the size of the DISCONNECT packet beyond the Maximum Packet Size specified by the receiver.
[MQTT-3.14.4-1]	After sending a DISCONNECT packet the sender MUST NOT send any more MQTT Control Packets on that Network Connection.
[MQTT-3.14.4-2]	After sending a DISCONNECT packet the sender MUST close the Network Connection.
[MQTT-3.14.4-3]	On receipt of DISCONNECT with a Reason Code of 0x00 (Success) the Server MUST discard any Will Message associated with the current Connection without publishing it.
[MQTT-3.15.1-1]	Bits 3,2,1 and 0 of the Fixed Header of the AUTH packet are reserved and MUST all be set to 0. The Client or Server MUST treat any other value as malformed and close the Network Connection.
[MQTT-3.15.2-1]	The sender of the AUTH Packet MUST use one of the Authenticate Reason Codes.
[MQTT-3.15.2-2]	The sender MUST NOT send this property if it would increase the size of the AUTH packet beyond the Maximum Packet Size specified by the receiver
[MQTT-3.15.2-3]	The sender MUST NOT send this property if it would increase the size of the AUTH packet beyond the Maximum Packet Size specified by the receiver.
[MQTT-4.1.0-1]	The Client and Server MUST NOT discard the Session State while the Network Connection is open.
[MQTT-4.2.0-1]	A Client or Server MUST support the use of one or more underlying transport protocols that provide an ordered, lossless, stream of bytes from the Client to Server and Server to Client.
[MQTT-4.1.0-2]	The Server MUST discard the Session State when the Network Connection is closed and the Session Expiry Interval has passed.
[MQTT-4.3.1-1]	In the QoS 0 delivery protocol, the sender MUST send a PUBLISH packet with QoS 0 and DUP flag set to 0.

[MQTT-4.3.2-1]	In the QoS 1 delivery protocol, the sender MUST assign an unused Packet Identifier each time it has a new Application Message to publish.
[MQTT-4.3.2-2]	In the QoS 1 delivery protocol, the sender MUST send a PUBLISH packet containing this Packet Identifier with QoS 1 and DUP flag set to 0.
[MQTT-4.3.2-3]	In the QoS 1 delivery protocol, the sender MUST treat the PUBLISH packet as “unacknowledged” until it has received the corresponding PUBACK packet from the receiver.
[MQTT-4.3.2-4]	In the QoS 1 delivery protocol, the receiver MUST respond with a PUBACK packet containing the Packet Identifier from the incoming PUBLISH packet, having accepted ownership of the Application Message.
[MQTT-4.3.2-5]	In the QoS 1 delivery protocol, the receiver after it has sent a PUBACK packet the receiver MUST treat any incoming PUBLISH packet that contains the same Packet Identifier as being a new Application Message, irrespective of the setting of its DUP flag.
[MQTT-4.3.3-1]	In the QoS 2 delivery protocol, the sender MUST assign an unused Packet Identifier when it has a new Application Message to publish.
[MQTT-4.3.3-2]	In the QoS 2 delivery protocol, the sender MUST send a PUBLISH packet containing this Packet Identifier with QoS 2 and DUP flag set to 0.
[MQTT-4.3.3-3]	In the QoS 2 delivery protocol, the sender MUST treat the PUBLISH packet as “unacknowledged” until it has received the corresponding PUBREC packet from the receiver.
[MQTT-4.3.3-4]	In the QoS 2 delivery protocol, the sender MUST send a PUBREL packet when it receives a PUBREC packet from the receiver with a Reason Code value less than 0x80. This PUBREL packet MUST contain the same Packet Identifier as the original PUBLISH packet.
[MQTT-4.3.3-5]	In the QoS 2 delivery protocol, the sender MUST treat the PUBREL packet as “unacknowledged” until it has received the corresponding PUBCOMP packet from the receiver.
[MQTT-4.3.3-6]	In the QoS 2 delivery protocol, the sender MUST NOT re-send the PUBLISH once it has sent the corresponding PUBREL packet.
[MQTT-4.3.3-7]	In the QoS 2 delivery protocol, the sender MUST NOT apply Application Message expiry if a PUBLISH packet has been sent.
[MQTT-4.3.3-8]	In the QoS 2 delivery protocol, the receiver MUST respond with a PUBREC containing the Packet Identifier from the incoming PUBLISH packet, having accepted ownership of the Application Message.
[MQTT-4.3.3-9]	In the QoS 2 delivery protocol, the receiver if it has sent a PUBREC with a Reason Code of 0x80 or greater, the receiver MUST treat any subsequent PUBLISH packet that contains that Packet Identifier as being a new Application Message.
[MQTT-4.3.3-10]	In the QoS 2 delivery protocol, the receiver until it has received the corresponding PUBREL packet, the receiver MUST acknowledge any subsequent PUBLISH packet with the same Packet Identifier by sending a PUBREC. It MUST NOT cause duplicate messages to be delivered to any onward recipients in this case.
[MQTT-4.3.3-11]	In the QoS 2 delivery protocol, the receiver MUST respond to a PUBREL packet by sending a PUBCOMP packet containing the same Packet Identifier as the PUBREL.

[MQTT-4.3.3-12]	In the QoS 2 delivery protocol, the receiver After it has sent a PUBCOMP, the receiver MUST treat any subsequent PUBLISH packet that contains that Packet Identifier as being a new Application Message.
[MQTT-4.3.3-13]	In the QoS 2 delivery protocol, the receiver MUST continue the QoS 2 acknowledgement sequence even if it has applied Application Message expiry.
[MQTT-4.4.0-1]	When a Client reconnects with Clean Start set to 0 and a session is present, both the Client and Server MUST resend any unacknowledged PUBLISH packets (where QoS > 0) and PUBREL packets using their original Packet Identifiers. This is the only circumstance where a Client or Server is REQUIRED to resend messages. Clients and Servers MUST NOT resend messages at any other time.
[MQTT-4.4.0-2]	If PUBACK or PUBREC is received containing a Reason Code of 0x80 or greater the corresponding PUBLISH packet is treated as acknowledged, and MUST NOT be retransmitted.
[MQTT-4.5.0-1]	When a Server takes ownership of an incoming Application Message it MUST add it to the Session State for those Clients that have matching Subscriptions.
[MQTT-4.5.0-2]	The Client MUST acknowledge any Publish packet it receives according to the applicable QoS rules regardless of whether it elects to process the Application Message that it contains.
[MQTT-4.6.0-1]	When the Client re-sends any PUBLISH packets, it MUST re-send them in the order in which the original PUBLISH packets were sent (this applies to QoS 1 and QoS 2 messages).
[MQTT-4.6.0-2]	The Client MUST send PUBACK packets in the order in which the corresponding PUBLISH packets were received (QoS 1 messages).
[MQTT-4.6.0-3]	The Client MUST send PUBREC packets in the order in which the corresponding PUBLISH packets were received (QoS 2 messages).
[MQTT-4.6.0-4]	The Client MUST send PUBREL packets in the order in which the corresponding PUBREC packets were received (QoS 2 messages).
[MQTT-4.6.0-5]	When a Server processes a message that has been published to an Ordered Topic, it MUST send PUBLISH packets to consumers (for the same Topic and QoS) in the order that they were received from any given Client.
[MQTT-4.6.0-6]	A Server MUST treat every, Topic as an Ordered Topic when it is forwarding messages on Non-shared Subscriptions.
[MQTT-4.7.0-1]	The wildcard characters can be used in Topic Filters, but MUST NOT be used within a Topic Name.
[MQTT-4.7.1-1]	The multi-level wildcard character MUST be specified either on its own or following a topic level separator. In either case it MUST be the last character specified in the Topic Filter.
[MQTT-4.7.1-2]	The single-level wildcard can be used at any level in the Topic Filter, including first and last levels. Where it is used, it MUST occupy an entire level of the filter.
[MQTT-4.7.2-1]	The Server MUST NOT match Topic Filters starting with a wildcard character (# or +) with Topic Names beginning with a \$ character.

[MQTT-4.7.3-1]	All Topic Names and Topic Filters MUST be at least one character long.
[MQTT-4.7.3-2]	Topic Names and Topic Filters MUST NOT include the null character (Unicode U+0000).
[MQTT-4.7.3-3]	Topic Names and Topic Filters are UTF-8 Encoded Strings; they MUST NOT encode to more than 65,535 bytes.
[MQTT-4.7.3-4]	When it performs subscription matching the Server MUST NOT perform any normalization of Topic Names or Topic Filters, or any modification or substitution of unrecognized characters.
[MQTT-4.8.2-1]	A Shared Subscription's Topic Filter MUST start with \$share/ and MUST contain a ShareName that is at least one character long.
[MQTT-4.8.2-2]	The ShareName MUST NOT contain the characters "/", "+" or "#", but MUST be followed by a "/" character. This "/" character MUST be followed by a Topic Filter.
[MQTT-4.8.2-3]	The Server MUST respect the granted QoS for the Clients subscription.
[MQTT-4.8.2-4]	The Server MUST complete the delivery of the message to that Client when it reconnects.
[MQTT-4.8.2-5]	If the Clients Session terminates before the Client reconnects, the Server MUST NOT send the Application Message to any other subscribed Client.
[MQTT-4.8.2-6]	If a Client responds with a PUBACK or PUBREC containing a Reason Code of 0x80 or greater to a PUBLISH packet from the Server, the Server MUST discard the Application Message and not attempt to send it to any other Subscriber.
[MQTT-4.9.0-1]	The Client or Server MUST set its initial send quota to a non-zero value not exceeding the Receive Maximum.
[MQTT-4.9.0-2]	Each time the Client or Server sends a PUBLISH packet at QoS > 0, it decrements the send quota. If the send quota reaches zero, the Client or Server MUST NOT send any more PUBLISH packets with QoS > 0.
[MQTT-4.9.0-3]	The Client and Server MUST continue to process and respond to all other MQTT Control Packets even if the quota is zero.
[MQTT-4.12.0-1]	If the Server does not support the Authentication Method supplied by the Client, it MAY send a CONNACK with a Reason Code of 0x8C (Bad authentication method) or 0x87 (Not Authorized) as described in section 4.13 and MUST close the Network Connection.
[MQTT-4.12.0-2]	If the Server requires additional information to complete the authorization, it can send an AUTH packet to the Client. This packet MUST contain a Reason Code of 0x18 (Continue authentication).
[MQTT-4.12.0-3]	The Client responds to an AUTH packet from the Server by sending a further AUTH packet. This packet MUST contain a Reason Code of 0x18 (Continue authentication).
[MQTT-4.12.0-4]	The Server can reject the authentication at any point in this process. It MAY send a CONNACK with a Reason Code of 0x80 or above as described in section 4.13, and MUST close the Network Connection.

[MQTT-4.12.0-5]	If the initial CONNECT packet included an Authentication Method property then all AUTH packets, and any successful CONNACK packet MUST include an Authentication Method Property with the same value as in the CONNECT packet.
[MQTT-4.12.0-6]	If the Client does not include an Authentication Method in the CONNECT, the Server MUST NOT send an AUTH packet, and it MUST NOT send an Authentication Method in the CONNACK packet.
[MQTT-4.12.0-7]	If the Client does not include an Authentication Method in the CONNECT, the Client MUST NOT send an AUTH packet to the Server.
[MQTT-4.12.1-1]	If the Client supplied an Authentication Method in the CONNECT packet it can initiate a re-authentication at any time after receiving a CONNACK. It does this by sending an AUTH packet with a Reason Code of 0x19 (Re-authentication). The Client MUST set the Authentication Method to the same value as the Authentication Method originally used to authenticate the Network Connection.
[MQTT-4.12.1-2]	If the re-authentication fails, the Client or Server SHOULD send DISCONNECT with an appropriate Reason Code and MUST close the Network Connection.
[MQTT-4.13.1-1]	When a Server detects a Malformed Packet or Protocol Error, and a Reason Code is given in the specification, it MUST close the Network Connection.
[MQTT-4.13.2-1]	The CONNACK and DISCONNECT packets allow a Reason Code of 0x80 or greater to indicate that the Network Connection will be closed. If a Reason Code of 0x80 or greater is specified, then the Network Connection MUST be closed whether or not the CONNACK or DISCONNECT is sent.
[MQTT-6.0.0-1]	MQTT Control Packets MUST be sent in WebSocket binary data frames. If any other type of data frame is received the recipient MUST close the Network Connection.
[MQTT-6.0.0-2]	A single WebSocket data frame can contain multiple or partial MQTT Control Packets. The receiver MUST NOT assume that MQTT Control Packets are aligned on WebSocket frame boundaries.
[MQTT-6.0.0-3]	The Client MUST include “mqtt” in the list of WebSocket Sub Protocols it offers.
[MQTT-6.0.0-4]	The WebSocket Subprotocol name selected and returned by the Server MUST be “mqtt”.

---

## Appendix C. Summary of new features in MQTT v5.0 (non-normative)

The following new features are added to MQTT v5.0

- Session expiry  
Split the Clean Session flag into a Clean Start flag which indicates that the session should start without using an existing session, and a Session Expiry interval which says how long to retain the session after a disconnect. The session expiry interval can be modified at disconnect. Setting of Clean Start to 1 and Session Expiry Interval to 0 is equivalent in MQTT v3.1.1 of setting Clean Session to 1.
- Message expiry  
Allow an expiry interval to be set when a message is published.
- Reason code on all ACKs  
Change all response packets to contain a reason code. This include CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK, DISCONNECT, and AUTH. This allows the invoker to determine whether the requested function succeeded.
- Reason string on all ACKs  
Change most packets with a reason code to also allow an optional reason string. This is designed for problem determination and is not intended to be parsed by the receiver.
- Server disconnect  
Allow DISCONNECT to be sent by the Server to indicate the reason the connection is closed.
- Payload format and content type  
Allow the payload format (binary, text) and a MIME style content type to be specified when a message is published. These are forwarded on to the receiver of the message.
- Request / Response  
Formalize the request/response pattern within MQTT and provide the Response Topic and Correlation Data properties to allow response messages to be routed back to the publisher of a request. Also, add the ability for the Client to get configuration information from the Server about how to construct the response topics.
- Shared Subscriptions  
Add shared subscription support allowing for load balanced consumers of a subscription
- Subscription ID  
Allow a numeric subscription identifier to be specified on a SUBSCRIBE, and returned on the message when it is delivered. This allows the Client to determine which subscription or subscriptions caused the message to be delivered.
- Topic Alias  
Decrease the size of the MQTT packet overhead by allowing the topic name to be abbreviated to a small integer. The Client and Server independently specify how many topic aliases they allow.
- Flow control  
Allow the Client and Server to independently specify the number of outstanding reliable messages (QoS>0) they allow. The sender pauses sending such messages to stay below this quota. This is used to limit the rate of reliable messages, and to limit how many are in flight at one time.



3960

3961 • User properties

3962 Add User Properties to most packets. User properties on PUBLISH are included with the message

3963 and are defined by the Client applications. The user properties on PUBLISH and Will Properties are

3964 forwarded by the Server to the receiver of the message. User properties on the CONNECT,

3965 SUBSCRIBE, and UNSUBSCRIBE packets are defined by the Server implementation. The user

3966 properties on CONNACK PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK and

3967 AUTH packets are defined by the sender, and are unique to the sender implementation. The meaning

3968 of user properties is not defined by MQTT.

3969

3970 • Maximum Packet Size

3971 Allow the Client and Server to independently specify the maximum packet size they support. It is an

3972 error for the session partner to send a larger packet.

3973

3974 • Optional Server feature availability

3975 Define a set of features which the Server does not allow and provide a mechanism for the Server to

3976 specify this to the Client. The features which can be specified in this way are: Maximum QoS, Retain

3977 Available, Wildcard Subscription Available, Subscription Identifier Available, and Shared Subscription

3978 Available. It is an error for the Client to use features that the Server has declared are not available.

3979

3980 It is possible in earlier versions of MQTT for a Server to not implement a feature by declaring that the

3981 Client is not authorized for that function. This feature allows such optional behavior to be declared

3982 and adds specific Reason Codes when the Client uses one of these features anyway.

3983

3984 • Enhanced authentication

3985 Provide a mechanism to enable challenge/response style authentication including mutual

3986 authentication. This allows SASL style authentication to be used if supported by both Client and

3987 Server, and includes the ability for a Client to re-authenticate within a connection.

3988

3989 • Subscription options

3990 Provide subscription options primarily defined to allow for message bridge applications. These include

3991 an option to not send messages originating on this Client (noLocal), and options for handling retained

3992 messages on subscribe.

3993

3994 • Will delay

3995 Add the ability to specify a delay between the end of the connection and sending the will message.

3996 This is designed so that if a connection to the session is re-established then the will message is not

3997 sent. This allows for brief interruptions of the connection without notification to others.

3998

3999 • Server Keep Alive

4000 Allow the Server to specify the value it wishes the Client to use as a keep alive. This allows the

4001 Server to set a maximum allowed keepalive and still have the Client honor it.

4002

4003 • Assigned ClientID

4004 In cases where the ClientID is assigned by the Server, return the assigned ClientID. This also lifts the

4005 restriction that Server assigned ClientIDs can only be used with Clean Session=1 connections.

4006

4007 • Server reference

4008 Allow the Server to specify an alternate Server to use on CONNACK or DISCONNECT. This can be

4009 used as a redirect or to do provisioning.

4010