



MQTT Version 5.0

Committee Specification Draft 02 / Public Review Draft 02

26 October 2017

Specification URIs

This version:

<http://docs.oasis-open.org/mqtt/mqtt/v5.0/csprd02/mqtt-v5.0-csprd02.docx> (Authoritative)
<http://docs.oasis-open.org/mqtt/mqtt/v5.0/csprd02/mqtt-v5.0-csprd02.html>
<http://docs.oasis-open.org/mqtt/mqtt/v5.0/csprd02/mqtt-v5.0-csprd02.pdf>

Previous version:

<http://docs.oasis-open.org/mqtt/mqtt/v5.0/csprd01/mqtt-v5.0-csprd01.docx> (Authoritative)
<http://docs.oasis-open.org/mqtt/mqtt/v5.0/csprd01/mqtt-v5.0-csprd01.html>
<http://docs.oasis-open.org/mqtt/mqtt/v5.0/csprd01/mqtt-v5.0-csprd01.pdf>

Latest version:

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

Technical Committee:

OASIS Message Queuing Telemetry Transport (MQTT) TC

Chairs:

Brian Raymor (brian.raymor@microsoft.com), Microsoft
Richard Coppen (coppen@uk.ibm.com), IBM

Editors:

Andrew Banks (andrew_banks@uk.ibm.com), IBM
Ed Briggs (edbriggs@microsoft.com), Microsoft
Ken Borgendale (kwb@us.ibm.com), IBM
Rahul Gupta (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.

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 Committee Specification Public Review Draft 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. 26 October 2017. OASIS Committee Specification Draft 02 / Public Review Draft 02.

<http://docs.oasis-open.org/mqtt/mqtt/v5.0/csprd02/mqtt-v5.0-csprd02.html>. Latest version:

<http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>.

Notices

Copyright © OASIS Open 2017. 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	IPR 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	19
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	27
2.4	Reason Code	28
3	MQTT Control Packets	31
3.1	CONNECT – Connection Request	31
3.1.1	CONNECT Fixed Header	31
3.1.2	CONNECT Variable Header	31
3.1.2.1	Protocol Name	31
3.1.2.2	Protocol Version	32
3.1.2.3	Connect Flags	32
3.1.2.4	Clean Start	33
3.1.2.5	Will Flag	33
3.1.2.6	Will QoS	34
3.1.2.7	Will Retain	34
3.1.2.8	User Name Flag	34

3.1.2.9 Password Flag	34
3.1.2.10 Keep Alive.....	35
3.1.2.11 CONNECT Properties.....	35
3.1.2.11.1 Property Length.....	35
3.1.2.11.2 Session Expiry Interval.....	36
3.1.2.11.3 Receive Maximum.....	37
3.1.2.11.4 Maximum Packet Size.....	37
3.1.2.11.5 Topic Alias Maximum	38
3.1.2.11.6 Request Response Information.....	38
3.1.2.11.7 Request Problem Information.....	38
3.1.2.11.8 User Property	39
3.1.2.11.9 Authentication Method.....	39
3.1.2.11.10 Authentication Data	39
3.1.2.12 Variable Header non-normative example.....	40
3.1.3 CONNECT Payload.....	41
3.1.3.1 Client Identifier (ClientID).....	41
3.1.3.2 Will Properties.....	41
3.1.3.2.1 Property Length.....	41
3.1.3.2.2 Will Delay Interval	42
3.1.3.2.3 Payload Format Indicator	42
3.1.3.2.4 Message Expiry Interval	42
3.1.3.2.5 Content Type.....	43
3.1.3.2.6 Response Topic	43
3.1.3.2.7 Correlation Data	43
3.1.3.2.8 User Property	43
3.1.3.3 Will Topic	43
3.1.3.4 Will Payload	44
3.1.3.5 User Name.....	44
3.1.3.6 Password	44
3.1.4 CONNECT Actions	44
3.2 CONNACK – Connect acknowledgement	45
3.2.1 CONNACK Fixed Header	46
3.2.2 CONNACK Variable Header	46
3.2.2.1 Connect Acknowledge Flags.....	46
3.2.2.1.1 Session Present.....	46
3.2.2.2 Connect Reason Code.....	47
3.2.2.3 CONNACK Properties.....	48
3.2.2.3.1 Property Length.....	48
3.2.2.3.2 Session Expiry Interval.....	48
3.2.2.3.3 Receive Maximum.....	49
3.2.2.3.4 Maximum QoS	49
3.2.2.3.5 Retain Available	49
3.2.2.3.6 Maximum Packet Size.....	50
3.2.2.3.7 Assigned Client Identifier	50
3.2.2.3.8 Topic Alias Maximum	50
3.2.2.3.9 Reason String	51
3.2.2.3.10 User Property	51
3.2.2.3.11 Wildcard Subscription Available	51
3.2.2.3.12 Subscription Identifiers Available	52

3.2.2.3.13 Shared Subscription Available	52
3.2.2.3.14 Server Keep Alive	52
3.2.2.3.15 Response Information	52
3.2.2.3.16 Server Reference	53
3.2.2.3.17 Authentication Method.....	53
3.2.2.3.18 Authentication Data	53
3.2.3 CONNACK Payload	53
3.3 PUBLISH – Publish message	54
3.3.1 PUBLISH Fixed Header	54
3.3.1.1 DUP	54
3.3.1.2 QoS.....	54
3.3.1.3 RETAIN.....	55
3.3.1.4 Remaining Length.....	56
3.3.2 PUBLISH Variable Header	56
3.3.2.1 Topic Name	56
3.3.2.2 Packet Identifier	57
3.3.2.3 PUBLISH Properties	57
3.3.2.3.1 Property Length.....	57
3.3.2.3.2 Payload Format Indicator	57
3.3.2.3.3 Message Expiry Interval`	57
3.3.2.3.4 Topic Alias.....	57
3.3.2.3.5 Response Topic	58
3.3.2.3.6 Correlation Data	59
3.3.2.3.7 User Property	59
3.3.2.3.8 Subscription Identifier.....	60
3.3.2.3.9 Content Type.....	60
3.3.3 PUBLISH Payload	61
3.3.4 PUBLISH Actions	61
3.4 PUBACK – Publish acknowledgement.....	63
3.4.1 PUBACK Fixed Header	63
3.4.2 PUBACK Variable Header.....	64
3.4.2.1 PUBACK Reason Code	64
3.4.2.2 PUBACK Properties.....	65
3.4.2.2.1 Property Length.....	65
3.4.2.2.2 Reason String	65
3.4.2.2.3 User Property	65
3.4.3 PUBACK Payload.....	65
3.4.4 PUBACK Actions	65
3.5 PUBREC – Publish received (QoS 2 delivery part 1)	65
3.5.1 PUBREC Fixed Header.....	66
3.5.2 PUBREC Variable Header	66
3.5.2.1 PUBREC Reason Code	66
3.5.2.2 PUBREC Properties.....	67
3.5.2.2.1 Property Length.....	67
3.5.2.2.2 Reason String	67
3.5.2.2.3 User Property	67
3.5.3 PUBREC Payload	68
3.5.4 PUBREC Actions.....	68

3.6 PUBREL – Publish release (QoS 2 delivery part 2)	68
3.6.1 PUBREL Fixed Header	68
3.6.2 PUBREL Variable Header	68
3.6.2.1 PUBREL Reason Code	69
3.6.2.2 PUBREL Properties	69
3.6.2.2.1 Property Length	69
3.6.2.2.2 Reason String	69
3.6.2.2.3 User Property	69
3.6.3 PUBREL Payload	69
3.6.4 PUBREL Actions	70
3.7 PUBCOMP – Publish complete (QoS 2 delivery part 3)	70
3.7.1 PUBCOMP Fixed Header	70
3.7.2 PUBCOMP Variable Header	70
3.7.2.1 PUBCOMP Reason Code	70
3.7.2.2 PUBCOMP Properties	71
3.7.2.2.1 Property Length	71
3.7.2.2.2 Reason String	71
3.7.2.2.3 User Property	71
3.7.3 PUBCOMP Payload	71
3.7.4 PUBCOMP Actions	71
3.8 SUBSCRIBE - Subscribe request	72
3.8.1 SUBSCRIBE Fixed Header	72
3.8.2 SUBSCRIBE Variable Header	72
3.8.2.1 SUBSCRIBE Properties	73
3.8.2.1.1 Property Length	73
3.8.2.1.2 Subscription Identifier	73
3.8.2.1.3 User Property	73
3.8.3 SUBSCRIBE Payload	73
3.8.3.1 Subscription Options	73
3.8.4 SUBSCRIBE Actions	75
3.9 SUBACK – Subscribe acknowledgement	78
3.9.1 SUBACK Fixed Header	78
3.9.2 SUBACK Variable Header	78
3.9.2.1 SUBACK Properties	78
3.9.2.1.1 Property Length	78
3.9.2.1.2 Reason String	78
3.9.2.1.3 User Property	78
3.9.3 SUBACK Payload	79
3.10 UNSUBSCRIBE – Unsubscribe request	80
3.10.1 UNSUBSCRIBE Fixed Header	80
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	81
3.10.4 UNSUBSCRIBE Actions	82
3.11 UNSUBACK – Unsubscribe acknowledgement	82

3.11.1 UNSUBACK Fixed Header	82
3.11.2 UNSUBACK Variable Header	83
3.11.2.1 UNSUBACK Properties.....	83
3.11.2.1.1 Property Length.....	83
3.11.2.1.2 Reason String	83
3.11.2.1.3 User Property	83
3.11.3 UNSUBACK Payload	83
3.12 PINGREQ – PING request	84
3.12.1 PINGREQ Fixed Header	84
3.12.2 PINGREQ Variable Header.....	85
3.12.3 PINGREQ Payload.....	85
3.12.4 PINGREQ Actions	85
3.13 PINGRESP – PING response	85
3.13.1 PINGRESP Fixed Header	85
3.13.2 PINGRESP Variable Header.....	85
3.13.3 PINGRESP Payload.....	85
3.13.4 PINGRESP Actions	85
3.14 DISCONNECT – Disconnect notification.....	86
3.14.1 DISCONNECT Fixed Header	86
3.14.2 DISCONNECT Variable Header.....	86
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	89
3.14.2.2.4 User Property	89
3.14.2.2.5 Server Reference	89
3.14.3 DISCONNECT Payload.....	90
3.14.4 DISCONNECT Actions	90
3.15 AUTH – Authentication exchange	90
3.15.1 AUTH Fixed Header	90
3.15.2 AUTH Variable Header.....	91
3.15.2.1 Authenticate Reason Code	91
3.15.2.2 AUTH Properties.....	91
3.15.2.2.1 Property Length.....	91
3.15.2.2.2 Authentication Method.....	91
3.15.2.2.3 Authentication Data	91
3.15.2.2.4 Reason String	92
3.15.2.2.5 User Property	92
3.15.3 AUTH Payload.....	92
3.15.4 AUTH Actions	92
4 Operational behavior	93
4.1 Session State.....	93
4.1.1 Storing Session State.....	93
4.1.2 Session State non-normative examples.....	94
4.2 Network Connections.....	94
4.3 Quality of Service levels and protocol flows	94

4.3.1	QoS 0: At most once delivery	95
4.3.2	QoS 1: At least once delivery	95
4.3.3	QoS 2: Exactly once delivery	96
4.4	Message delivery retry.....	97
4.5	Message receipt	98
4.6	Message ordering	98
4.7	Topic Names and Topic Filters	99
4.7.1	Topic wildcards.....	99
4.7.1.1	Topic level separator.....	99
4.7.1.2	Multi-level wildcard.....	99
4.7.1.3	Single-level wildcard	100
4.7.2	Topics beginning with \$.....	100
4.7.3	Topic semantic and usage	101
4.8	Subscriptions	102
4.8.1	Non-shared Subscriptions	102
4.8.2	Shared Subscriptions	102
4.9	Flow Control.....	104
4.10	Request / Response	105
4.10.1	Basic Request Response (non-normative).....	105
4.10.2	Determining a Response Topic value (non-normative)	106
4.11	Server redirection	107
4.12	Enhanced authentication	107
4.12.1	Re-authentication	109
4.13	Handling errors	110
4.13.1	Malformed Packet and Protocol Errors	110
4.13.2	Other errors	111
5	Security (non-normative)	112
5.1	Introduction	112
5.2	MQTT solutions: security and certification.....	112
5.3	Lightweight cryptography and constrained devices	113
5.4	Implementation notes	113
5.4.1	Authentication of Clients by the Server	113
5.4.2	Authorization of Clients by the Server	113
5.4.3	Authentication of the Server by the Client.....	114
5.4.4	Integrity of Application Messages and MQTT Control Packets.....	114
5.4.5	Privacy of Application Messages and MQTT Control Packets.....	114
5.4.6	Non-repudiation of message transmission.....	115
5.4.7	Detecting compromise of Clients and Servers	115
5.4.8	Detecting abnormal behaviors.....	115
5.4.9	Other security considerations	116
5.4.10	Use of SOCKS	116
5.4.11	Security profiles.....	116
5.4.11.1	Clear communication profile.....	116
5.4.11.2	Secured network communication profile	116
5.4.11.3	Secured transport profile.....	117
5.4.11.4	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 IPR Policy

This Committee Specification Public Review Draft is being developed 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.

42

43 **Server:**

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

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

51

52 **Session:**

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

55

56 **Subscription:**

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

60

61 **Shared Subscription:**

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

67

68 **Wildcard Subscription:**

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

72

73 **Topic Name:**

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

76

77 **Topic Filter:**

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

80

81 **MQTT Control Packet:**

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

85

86 **Malformed Packet:**

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

89
90 **Protocol Error:**
91 An error that is detected after the packet has been parsed and found to contain data that is not allowed by
92 the protocol or is inconsistent with the state of the Client or Server. Refer to [section 4.13](#) for information
93 about error handling.

94
95 **Will Message:**
96 An Application Message which is published by the Server after the Network Connection is closed in cases
97 where the Network Connection is not closed normally. Refer to [section 3.1.2.5](#) for information about Will
98 Messages.

99

100 **1.3 Normative references**

101 **[RFC2119]**
102 Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI
103 10.17487/RFC2119, March 1997,
104 <http://www.rfc-editor.org/info/rfc2119>

105

106 **[RFC3629]**
107 Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI
108 10.17487/RFC3629, November 2003,
109 <http://www.rfc-editor.org/info/rfc3629>

110

111 **[RFC6455]**
112 Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December
113 2011,
114 <http://www.rfc-editor.org/info/rfc6455>

115

116 **[Unicode]**
117 The Unicode Consortium. The Unicode Standard,
118 <http://www.unicode.org/versions/latest/>

119

120 **1.4 Non-normative references**

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

124

125 **[RFC5246]**
126 Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI
127 10.17487/RFC5246, August 2008,
128 <http://www.rfc-editor.org/info/rfc5246>

129

130 **[AES]**
131 Advanced Encryption Standard (AES) (FIPS PUB 197).

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

175
176 **[PCIDSS]**
177 PCI-DSS Payment Card Industry Data Security Standard
178 https://www.pcisecuritystandards.org/pci_security/
179
180 **[RFC1928]**
181 Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC
182 1928, DOI 10.17487/RFC1928, March 1996,
183 <http://www.rfc-editor.org/info/rfc1928>
184
185 **[RFC4511]**
186 Sermersheim, J., Ed., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, DOI
187 10.17487/RFC4511, June 2006,
188 <http://www.rfc-editor.org/info/rfc4511>
189
190 **[RFC5280]**
191 Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key
192 Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI
193 10.17487/RFC5280, May 2008,
194 <http://www.rfc-editor.org/info/rfc5280>
195
196 **[RFC6066]**
197 Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI
198 10.17487/RFC6066, January 2011,
199 <http://www.rfc-editor.org/info/rfc6066>
200
201 **[RFC6749]**
202 Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October
203 2012,
204 <http://www.rfc-editor.org/info/rfc6749>
205
206 **[RFC6960]**
207 Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public
208 Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June
209 2013,
210 <http://www.rfc-editor.org/info/rfc6960>
211
212 **[SARBANES]**
213 Sarbanes-Oxley Act of 2002.
214 <http://www.gpo.gov/fdsys/pkg/PLAW-107publ204/html/PLAW-107publ204.htm>
215
216 **[USEUPRIVSH]**
217 U.S.-EU Privacy Shield Framework
218 <https://www.privacyshield.gov>
219

220 **[RFC3986]**
221 Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD
222 66, RFC 3986, DOI 10.17487/RFC3986, January 2005,
223 <http://www.rfc-editor.org/info/rfc3986>

224
225 **[RFC1035]**
226 Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI
227 10.17487/RFC1035, November 1987,
228 <http://www.rfc-editor.org/info/rfc1035>

229
230 **[RFC2782]**
231 Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)",
232 RFC 2782, DOI 10.17487/RFC2782, February 2000,
233 <http://www.rfc-editor.org/info/rfc2782>

235 **1.5 Data representation**

236 **1.5.1 Bits**
237 Bits in a byte are labelled 7 to 0. Bit number 7 is the most significant bit, the least significant bit is
238 assigned bit number 0.
239

240 **1.5.2 Two Byte Integer**
241 Two Byte Integer data values are 16-bit unsigned integers in big-endian order: the high order byte
242 precedes the lower order byte. This means that a 16-bit word is presented on the network as Most
243 Significant Byte (MSB), followed by Least Significant Byte (LSB).
244

245 **1.5.3 Four Byte Integer**
246 Four Byte Integer data values are 32-bit unsigned integers in big-endian order: the high order byte
247 precedes the successively lower order bytes. This means that a 32-bit word is presented on the network
248 as Most Significant Byte (MSB), followed by the next most Significant Byte (MSB), followed by the next
249 most Significant Byte (MSB), followed by Least Significant Byte (LSB).
250

251 **1.5.4 UTF-8 Encoded String**
252 Text fields within the MQTT Control Packets described later are encoded as UTF-8 strings. UTF-8
253 [\[RFC3629\]](#) is an efficient encoding of Unicode [\[Unicode\]](#) characters that optimizes the encoding of ASCII
254 characters in support of text-based communications.
255

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

259
260 Unless stated otherwise all UTF-8 encoded strings can have any length in the range 0 to 65,535 bytes.
261

262 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.							

263

264 The character data in a UTF-8 Encoded String MUST be well-formed UTF-8 as defined by the Unicode
 265 specification [Unicode] and restated in RFC 3629 [RFC3629]. In particular, the character data MUST NOT
 266 include encodings of code points between U+D800 and U+DFFF [MQTT-1.5.4-1]. If the Client or Server
 267 receives an MQTT Control Packet containing ill-formed UTF-8 it is a Malformed Packet. Refer to section
 268 4.13 for information about handling errors.

269

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

273

274 The data SHOULD NOT include encodings of the Unicode [Unicode] code points listed below. If a
 275 receiver (Server or Client) receives an MQTT Control Packet containing any of them it MAY treat it as a
 276 Malformed Packet.

277

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

282

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

286

287 **Non-normative example**

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

290

291 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

292

293 1.5.5 Variable Byte Integer

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

301

302 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)

303

304 Non-normative comment

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

307

```

308 do
309     encodedByte = X MOD 128
310     X = X DIV 128
311     // if there are more data to encode, set the top bit of this byte
312     if (X > 0)
313         encodedByte = encodedByte OR 128
314     endif
315     'output' encodedByte
316 while (X > 0)
  
```

317

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

320

321 **Non-normative comment**

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

323

```
324 multiplier = 1
325 value = 0
326 do
327     encodedByte = 'next byte from stream'
328     value += (encodedByte AND 127) * multiplier
329     if (multiplier > 128*128*128)
330         throw Error(Malformed Variable Byte Integer)
331     multiplier *= 128
332 while ((encodedByte AND 128) != 0)
```

333

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

335

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

337

338 **1.5.6 Binary Data**

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

342

343 **1.5.7 UTF-8 String Pair**

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

346

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

350

351 **1.6 Security**

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

356

357 **1.7 Editing convention**

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

361

362 **1.8 Change history**

363 **1.8.1 MQTT v3.1.1**

364 MQTT v3.1.1 was the first OASIS standard version of MQTT [[MQTTV311](#)].[\[MQTTV311](#).

365 MQTT v3.1.1 is also standardized as ISO/IEC 20922:2016 [[ISO20922](#)].

366

367 **1.8.2 MQTT v5.0**

368 MQTT v5.0 adds a significant number of new features to MQTT while keeping much of the core in place.

369 The major functional objectives are:

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

375

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

377

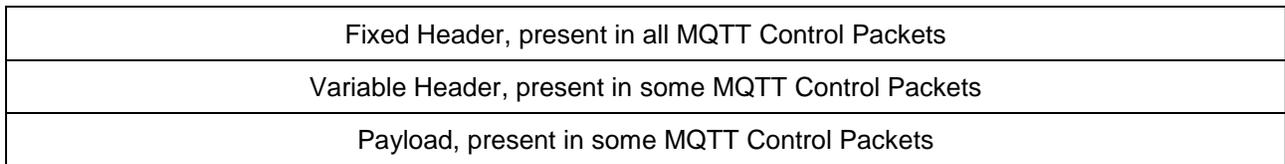
378 **2 MQTT Control Packet format**

379 **2.1 Structure of an MQTT Control Packet**

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

382
383 An MQTT Control Packet consists of up to three parts, always in the following order as shown below.
384

385 Figure 2-1 Structure of an MQTT Control Packet



386
387 **2.1.1 Fixed Header**

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

389
390 *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							

391
392 **2.1.2 MQTT Control Packet type**

393 **Position:** byte 1, bits 7-4.
394 Represented as a 4-bit unsigned value, the values are shown below.

395
396 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

397

398 2.1.3 Flags

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

403

404 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

405

406 DUP = Duplicate delivery of a PUBLISH packet

407 QoS = PUBLISH Quality of Service

408 RETAIN = PUBLISH retained message flag

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

410

411 2.1.4 Remaining Length

412 **Position:** starts at byte 2.

413

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

419

420 2.2 Variable Header

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

424

425 2.2.1 Packet Identifier

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

429

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

431

432 *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

433

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

435

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

438

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

441

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

446

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

450

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

455

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

458

459 **Non-normative comment**

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

463
 464 Client Server
 465 PUBLISH Packet Identifier=0x1234 →→
 466 ←← PUBLISH Packet Identifier=0x1234
 467 PUBACK Packet Identifier=0x1234 →→
 468 ←← PUBACK Packet Identifier=0x1234
 469
 470

471 2.2.2 Properties

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

476
 477 The set of Properties is composed of a Property Length followed by the Properties.
 478

479 2.2.2.1 Property Length

480 The Property Length is encoded as a Variable Byte Integer. The Property Length does not include the
 481 bytes used to encode itself, but includes the length of the Properties. **If there are no properties, this MUST**
 482 **be indicated by including a Property Length of zero [MQTT-2.2.2-1].**
 483

484 2.2.2.2 Property

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

491
 492 *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

493

494 **Non-normative comment**

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

497

498 **2.3 Payload**

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

501

502 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

503

504 **2.4 Reason Code**

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

508

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

512

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

514

515 *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 Subscription 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 Subscription not supported	SUBACK, DISCONNECT

516

517

Non-normative comment

518

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.

519

520

521

522
523

3 MQTT Control Packets

524

3.1 CONNECT – Connection Request

525
526
527

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].

528
529
530

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.

531

532
533
534

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.

535

536

3.1.1 CONNECT Fixed Header

537

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							

538

539

Remaining Length field

540
541

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

542

543

3.1.2 CONNECT Variable Header

544
545
546

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.

547

548

3.1.2.1 Protocol Name

549

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

550

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

553

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

559

Non-normative comment

560

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

562

3.1.2.2 Protocol Version

563

564 *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

565

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

568

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

573

3.1.2.3 Connect Flags

574

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

577 *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

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

581

582 3.1.2.4 Clean Start

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

584
585 This bit specifies whether the Connection starts a new Session or is a continuation of an existing Session.
586 Refer to section 4.1 for a definition of the Session State.

587

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

591

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

596

597 3.1.2.5 Will Flag

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

599

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

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

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

614

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

619

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

625
626
627
628
629
630
631
632

633
634
635
636
637
638
639
640
641

642
643
644
645
646
647
648
649
650
651

652
653
654
655
656
657

658
659
660
661
662
663
664
665
666
667

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.

668 **3.1.2.10 Keep Alive**

669 *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							

670
671 The Keep Alive is a Two Byte Integer which is a time interval measured in seconds. It is the maximum
672 time interval that is permitted to elapse between the point at which the Client finishes transmitting one
673 MQTT Control Packet and the point it starts sending the next. It is the responsibility of the Client to ensure
674 that the interval between MQTT Control Packets being sent does not exceed the Keep Alive value. **If**
675 **Keep Alive is non-zero and in the absence of sending any other MQTT Control Packets, the Client MUST**
676 **send a PINGREQ packet [MQTT-3.1.2-20].**

677
678 **If the Server returns a Server Keep Alive on the CONNACK packet, the Client MUST use that value**
679 **instead of the value it sent as the Keep Alive [MQTT-3.1.2-21].**

680
681 The Client can send PINGREQ at any time, irrespective of the Keep Alive value, and check for a
682 corresponding PINGRESP to determine that the network and the Server are available.

683
684 **If the Keep Alive value is non-zero and the Server does not receive an MQTT Control Packet from the**
685 **Client within one and a half times the Keep Alive time period, it MUST close the Network Connection to**
686 **the Client as if the network had failed [MQTT-3.1.2-22].**

687
688 If a Client does not receive a PINGRESP packet within a reasonable amount of time after it has sent a
689 PINGREQ, it SHOULD close the Network Connection to the Server.

690
691 A Keep Alive value of 0 has the effect of turning off the Keep Alive mechanism. If Keep Alive is 0 the
692 Client is not obliged to send MQTT Control Packets on any particular schedule.

693
694 **Non-normative comment**
695 The Server may have other reasons to disconnect the Client, for instance because it is shutting
696 down. Setting Keep Alive does not guarantee that the Client will remain connected.

697
698 **Non-normative comment**
699 The actual value of the Keep Alive is application specific; typically, this is a few minutes. The
700 maximum value of 65,535 is 18 hours 12 minutes and 15 seconds.

701
702 **3.1.2.11 CONNECT Properties**

703 **3.1.2.11.1 Property Length**

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

707 3.1.2.11.2 Session Expiry Interval

708 **17 (0x11) Byte**, Identifier of the Session Expiry Interval.

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

711
712 If the Session Expiry Interval is absent the value 0 is used. If it is set to 0, or is absent, the Session ends
713 when the Network Connection is closed.

714
715 If the Session Expiry Interval is 0xFFFFFFFF (UINT_MAX), the Session does not expire.

716
717 The Client and Server MUST store the Session State after the Network Connection is closed if the
718 Session Expiry Interval is greater than 0 [MQTT-3.1.2-23].

719
720 **Non-normative comment**

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

724
725 Refer to [section 4.1](#) for more information about Sessions. Refer to [section 4.1.1](#) for details and limitations
726 of stored state.

727
728 When the Session expires the Client and Server need not process the deletion of state atomically.

729
730 **Non-normative comment**

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

734
735 **Non-normative comment**

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

739
740 **Non-normative comment**

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

745
746 **Non-normative comment**

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

752
753 **Non-normative comment**

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

756

757 **Non-normative comment**

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

762

763 **3.1.2.11.3 Receive Maximum**

764 **33 (0x21) Byte**, Identifier of the Receive Maximum.

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

767

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

770

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

773

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

775

776 **3.1.2.11.4 Maximum Packet Size**

777 **39 (0x27) Byte**, Identifier of the Maximum Packet Size.

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

781

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

784

785 **Non-normative comment**

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

788

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

792

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

796

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

799

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

803

804 **Non-normative comment**

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

808

809 **3.1.2.11.5 Topic Alias Maximum**

810 **34 (0x22) Byte**, Identifier of the Topic Alias Maximum.

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

814

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

821

822 **3.1.2.11.6 Request Response Information**

823 **25 (0x19) Byte**, Identifier of the Request Response Information.

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

827

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

831

832 **Non-normative comment**

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

835

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

837

838 **3.1.2.11.7 Request Problem Information**

839 **23 (0x17) Byte**, Identifier of the Request Problem Information.

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

843

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

846

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

853

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

856

857 **3.1.2.11.8 User Property**

858 **38 (0x26) Byte**, Identifier of the User Property.

859 Followed by a UTF-8 String Pair.

860

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

863

864 **Non-normative comment**

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

867

868 **3.1.2.11.9 Authentication Method**

869 **21 (0x15) Byte**, Identifier of the Authentication Method.

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

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

873

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

876

877 **3.1.2.11.10 Authentication Data**

878 **22 (0x16) Byte**, Identifier of the Authentication Data.

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

882

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

885

886 **3.1.2.12 Variable Header non-normative example**

887 *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)								
	Password Flag (1)								
	Will Retain (0)								
	Will QoS (01)	1	1	0	0	1	1	1	0
	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

888

889 3.1.3 CONNECT Payload

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

893

894 3.1.3.1 Client Identifier (ClientID)

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

899

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

901

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

903

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

906 "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ" [MQTT-3.1.3-5].

907

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

910

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

915

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

919

920 **Non-normative comment**

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

923

924 3.1.3.2 Will Properties

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

929

930 3.1.3.2.1 Property Length

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

932

933 **3.1.3.2.2 Will Delay Interval**

934 **24 (0x18) Byte**, Identifier of the Will Delay Interval.

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

938

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

942

943 **Non-normative comment**

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

947

948 **Non-normative comment**

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

954

955 **3.1.3.2.3 Payload Format Indicator**

956 **1 (0x01) Byte**, Identifier of the Payload Format Indicator.

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

- 958 • 0 (0x00) Byte Indicates that the Will Message is unspecified bytes, which is equivalent to not
959 sending a Payload Format Indicator.
- 960 • 1 (0x01) Byte Indicates that the Will Message is UTF-8 Encoded Character Data. The UTF-8 data
961 in the Payload MUST be well-formed UTF-8 as defined by the Unicode specification
962 [Unicode] and restated in RFC 3629 [RFC3629].
963

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

967

968 **3.1.3.2.4 Message Expiry Interval**

969 **2 (0x02) Byte**, Identifier of the Message Expiry Interval.

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

972

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

975

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

977

978 **3.1.3.2.5 Content Type**

979 **3 (0x03)** Identifier of the Content Type.

980 Followed by a UTF-8 Encoded String describing the content of the Will Message. It is a Protocol Error to
981 include the Content Type more than once. The value of the Content Type is defined by the sending and
982 receiving application.
983

984 **3.1.3.2.6 Response Topic**

985 **8 (0x08) Byte**, Identifier of the Response Topic.

986 Followed by a UTF-8 Encoded String which is used as the Topic Name for a response message. It is a
987 Protocol Error to include the Response Topic more than once. The presence of a Response Topic
988 identifies the Will Message as a Request.
989

990 Refer to [section 4.10](#) for more information about Request / Response.
991

992 **3.1.3.2.7 Correlation Data**

993 **9 (0x09) Byte**, Identifier of the Correlation Data.

994 Followed by Binary Data. The Correlation Data is used by the sender of the Request Message to identify
995 which request the Response Message is for when it is received. It is a Protocol Error to include
996 Correlation Data more than once. If the Correlation Data is not present, the Requester does not require
997 any correlation data.
998

999 The value of the Correlation Data only has meaning to the sender of the Request Message and receiver
1000 of the Response Message.
1001

1002 Refer to [section 4.10](#) for more information about Request / Response
1003

1004 **3.1.3.2.8 User Property**

1005 **38 (0x26) Byte**, Identifier of the User Property.

1006 Followed by a UTF-8 String Pair. The User Property is allowed to appear multiple times to represent
1007 multiple name, value pairs. The same name is allowed to appear more than once.
1008

1009 **The Server MUST maintain the order of User Properties when publishing the Will Message [MQTT-3.1.3-**
1010 **10].**

1011

1012 **Non-normative comment**

1013 This property is intended to provide a means of transferring application layer name-value tags
1014 whose meaning and interpretation are known only by the application programs responsible for
1015 sending and receiving them.
1016

1017 **3.1.3.3 Will Topic**

1018 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**
1019 **Encoded String** as defined in [section 1.5.4 \[MQTT-3.1.3-11\]](#).
1020

1021 3.1.3.4 Will Payload

1022 If the Will Flag is set to 1 the Will Payload is the next field in the Payload. The Will Payload defines the
1023 Application Message Payload that is to be published to the Will Topic as described in [section 3.1.2.5](#). This
1024 field consists of Binary Data.

1025

1026 3.1.3.5 User Name

1027 If the User Name Flag is set to 1, the User Name is the next field in the Payload. The User Name MUST
1028 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
1029 authentication and authorization.

1030

1031 3.1.3.6 Password

1032 If the Password Flag is set to 1, the Password is the next field in the Payload. The Password field is
1033 Binary Data. Although this field is called Password, it can be used to carry any credential information.

1034

1035 3.1.4 CONNECT Actions

1036 Note that a Server MAY support multiple protocols (including other versions of the MQTT protocol) on the
1037 same TCP port or other network endpoint. If the Server determines that the protocol is MQTT v5.0 then it
1038 validates the connection attempt as follows.

1039

- 1040 1. If the Server does not receive a CONNECT packet within a reasonable amount of time after the
1041 Network Connection is established, the Server SHOULD close the Network Connection.
- 1042 2. The Server MUST validate that the CONNECT packet matches the format described in [section](#)
1043 [3.1](#) and close the Network Connection if it does not match [\[MQTT-3.1.4-1\]](#). The Server MAY send
1044 a CONNACK with a Reason Code of 0x80 or greater as described in [section 4.13](#) before closing
1045 the Network Connection.
- 1046 3. The Server MAY check that the contents of the CONNECT packet meet any further restrictions and
1047 SHOULD perform authentication and authorization checks. If any of these checks fail, it MUST
1048 close the Network Connection [\[MQTT-3.1.4-2\]](#). Before closing the Network Connection, it MAY
1049 send an appropriate CONNACK response with a Reason Code of 0x80 or greater as described in
1050 [section 3.2](#) and [section 4.13](#).

1051

1052 If validation is successful, the Server performs the following steps.

1053

- 1054 1. If the ClientID represents a Client already connected to the Server, the Server sends a
1055 DISCONNECT packet to the existing Client with Reason Code of 0x8E (Session taken over) as
1056 described in [section 4.13](#) and MUST close the Network Connection of the existing Client [\[MQTT-](#)
1057 [3.1.4-3\]](#). If the existing Client has a Will Message, that Will Message is published as described in
1058 [section 3.1.2.5](#).

1059

1060 Non-normative comment

1061 If the Will Delay Interval of the existing Network Connection is 0 and there is a Will Message, it
1062 will be sent because the Network Connection is closed. If the Session Expiry Interval of the
1063 existing Network Connection is 0, or the new Network Connection has Clean Start set to 1 then if
1064 the existing Network Connection has a Will Message it will be sent because the original Session
1065 is ended on the takeover.

1066

1067 2. The Server MUST perform the processing of Clean Start that is described in section 3.1.2.4
1068 [MQTT-3.1.4-4].

1069
1070 3. The Server MUST acknowledge the CONNECT packet with a CONNACK packet containing a
1071 0x00 (Success) Reason Code [MQTT-3.1.4-5].
1072

1073 **Non-normative comment**

1074 It is recommended that authentication and authorization checks be performed if the Server is
1075 being used to process any form of business critical data. If these checks succeed, the Server
1076 responds by sending CONNACK with a 0x00 (Success) Reason Code. If they fail, it is suggested
1077 that the Server does not to send a CONNACK at all, as this could alert a potential attacker to the
1078 presence of the MQTT Server and encourage such an attacker to launch a denial of service or
1079 password-guessing attack.

1080
1081 4. Start message delivery and Keep Alive monitoring.
1082

1083 Clients are allowed to send further MQTT Control Packets immediately after sending a CONNECT
1084 packet; Clients need not wait for a CONNACK packet to arrive from the Server. If the Server rejects the
1085 CONNECT, it MUST NOT process any data sent by the Client after the CONNECT packet except AUTH
1086 packets [MQTT-3.1.4-6].
1087

1088 **Non-normative comment**

1089 Clients typically wait for a CONNACK packet, However, if the Client exploits its freedom to send
1090 MQTT Control Packets before it receives a CONNACK, it might simplify the Client implementation
1091 as it does not have to police the connected state. The Client accepts that any data that it sends
1092 before it receives a CONNACK packet from the Server will not be processed if the Server rejects
1093 the connection.

1094
1095 **Non-normative comment**

1096 Clients that send MQTT Control Packets before they receive CONNACK will be unaware of the
1097 Server constraints and whether any existing Session is being used.

1098
1099 **Non-normative comment**

1100 The Server can limit reading from the Network Connection or close the Network Connection if the
1101 Client sends too much data before authentication is complete. This is suggested as a way of
1102 avoiding denial of service attacks.
1103

1104 3.2 CONNACK – Connect acknowledgement

1105 The CONNACK packet is the packet sent by the Server in response to a CONNECT packet received from
1106 a Client. The Server MUST send a CONNACK with a 0x00 (Success) Reason Code before sending any
1107 Packet other than AUTH [MQTT-3.2.0-1]. The Server MUST NOT send more than one CONNACK in a
1108 Network Connection [MQTT-3.2.0-2].
1109

1110 If the Client does not receive a CONNACK packet from the Server within a reasonable amount of time, the
1111 Client SHOULD close the Network Connection. A "reasonable" amount of time depends on the type of
1112 application and the communications infrastructure.
1113

1114 **3.2.1 CONNACK Fixed Header**

1115 The Fixed Header format is illustrated in Figure 3-7.

1116 *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							

1117

1118 **Remaining Length field**

1119 This is the length of the Variable Header encoded as a Variable Byte Integer.

1120

1121 **3.2.2 CONNACK Variable Header**

1122 The Variable Header of the CONNACK Packet contains the following fields in the order: Connect
1123 Acknowledge Flags, Connect Reason Code, and Properties. The rules for encoding Properties are
1124 described in [section 2.2.2](#).

1125

1126 **3.2.2.1 Connect Acknowledge Flags**

1127 **Byte 1 is the "Connect Acknowledge Flags". Bits 7-1 are reserved and MUST be set to 0 [MQTT-3.2.2-1].**

1128

1129 Bit 0 is the Session Present Flag.

1130

1131 **3.2.2.1.1 Session Present**

1132 Position: bit 0 of the Connect Acknowledge Flags.

1133

1134 The Session Present flag informs the Client whether the Server is using Session State from a previous
1135 connection for this ClientID. This allows the Client and Server to have a consistent view of the Session
1136 State.

1137

1138 **If the Server accepts a connection with Clean Start set to 1, the Server MUST set Session Present to 0 in**
1139 **the CONNACK packet in addition to setting a 0x00 (Success) Reason Code in the CONNACK packet**
1140 **[MQTT-3.2.2-2].**

1141

1142 **If the Server accepts a connection with Clean Start set to 0 and the Server has Session State for the**
1143 **ClientID, it MUST set Session Present to 1 in the CONNACK packet, otherwise it MUST set Session**
1144 **Present to 0 in the CONNACK packet. In both cases it MUST set a 0x00 (Success) Reason Code in the**
1145 **CONNACK packet [MQTT-3.2.2-3].**

1146

1147 If the value of Session Present received by the Client from the Server is not as expected, the Client
1148 proceeds as follows:

- 1149 • **If the Client does not have Session State and receives Session Present set to 1 it MUST close**
1150 **the Network Connection [MQTT-3.2.2-4].** If it wishes to restart with a new Session the Client can
1151 reconnect using Clean Start set to 1.

1152 • If the Client does have Session State and receives Session Present set to 0 it MUST discard its
1153 Session State if it continues with the Network Connection [MQTT-3.2.2-5].

1154
1155
1156 If a Server sends a CONNACK packet containing a non-zero Reason Code it MUST set Session Present
1157 to 0 [MQTT-3.2.2-6].
1158

1159 3.2.2.2 Connect Reason Code

1160 Byte 2 in the Variable Header is the Connect Reason Code.

1161
1162 The values the Connect Reason Code are shown below. If a well formed CONNECT packet is received
1163 by the Server, but the Server is unable to complete the Connection the Server MAY send a CONNACK
1164 packet containing the appropriate Connect Reason code from this table. If a Server sends a CONNACK
1165 packet containing a Reason code of 128 or greater it MUST then close the Network Connection [MQTT-
1166 3.2.2-7].
1167

1168 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.

1169

1170 The Server sending the CONNACK packet MUST use one of the Connect Reason Code values [T-3.2.2-
1171 8].

1172

1173 **Non-normative comment**

1174 Reason Code 0x80 (Unspecified error) may be used where the Server knows the reason for the
1175 failure but does not wish to reveal it to the Client, or when none of the other Reason Code values
1176 applies.

1177

1178 The Server may choose to close the Network Connection without sending a CONNACK to
1179 enhance security in the case where an error is found on the CONNECT. For instance, when on a
1180 public network and the connection has not been authorized it might be unwise to indicate that this
1181 is an MQTT Server.

1182

1183 **3.2.2.3 CONNACK Properties**

1184 **3.2.2.3.1 Property Length**

1185 This is the length of the Properties in the CONNACK packet Variable Header encoded as a Variable Byte
1186 Integer.

1187

1188 **3.2.2.3.2 Session Expiry Interval**

1189 **17 (0x11) Byte**, Identifier of the Session Expiry Interval.

1190 Followed by the Four Byte Integer representing the Session Expiry Interval in seconds. It is a Protocol
1191 Error to include the Session Expiry Interval more than once.

1192

1193 If the Session Expiry Interval is absent the value in the CONNECT Packet used. The server uses this
1194 property to inform the Client that it is using a value other than that sent by the Client in the CONNACK.
1195 Refer to section 3.1.2.11.2 for a description of the use of Session Expiry Interval.

1196

1197 **3.2.2.3.3 Receive Maximum**

1198 **33 (0x21) Byte**, Identifier of the Receive Maximum.

1199 Followed by the Two Byte Integer representing the Receive Maximum value. It is a Protocol Error to
1200 include the Receive Maximum value more than once or for it to have the value 0.

1201

1202 The Server uses this value to limit the number of QoS 1 and QoS 2 publications that it is willing to
1203 process concurrently for the Client. It does not provide a mechanism to limit the QoS 0 publications that
1204 the Client might try to send.

1205

1206 If the Receive Maximum value is absent, then its value defaults to 65,535.

1207

1208 Refer to [section 4.9](#) Flow Control for details of how the Receive Maximum is used.

1209

1210 **3.2.2.3.4 Maximum QoS**

1211 **36 (0x24) Byte**, Identifier of the Maximum QoS.

1212 Followed by a Byte with a value of either 0 or 1. It is a Protocol Error to include Maximum QoS more than
1213 once, or to have a value other than 0 or 1. If the Maximum QoS is absent, the Client uses a Maximum
1214 QoS of 2.

1215

1216 If a Server does not support QoS 1 or QoS 2 PUBLISH packets it MUST send a Maximum QoS in the
1217 CONNACK packet specifying the highest QoS it supports [\[MQTT-3.2.2-9\]](#). A Server that does not support
1218 QoS 1 or QoS 2 PUBLISH packets MUST still accept SUBSCRIBE packets containing a Requested QoS
1219 of 0, 1 or 2 [\[MQTT-3.2.2-10\]](#).

1220

1221 If a Client receives a Maximum QoS from a Server, it MUST NOT send PUBLISH packets at a QoS level
1222 exceeding the Maximum QoS level specified [\[MQTT-3.2.2-11\]](#). It is a Protocol Error if the Server receives
1223 a PUBLISH packet with a QoS greater than the Maximum QoS it specified. In this case use
1224 DISCONNECT with Reason Code 0x9B (QoS not supported) as described in [section 4.13](#) Handling
1225 errors.

1226

1227 If a Server receives a CONNECT packet containing a Will QoS that exceeds its capabilities, it MUST
1228 reject the connection. It SHOULD use a CONNACK packet with Reason Code 0x9B (QoS not supported)
1229 as described in [section 4.13](#) Handling errors, and MUST close the Network Connection [\[MQTT-3.2.2-12\]](#).

1230

1231 **Non-normative comment**

1232 A Client does not need to support QoS 1 or QoS 2 PUBLISH packets. If this is the case, the
1233 Client simply restricts the maximum QoS field in any SUBSCRIBE commands it sends to a value
1234 it can support.

1235

1236 **3.2.2.3.5 Retain Available**

1237 **37 (0x25) Byte**, Identifier of Retain Available.

1238 Followed by a Byte field. If present, this byte declares whether the Server supports retained messages. A
1239 value of 0 means that retained messages are not supported. A value of 1 means retained messages are
1240 supported. If not present, then retained messages are supported. It is a Protocol Error to include Retain
1241 Available more than once or to use a value other than 0 or 1.

1242

1243 If a Server receives a CONNECT packet containing a Will Message with the Will Retain set to 1, and it
1244 does not support retained messages, the Server MUST reject the connection request. It SHOULD send
1245 CONNACK with Reason Code 0x9A (Retain not supported) and then it MUST close the Network
1246 Connection [MQTT-3.2.2-13].

1247

1248 A Client receiving Retain Available from the Server MUST NOT send a PUBLISH packet with the RETAIN
1249 flag set to 1 [MQTT-3.2.2-14]. If the Server receives such a packet, this is a Protocol Error. The Server
1250 SHOULD send a DISCONNECT with Reason Code of 0x9A (Retain not supported) as described in
1251 section 4.13.

1252

1253 3.2.2.3.6 Maximum Packet Size

1254 **39 (0x27) Byte**, Identifier of the Maximum Packet Size.

1255 Followed by a Four Byte Integer representing the Maximum Packet Size the Server is willing to accept. If
1256 the Maximum Packet Size is not present, there is no limit on the packet size imposed beyond the
1257 limitations in the protocol as a result of the remaining length encoding and the protocol header sizes.

1258

1259 It is a Protocol Error to include the Maximum Packet Size more than once, or for the value to be set to
1260 zero.

1261

1262 The packet size is the total number of bytes in an MQTT Control Packet, as defined in section 2.1.4. The
1263 Server uses the Maximum Packet Size to inform the Client that it will not process packets whose size
1264 exceeds this limit.

1265

1266 The Client MUST NOT send packets exceeding Maximum Packet Size to the Server [MQTT-3.2.2-15]. If
1267 a Server receives a packet whose size exceeds this limit, this is a Protocol Error, the Server uses
1268 DISCONNECT with Reason Code 0x95 (Packet too large), as described in section 4.13.

1269

1270 3.2.2.3.7 Assigned Client Identifier

1271 **18 (0x12) Byte**, Identifier of the Assigned Client Identifier.

1272 Followed by the UTF-8 string which is the Assigned Client Identifier. It is a Protocol Error to include the
1273 Assigned Client Identifier more than once.

1274

1275 The Client Identifier which was assigned by the Server because a zero length Client Identifier was found
1276 in the CONNECT packet.

1277

1278 If the Client connects using a zero length Client Identifier, the Server MUST respond with a CONNACK
1279 containing an Assigned Client Identifier. The Assigned Client Identifier MUST be a new Client Identifier
1280 not used by any other Session currently in the Server [MQTT-3.2.2-16].

1281

1282 3.2.2.3.8 Topic Alias Maximum

1283 **34 (0x22) Byte**, Identifier of the Topic Alias Maximum.

1284 Followed by the Two Byte Integer representing the Topic Alias Maximum value. It is a Protocol Error to
1285 include the Topic Alias Maximum value more than once. If the Topic Alias Maximum property is absent,
1286 the default value is 0.

1287
1288 This value indicates the highest value that the Server will accept as a Topic Alias sent by the Client. The
1289 Server uses this value to limit the number of Topic Aliases that it is willing to hold on this Connection. **The**
1290 **Client MUST NOT send a Topic Alias in a PUBLISH packet to the Server greater than this value [MQTT-**
1291 **3.2.2-17]. A value of 0 indicates that the Server does not accept any Topic Aliases on this connection. If**
1292 **Topic Alias Maximum is absent or 0, the Client MUST NOT send any Topic Aliases on to the Server**
1293 **[MQTT-3.2.2-18].**
1294

1295 **3.2.2.3.9 Reason String**

1296 **31 (0x1F) Byte** Identifier of the Reason String.

1297 Followed by the UTF-8 Encoded String representing the reason associated with this response. This
1298 Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the
1299 Client.

1300

1301 The Server uses this value to give additional information to the Client. **The Server MUST NOT send this**
1302 **property if it would increase the size of the CONNACK packet beyond the Maximum Packet Size specified**
1303 **by the Client [MQTT-3.2.2-19].** It is a Protocol Error to include the Reason String more than once.

1304

1305 **Non-normative comment**

1306 Proper uses for the reason string in the Client would include using this information in an exception
1307 thrown by the Client code, or writing this string to a log.

1308

1309 **3.2.2.3.10 User Property**

1310 **38 (0x26) Byte**, Identifier of User Property.

1311 Followed by a UTF-8 String Pair. This property can be used to provide additional information to the Client
1312 including diagnostic information. **The Server MUST NOT send this property if it would increase the size of**
1313 **the CONNACK packet beyond the Maximum Packet Size specified by the Client [MQTT-3.2.2-20].** The
1314 User Property is allowed to appear multiple times to represent multiple name, value pairs. The same
1315 name is allowed to appear more than once.

1316

1317 The content and meaning of this property is not defined by this specification. The receiver of a CONNACK
1318 containing this property MAY ignore it.

1319

1320 **3.2.2.3.11 Wildcard Subscription Available**

1321 **40 (0x28) Byte**, Identifier of Wildcard Subscription Available.

1322 Followed by a Byte field. If present, this byte declares whether the Server supports Wildcard
1323 Subscriptions. A value is 0 means that Wildcard Subscriptions are not supported. A value of 1 means
1324 Wildcard Subscriptions are supported. If not present, then Wildcard Subscriptions are supported. It is a
1325 Protocol Error to include the Wildcard Subscription Available more than once or to send a value other
1326 than 0 or 1.

1327

1328 If the Server receives a SUBSCRIBE packet containing a Wildcard Subscription and it does not support
1329 Wildcard Subscriptions, this is a Protocol Error. The Server uses DISCONNECT with Reason Code 0xA2
1330 (Wildcard subscription not supported) as described in [section 4.13](#).

1331

1332 If a Server supports Wildcard Subscriptions, it can still reject a particular subscribe request containing a
1333 Wildcard Subscription. In this case the Server MAY send a SUBACK Control Packet with a Reason Code
1334 0xA2 (Wildcard Subscriptions not supported).
1335

1336 **3.2.2.3.12 Subscription Identifiers Available**

1337 **41 (0x29) Byte**, Identifier of Subscription Identifier Available.

1338 Followed by a Byte field. If present, this byte declares whether the Server supports Subscription
1339 Identifiers. A value is 0 means that Subscription Identifiers are not supported. A value of 1 means
1340 Subscription Identifiers are supported. If not present, then Subscription Identifiers are supported. It is a
1341 Protocol Error to include the Subscription Identifier Available more than once, or to send a value other
1342 than 0 or 1.

1343

1344 If the Server receives a SUBSCRIBE packet containing Subscription Identifier and it does not support
1345 Subscription Identifiers, this is a Protocol Error. The Server uses DISCONNECT with Reason Code of
1346 0xA1 (Subscription Identifiers not supported) as described in [section 4.13](#).

1347

1348 **3.2.2.3.13 Shared Subscription Available**

1349 **42 (0x2A) Byte**, Identifier of Shared Subscription Available.

1350 Followed by a Byte field. If present, this byte declares whether the Server supports Shared Subscriptions.
1351 A value is 0 means that Shared Subscriptions are not supported. A value of 1 means Shared
1352 Subscriptions are supported. If not present, then Shared Subscriptions are supported. It is a Protocol
1353 Error to include the Shared Subscription Available more than once or to send a value other than 0 or 1.

1354

1355 If the Server receives a SUBSCRIBE packet containing Shared Subscriptions and it does not support
1356 Shared Subscriptions, this is a Protocol Error. The Server uses DISCONNECT with Reason Code 0x9E
1357 (Shared Subscription not supported) as described in [section 4.13](#).

1358

1359 **3.2.2.3.14 Server Keep Alive**

1360 **19 (0x13) Byte**, Identifier of the Server Keep Alive.

1361 Followed by a Two Byte Integer with the Keep Alive time assigned by the Server. If the Server sends a
1362 Server Keep Alive on the CONNACK packet, the Client MUST use this value instead of the Keep Alive
1363 value the Client sent on CONNECT [MQTT-3.2.2-21]. If the Server does not send the Server Keep Alive,
1364 the Server MUST use the Keep Alive value set by the Client on CONNECT [MQTT-3.2.2-22]. It is a
1365 Protocol Error to include the Server Keep Alive more than once.

1366

1367 **Non-normative comment**

1368 The primary use of the Server Keep Alive is for the Server to inform the Client that it will
1369 disconnect the Client for inactivity sooner than the Keep Alive specified by the Client.

1370

1371 **3.2.2.3.15 Response Information**

1372 **26 (0x1A) Byte**, Identifier of the Response Information.

1373 Followed by a UTF-8 Encoded String which is used as the basis for creating a Response Topic. The way
1374 in which the Client creates a Response Topic from the Response Information is not defined by this
1375 specification. It is a Protocol Error to include the Response Information more than once.

1376
1377 If the Client sends a Request Response Information with a value 1, it is OPTIONAL for the Server to send
1378 the Response Information in the CONNACK.

1379
1380 **Non-normative comment**
1381 A common use of this is to pass a globally unique portion of the topic tree which is reserved for
1382 this Client for at least the lifetime of its Session. This often cannot just be a random name as both
1383 the requesting Client and the responding Client need to be authorized to use it. It is normal to use
1384 this as the root of a topic tree for a particular Client. For the Server to return this information, it
1385 normally needs to be correctly configured. Using this mechanism allows this configuration to be
1386 done once in the Server rather than in each Client.

1387
1388 Refer to [section 4.10](#) for more information about Request / Response.
1389

1390 **3.2.2.3.16 Server Reference**

1391 **28 (0x1C) Byte**, Identifier of the Server Reference.
1392 Followed by a UTF-8 Encoded String which can be used by the Client to identify another Server to use. It
1393 is a Protocol Error to include the Server Reference more than once.

1394
1395 The Server uses a Server Reference in either a CONNACK or DISCONNECT packet with Reason code
1396 of 0x9C (Use another server) or Reason Code 0x9D (Server moved) as described in [section 4.13](#).

1397
1398 Refer to [section 4.11](#) Server redirection for information about how Server Reference is used.
1399

1400 **3.2.2.3.17 Authentication Method**

1401 **21 (0x15) Byte**, Identifier of the Authentication Method.
1402 Followed by a UTF-8 Encoded String containing the name of the authentication method. It is a Protocol
1403 Error to include the Authentication Method more than once. Refer to [section 4.12](#) for more information
1404 about extended authentication.

1405

1406 **3.2.2.3.18 Authentication Data**

1407 **22 (0x16) Byte**, Identifier of the Authentication Data.
1408 Followed by Binary Data containing authentication data. The contents of this data are defined by the
1409 authentication method and the state of already exchanged authentication data. It is a Protocol Error to
1410 include the Authentication Data more than once. Refer to [section 4.12](#) for more information about
1411 extended authentication.

1412

1413 **3.2.3 CONNACK Payload**

1414 The CONNACK packet has no Payload.

1415

1416 **3.3 PUBLISH – Publish message**

1417 A PUBLISH packet is sent from a Client to a Server or from a Server to a Client to transport an
1418 Application Message.

1419

1420 **3.3.1 PUBLISH Fixed Header**

1421 *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							

1422

1423 **3.3.1.1 DUP**

1424 **Position:** byte 1, bit 3.

1425 If the DUP flag is set to 0, it indicates that this is the first occasion that the Client or Server has attempted
1426 to send this PUBLISH packet. If the DUP flag is set to 1, it indicates that this might be re-delivery of an
1427 earlier attempt to send the packet.

1428

1429 The DUP flag MUST be set to 1 by the Client or Server when it attempts to re-deliver a PUBLISH packet
1430 [MQTT-3.3.1-1]. The DUP flag MUST be set to 0 for all QoS 0 messages [MQTT-3.3.1-2].

1431

1432 The value of the DUP flag from an incoming PUBLISH packet is not propagated when the PUBLISH
1433 packet is sent to subscribers by the Server. The DUP flag in the outgoing PUBLISH packet is set
1434 independently to the incoming PUBLISH packet, its value MUST be determined solely by whether the
1435 outgoing PUBLISH packet is a retransmission [MQTT-3.3.1-3].

1436

1437 **Non-normative comment**

1438 The receiver of an MQTT Control Packet that contains the DUP flag set to 1 cannot assume that
1439 it has seen an earlier copy of this packet.

1440

1441 **Non-normative comment**

1442 It is important to note that the DUP flag refers to the MQTT Control Packet itself and not to the
1443 Application Message that it contains. When using QoS 1, it is possible for a Client to receive a
1444 PUBLISH packet with DUP flag set to 0 that contains a repetition of an Application Message that
1445 it received earlier, but with a different Packet Identifier. Section 2.2.1 provides more information
1446 about Packet Identifiers.

1447

1448 **3.3.1.2 QoS**

1449 **Position:** byte 1, bits 2-1.

1450 This field indicates the level of assurance for delivery of an Application Message. The QoS levels are
1451 shown below.

1452

1453 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

1454

1455 If the Server included a Maximum QoS in its CONNACK response to a Client and it receives a PUBLISH
 1456 packet with a QoS greater than this, then it uses DISCONNECT with Reason Code 0x9B (QoS not
 1457 supported) as described in [section 4.13](#) Handling errors.

1458

1459 **A PUBLISH Packet MUST NOT have both QoS bits set to 1 [MQTT-3.3.1-4].** If a Server or Client receives
 1460 a PUBLISH packet which has both QoS bits set to 1 it is a Malformed Packet. Use DISCONNECT with
 1461 Reason Code 0x81 (Malformed Packet) as described in [section 4.13](#).

1462

1463 3.3.1.3 RETAIN

1464 **Position:** byte 1, bit 0.

1465

1466 **If the RETAIN flag is set to 1 in a PUBLISH packet sent by a Client to a Server, the Server MUST replace**
 1467 **any existing retained message for this topic and store the Application Message [MQTT-3.3.1-5], so that it**
 1468 **can be delivered to future subscribers whose subscriptions match its Topic Name. If the Payload contains**
 1469 **zero bytes it is processed normally by the Server but any retained message with the same topic name**
 1470 **MUST be removed and any future subscribers for the topic will not receive a retained message [MQTT-**
 1471 **3.3.1-6]. A retained message with a Payload containing zero bytes MUST NOT be stored as a retained**
 1472 **message on the Server [MQTT-3.3.1-7].**

1473

1474 **If the RETAIN flag is 0 in a PUBLISH packet sent by a Client to a Server, the Server MUST NOT store the**
 1475 **message as a retained message and MUST NOT remove or replace any existing retained message**
 1476 **[MQTT-3.3.1-8].**

1477

1478 If the Server included Retain Available in its CONNACK response to a Client with its value set to 0 and it
 1479 receives a PUBLISH packet with the RETAIN flag is set to 1, then it uses the DISCONNECT Reason
 1480 Code of 0x9A (Retain not supported) as described in [section 4.13](#).

1481

1482 When a new Non-shared Subscription is made, the last retained message, if any, on each matching topic
 1483 name is sent to the Client as directed by the Retain Handling Subscription Option. These messages are
 1484 sent with the RETAIN flag set to 1. Which retained messages are sent is controlled by the Retain
 1485 Handling Subscription Option. At the time of the Subscription:

- 1486 • **If Retain Handling is set to 0 the Server MUST send the retained messages matching the Topic**
 1487 **Filter of the subscription to the Client [MQTT-3.3.1-9].**
- 1488 • **If Retain Handling is set to 1 then if the subscription did not already exist, the Server MUST send**
 1489 **all retained message matching the Topic Filter of the subscription to the Client, and if the**
 1490 **subscription did exist the Server MUST NOT send the retained messages. [MQTT-3.3.1-10].**
- 1491 • **If Retain Handling is set to 2, the Server MUST NOT send the retained messages [MQTT-3.3.1-**
 1492 **11].**

1493

1494 Refer to [section 3.8.3.1](#) for a definition of the Subscription Options.

1495

1496 If the Server receives a PUBLISH packet with the RETAIN flag set to 1, and QoS 0 it SHOULD store the
1497 new QoS 0 message as the new retained message for that topic, but MAY choose to discard it at any
1498 time. If this happens there will be no retained message for that topic.

1499

1500 If the current retained message for a Topic expires, it is discarded and there will be no retained message
1501 for that topic.

1502

1503 The setting of the RETAIN flag in an Application Message forwarded by the Server from an established
1504 connection is controlled by the Retain As Published subscription option. Refer to [section 3.8.3.1](#) for a
1505 definition of the Subscription Options.

1506

- 1507 • If the value of Retain As Published subscription option is set to 0, the Server MUST set the RETAIN
1508 flag to 0 when forwarding an Application Message regardless of how the RETAIN flag was set in the
1509 received PUBLISH packet [\[MQTT-3.3.1-12\]](#).
- 1510 • If the value of Retain As Published subscription option is set to 1, the Server MUST set the RETAIN
1511 flag equal to the RETAIN flag in the received PUBLISH packet [\[MQTT-3.3.1-13\]](#).

1512

1513 **Non-normative comment**

1514 Retained messages are useful where publishers send state messages on an irregular basis. A new
1515 non-shared subscriber will receive the most recent state.

1516

1517 **3.3.1.4 Remaining Length**

1518 This is the length of Variable Header plus the length of the Payload, encoded as a Variable Byte Integer.

1519

1520 **3.3.2 PUBLISH Variable Header**

1521 The Variable Header of the PUBLISH Packet contains the following fields in the order: Topic Name,
1522 Packet Identifier, and Properties. The rules for encoding Properties are described in [section 2.2.2](#).

1523

1524 **3.3.2.1 Topic Name**

1525 The Topic Name identifies the information channel to which Payload data is published.

1526

1527 The Topic Name MUST be present as the first field in the PUBLISH packet Variable Header. It MUST be
1528 a UTF-8 Encoded String as defined in [section 1.5.4](#) [\[MQTT-3.3.2-1\]](#).

1529

1530 The Topic Name in the PUBLISH packet MUST NOT contain wildcard characters [\[MQTT-3.3.2-2\]](#).

1531

1532 The Topic Name in a PUBLISH packet sent by a Server to a subscribing Client MUST match the
1533 Subscription's Topic Filter according to the matching process defined in [section 4.7](#) [\[MQTT-3.3.2-3\]](#).

1534 However, as the Server is permitted to map the Topic Name to another name, it might not be the same as
1535 the Topic Name in the original PUBLISH packet.

1536

1537 To reduce the size of the PUBLISH packet the sender can use a Topic Alias. The Topic Alias is described
1538 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.
1539

1540 **3.3.2.2 Packet Identifier**

1541 The Packet Identifier field is only present in PUBLISH packets where the QoS level is 1 or 2. [Section](#)
1542 [2.2.1](#) provides more information about Packet Identifiers.
1543

1544 **3.3.2.3 PUBLISH Properties**

1545 **3.3.2.3.1 Property Length**

1546 The length of the Properties in the PUBLISH packet Variable Header encoded as a Variable Byte Integer.
1547

1548 **3.3.2.3.2 Payload Format Indicator**

1549 **1 (0x01) Byte**, Identifier of the Payload Format Indicator.

1550 Followed by the value of the Payload Format Indicator, either of:

- 1551 • 0 (0x00) Byte Indicates that the Payload is unspecified bytes, which is equivalent to not sending a
1552 Payload Format Indicator.
- 1553 • 1 (0x01) Byte Indicates that the Payload is UTF-8 Encoded Character Data. The UTF-8 data in
1554 the Payload MUST be well-formed UTF-8 as defined by the Unicode specification [[Unicode](#)]
1555 and restated in RFC 3629 [[RFC3629](#)].
1556

1557 A Server MUST send the Payload Format Indicator unaltered to all subscribers receiving the Application
1558 Message [[MQTT-3.3.2-4](#)]. The receiver MAY validate that the Payload is of the format indicated, and if it
1559 is not send a PUBACK, PUBREC, or DISCONNECT with Reason Code of 0x99 (Payload format invalid)
1560 as described in [section 4.13](#).
1561

1562 **3.3.2.3.3 Message Expiry Interval**

1563 **2 (0x02) Byte**, Identifier of the Message Expiry Interval.

1564 Followed by the Four Byte Integer representing the Message Expiry Interval.
1565

1566 If present, the Four Byte value is the lifetime of the Application Message in seconds. If the Message
1567 Expiry Interval has passed and the Server has not managed to start onward delivery to a matching
1568 subscriber, then it MUST delete the copy of the message for that subscriber [[MQTT-3.3.2-5](#)].
1569

1570 If absent, the Application Message does not expire.
1571

1572 The PUBLISH packet sent to a Client by the Server MUST contain a Message Expiry Interval set to the
1573 received value minus the time that the Application Message has been waiting in the Server [[MQTT-3.3.2-](#)
1574 [6](#)]. Refer to [section 4.1](#) for details and limitations of stored state.
1575

1576 **3.3.2.3.4 Topic Alias**

1577 **35 (0x23) Byte**, Identifier of the Topic Alias.

1578 Followed by the Two Byte integer representing the Topic Alias value. It is a Protocol Error to include the
1579 Topic Alias value more than once.

1580

1581 A Topic Alias is an integer value that is used to identify the Topic instead of using the Topic Name. This
1582 reduces the size of the PUBLISH packet, and is useful when the Topic Names are long and the same
1583 Topic Names are used repetitively within a Network Connection.

1584

1585 The sender decides whether to use a Topic Alias and chooses the value. It sets a Topic Alias mapping by
1586 including a non-zero length Topic Name and a Topic Alias in the PUBLISH packet. The receiver
1587 processes the PUBLISH as normal but also sets the specified Topic Alias mapping to this Topic Name.

1588

1589 If a Topic Alias mapping has been set at the receiver, a sender can send a PUBLISH packet that contains
1590 that Topic Alias and a zero length Topic Name. The receiver then treats the incoming PUBLISH as if it
1591 had contained the Topic Name of the Topic Alias.

1592

1593 A sender can modify the Topic Alias mapping by sending another PUBLISH in the same Network
1594 Connection with the same Topic Alias value and a different non-zero length Topic Name.

1595

1596 Topic Alias mappings exist only within a Network Connection and last only for the lifetime of that Network
1597 Connection. A receiver MUST NOT carry forward any Topic Alias mappings from one Network
1598 Connection to another [MQTT-3.3.2-7].

1599

1600 A Topic Alias of 0 is not permitted. A sender MUST NOT send a PUBLISH packet containing a Topic
1601 Alias which has the value 0 [MQTT-3.3.2-8].

1602

1603 A Client MUST NOT send a PUBLISH packet with a Topic Alias greater than the Topic Alias Maximum
1604 value returned by the Server in the CONNACK packet [MQTT-3.3.2-9]. A Client MUST accept all Topic
1605 Alias values greater than 0 and less than or equal to the Topic Alias Maximum value that it sent in the
1606 CONNECT packet [MQTT-3.3.2-10].

1607

1608 A Server MUST NOT send a PUBLISH packet with a Topic Alias greater than the Topic Alias Maximum
1609 value sent by the Client in the CONNECT packet [MQTT-3.3.2-11]. A Server MUST accept all Topic Alias
1610 values greater than 0 and less than or equal to the Topic Alias Maximum value that it returned in the
1611 CONNACK packet [MQTT-3.3.2-12].

1612

1613 The Topic Alias mappings used by the Client and Server are independent from each other. Thus, when a
1614 Client sends a PUBLISH containing a Topic Alias value of 1 to a Server and the Server sends a PUBLISH
1615 with a Topic Alias value of 1 to that Client they will in general be referring to different Topics.

1616

1617 3.3.2.3.5 Response Topic

1618 8 (0x08) Byte, Identifier of the Response Topic.

1619 Followed by a UTF-8 Encoded String which is used as the Topic Name for a response message. The
1620 Response Topic MUST be a UTF-8 Encoded String as defined in section 1.5.4 [MQTT-3.3.2-13]. The
1621 Response Topic MUST NOT contain wildcard characters [MQTT-3.3.2-14]. It is a Protocol Error to include
1622 the Response Topic more than once. The presence of a Response Topic identifies the Message as a
1623 Request.

1624

1625 Refer to section 4.10 for more information about Request / Response.

1626
1627 The Server MUST send the Response Topic unaltered to all subscribers receiving the Application
1628 Message [MQTT-3.3.2-15].

1629
1630 **Non-normative comment:**
1631 The receiver of an Application Message with a Response Topic sends a response by using the
1632 Response Topic as the Topic Name of a PUBLISH. If the Request Message contains a
1633 Correlation Data, the receiver of the Request Message should also include this Correlation Data
1634 as a property in the PUBLISH packet of the Response Message.
1635

1636 3.3.2.3.6 Correlation Data

1637 **9 (0x09) Byte**, Identifier of the Correlation Data.

1638 Followed by Binary Data. The Correlation Data is used by the sender of the Request Message to identify
1639 which request the Response Message is for when it is received. It is a Protocol Error to include
1640 Correlation Data more than once. If the Correlation Data is not present, the Requester does not require
1641 any correlation data.

1642
1643 The Server MUST send the Correlation Data unaltered to all subscribers receiving the Application
1644 Message [MQTT-3.3.2-16]. The value of the Correlation Data only has meaning to the sender of the
1645 Request Message and receiver of the Response Message.

1646
1647 **Non-normative comment**
1648 The receiver of an Application Message which contains both a Response Topic and a Correlation
1649 Data sends a response by using the Response Topic as the Topic Name of a PUBLISH. The
1650 Client should also send the Correlation Data unaltered as part of the PUBLISH of the responses.

1651
1652 **Non-normative comment**
1653 If the Correlation Data contains information which can cause application failures if modified by the
1654 Client responding to the request, it should be encrypted and/or hashed to allow any alteration to
1655 be detected.

1656
1657 Refer to [section 4.10](#) for more information about Request / Response
1658

1659 3.3.2.3.7 User Property

1660 **38 (0x26) Byte**, Identifier of the User Property.

1661 Followed by a UTF-8 String Pair. The User Property is allowed to appear multiple times to represent
1662 multiple name, value pairs. The same name is allowed to appear more than once.

1663
1664 The Server MUST send all User Properties unaltered in a PUBLISH packet when forwarding the
1665 Application Message to a Client [MQTT-3.3.2-17]. The Server MUST maintain the order of User
1666 Properties when forwarding the Application Message [MQTT-3.3.2-18].

1667
1668 **Non-normative comment**
1669 This property is intended to provide a means of transferring application layer name-value tags
1670 whose meaning and interpretation are known only by the application programs responsible for
1671 sending and receiving them.

1673 **3.3.2.3.8 Subscription Identifier**

1674 **11 (0x0B)**, Identifier of the Subscription Identifier.

1675 Followed by a Variable Byte Integer representing the identifier of the subscription.

1676
1677 The Subscription Identifier can have the value of 1 to 268,435,455. It is a Protocol Error if the
1678 Subscription Identifier has a value of 0. Multiple Subscription Identifiers will be included if the publication
1679 is the result of a match to more than one subscription, in this case their order is not significant.
1680

1681 **3.3.2.3.9 Content Type**

1682 **3 (0x03)** Identifier of the Content Type.

1683 Followed by a UTF-8 Encoded String describing the content of the Application Message. **The Content**
1684 **Type MUST be a UTF-8 Encoded String** as defined in [section 1.5.4 \[MQTT-3.3.2-19\]](#).

1685 It is a Protocol Error to include the Content Type more than once. The value of the Content Type is
1686 defined by the sending and receiving application.
1687

1688 **A Server MUST send the Content Type unaltered to all subscribers receiving the Application Message**
1689 **[MQTT-3.3.2-20]**.

1690

1691 **Non-normative comment**

1692 The UTF-8 Encoded String may use a MIME content type string to describe the contents of the
1693 Application message. However, since the sending and receiving applications are responsible for
1694 the definition and interpretation of the string, MQTT performs no validation of the string except to
1695 insure it is a valid UTF-8 Encoded String.
1696

1697 **Non-normative example**

1698 Figure 3-9 shows an example of a PUBLISH packet with the Topic Name set to “a/b”, the Packet
1699 Identifier set to 10, and having no properties.
1700

1700

1701 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

1702

1703 **3.3.3 PUBLISH Payload**

1704 The Payload contains the Application Message that is being published. The content and format of the
1705 data is application specific. The length of the Payload can be calculated by subtracting the length of the
1706 Variable Header from the Remaining Length field that is in the Fixed Header. It is valid for a PUBLISH
1707 packet to contain a zero length Payload.

1708

1709 **3.3.4 PUBLISH Actions**

1710 The receiver of a PUBLISH Packet MUST respond with the packet as determined by the QoS in the
1711 PUBLISH Packet [MQTT-3.3.4-1].

1712

1713 Table 3-3 Expected PUBLISH packet response

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

1714

1715 The Client uses a PUBLISH packet to send an Application Message to the Server, for distribution to
1716 Clients with matching subscriptions.

1717

1718 The Server uses a PUBLISH packet to send an Application Message to each Client which has a matching
1719 subscription. The PUBLISH packet includes the Subscription Identifier carried in the SUBSCRIBE packet,
1720 if there was one.

1721

1722 When Clients make subscriptions with Topic Filters that include wildcards, it is possible for a Client's
1723 subscriptions to overlap so that a published message might match multiple filters. In this case the Server
1724 MUST deliver the message to the Client respecting the maximum QoS of all the matching subscriptions
1725 [MQTT-3.3.4-2]. In addition, the Server MAY deliver further copies of the message, one for each
1726 additional matching subscription and respecting the subscription's QoS in each case.

1727

1728 If a Client receives an unsolicited Application Message (not resulting from a subscription) which has a
1729 QoS greater than Maximum QoS, it uses a DISCONNECT packet with Reason Code 0x9B (QoS not
1730 supported) as described in section 4.13 Handling errors.

1731

1732 If the Client specified a Subscription Identifier for any of the overlapping subscriptions the Server MUST
1733 send those Subscription Identifiers in the message which is published as the result of the subscriptions
1734 [MQTT-3.3.4-3]. If the Server sends a single copy of the message it MUST include in the PUBLISH
1735 packet the Subscription Identifiers for all matching subscriptions which have a Subscription Identifiers,
1736 their order is not significant [MQTT-3.3.4-4]. If the Server sends multiple PUBLISH packets it MUST send,
1737 in each of them, the Subscription Identifier of the matching subscription if it has a Subscription Identifier
1738 [MQTT-3.3.4-5].

1739

1740 It is possible that the Client made several subscriptions which match a publication and that it used the
1741 same identifier for more than one of them. In this case the PUBLISH packet will carry multiple identical
1742 Subscription Identifiers.

1743

1744 It is a Protocol Error for a PUBLISH packet to contain any Subscription Identifier other than those
1745 received in SUBSCRIBE packet which caused it to flow. A PUBLISH packet sent from a Client to a Server
1746 MUST NOT contain a Subscription Identifier [MQTT-3.3.4-6].

1747
1748 If the subscription was shared, then only the Subscription Identifiers that were present in the SUBSCRIBE
1749 packet from the Client which is receiving the message are returned in the PUBLISH packet.

1750
1751 The action of the recipient when it receives a PUBLISH packet depends on the QoS level as described in
1752 [section 4.3](#).

1753
1754 If the PUBLISH packet contains a Topic Alias, the receiver processes it as follows:

- 1755 1) A Topic Alias value of 0 or greater than the Maximum Topic Alias is a Protocol Error, the receiver
1756 uses DISCONNECT with Reason Code of 0x94 (Topic Alias invalid) as described in [section 4.13](#).
1757
- 1758 2) If the receiver has already established a mapping for the Topic Alias, then
 - 1759 a) If the packet has a zero length Topic Name, the receiver processes it using the Topic Name that
1760 corresponds to the Topic Alias
 - 1761 b) If the packet contains a non-zero length Topic Name, the receiver processes the packet using
1762 that Topic Name and updates its mapping for the Topic Alias to the Topic Name from the
1763 incoming packet
- 1764 3) If the receiver does not already have a mapping for this Topic Alias
 - 1765 a) If the packet has a zero length Topic Name field it is a Protocol Error and the receiver uses
1766 DISCONNECT with Reason Code of 0x82 (Protocol Error) as described in [section 4.13](#).
1767
 - 1768 b) If the packet contains a Topic Name with a non-zero length, the receiver processes the packet
1769 using that Topic Name and sets its mappings for the Topic Alias to Topic Name from the
1770 incoming packet.

1771
1772 **Non-normative Comment**

1773 If the Server distributes Application Messages to Clients at different protocol levels (such as
1774 MQTT V3.1.1) which do not support properties or other features provided by this specification,
1775 some information in the Application Message can be lost, and applications which depend on this
1776 information might not work correctly.

1777
1778 The Client MUST NOT send more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets for which
1779 it has not received PUBACK, PUBCOMP, or PUBREC with a Reason Code of 128 or greater from the
1780 Server [MQTT-3.3.4-7]. If it receives more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets
1781 where it has not sent a PUBACK or PUBCOMP in response, the Server uses a DISCONNECT packet
1782 with Reason Code 0x93 (Receive Maximum exceeded) as described in [section 4.13](#) Handling errors.
1783 Refer to [section 4.9](#) for more information about flow control.

1784
1785 The Client MUST NOT delay the sending of any packets other than PUBLISH packets due to having sent
1786 Receive Maximum PUBLISH packets without receiving acknowledgements for them [MQTT-3.3.4-8]. The
1787 value of Receive Maximum applies only to the current Network Connection.

1788
1789 **Non-normative comment**

1790 The Client might choose to send fewer than Receive Maximum messages to the Server without
1791 receiving acknowledgement, even if it has more than this number of messages available to send.

1792
1793 **Non-normative comment**

1794 The Client might choose to suspend the sending of QoS 0 PUBLISH packets when it suspends
1795 the sending of QoS 1 and QoS 2 PUBLISH packets.

1796

1797 **Non-normative comment**

1798 If the Client sends QoS 1 or QoS 2 PUBLISH packets before it has received a CONNACK packet,
1799 it risks being disconnected because it has sent more than Receive Maximum publications.

1800

1801 The Server MUST NOT send more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets for
1802 which it has not received PUBACK, PUBCOMP, or PUBREC with a Reason Code of 128 or greater from
1803 the Client [MQTT-3.3.4-9]. If it receives more than Receive Maximum QoS 1 and QoS 2 PUBLISH
1804 packets where it has not sent a PUBACK or PUBCOMP in response, the Client uses DISCONNECT with
1805 Reason Code 0x93 (Receive Maximum exceeded) as described in section 4.13 Handling errors. Refer to
1806 section 4.9 for more information about flow control.

1807

1808 The Server MUST NOT delay the sending of any packets other than PUBLISH packets due to having
1809 sent Receive Maximum PUBLISH packets without receiving acknowledgements for them [MQTT-3.3.4-
1810 10].

1811

1812 **Non-normative comment**

1813 The Server might choose to send fewer than Receive Maximum messages to the Client without
1814 receiving acknowledgement, even if it has more than this number of messages available to send.

1815

1816 **Non-normative comment**

1817 The Server might choose to suspend the sending of QoS 0 PUBLISH packets when it suspends
1818 the sending of QoS 1 and QoS 2 PUBLISH packets.

1819

1820 **3.4 PUBACK – Publish acknowledgement**

1821 A PUBACK packet is the response to a PUBLISH packet with QoS 1.

1822

1823 **3.4.1 PUBACK Fixed Header**

1824 *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							

1825

1826 **Remaining Length field**

1827 This is the length of the Variable Header, encoded as a Variable Byte Integer.

1828

1829 **3.4.2 PUBACK Variable Header**

1830 The Variable Header of the PUBACK Packet contains the following fields in the order: Packet Identifier
 1831 from the PUBLISH packet that is being acknowledged, PUBACK Reason Code, Property Length, and the
 1832 Properties. The rules for encoding Properties are described in [section 2.2.2](#).

1833

1834 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							

1835

1836 **3.4.2.1 PUBACK Reason Code**

1837 Byte 3 in the Variable Header is the PUBACK Reason Code. If the Remaining Length is 2, then there is
 1838 no Reason Code and the value of 0x00 (Success) is used.

1839

1840 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.

1841
1842 **The Client or Server sending the PUBACK packet MUST use one of the PUBACK Reason Codes [MQTT-**
1843 **3.4.2-1].** The Reason Code and Property Length can be omitted if the Reason Code is 0x00 (Success)
1844 and there are no Properties. In this case the PUBACK has a Remaining Length of 2.

1845

1846 **3.4.2.2 PUBACK Properties**

1847 **3.4.2.2.1 Property Length**

1848 The length of the Properties in the PUBACK packet Variable Header encoded as a Variable Byte Integer.
1849 If the Remaining Length is less than 4 there is no Property Length and the value of 0 is used.

1850

1851 **3.4.2.2.2 Reason String**

1852 **31 (0x1F) Byte**, Identifier of the Reason String.

1853 Followed by the UTF-8 Encoded String representing the reason associated with this response. This
1854 Reason String is a human readable string designed for diagnostics and is not intended to be parsed by
1855 the receiver.

1856

1857 The sender uses this value to give additional information to the receiver. **The sender MUST NOT send**
1858 **this property if it would increase the size of the PUBACK packet beyond the Maximum Packet Size**
1859 **specified by the receiver [MQTT-3.4.2-2].** It is a Protocol Error to include the Reason String more than
1860 once.

1861

1862 **3.4.2.2.3 User Property**

1863 **38 (0x26) Byte**, Identifier of the User Property.

1864 Followed by UTF-8 String Pair. This property can be used to provide additional diagnostic or other
1865 information. **The sender MUST NOT send this property if it would increase the size of the PUBACK**
1866 **packet beyond the Maximum Packet Size specified by the receiver [MQTT-3.4.2-3].** The User Property is
1867 allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to
1868 appear more than once.

1869

1870 **3.4.3 PUBACK Payload**

1871 The PUBACK packet has no Payload.

1872

1873 **3.4.4 PUBACK Actions**

1874 This is described in [section 4.3.2](#).

1875

1876 **3.5 PUBREC – Publish received (QoS 2 delivery part 1)**

1877 A PUBREC packet is the response to a PUBLISH packet with QoS 2. It is the second packet of the QoS 2
1878 protocol exchange.

1879

1880 **3.5.1 PUBREC Fixed Header**

1881 *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							

1882

1883 **Remaining Length field**

1884 This is the length of the Variable Header, encoded as a Variable Byte Integer.

1885

1886 **3.5.2 PUBREC Variable Header**

1887 The Variable Header of the PUBREC Packet consists of the following fields in the order: the Packet
 1888 Identifier from the PUBLISH packet that is being acknowledged, PUBREC Reason Code, and Properties.
 1889 The rules for encoding Properties are described in [section 2.2.2](#).

1890

1891 *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							

1892

1893 **3.5.2.1 PUBREC Reason Code**

1894 Byte 3 in the Variable Header is the PUBREC Reason Code. If the Remaining Length is 2, then the
 1895 Publish Reason Code has the value 0x00 (Success).

1896

1897 *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.

1898

1899 **The Client or Server sending the PUBREC packet MUST use one of the PUBREC Reason Code values.**

1900 **[MQTT-3.5.2-1].** The Reason Code and Property Length can be omitted if the Reason Code is 0x00

1901 (Success) and there are no Properties. In this case the PUBREC has a Remaining Length of 2.

1902

1903 3.5.2.2 PUBREC Properties

1904 3.5.2.2.1 Property Length

1905 The length of the Properties in the PUBREC packet Variable Header encoded as a Variable Byte Integer.

1906 If the Remaining Length is less than 4 there is no Property Length and the value of 0 is used.

1907

1908 3.5.2.2.2 Reason String

1909 **31 (0x1F) Byte**, Identifier of the Reason String.

1910 Followed by the UTF-8 Encoded String representing the reason associated with this response. This

1911 Reason String is human readable, designed for diagnostics and SHOULD NOT be parsed by the

1912 receiver.

1913

1914 The sender uses this value to give additional information to the receiver. **The sender MUST NOT send**

1915 **this property if it would increase the size of the PUBREC packet beyond the Maximum Packet Size**

1916 **specified by the receiver [MQTT-3.5.2-2].** It is a Protocol Error to include the Reason String more than

1917 once.

1918

1919 3.5.2.2.3 User Property

1920 **38 (0x26) Byte**, Identifier of the User Property.

1921 Followed by UTF-8 String Pair. This property can be used to provide additional diagnostic or other

1922 information. **The sender MUST NOT send this property if it would increase the size of the PUBREC**

1923 **packet beyond the Maximum Packet Size specified by the receiver [MQTT-3.5.2-3].** The User Property is

1924 allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to

1925 appear more than once.

1926

1927 **3.5.3 PUBREC Payload**

1928 The PUBREC packet has no Payload.

1929 **3.5.4 PUBREC Actions**

1930 This is described in [section 4.3.3](#).

1931

1932 **3.6 PUBREL – Publish release (QoS 2 delivery part 2)**

1933 A PUBREL packet is the response to a PUBREC packet. It is the third packet of the QoS 2 protocol
1934 exchange.

1935

1936 **3.6.1 PUBREL Fixed Header**

1937 *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							

1938

1939 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
1940 respectively. The Server MUST treat any other value as malformed and close the Network Connection
1941 [\[MQTT-3.6.1-1\]](#).

1942

1943 **Remaining Length field**

1944 This is the length of the Variable Header, encoded as a Variable Byte Integer.

1945

1946 **3.6.2 PUBREL Variable Header**

1947 The Variable Header of the PUBREL Packet contains the following fields in the order: the Packet
1948 Identifier from the PUBREC packet that is being acknowledged, PUBREL Reason Code, and Properties.
1949 The rules for encoding Properties are described in [section 2.2.2](#).

1950

1951 *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							

1952

1953 **3.6.2.1 PUBREL Reason Code**

1954 Byte 3 in the Variable Header is the PUBREL Reason Code. If the Remaining Length is 2, the value of
1955 0x00 (Success) is used.

1956

1957 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.

1958

1959 **The Client or Server sending the PUBREL packet MUST use one of the PUBREL Reason Code values**
1960 **[MQTT-3.6.2-1]**. The Reason Code and Property Length can be omitted if the Reason Code is 0x00
1961 (Success) and there are no Properties. In this case the PUBREL has a Remaining Length of 2.

1962

1963 **3.6.2.2 PUBREL Properties**

1964 **3.6.2.2.1 Property Length**

1965 The length of the Properties in the PUBREL packet Variable Header encoded as a Variable Byte Integer.
1966 If the Remaining Length is less than 4 there is no Property Length and the value of 0 is used.

1967

1968 **3.6.2.2.2 Reason String**

1969 **31 (0x1F) Byte**, Identifier of the Reason String.

1970 Followed by the UTF-8 Encoded String representing the reason associated with this response. This
1971 Reason String is human readable, designed for diagnostics and SHOULD NOT be parsed by the
1972 receiver.

1973

1974 The sender uses this value to give additional information to the receiver. **The sender MUST NOT send**
1975 **this Property if it would increase the size of the PUBREL packet beyond the Maximum Packet Size**
1976 **specified by the receiver [MQTT-3.6.2-2]**. It is a Protocol Error to include the Reason String more than
1977 once.

1978

1979 **3.6.2.2.3 User Property**

1980 **38 (0x26) Byte**, Identifier of the User Property.

1981 Followed by UTF-8 String Pair. This property can be used to provide additional diagnostic or other
1982 information for the PUBREL. **The sender MUST NOT send this property if it would increase the size of the**
1983 **PUBREL packet beyond the Maximum Packet Size specified by the receiver [MQTT-3.6.2-3]**. The User
1984 Property is allowed to appear multiple times to represent multiple name, value pairs. The same name is
1985 allowed to appear more than once.

1986

1987 **3.6.3 PUBREL Payload**

1988 The PUBREL packet has no Payload.

1989

1990 3.6.4 PUBREL Actions

1991 This is described in [section 4.3.3](#).

1992

1993 3.7 PUBCOMP – Publish complete (QoS 2 delivery part 3)

1994 The PUBCOMP packet is the response to a PUBREL packet. It is the fourth and final packet of the QoS 2
1995 protocol exchange.

1996

1997 3.7.1 PUBCOMP Fixed Header

1998 *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							

1999

2000 Remaining Length field

2001 This is the length of the Variable Header, encoded as a Variable Byte Integer.

2002

2003 3.7.2 PUBCOMP Variable Header

2004 The Variable Header of the PUBCOMP Packet contains the following fields in the order: Packet Identifier
2005 from the PUBREL packet that is being acknowledged, PUBCOMP Reason Code, and Properties. The
2006 rules for encoding Properties are described in [section 2.2.2](#).

2007

2008 *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							

2009

2010 3.7.2.1 PUBCOMP Reason Code

2011 Byte 3 in the Variable Header is the PUBCOMP Reason Code. If the Remaining Length is 2, then the
2012 value 0x00 (Success) is used.

2013

2014 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.

2015

2016 The Client or Server sending the PUBCOMP packet MUST use one of the PUBCOMP Reason Code
 2017 values [MQTT-3.7.2-1]. The Reason Code and Property Length can be omitted if the Reason Code is
 2018 0x00 (Success) and there are no Properties. In this case the PUBCOMP has a Remaining Length of 2.
 2019

2019

2020 3.7.2.2 PUBCOMP Properties

2021 3.7.2.2.1 Property Length

2022 The length of the Properties in the PUBCOMP packet Variable Header encoded as a Variable Byte
 2023 Integer. If the Remaining Length is less than 4 there is no Property Length and the value of 0 is used.
 2024

2024

2025 3.7.2.2.2 Reason String

2026 **31 (0x1F) Byte**, Identifier of the Reason String.

2027 Followed by the UTF-8 Encoded String representing the reason associated with this response. This
 2028 Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the
 2029 receiver.
 2030

2030

2031 The sender uses this value to give additional information to the receiver. The sender MUST NOT send
 2032 this Property if it would increase the size of the PUBCOMP packet beyond the Maximum Packet Size
 2033 specified by the receiver [MQTT-3.7.2-2]. It is a Protocol Error to include the Reason String more than
 2034 once.
 2035

2035

2036 3.7.2.2.3 User Property

2037 **38 (0x26) Byte**, Identifier of the User Property.

2038 Followed by UTF-8 String Pair. This property can be used to provide additional diagnostic or other
 2039 information. The sender MUST NOT send this property if it would increase the size of the PUBCOMP
 2040 packet beyond the Maximum Packet Size specified by the receiver [MQTT-3.7.2-3]. The User Property is
 2041 allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to
 2042 appear more than once.
 2043

2043

2044 3.7.3 PUBCOMP Payload

2045 The PUBCOMP packet has no Payload.
 2046

2046

2047 3.7.4 PUBCOMP Actions

2048 This is described in [section 4.3.3](#).

2049

2050 3.8 SUBSCRIBE - Subscribe request

2051 The SUBSCRIBE packet is sent from the Client to the Server to create one or more Subscriptions. Each
2052 Subscription registers a Client's interest in one or more Topics. The Server sends PUBLISH packets to
2053 the Client to forward Application Messages that were published to Topics that match these Subscriptions.
2054 The SUBSCRIBE packet also specifies (for each Subscription) the maximum QoS with which the Server
2055 can send Application Messages to the Client.

2056

2057 3.8.1 SUBSCRIBE Fixed Header

2058 *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							

2059

2060 Bits 3,2,1 and 0 of the Fixed Header of the SUBSCRIBE packet are reserved and MUST be set to 0,0,1
2061 and 0 respectively. The Server MUST treat any other value as malformed and close the Network
2062 Connection [MQTT-3.8.1-1].

2063

2064 Remaining Length field

2065 This is the length of Variable Header plus the length of the Payload, encoded as a Variable Byte Integer.

2066

2067 3.8.2 SUBSCRIBE Variable Header

2068 The Variable Header of the SUBSCRIBE Packet contains the following fields in the order: Packet
2069 Identifier, and Properties. Section 2.2.1 provides more information about Packet Identifiers. The rules for
2070 encoding Properties are described in section 2.2.2.

2071

2072 Non-normative example

2073 Figure 3-19 shows an example of a SUBSCRIBE variable header with a Packet Identifier of 10
2074 and no properties.

2075

2076 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

2077

2078 **3.8.2.1 SUBSCRIBE Properties**

2079 **3.8.2.1.1 Property Length**

2080 The length of Properties in the SUBSCRIBE packet Variable Header encoded as a Variable Byte Integer.
2081

2082 **3.8.2.1.2 Subscription Identifier**

2083 **11 (0x0B) Byte**, Identifier of the Subscription Identifier.

2084 Followed by a Variable Byte Integer representing the identifier of the subscription. The Subscription
2085 Identifier can have the value of 1 to 268,435,455. It is a Protocol Error if the Subscription Identifier has a
2086 value of 0. It is a Protocol Error to include the Subscription Identifier more than once.

2087
2088 The Subscription Identifier is associated with any subscription created or modified as the result of this
2089 SUBSCRIBE packet. If there is a Subscription Identifier, it is stored with the subscription. If this property is
2090 not specified, then the absence of a Subscription Identifier is stored with the subscription.

2091
2092 Refer to [section 3.8.3.1](#) for more information about the handling of Subscription Identifiers.
2093

2094 **3.8.2.1.3 User Property**

2095 **38 (0x26) Byte**, Identifier of the User Property.

2096 Followed by a UTF-8 String Pair.

2097
2098 The User Property is allowed to appear multiple times to represent multiple name, value pairs. The same
2099 name is allowed to appear more than once.

2100
2101 **Non-normative comment**

2102 User Properties on the SUBSCRIBE packet can be used to send subscription related properties
2103 from the Client to the Server. The meaning of these properties is not defined by this specification.

2104

2105 **3.8.3 SUBSCRIBE Payload**

2106 The Payload of a SUBSCRIBE packet contains a list of Topic Filters indicating the Topics to which the
2107 Client wants to subscribe. **The Topic Filters MUST be a UTF-8 Encoded String [MQTT-3.8.3-1].** Each
2108 Topic Filter is followed by a Subscription Options byte.

2109
2110 **The Payload MUST contain at least one Topic Filter and Subscription Options pair [MQTT-3.8.3-2].** A
2111 SUBSCRIBE packet with no Payload is a Protocol Error. Refer to [section 4.13](#) for information about
2112 handling errors.

2113

2114 **3.8.3.1 Subscription Options**

2115 Bits 0 and 1 of the Subscription Options represent Maximum QoS field. This gives the maximum QoS
2116 level at which the Server can send Application Messages to the Client. It is a Protocol Error if the
2117 Maximum QoS field has the value 3.

2118

2119 Bit 2 of the Subscription Options represents the No Local option. If the value is 1, Application Messages
 2120 MUST NOT be forwarded to a connection with a ClientID equal to the ClientID of the publishing
 2121 connection [MQTT-3.8.3-3]. It is a Protocol Error to set the No Local bit to 1 on a Shared Subscription
 2122 [MQTT-3.8.3-4].

2123
 2124 Bit 3 of the Subscription Options represents the Retain As Published option. If 1, Application Messages
 2125 forwarded using this subscription keep the RETAIN flag they were published with. If 0, Application
 2126 Messages forwarded using this subscription have the RETAIN flag set to 0. Retained messages sent
 2127 when the subscription is established have the RETAIN flag set to 1.

2128
 2129 Bits 4 and 5 of the Subscription Options represent the Retain Handling option. This option specifies
 2130 whether retained messages are sent when the subscription is established. This does not affect the
 2131 sending of retained messages at any point after the subscribe. If there are no retained messages
 2132 matching the Topic Filter, all of these values act the same. The values are:

- 2133 0 = Send retained messages at the time of the subscribe
- 2134 1 = Send retained messages at subscribe only if the subscription does not currently exist
- 2135 2 = Do not send retained messages at the time of the subscribe
- 2136 It is a Protocol Error to send a Retain Handling value of 3.

2137
 2138 Bits 6 and 7 of the Subscription Options byte are reserved for future use. The Server MUST treat a
 2139 SUBSCRIBE packet as malformed if any of Reserved bits in the Payload are non-zero [MQTT-3.8.3-5].

2140
 2141 **Non-normative comment**
 2142 The No Local and Retain As Published subscription options can be used to implement bridging
 2143 where the Client is sending the message on to another Server.

2144
 2145 **Non-normative comment**
 2146 Not sending retained messages for an existing subscription is useful when a reconnect is done
 2147 and the Client is not certain whether the subscriptions were completed in the previous connection
 2148 to the Session.

2149
 2150 **Non-normative comment**
 2151 Not sending stored retained messages because of a new subscription is useful where a Client
 2152 wishes to receive change notifications and does not need to know the initial state.

2153
 2154 **Non-normative comment**
 2155 For a Server that indicates it does not support retained messages, all valid values of Retain As
 2156 Published and Retain Handling give the same result which is to not send any retained messages
 2157 at subscribe and to set the RETAIN flag to 0 for all messages.

2158
 2159 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

2160 RAP means Retain as Published.

2161 NL means No Local.

2162

2163 **Non-normative example**

2164 Figure 3.21 show the SUBSCRIBE Payload example with two Topic Filters. The first is “a/b” with
 2165 QoS 1, and the second is “c/d” with QoS 2.

2166

2167 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

2168

2169 **3.8.4 SUBSCRIBE Actions**

2170 When the Server receives a SUBSCRIBE packet from a Client, the Server MUST respond with a
 2171 SUBACK packet [MQTT-3.8.4-1]. The SUBACK packet MUST have the same Packet Identifier as the
 2172 SUBSCRIBE packet that it is acknowledging [MQTT-3.8.4-2].

2173

2174 The Server is permitted to start sending PUBLISH packets matching the Subscription before the Server
2175 sends the SUBACK packet.

2176

2177 If a Server receives a SUBSCRIBE packet containing a Topic Filter that is identical to a Non-shared
2178 Subscription's Topic Filter for the current Session, then it MUST replace that existing Subscription with a
2179 new Subscription [MQTT-3.8.4-3]. The Topic Filter in the new Subscription will be identical to that in the
2180 previous Subscription, although its Subscription Options could be different. If the Retain Handling option
2181 is 0, any existing retained messages matching the Topic Filter MUST be re-sent, but Applicaton
2182 Messages MUST NOT be lost due to replacing the Subscription [MQTT-3.8.4-4].

2183

2184 If a Server receives a Non-shared Topic Filter that is not identical to any Topic Filter for the current
2185 Session, a new Non-shared Subscription is created. If the Retain Handling option is not 2, all matching
2186 retained messages are sent to the Client.

2187

2188 If a Server receives a Topic Filter that is identical to the Topic Filter for a Shared Subscription that already
2189 exists on the Server, the Session is added as a subscriber to that Shared Subscription. No retained
2190 messages are sent.

2191

2192 If a Server receives a Shared Subscription Topic Filter that is not identical to any existing Shared
2193 Subscription's Topic Filter, a new Shared Subscription is created. The Session is added as a subscriber
2194 to that Shared Subscription. No retained messages are sent.

2195

2196 Refer to [section 4.8](#) for more details on Shared Subscriptions.

2197

2198 If a Server receives a SUBSCRIBE packet that contains multiple Topic Filters it MUST handle that packet
2199 as if it had received a sequence of multiple SUBSCRIBE packets, except that it combines their responses
2200 into a single SUBACK response [MQTT-3.8.4-5].

2201

2202 The SUBACK packet sent by the Server to the Client MUST contain a Reason Code for each Topic
2203 Filter/Subscription Option pair [MQTT-3.8.4-6]. This Reason Code MUST either show the maximum QoS
2204 that was granted for that Subscription or indicate that the subscription failed [MQTT-3.8.4-7]. The Server
2205 might grant a lower Maximum QoS than the subscriber requested. The QoS of Application Messages sent
2206 in response to a Subscription MUST be the minimum of the QoS of the originally published message and
2207 the Maximum QoS granted by the Server [MQTT-3.8.4-8]. The server is permitted to send duplicate
2208 copies of a message to a subscriber in the case where the original message was published with QoS 1
2209 and the maximum QoS granted was QoS 0.

2210

2211 **Non-normative comment**

2212 If a subscribing Client has been granted maximum QoS 1 for a particular Topic Filter, then a QoS
2213 0 Application Message matching the filter is delivered to the Client at QoS 0. This means that at
2214 most one copy of the message is received by the Client. On the other hand, a QoS 2 Message
2215 published to the same topic is downgraded by the Server to QoS 1 for delivery to the Client, so
2216 that Client might receive duplicate copies of the Message.

2217

2218 **Non-normative comment**

2219 If the subscribing Client has been granted maximum QoS 0, then an Application Message
2220 originally published as QoS 2 might get lost on the hop to the Client, but the Server should never
2221 send a duplicate of that Message. A QoS 1 Message published to the same topic might either get
2222 lost or duplicated on its transmission to that Client.

2223

2224 **Non-normative comment**

2225 Subscribing to a Topic Filter at QoS 2 is equivalent to saying "I would like to receive Messages
2226 matching this filter at the QoS with which they were published". This means a publisher is
2227 responsible for determining the maximum QoS a Message can be delivered at, but a subscriber is
2228 able to require that the Server downgrades the QoS to one more suitable for its usage.

2229
2230 The Subscription Identifiers are part of the Session State in the Server and are returned to the Client
2231 receiving a matching PUBLISH packet. They are removed from the Server's Session State when the
2232 Server receives an UNSUBSCRIBE packet, when the Server receives a SUBSCRIBE packet from the
2233 Client for the same Topic Filter but with a different Subscription Identifier or with no Subscription Identifier,
2234 or when the Server sends Session Present 0 in a CONNACK packet.

2235
2236 The Subscription Identifiers do not form part of the Client's Session State in the Client. In a useful
2237 implementation, a Client will associate the Subscription Identifiers with other Client side state, this state is
2238 typically removed when the Client unsubscribes, when the Client subscribes for the same Topic Filter with
2239 a different identifier or no identifier, or when the Client receives Session Present 0 in a CONNACK
2240 packet.

2241
2242 The Server need not use the same set of Subscription Identifiers in the retransmitted PUBLISH packet.
2243 The Client can remake a Subscription by sending a SUBSCRIBE packet containing a Topic Filter that is
2244 identical to the Topic Filter of an existing Subscription in the current Session. If the Client remade a
2245 subscription after the initial transmission of a PUBLISH packet and used a different Subscription Identifier,
2246 then the Server is allowed to use the identifiers from the first transmission in any retransmission.
2247 Alternatively, the Server is allowed to use the new identifiers during a retransmission. The Server is not
2248 allowed to revert to the old identifier after it has sent a PUBLISH packet containing the new one.

2249
2250 **Non-normative comment**

2251 Usage scenarios, for illustration of Subscription Identifiers.

- 2252 • The Client implementation indicates via its programming interface that a publication matched
2253 more than one subscription. The Client implementation generates a new identifier each time
2254 a subscription is made. If the returned publication carries more than one Subscription
2255 Identifier, then the publication matched more than one subscription.
2256
- 2257 • The Client implementation allows the subscriber to direct messages to a callback associated
2258 with the subscription. The Client implementation generates an identifier which uniquely maps
2259 the identifier to the callback. When a publication is received it uses the Subscription Identifier
2260 to determine which callback is driven.
2261
- 2262 • The Client implementation returns the topic string used to make the subscription to the
2263 application when it delivers the published message. To achieve this the Client generates an
2264 identifier which uniquely identifies the Topic Filter. When a publication is received the Client
2265 implementation uses the identifiers to look up the original Topic Filters and return them to the
2266 Client application.
2267
- 2268 • A gateway forwards publications received from a Server to Clients that have made
2269 subscriptions to the gateway. The gateway implementation maintains a map of each unique
2270 Topic Filter it receives to the set of ClientID, Subscription Identifier pairs that it also received.
2271 It generates a unique identifier for each Topic Filter that it forwards to the Server. When a
2272 publication is received, the gateway uses the Subscription Identifiers it received from the
2273 Server to look up the Client Identifier, Subscription Identifier pairs associated with them. It
2274 adds these to the PUBLISH packets it sends to the Clients. If the upstream Server sent
2275 multiple PUBLISH packets because the message matched multiple subscriptions, then this
2276 behavior is mirrored to the Clients.

2277

2278 **3.9 SUBACK – Subscribe acknowledgement**

2279 A SUBACK packet is sent by the Server to the Client to confirm receipt and processing of a SUBSCRIBE
2280 packet.

2281

2282 A SUBACK packet contains a list of Reason Codes, that specify the maximum QoS level that was
2283 granted or the error which was found for each Subscription that was requested by the SUBSCRIBE.
2284

2285 **3.9.1 SUBACK Fixed Header**

2286 *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							

2287

2288 **Remaining Length field**

2289 This is the length of Variable Header plus the length of the Payload, encoded as a Variable Byte Integer.

2290

2291 **3.9.2 SUBACK Variable Header**

2292 The Variable Header of the SUBACK Packet contains the following fields in the order: the Packet
2293 Identifier from the SUBSCRIBE Packet that is being acknowledged, and Properties.
2294

2295 **3.9.2.1 SUBACK Properties**

2296 **3.9.2.1.1 Property Length**

2297 The length of Properties in the SUBACK packet Variable Header encoded as a Variable Byte Integer

2298

2299 **3.9.2.1.2 Reason String**

2300 **31 (0x1F) Byte**, Identifier of the Reason String.

2301 Followed by the UTF-8 Encoded String representing the reason associated with this response. This
2302 Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the
2303 Client.

2304

2305 The Server uses this value to give additional information to the Client. **The Server MUST NOT send this**
2306 **Property if it would increase the size of the SUBACK packet beyond the Maximum Packet Size specified**
2307 **by the Client [MQTT-3.9.2-1].** It is a Protocol Error to include the Reason String more than once.

2308

2309 **3.9.2.1.3 User Property**

2310 **38 (0x26) Byte**, Identifier of the User Property.

2311 Followed by UTF-8 String Pair. This property can be used to provide additional diagnostic or other
 2312 information. The Server MUST NOT send this property if it would increase the size of the SUBACK packet
 2313 beyond the Maximum Packet Size specified by Client [MQTT-3.9.2-2]. The User Property is allowed to
 2314 appear multiple times to represent multiple name, value pairs. The same name is allowed to appear more
 2315 than once.

2316

2317 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							

2318

2319 3.9.3 SUBACK Payload

2320 The Payload contains a list of Reason Codes. Each Reason Code corresponds to a Topic Filter in the
 2321 SUBSCRIBE packet being acknowledged. The order of Reason Codes in the SUBACK packet MUST
 2322 match the order of Topic Filters in the SUBSCRIBE packet [MQTT-3.9.3-1].

2323

2324 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 Subscription 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 subscription; the subscription is not accepted.

2325

2326 The Server sending a SUBACK packet MUST use one of the Subscribe Reason Codes for each Topic
2327 Filter received [MQTT-3.9.3-2].

2328

2329 **Non-normative comment**

2330 There is always one Reason Code for each Topic Filter in the corresponding SUBSCRIBE
2331 packet. If the Reason Code is not specific to a Topic Filters (such as 0x91 (Packet Identifier in
2332 use)) it is set for each Topic Filter.

2333

2334 **3.10 UNSUBSCRIBE – Unsubscribe request**

2335 An UNSUBSCRIBE packet is sent by the Client to the Server, to unsubscribe from topics.

2336

2337 **3.10.1 UNSUBSCRIBE Fixed Header**

2338 *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							

2339

2340 Bits 3,2,1 and 0 of the Fixed Header of the UNSUBSCRIBE packet are reserved and MUST be set to
2341 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network
2342 Connection [MQTT-3.10.1-1].

2343

2344 **Remaining Length field**

2345 This is the length of Variable Header (2 bytes) plus the length of the Payload, encoded as a Variable Byte
2346 Integer.

2347

2348 **3.10.2 UNSUBSCRIBE Variable Header**

2349 The Variable Header of the UNSUBSCRIBE Packet contains the following fields in the order: Packet
2350 Identifier, and Properties. Section 2.2.1 provides more information about Packet Identifiers. The rules for
2351 encoding Properties are described in section 2.2.2.

2352

2353 **3.10.2.1 UNSUBSCRIBE Properties**

2354 **3.10.2.1.1 Property Length**

2355 The length of Properties in the SUBSCRIBE packet Variable Header encoded as a Variable Byte Integer.

2356

2357 **3.10.2.1.2 User Property**

2358 **38 (0x26) Byte**, Identifier of the User Property.

2359 Followed by a UTF-8 String Pair.

2360

2361 The User Property is allowed to appear multiple times to represent multiple name, value pairs. The same
2362 name is allowed to appear more than once.

2363

2364 **Non-normative comment**

2365 User Properties on the UNSUBSCRIBE packet can be used to send subscription related
2366 properties from the Client to the Server. The meaning of these properties is not defined by this
2367 specification.

2368

2369 **3.10.3 UNSUBSCRIBE Payload**

2370 The Payload for the UNSUBSCRIBE packet contains the list of Topic Filters that the Client wishes to
2371 unsubscribe from. **The Topic Filters in an UNSUBSCRIBE packet MUST be UTF-8 Encoded Strings**
2372 **[MQTT-3.10.3-1]** as defined in [section 1.5.4](#), packed contiguously.

2373

2374 **The Payload of an UNSUBSCRIBE packet MUST contain at least one Topic Filter [MQTT-3.10.3-2].** An
2375 UNSUBSCRIBE packet with no Payload is a Protocol Error. Refer to [section 4.13](#) for information about
2376 handling errors.

2377

2378 **Non-normative example**

2379 Figure 3.30 shows the Payload for an UNSUBSCRIBE packet with two Topic Filters “a/b” and “c/d”.

2380

2381 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

2382

2383 **3.10.4 UNSUBSCRIBE Actions**

2384 The Topic Filters (whether they contain wildcards or not) supplied in an UNSUBSCRIBE packet MUST be
 2385 compared character-by-character with the current set of Topic Filters held by the Server for the Client. If
 2386 any filter matches exactly then its owning Subscription MUST be deleted [MQTT-3.10.4-1], otherwise no
 2387 additional processing occurs.
 2388

2389 When a Server receives UNSUBSCRIBE :

- 2390 • It MUST stop adding any new messages which match the Topic Filters, for delivery to the Client
 2391 [MQTT-3.10.4-2].
- 2392 • It MUST complete the delivery of any QoS 1 or QoS 2 messages which match the Topic Filters
 2393 and it has started to send to the Client [MQTT-3.10.4-3].
- 2394 • It MAY continue to deliver any existing messages buffered for delivery to the Client.

2395
 2396 The Server MUST respond to an UNSUBSCRIBE request by sending an UNSUBACK packet [MQTT-
 2397 3.10.4-4]. The UNSUBACK packet MUST have the same Packet Identifier as the UNSUBSCRIBE packet.
 2398 Even where no Topic Subscriptions are deleted, the Server MUST respond with an UNSUBACK [MQTT-
 2399 3.10.4-5].

2400
 2401 If a Server receives an UNSUBSCRIBE packet that contains multiple Topic Filters, it MUST process that
 2402 packet as if it had received a sequence of multiple UNSUBSCRIBE packets, except that it sends just one
 2403 UNSUBACK response [MQTT-3.10.4-6].
 2404

2405 If a Topic Filter represents a Shared Subscription, this Session is detached from the Shared Subscription.
 2406 If this Session was the only Session that the Shared Subscription was associated with, the Shared
 2407 Subscription is deleted. Refer to section 4.8.2 for a description of Shared Subscription handling.
 2408

2409 **3.11 UNSUBACK – Unsubscribe acknowledgement**

2410 The UNSUBACK packet is sent by the Server to the Client to confirm receipt of an UNSUBSCRIBE
 2411 packet.
 2412

2413 **3.11.1 UNSUBACK Fixed Header**

2414 *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							

2415
 2416 **Remaining Length field**
 2417 This is the length of the Variable Header plus the length of the Payload, encoded as a Variable Byte
 2418 Integer.
 2419

2420 **3.11.2 UNSUBACK Variable Header**

2421 The Variable Header of the UNSUBACK Packet the following fields in the order: the Packet Identifier from
2422 the UNSUBSCRIBE Packet that is being acknowledged, and Properties. The rules for encoding
2423 Properties are described in [section 2.2.2](#).

2424

2425 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							

2426

2427 **3.11.2.1 UNSUBACK Properties**

2428 **3.11.2.1.1 Property Length**

2429 The length of the Properties in the UNSUBACK packet Variable Header encoded as a Variable Byte
2430 Integer.

2431

2432 **3.11.2.1.2 Reason String**

2433 **31 (0x1F) Byte**, Identifier of the Reason String.

2434 Followed by the UTF-8 Encoded String representing the reason associated with this response. This
2435 Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the
2436 Client.

2437

2438 The Server uses this value to give additional information to the Client. **The Server MUST NOT send this**
2439 **Property if it would increase the size of the UNSUBACK packet beyond the Maximum Packet Size**
2440 **specified by the Client [MQTT-3.11.2-1].** It is a Protocol Error to include the Reason String more than
2441 once.

2442

2443 **3.11.2.1.3 User Property**

2444 **38 (0x26) Byte**, Identifier of the User Property.

2445 Followed by UTF-8 String Pair. This property can be used to provide additional diagnostic or other
2446 information. **The Server MUST NOT send this property if it would increase the size of the UNSUBACK**
2447 **packet beyond the Maximum Packet Size specified by the Client [MQTT-3.11.2-2].** The User Property is
2448 allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to
2449 appear more than once.

2450

2451 **3.11.3 UNSUBACK Payload**

2452 The Payload contains a list of Reason Codes. Each Reason Code corresponds to a Topic Filter in the
2453 UNSUBSCRIBE packet being acknowledged. **The order of Reason Codes in the UNSUBACK packet**
2454 **MUST match the order of Topic Filters in the UNSUBSCRIBE packet [MQTT-3.11.3-1].**

2455

2456 The values for the one byte unsigned Unsubscribe Reason Codes are shown below. The Server sending
 2457 an UNSUBACK packet MUST use one of the Unsubscribe Reason Code values for each Topic Filter
 2458 received [MQTT-3.11.3-2].

2459

2460 Table 3-9 - Unsubscribe Reason Codes

Value	Hex	Reason Code name	Description
0	0x00	Success	The subscription is deleted.
17	0x11	No subscription found	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.

2461

2462 **Non-normative comment**

2463 There is always one Reason Code for each Topic Filter in the corresponding UNSUBSCRIBE
 2464 packet. If the Reason Code is not specific to a Topic Filters (such as 0x91 (Packet Identifier in
 2465 use)) it is set for each Topic Filter.

2466

2467 **3.12 PINGREQ – PING request**

2468 The PINGREQ packet is sent from a Client to the Server. It can be used to:

- 2469 • Indicate to the Server that the Client is alive in the absence of any other MQTT Control Packets being
 2470 sent from the Client to the Server.
- 2471 • Request that the Server responds to confirm that it is alive.
- 2472 • Exercise the network to indicate that the Network Connection is active.

2473

2474 This packet is used in Keep Alive processing. Refer to [section 3.1.2.10](#) for more details.

2475

2476 **3.12.1 PINGREQ Fixed Header**

2477 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

2478

2479 3.12.2 PINGREQ Variable Header

2480 The PINGREQ packet has no Variable Header.

2481

2482 3.12.3 PINGREQ Payload

2483 The PINGREQ packet has no Payload.

2484

2485 3.12.4 PINGREQ Actions

2486 The Server MUST send a PINGRESP packet in response to a PINGREQ packet [MQTT-3.12.4-1].

2487

2488 3.13 PINGRESP – PING response

2489 A PINGRESP Packet is sent by the Server to the Client in response to a PINGREQ packet. It indicates
2490 that the Server is alive.

2491

2492 This packet is used in Keep Alive processing. Refer to [section 3.1.2.10](#) for more details.

2493

2494 3.13.1 PINGRESP Fixed Header

2495 *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

2496

2497 3.13.2 PINGRESP Variable Header

2498 The PINGRESP packet has no Variable Header.

2499

2500 3.13.3 PINGRESP Payload

2501 The PINGRESP packet has no Payload.

2502

2503 3.13.4 PINGRESP Actions

2504 The Client takes no action on receiving this packet

2505

2506 **3.14 DISCONNECT – Disconnect notification**

2507 The DISCONNECT packet is the final MQTT Control Packet sent from the Client or the Server. It
 2508 indicates the reason why the Network Connection is being closed. The Client or Server MAY send a
 2509 DISCONNECT packet before closing the Network Connection. If the Network Connection is closed
 2510 without the Client first sending a DISCONNECT packet with Reason Code 0x00 (Normal disconnection)
 2511 and the Connection has a Will Message, the Will Message is published. Refer to [section 3.1.2.5](#) for
 2512 further details.

2513
 2514 **A Server MUST NOT send a DISCONNECT until after it has sent a CONNACK with Reason Code of less**
 2515 **than 0x80 [MQTT-3.14.0-1].**
 2516

2517 **3.14.1 DISCONNECT Fixed Header**

2518 *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							

2519 **The Client or Server MUST validate that reserved bits are set to 0. If they are not zero it sends a**
 2520 **DISCONNECT packet with a Reason code of 0x81 (Malformed Packet) as described in [section 4.13](#)**
 2521 **[MQTT-3.14.1-1].**

2522
 2523 **Remaining Length field**

2524 This is the length of the Variable Header encoded as a Variable Byte Integer.
 2525

2526 **3.14.2 DISCONNECT Variable Header**

2527 The Variable Header of the DISCONNECT Packet contains the following fields in the order: Disconnect
 2528 Reason Code, and Properties. The rules for encoding Properties are described in [section 2.2.2](#).

2529
 2530 **3.14.2.1 Disconnect Reason Code**

2531 Byte 1 in the Variable Header is the Disconnect Reason Code. If the Remaining Length is less than 1 the
 2532 value of 0x00 (Normal disconnection) is used.

2533
 2534 The values for the one byte unsigned Disconnect Reason Code field are shown below.

2535
 2536 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 Subscription 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 subscription; the subscription is not accepted.

2537

2538 The Client or Server sending the DISCONNECT packet MUST use one of the DISCONNECT Reason
 2539 Code values [MQTT-3.14.2-1]. The Reason Code and Property Length can be omitted if the Reason
 2540 Code is 0x00 (Normal disconnect) and there are no Properties. In this case the DISCONNECT has a
 2541 Remaining Length of 0.

2542

2543 **Non-normative comment**

2544 The DISCONNECT packet is used to indicate the reason for a disconnect for cases where there
 2545 is no acknowledge packet (such as a QoS 0 publish) or when the Client or Server is unable to
 2546 continue processing the Connection.

2547

2548 **Non-normative comment**

2549 The information can be used by the Client to decide whether to retry the connection, and how
 2550 long it should wait before retrying the connection.

2551

2552 **3.14.2.2 DISCONNECT Properties**

2553 **3.14.2.2.1 Property Length**

2554 The length of Properties in the DISCONNECT packet Variable Header encoded as a Variable Byte
 2555 Integer. If the Remaining Length is less than 2, a value of 0 is used.

2556

2557 **3.14.2.2.2 Session Expiry Interval**

2558 **17 (0x11) Byte**, Identifier of the Session Expiry Interval.

2559 Followed by the Four Byte Integer representing the Session Expiry Interval in seconds. It is a Protocol
 2560 Error to include the Session Expiry Interval more than once.

2561

2562 If the Session Expiry Interval is absent, the Session Expiry Interval in the CONNECT packet is used.

2563

2564 The Session Expiry Interval MUST NOT be sent on a DISCONNECT by the Server [MQTT-3.14.2-2].

2565

2566 If the Session Expiry Interval in the CONNECT packet was zero, then it is a Protocol Error to set a non-zero Session Expiry Interval in the DISCONNECT packet sent by the Client. If such a non-zero Session Expiry Interval is received by the Server, it does not treat it as a valid DISCONNECT packet. The Server uses DISCONNECT with Reason Code 0x82 (Protocol Error) as described in section 4.13.

2570

2571 3.14.2.2.3 Reason String

2572 31 (0x1F) Byte, Identifier of the Reason String.

2573 Followed by the UTF-8 Encoded String representing the reason for the disconnect. This Reason String is human readable, designed for diagnostics and SHOULD NOT be parsed by the receiver.

2575

2576 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-3.14.2-3]. It is a Protocol Error to include the Reason String more than once.

2579

2580 3.14.2.2.4 User Property

2581 38 (0x26) Byte, Identifier of the User Property.

2582 Followed by UTF-8 String Pair. This property may be used to provide additional diagnostic or other information. 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-3.14.2-4]. 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.

2587

2588 3.14.2.2.5 Server Reference

2589 28 (0x1C) Byte, Identifier of the Server Reference.

2590 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.

2592

2593 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.

2595

2596 Refer to section 4.11 Server Redirection for information about how Server Reference is used.

2597

2598 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

2599

2600 3.14.3 DISCONNECT Payload

2601 The DISCONNECT packet has no Payload.

2602

2603 3.14.4 DISCONNECT Actions

2604 After sending a DISCONNECT packet the sender:

- 2605 • MUST NOT send any more MQTT Control Packets on that Network Connection [MQTT-3.14.4-1].
- 2606 • MUST close the Network Connection [MQTT-3.14.4-2].

2607

2608 On receipt of DISCONNECT with a Reason Code of 0x00 (Success) the Server:

- 2609 • 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.

2610

2611 On receipt of DISCONNECT, the receiver:

- 2612 • SHOULD close the Network Connection.

2613

2615 3.15 AUTH – Authentication exchange

2616 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.

2619

2620 3.15.1 AUTH Fixed Header

2621 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							

2622

2623 Bits 3,2,1 and 0 of the Fixed Header of the AUTH packet are reserved and MUST all be set to 0. The
2624 Client or Server MUST treat any other value as malformed and close the Network Connection [MQTT-
2625 3.15.1-1].

2626

2627 Remaining Length field

2628 This is the length of the Variable Header encoded as a Variable Byte Integer.

2629

2630 3.15.2 AUTH Variable Header

2631 The Variable Header of the AUTH Packet contains the following fields in the order: Authenticate Reason
2632 Code, and Properties. The rules for encoding Properties are described in [section 2.2.2](#).

2633

2634 3.15.2.1 Authenticate Reason Code

2635 Byte 0 in the Variable Header is the Authenticate Reason Code. The values for the one byte unsigned
2636 Authenticate Reason Code field are shown below. The sender of the AUTH Packet MUST use one of the
2637 Authenticate Reason Codes [MQTT-3.15.2-1].

2638

2639 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

2640 The Reason Code and Property Length can be omitted if the Reason Code is 0x00 (Success) and there
2641 are no Properties. In this case the AUTH has a Remaining Length of 0.

2642

2643 3.15.2.2 AUTH Properties

2644 3.15.2.2.1 Property Length

2645 The length of Properties in the AUTH packet Variable Header encoded as a Variable Byte Integer.

2646

2647 3.15.2.2.2 Authentication Method

2648 **21 (0x15) Byte**, Identifier of the Authentication Method.

2649 Followed by a UTF-8 Encoded String containing the name of the authentication method. It is a Protocol
2650 Error to omit the Authentication Method or to include it more than once. Refer to [section 4.12](#) for more
2651 information about extended authentication.

2652

2653 3.15.2.2.3 Authentication Data

2654 **22 (0x16) Byte**, Identifier of the Authentication Data.

2655 Followed by Binary Data containing authentication data. It is a Protocol Error to include Authentication
2656 Data more than once. The contents of this data are defined by the authentication method. Refer to
2657 [section 4.12](#) for more information about extended authentication.
2658

2659 **3.15.2.2.4 Reason String**

2660 **31 (0x1F) Byte**, Identifier of the Reason String.

2661 Followed by the UTF-8 Encoded String representing the reason for the disconnect. This Reason String is
2662 human readable, designed for diagnostics and SHOULD NOT be parsed by the receiver.

2663
2664 The sender MUST NOT send this property if it would increase the size of the AUTH packet beyond the
2665 Maximum Packet Size specified by the receiver [MQTT-3.15.2-2]. It is a Protocol Error to include the
2666 Reason String more than once.

2667

2668 **3.15.2.2.5 User Property**

2669 **38 (0x26) Byte**, Identifier of the User Property.

2670 Followed by UTF-8 String Pair. This property may be used to provide additional diagnostic or other
2671 information. The sender MUST NOT send this property if it would increase the size of the AUTH packet
2672 beyond the Maximum Packet Size specified by the receiver [MQTT-3.15.2-3]. The User Property is
2673 allowed to appear multiple times to represent multiple name, value pairs. The same name is allowed to
2674 appear more than once.

2675

2676 **3.15.3 AUTH Payload**

2677 The AUTH packet has no Payload.

2678

2679 **3.15.4 AUTH Actions**

2680 Refer to [section 4.12](#) for more information about extended authentication.

2681

4 Operational behavior

2682

4.1 Session State

2683 In order to implement QoS 1 and QoS 2 protocol flows the Client and Server need to associate state with
2684 the Client Identifier, this is referred to as the Session State. The Server also stores the subscriptions as
2685 part of the Session State.

2686

2687 The session can continue across a sequence of Network Connections. It lasts as long as the latest
2688 Network Connection plus the Session Expiry Interval.

2689

2690 The Session State in the Client consists of:

- 2691 • QoS 1 and QoS 2 messages which have been sent to the Server, but have not been completely
2692 acknowledged.
- 2693 • QoS 2 messages which have been received from the Server, but have not been completely
2694 acknowledged.

2695

2696 The Session State in the Server consists of:

- 2697 • The existence of a Session, even if the rest of the Session State is empty.
- 2698 • The Clients subscriptions, including any Subscription Identifiers.
- 2699 • QoS 1 and QoS 2 messages which have been sent to the Client, but have not been completely
2700 acknowledged.
- 2701 • QoS 1 and QoS 2 messages pending transmission to the Client and OPTIONALLY QoS 0 messages
2702 pending transmission to the Client.
- 2703 • QoS 2 messages which have been received from the Client, but have not been completely
2704 acknowledged. The Will Message and the Will Delay Interval
- 2705 • If the Session is currently not connected, the time at which the Session will end and Session State will
2706 be discarded.

2707

2708 Retained messages do not form part of the Session State in the Server, they are not deleted as a result of
2709 a Session ending.

2710

4.1.1 Storing Session State

2712 The Client and Server MUST NOT discard the Session State while the Network Connection is open
2713 [MQTT-4.1.0-1]. The Server MUST discard the Session State when the Network Connection is closed and
2714 the Session Expiry Interval has passed [MQTT-4.1.0-2].

2715

Non-normative comment

2717 The storage capabilities of Client and Server implementations will of course have limits in terms
2718 of capacity and may be subject to administrative policies. Stored Session State can be discarded
2719 as a result of an administrator action, including an automated response to defined conditions.
2720 This has the effect of terminating the Session. These actions might be prompted by resource
2721 constraints or for other operational reasons. It is possible that hardware or software failures may
2722 result in loss or corruption of Session State stored by the Client or Server. It is prudent to
2723 evaluate the storage capabilities of the Client and Server to ensure that they are sufficient.

2724

2725 **4.1.2 Session State non-normative examples**

2726 For example, an electricity meter reading solution might use QoS 1 messages to protect the readings
2727 against loss over the network. The solution developer might have determined that the power supply is
2728 sufficiently reliable that, in this case, the data in the Client and Server can be stored in volatile memory
2729 without too much risk of its loss.

2730

2731 Conversely a parking meter payment application provider might decide that the payment messages
2732 should never be lost due to a network or Client failure. Thus, they require that all data be written to non-
2733 volatile memory before it is transmitted across the network.

2734

2735 **4.2 Network Connections**

2736 The MQTT protocol requires an underlying transport that provides an ordered, lossless, stream of bytes
2737 from the Client to Server and Server to Client. This specification does not require the support of any
2738 specific transport protocol. A Client or Server MAY support any of the transport protocols listed here, or
2739 any other transport protocol that meets the requirements of this [section](#).

2740

2741 A Client or Server MUST support the use of one or more underlying transport protocols that provide an
2742 ordered, lossless, stream of bytes from the Client to Server and Server to Client [\[MQTT-4.2-1\]](#).

2743

2744 **Non-normative comment**

2745 TCP/IP as defined in [\[RFC0793\]](#) can be used for MQTT v5.0. The following transport protocols
2746 are also suitable:

- 2747 • TLS [\[RFC5246\]](#)
- 2748 • WebSocket [\[RFC6455\]](#)

2749

2750 **Non-normative comment**

2751 TCP ports 8883 and 1883 are registered with IANA for MQTT TLS and non-TLS communication
2752 respectively.

2753

2754 **Non-normative comment**

2755 Connectionless network transports such as User Datagram Protocol (UDP) are not suitable on
2756 their own because they might lose or reorder data.

2757

2758 **4.3 Quality of Service levels and protocol flows**

2759 MQTT delivers Application Messages according to the Quality of Service (QoS) levels defined in the
2760 following sections. The delivery protocol is symmetric, in the description below the Client and Server can
2761 each take the role of either sender or receiver. The delivery protocol is concerned solely with the delivery
2762 of an application message from a single sender to a single receiver. When the Server is delivering an
2763 Application Message to more than one Client, each Client is treated independently. The QoS level used
2764 to deliver an Application Message outbound to the Client could differ from that of the inbound Application
2765 Message.

2766

2767 **4.3.1 QoS 0: At most once delivery**

2768 The message is delivered according to the capabilities of the underlying network. No response is sent by
 2769 the receiver and no retry is performed by the sender. The message arrives at the receiver either once or
 2770 not at all.

2771

2772 In the QoS 0 delivery protocol, the sender

- 2773 • MUST send a PUBLISH packet with QoS 0 and DUP flag set to 0 [MQTT-4.3.1-1].

2774

2775 In the QoS 0 delivery protocol, the receiver

- 2776 • Accepts ownership of the message when it receives the PUBLISH packet.

2777

2778 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)

2779

2780 **4.3.2 QoS 1: At least once delivery**

2781 This Quality of Service level ensures that the message arrives at the receiver at least once. A QoS 1
 2782 PUBLISH packet has a Packet Identifier in its Variable Header and is acknowledged by a PUBACK packet.
 2783 Section 2.2.1 provides more information about Packet Identifiers.

2784

2785 In the QoS 1 delivery protocol, the sender

- 2786 • MUST assign an unused Packet Identifier each time it has a new Application Message to publish
 2787 [MQTT-4.3.2-1].
- 2788 • MUST send a PUBLISH packet containing this Packet Identifier with QoS 1 and DUP flag set to
 2789 0 [MQTT-4.3.2-2].
- 2790 • MUST treat the PUBLISH packet as “unacknowledged” until it has received the corresponding
 2791 PUBACK packet from the receiver. Refer to section 4.4 for a discussion of unacknowledged
 2792 messages [MQTT-4.3.2-3].

2793

2794 The Packet Identifier becomes available for reuse once the sender has received the PUBACK packet.

2795

2796 Note that a sender is permitted to send further PUBLISH packets with different Packet Identifiers while it is
 2797 waiting to receive acknowledgements.

2798

2799 In the QoS 1 delivery protocol, the receiver

- 2800 • MUST respond with a PUBACK packet containing the Packet Identifier from the incoming
 2801 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 ¹
	<-----	Send PUBACK <Packet Identifier>
Discard message		

¹ 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].

2837 The Packet Identifier becomes available for reuse once the sender has received the PUBCOMP packet or
 2838 a PUBREC with a Reason Code of 0x80 or greater.

2839

2840 Note that a sender is permitted to send further PUBLISH packets with different Packet Identifiers while it is
 2841 waiting to receive acknowledgements, subject to flow control as described in [section 4.9](#).

2842

2843 In the QoS 2 delivery protocol, the receiver:

- 2844 • MUST respond with a PUBREC containing the Packet Identifier from the incoming PUBLISH
 2845 packet, having accepted ownership of the Application Message [\[MQTT-4.3.3-8\]](#).
- 2846 • If it has sent a PUBREC with a Reason Code of 0x80 or greater, the receiver MUST treat any
 2847 subsequent PUBLISH packet that contains that Packet Identifier as being a new Application
 2848 Message [\[MQTT-4.3.3-9\]](#).
- 2849 • Until it has received the corresponding PUBREL packet, the receiver MUST acknowledge any
 2850 subsequent PUBLISH packet with the same Packet Identifier by sending a PUBREC. It MUST
 2851 NOT cause duplicate messages to be delivered to any onward recipients in this case [\[MQTT-
 2852 4.3.3-10\]](#).
- 2853 • MUST respond to a PUBREL packet by sending a PUBCOMP packet containing the same
 2854 Packet Identifier as the PUBREL [\[MQTT-4.3.3-11\]](#).
- 2855 • After it has sent a PUBCOMP, the receiver MUST treat any subsequent PUBLISH packet that
 2856 contains that Packet Identifier as being a new Application Message [\[MQTT-4.3.3-12\]](#).
- 2857 • MUST continue the QoS 2 acknowledgement sequence even if it has applied message expiry
 2858 [\[MQTT-4.3.3-13\]](#).

2859

2860 4.4 Message delivery retry

2861 When a Client reconnects with Clean Start set to 0 and a session is present, both the Client and Server
 2862 MUST resend any unacknowledged PUBLISH packets (where QoS > 0) and PUBREL packets using their
 2863 original Packet Identifiers. This is the only circumstance where a Client or Server is REQUIRED to resend
 2864 messages. Clients and Servers MUST NOT resend messages at any other time [\[MQTT-4.4.0-1\]](#).

2865

2866 If PUBACK or PUBREC is received containing a Reason Code of 0x80 or greater the corresponding
 2867 PUBLISH packet is treated as acknowledged, and MUST NOT be retransmitted [\[MQTT-4.4.0-2\]](#).

2868

2869 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 ¹
		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		

2870
2871
2872
2873
2874
2875
2876
2877

¹ 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.

2878 4.5 Message receipt

2879 When a Server takes ownership of an incoming Application Message it MUST add it to the Session State
2880 for those Clients that have matching Subscriptions [MQTT-4.5.0-1]. Matching rules are defined in section
2881 4.7.

2882
2883 Under normal circumstances Clients receive messages in response to Subscriptions they have created. A
2884 Client could also receive messages that do not match any of its explicit Subscriptions. This can happen if
2885 the Server automatically assigned a subscription to the Client. A Client could also receive messages
2886 while an UNSUBSCRIBE operation is in progress. The Client MUST acknowledge any Publish packet it
2887 receives according to the applicable QoS rules regardless of whether it elects to process the Application
2888 Message that it contains [MQTT-4.5.0-2].

2889

2890 4.6 Message ordering

2891 The following these rules apply to the Client when implementing the protocol flows defined in section 4.3.

- 2892 • When the Client re-sends any PUBLISH packets, it MUST re-send them in the order in which the
2893 original PUBLISH packets were sent (this applies to QoS 1 and QoS 2 messages) [MQTT-4.6.0-
2894 1]
- 2895 • The Client MUST send PUBACK packets in the order in which the corresponding PUBLISH
2896 packets were received (QoS 1 messages) [MQTT-4.6.0-2]
- 2897 • The Client MUST send PUBREC packets in the order in which the corresponding PUBLISH
2898 packets were received (QoS 2 messages) [MQTT-4.6.0-3]
- 2899 • The Client MUST send PUBREL packets in the order in which the corresponding PUBREC
2900 packets were received (QoS 2 messages) [MQTT-4.6.0-4]

2901

2902 An Ordered Topic is a Topic where the Client can be certain that the Application Messages in that Topic
2903 from the same Client and at the same QoS are received are in the order they were published. **When a**
2904 **Server processes a message that has been published to an Ordered Topic, it MUST send PUBLISH**
2905 **packets to consumers (for the same Topic and QoS) in the order that they were received from any given**
2906 **Client [MQTT-4.6.0-5].** This is addition to the rules listed above.

2907
2908 **By default, a Server MUST treat every Topic as an Ordered Topic when it is forwarding messages on**
2909 **Non-shared Subscriptions. [MQTT-4.6.0-6].** A Server MAY provide an administrative or other mechanism
2910 to allow one or more Topics to not be treated as an Ordered Topic.

2911
2912 **Non-normative comment**

2913 The rules listed above ensure that when a stream of messages is published and subscribed to an
2914 Ordered Topic with QoS 1, the final copy of each message received by the subscribers will be in
2915 the order that they were published. If the message is re-sent the duplicate message can be
2916 received after one of the earlier messages is received. For example, a publisher might send
2917 messages in the order 1,2,3,4 but the subscriber might receive them in the order 1,2,3,2,3,4 if
2918 there is a network disconnection after message 3 has been sent.

2919
2920 If both Client and Server set Receive Maximum to 1, they make sure that no more than one
2921 message is “in-flight” at any one time. In this case no QoS 1 message will be received after any
2922 later one even on re-connection. For example a subscriber might receive them in the order
2923 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
2924 Maximum is used.

2925

2926 **4.7 Topic Names and Topic Filters**

2927 **4.7.1 Topic wildcards**

2928 The topic level separator is used to introduce structure into the Topic Name. If present, it divides the
2929 Topic Name into multiple “topic levels”.

2930 A subscription’s Topic Filter can contain special wildcard characters, which allow a Client to subscribe to
2931 multiple topics at once.

2932 **The wildcard characters can be used in Topic Filters, but MUST NOT be used within a Topic Name**
2933 **[MQTT-4.7.0-1].**

2934

2935 **4.7.1.1 Topic level separator**

2936 The forward slash (‘/’ U+002F) is used to separate each level within a topic tree and provide a hierarchical
2937 structure to the Topic Names. The use of the topic level separator is significant when either of the two
2938 wildcard characters is encountered in Topic Filters specified by subscribing Clients. Topic level separators
2939 can appear anywhere in a Topic Filter or Topic Name. Adjacent Topic level separators indicate a zero-
2940 length topic level.

2941

2942 **4.7.1.2 Multi-level wildcard**

2943 The number sign (‘#’ U+0023) is a wildcard character that matches any number of levels within a topic.
2944 The multi-level wildcard represents the parent and any number of child levels. **The multi-level wildcard**
2945 **character MUST be specified either on its own or following a topic level separator. In either case it MUST**
2946 **be the last character specified in the Topic Filter [MQTT-4.7.1-1].**

2947

2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961

2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978

2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990

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

- 2991 • A subscription to “#” will not receive any messages published to a topic beginning with a \$
- 2992 • A subscription to “+/monitor/Clients” will not receive any messages published to
- 2993 “\$SYS/monitor/Clients”
- 2994 • A subscription to “\$SYS/#” will receive messages published to topics beginning with “\$SYS/”
- 2995 • A subscription to “\$SYS/monitor/+” will receive messages published to
- 2996 “\$SYS/monitor/Clients”
- 2997 • For a Client to receive messages from topics that begin with \$SYS/ and from topics that don't
- 2998 begin with a \$, it has to subscribe to both “#” and “\$SYS/#”
- 2999

3000 4.7.3 Topic semantic and usage

3001 The following rules apply to Topic Names and Topic Filters:

- 3002 • All Topic Names and Topic Filters MUST be at least one character long [MQTT-4.7.3-1]
- 3003 • Topic Names and Topic Filters are case sensitive
- 3004 • Topic Names and Topic Filters can include the space character
- 3005 • A leading or trailing '/' creates a distinct Topic Name or Topic Filter
- 3006 • A Topic Name or Topic Filter consisting only of the '/' character is valid
- 3007 • Topic Names and Topic Filters MUST NOT include the null character (Unicode U+0000) [Unicode]
- 3008 [MQTT-4.7.3-2]
- 3009 • Topic Names and Topic Filters are UTF-8 Encoded Strings; they MUST NOT encode to more than
- 3010 65,535 bytes [MQTT-4.7.3-3]. Refer to section 1.5.4.
- 3011

3012 There is no limit to the number of levels in a Topic Name or Topic Filter, other than that imposed by the

3013 overall length of a UTF-8 Encoded String.

3014

3015 When it performs subscription matching the Server MUST NOT perform any normalization of Topic

3016 Names or Topic Filters, or any modification or substitution of unrecognized characters [MQTT-4.7.3-4].

3017 Each non-wildcarded level in the Topic Filter has to match the corresponding level in the Topic Name

3018 character for character for the match to succeed.

3019

3020 Non-normative comment

3021 The UTF-8 encoding rules mean that the comparison of Topic Filter and Topic Name could be

3022 performed either by comparing the encoded UTF-8 bytes, or by comparing decoded Unicode

3023 characters

3024

3025 Non-normative comment

- 3026 • “ACCOUNTS” and “Accounts” are two different Topic Names
- 3027 • “Accounts payable” is a valid Topic Name
- 3028 • “/finance” is different from “finance”
- 3029

3030 An Application Message is sent to each Client Subscription whose Topic Filter matches the Topic Name

3031 attached to an Application Message. The topic resource MAY be either predefined in the Server by an

3032 administrator or it MAY be dynamically created by the Server when it receives the first subscription or an

3033 Application Message with that Topic Name. The Server MAY also use a security component to authorize

3034 particular actions on the topic resource for a given Client.

3035

3036 **4.8 Subscriptions**

3037 MQTT provides two kinds of Subscription, Shared and Non-shared.

3038

3039 **Non-normative comment**

3040 In earlier versions of MQTT all Subscriptions are Non-shared.

3041

3042 **4.8.1 Non-shared Subscriptions**

3043 A Non-shared Subscription is associated only with the MQTT Session that created it. Each Subscription
3044 includes a Topic Filter, indicating the topic(s) for which messages are to be delivered on that Session,
3045 and Subscription Options. The Server is responsible for collecting messages that match the filter and
3046 transmitting them on the Session's MQTT connection if and when that connection is active.

3047

3048 A Session cannot have more than one Non-shared Subscription with the same Topic Filter, so the Topic
3049 Filter can be used as a key to identify the subscription within that Session.

3050

3051 If there are multiple Clients, each with its own Non-shared Subscription to the same Topic, each Client
3052 gets its own copy of the Application Messages that are published on that Topic. This means that the
3053 Non-shared Subscriptions cannot be used to load-balance Application Messages across multiple
3054 consuming Clients as in such cases every message is delivered to every subscribing Client.

3055

3056 **4.8.2 Shared Subscriptions**

3057 A Shared Subscription can be associated with multiple subscribing MQTT Sessions. Like a Non-shared
3058 Subscription, it has a Topic Filter and Subscription Options; however, a publication that matches its Topic
3059 Filter is only sent to one of its subscribing Sessions. Shared Subscriptions are useful where several
3060 consuming Clients share the processing of the publications in parallel.

3061

3062 A Shared Subscription is identified using a special style of Topic Filter. The format of this filter is:

3063

3064 `$share/{ShareName}/{filter}`

- 3065 • `$share` is a literal string that marks the Topic Filter as being a Shared Subscription Topic Filter.
- 3066 • `{ShareName}` is a character string that does not include `/`, `+` or `#`
- 3067 • `{filter}` The remainder of the string has the same syntax and semantics as a Topic Filter in a non-
3068 shared subscription. Refer to [section 4.7](#).

3069

3070 A Shared Subscription's Topic Filter MUST start with `$share/` and MUST contain a `ShareName` that is at
3071 least one character long [\[MQTT-4.8.2-1\]](#). The `ShareName` MUST NOT contain the characters `/`, `+` or
3072 `#`, but MUST be followed by a `/` character. This `/` character MUST be followed by a Topic Filter
3073 [\[MQTT-4.8.2-2\]](#) as described in [section 4.7](#).

3074

3075 **Non-normative comment**

3076 Shared Subscriptions are defined at the scope of the MQTT Server, rather than of a Session. A
3077 `ShareName` is included in the Shared Subscription's Topic Filter so that there can be more than
3078 one Shared Subscription on a Server that has the same `{filter}` component. Typically, applications
3079 use the `ShareName` to represent the group of subscribing Sessions that are sharing the
3080 subscription.

3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133

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.

- 3134 • Different subscribing Clients are permitted to ask for different Requested QoS levels in their
3135 SUBSCRIBE packets. The Server decides which Maximum QoS to grant to each Client, and it is
3136 permitted to grant different Maximum QoS levels to different subscribers. When sending an
3137 Application Message to a Client, **the Server MUST respect the granted QoS for the Client's**
3138 **subscription** [MQTT-4.8.2-3], in the same that it does when sending a message to a -Subscriber.
3139
- 3140 • If the Server is in the process of sending a QoS 2 message to its chosen subscribing Client and the
3141 connection to the Client breaks before delivery is complete, **the Server MUST complete the delivery**
3142 **of the message to that Client when it reconnects** [MQTT-4.8.2-4] as described in [section 4.3.3](#). If the
3143 Client's Session terminates before the Client reconnects, **the Server MUST NOT send the Application**
3144 **Message to any other subscribed Client** [MQTT-4.8.2-5].
3145
- 3146 • If the Server is in the process of sending a QoS 1 message to its chosen subscribing Client and the
3147 connection to that Client breaks before the Server has received an acknowledgement from the Client,
3148 the Server MAY wait for the Client to reconnect and retransmit the message to that Client. If the
3149 Client's Session terminates before the Client reconnects, the Server SHOULD send the Application
3150 Message to another Client that is subscribed to the same Shared Subscription. It MAY attempt to
3151 send the message to another Client as soon as it loses its connection to the first Client.
3152
- 3153 • **If a Client responds with a PUBACK or PUBREC containing a Reason Code of 0x80 or greater to a**
3154 **PUBLISH packet from the Server, the Server MUST discard the Application Message and not attempt**
3155 **to send it to any other Subscriber** [MQTT-4.8.2-6].
3156
- 3157 • A Client is permitted to submit a second SUBSCRIBE request to a Shared Subscription on a Session
3158 that's already subscribed to that Shared Subscription. For example, it might do this to change the
3159 Requested QoS for its subscription or because it was uncertain that the previous subscribe
3160 completed before the previous connection was closed. This does not increase the number of times
3161 that the Session is associated with the Shared Subscription, so the Session will leave the Shared
3162 Subscription on its first UNSUBSCRIBE.
3163
- 3164 • Each Shared Subscription is independent from any other. It is possible to have two Shared
3165 Subscriptions with overlapping filters. In such cases a message that matches both Shared
3166 Subscriptions will be processed separately by both of them. If a Client has a Shared Subscription and
3167 a Non-shared Subscription and a message matches both of them, the Client will receive a copy of the
3168 message by virtue of it having the Non-shared Subscription. A second copy of the message will be
3169 delivered to one of the subscribers to the Shared Subscription, and this could result in a second copy
3170 being sent to this Client.

3171

3172 4.9 Flow Control

3173 Clients and Servers control the number of unacknowledged PUBLISH packets they receive by using a
3174 Receive Maximum value as described in [section 3.1.2.11.4](#) and [section 3.2.2.3.2](#). The Receive Maximum
3175 establishes a send quota which is used to limit the number of PUBLISH QoS > 0 packets which can be
3176 sent without receiving an PUBACK (for QoS 1) or PUBCOMP (for QoS 2). The PUBACK and PUBCOMP
3177 replenish the quota in the manner described below.

3178

3179 **The Client or Server MUST set its initial send quota to a non-zero value not exceeding the Receive**
3180 **Maximum** [MQTT-4.9.0-1].

3181

3182 **Each time the Client or Server sends a PUBLISH packet at QoS > 0, it decrements the send quota. If the**
3183 **send quota reaches zero, the Client or Server MUST NOT send any more PUBLISH packets with QoS >**
3184 **0** [MQTT-4.9.0-2]. It MAY continue to send PUBLISH packets with QoS 0, or it MAY choose to suspend
3185 sending these as well. **The Client and Server MUST continue to process and respond to all other MQTT**
3186 **Control Packets even if the quota is zero** [MQTT-4.9.0-3].

3187
3188 The send quota is incremented by 1:
3189 • Each time a PUBACK or PUBCOMP packet is received, regardless of whether the PUBACK or
3190 PUBCOMP carried an error code.
3191 • Each time a PUBREC packet is received with a Return Code of 0x80 or greater.
3192
3193 The send quota is not incremented if it is already equal to the initial send quota. The attempt to increment
3194 above the initial send quota might be caused by the re-transmission of a PUBREL packet after a new
3195 Network Connection is established.
3196
3197 Refer to [section 3.3.4](#) for a description of how Clients and Servers react if they are sent more PUBLISH
3198 packets than the Receive Maximum allows.
3199
3200 The send quota and Receive Maximum value are not preserved across Network Connections, and are re-
3201 initialized with each new Network Connection as described above. They are not part of the session state.
3202

3203 **4.10 Request / Response**

3204 Some applications or standards might wish to run a Request/Response interaction over MQTT. This
3205 version of MQTT includes three properties that can be used for this purpose:

- 3206 • Response Topic, described in [section 3.3.2.3.5](#)
- 3207 • Correlation Data, described in [section 3.3.2.3.6](#)
- 3208 • Request Response Information, described in [section 3.1.2.11.7](#)
- 3209 • Response Information, described in [section 3.2.2.3.14](#)

3210 The following non-normative sections describe how these properties can be used.

3211
3212 A Client sends a Request Message by publishing an Application Message which has a Response Topic
3213 set as described in [section 3.3.2.3.5](#). The Request can include a Correlation Data property as described
3214 in [section 3.3.2.3.6](#).

3215

3216 **4.10.1 Basic Request Response (non-normative)**

3217 Request/Response interaction proceeds as follows:

- 3218 1. An MQTT Client (the Requester) publishes a Request Message to a topic. A Request Message
3219 is an Application Message with a Response Topic.
- 3220 2. Another MQTT Client (the Responder) has subscribed to a Topic Filter which matches the Topic
3221 Name used when the Request Message was published. As a result, it receives the Request
3222 Message. There could be multiple Responders subscribed to this Topic Name or there could be
3223 none.
- 3224 3. The Responder takes the appropriate action based on the Request Message, and then publishes
3225 a Response Message to the Topic Name in the Response Topic property that was carried in the
3226 Request Message.
- 3227 4. In typical usage the Requester has subscribed to the Response Topic and thereby receives the
3228 Response Message. However, some other Client might be subscribed to the Response Topic in
3229 which case the Response Message will also be received and processed by that Client. As with
3230 the Request Message, the topic on which the Response Message is sent could be subscribed to
3231 by multiple Clients, or by none.

3232

3233 If the Request Message contains a Correlation Data property, the Responder copies this property into the
3234 Response Message and this is used by the receiver of the Response Message to associate the
3235 Response Message with the original request. The Response Message does not include a Response
3236 Topic property.

3237
3238 The MQTT Server forwards the Response Topic and Correlation Data Property in the Request Message
3239 and the Correlation Data in the Response Message. The Server treats the Request Message and the
3240 Response Message like any other Application Message.

3241
3242 The Requester normally subscribes to the Response Topic before publishing a Request Message. If there
3243 are no subscribers to the Response Topic when the Response Message is sent, the Response Message
3244 will not be delivered to any Client.

3245
3246 The Request Message and Response Message can be of any QoS, and the Responder can be using a
3247 Session with a non-zero Session Expiry Interval. It is common to send Request Messages at QoS 0 and
3248 only when the Responder is expected to be connected. However, this is not necessary.

3249
3250 The Responder can use a Shared Subscription to allow for a pool of responding Clients. Note however
3251 that when using Shared Subscriptions that the order of message delivery is not guaranteed between
3252 multiple Clients.

3253
3254 It is the responsibility of the Requester to make sure it has the necessary authority to publish to the
3255 request topic, and to subscribe to the Topic Name that it sets in the Response Topic property. It is the
3256 responsibility of the Responder to make sure it has the authority to subscribe to the request topic and
3257 publish to the Response Topic. While topic authorization is outside of this specification, it is
3258 recommended that Servers implement such authorization.

3259

3260 **4.10.2 Determining a Response Topic value (non-normative)**

3261 Requesters can determine a Topic Name to use as their Response Topic in any manner they choose
3262 including via local configuration. To avoid clashes between different Requesters, it is desirable that the
3263 Response Topic used by a Requester Client be unique to that Client. As the Requester and Responder
3264 commonly need to be authorized to these topics, it can be an authorization challenge to use a random
3265 Topic Name.

3266
3267 To help with this problem, this specification defines a property in the CONNACK packet called Response
3268 Information. The Server can use this property to guide the Client in its choice for the Response Topic to
3269 use. This mechanism is optional for both the Client and the Server. At connect time, the Client requests
3270 that the Server send a Response Information by setting the Request Response Information property in
3271 the CONNECT packet. This causes the Server to insert a Response Information property (a UTF-8
3272 Encoded String) sent in the CONNACK packet.

3273
3274 This specification does not define the contents of the Response Information but it could be used to pass a
3275 globally unique portion of the topic tree which is reserved for that Client for at least the lifetime of its
3276 Session. Using this mechanism allows this configuration to be done once in the Server rather than in
3277 each Client.

3278
3279 Refer to [section 3.1.2.11.7](#) for the definition of the Response Information.

3280

3281 4.11 Server redirection

3282 A Server can request that the Client uses another Server by sending CONNACK or DISCONNECT with
3283 Reason Codes 0x9C (Use another server), or 0x9D (Server moved) as described in [section 4.13](#). When
3284 sending one of these Reason Codes, the Server MAY also include a Server Reference property to
3285 indicate the location of the Server or Servers the Client SHOULD use.

3286
3287 The Reason Code 0x9C (Use another server) specifies that the Client SHOULD temporarily switch to
3288 using another Server. The other Server is either already known to the Client, or is specified using a
3289 Server Reference.

3290
3291 The Reason Code 0x9D (Server moved) specifies that the Client SHOULD permanently switch to using
3292 another Server. The other Server is either already known to the Client, or is specified using a Server
3293 Reference.

3294
3295 The Server Reference is a UTF-8 Encoded String. The value of this string is a space separated list of
3296 references. The format of references is not specified here.

3297
3298 **Non-normative comment**

3299 It is recommended that each reference consists of a name optionally followed by a colon and a
3300 port number. If the name contains a colon the name string can be enclosed within square
3301 brackets (“[“ and ”]”). A name enclosed by square brackets cannot contain the right square
3302 bracket (“]”) character. This is used to represent an IPv6 literal address which uses colon
3303 separators. This is a simplified version of an URI authority as described in [\[RFC3986\]](#).

3304
3305 **Non-normative comment**

3306 The name within a Server Reference commonly represents a host name, DNS name [\[RFC1035\]](#),
3307 SRV name [\[RFC2782\]](#) , or literal IP address. The value following the colon separator is commonly
3308 a port number in decimal. This is not needed where the port information comes from the name
3309 resolution (such as with SRV) or is defaulted.

3310
3311 **Non-normative comment**

3312 If multiple references are given, the expectation is that that Client will choose one of them.

3313
3314 **Non-normative comment**

3315 Examples of the Server Reference are:

```
3316 myserver.xyz.org  
3317 myserver.xyz.org:8883  
3318 10.10.151.22:8883 [fe80::9610:3eff:fe1c]:1883
```

3319
3320 The Server is allowed to not ever send a Server Reference, and the Client is allowed to ignore a Server
3321 Reference. This feature can be used to allow for load balancing, Server relocation, and Client
3322 provisioning to a Server.

3323 3324 4.12 Enhanced authentication

3325 The MQTT CONNECT packet supports basic authentication of a Network Connection using the User
3326 Name and Password fields. While these fields are named for a simple password authentication, they can
3327 be used to carry other forms of authentication such as passing a token as the Password.

3328

3329 Enhanced authentication extends this basic authentication to include challenge / response style
3330 authentication. It might involve the exchange of AUTH packets between the Client and the Server after
3331 the CONNECT and before the CONNACK packets.

3332

3333 To begin an enhanced authentication, the Client includes an Authentication Method in the CONNECT
3334 packet. This specifies the authentication method to use. If the Server does not support the Authentication
3335 Method supplied by the Client, it MAY send a CONNACK with a Reason Code of 0x8C (Bad
3336 authentication method) or 0x87 (Not Authorized) as described in section 4.13 and MUST close the
3337 Network Connection [MQTT-4.12.0-1].

3338

3339 The Authentication Method is an agreement between the Client and Server about the meaning of the data
3340 sent in the Authentication Data and any of the other fields in CONNECT, and the exchanges and
3341 processing needed by the Client and Server to complete the authentication.

3342

3343 **Non-normative comment**

3344 The Authentication Method is commonly a SASL mechanism, and using such a registered name
3345 aids interchange. However, the Authentication Method is not constrained to using registered
3346 SASL mechanisms.

3347

3348 If the Authentication Method selected by the Client specifies that the Client sends data first, the Client
3349 SHOULD include an Authentication Data property in the CONNECT packet. This property can be used to
3350 provide data as specified by the Authentication Method. The contents of the Authentication Data are
3351 defined by the authentication method.

3352

3353 If the Server requires additional information to complete the authentication, it can send an AUTH packet
3354 to the Client. This packet MUST contain a Reason Code of 0x18 (Continue authentication) [MQTT-4.12.0-
3355 2]. If the authentication method requires the Server to send authentication data to the Client, it is sent in
3356 the Authentication Data.

3357

3358 The Client responds to an AUTH packet from the Server by sending a further AUTH packet. This packet
3359 MUST contain a Reason Code of 0x18 (Continue authentication) [MQTT-4.12.0-3]. If the authentication
3360 method requires the Client to send authentication data for the Server, it is sent in the Authentication Data.

3361

3362 The Client and Server exchange AUTH packets as needed until the Server accepts the authentication by
3363 sending a CONNACK with a Reason Code of 0. If the acceptance of the authentication requires data to
3364 be sent to the Client, it is sent in the Authentication Data.

3365

3366 The Client can close the connection at any point in this process. It MAY send a DISCONNECT packet
3367 before doing so. The Server can reject the authentication at any point in this process. It MAY send a
3368 CONNACK with a Reason Code of 0x80 or above as described in section 4.13, and MUST close the
3369 Network Connection [MQTT-4.12.0-4].

3370

3371 If the initial CONNECT packet included an Authentication Method property then all AUTH packets, and
3372 any successful CONNACK packet MUST include an Authentication Method Property with the same value
3373 as in the CONNECT packet [MQTT-4.12.0-5].

3374

3375 The implementation of enhanced authentication is OPTIONAL for both Clients and Servers. If the Client
3376 does not include an Authentication Method in the CONNECT, the Server MUST NOT send an AUTH

3377 packet, and it MUST NOT send an Authentication Method in the CONNACK packet [MQTT-4.12.0-6]. If
3378 the Client does not include an Authentication Method in the CONNECT, the Client MUST NOT send an
3379 AUTH packet to the Server [MQTT-4.12.0-7].

3380

3381 If the Client does not include an Authentication Method in the CONNECT packet, the Server SHOULD
3382 authenticate using some or all of the information in the CONNECT packet, TLS session, and Network
3383 Connection.

3384

3385 **Non-normative example showing a SCRAM challenge**

- 3386 • Client to Server: CONNECT Authentication Method="SCRAM-SHA-1" Authentication
3387 Data=client-first-data
- 3388 • Server to Client: AUTH rc=0x18 Authentication Method="SCRAM-SHA-1" Authentication
3389 Data=server-first-data
- 3390 • Client to Server AUTH rc=0x18 Authentication Method="SCRAM-SHA-1" Authentication
3391 Data=client-final-data
- 3392 • Server to Client CONNACK rc=0 Authentication Method="SCRAM-SHA-1" Authentication
3393 Data=server-final-data

3394

3395 **Non-normative example showing a Kerberos challenge**

- 3396 • Client to Server CONNECT Authentication Method="GS2-KRB5"
- 3397 • Server to Client AUTH rc=0x18 Authentication Method="GS2-KRB5"
- 3398 • Client to Server AUTH rc=0x18 Authentication Method="GS2-KRB5" Authentication
3399 Data=initial context token
- 3400 • Server to Client AUTH rc=0x18 Authentication Method="GS2-KRB5" Authentication
3401 Data=reply context token
- 3402 • Client to Server AUTH rc=0x18 Authentication Method="GS2-KRB5"
- 3403 • Server to Client CONNACK rc=0 Authentication Method="GS2-KRB5" Authentication
3404 Data=outcome of authentication

3405

3406 **4.12.1 Re-authentication**

3407 If the Client supplied an Authentication Method in the CONNECT packet it can initiate a re-authentication
3408 at any time after receiving a CONNACK. It does this by sending an AUTH packet with a Reason Code of
3409 0x19 (Re-authentication). The Client MUST set the Authentication Method to the same value as the
3410 Authentication Method originally used to authenticate the Network Connection [MQTT-4.12.1-1]. If the
3411 authentication method requires Client data first, this AUTH packet contains the first piece of
3412 authentication data as the Authentication Data.

3413

3414 The Server responds to this re-authentication request by sending an AUTH packet to the Client with a
3415 Reason Code of 0x00 (Success) to indicate that the re-authentication is complete, or a Reason Code of
3416 0x18 (Continue authentication) to indicate that more authentication data is needed. The Client can
3417 respond with additional authentication data by sending an AUTH packet with a Reason Code of 0x18
3418 (Continue authentication). This flow continues as with the original authentication until the re-
3419 authentication is complete or the re-authentication fails.

3420

3421 If the re-authentication fails, the Client or Server SHOULD send DISCONNECT with an appropriate
3422 Reason Code as described in section 4.13, and MUST close the Network Connection [MQTT-4.12.1-2].

3423

3424 During this re-authentication sequence, the flow of other packets between the Client and Server can
3425 continue using the previous authentication.

3426
3427
3428
3429
3430
3431

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

3434 Definitions of Malformed Packet and Protocol Errors are contained in [section 1.2 Terminology](#), some but
3435 not all, of these error cases are noted throughout the specification. The rigor with which a Client or Server
3436 checks an MQTT Control Packet it has received will be a compromise between:

- 3437 • The size of the Client or Server implementation.
- 3438 • The capabilities that the implementation supports.
- 3439 • The degree to which the receiver trusts the sender to send correct MQTT Control Packets.
- 3440 • The degree to which the receiver trusts the network to deliver MQTT Control Packets correctly.
- 3441 • The consequences of continuing to process a packet that is incorrect.

3442

3443 If the sender is compliant with this specification it will not send Malformed Packets or cause Protocol
3444 Errors. However, if a Client sends MQTT Control Packets before it receives CONNACK, it might cause a
3445 Protocol Error because it made an incorrect assumption about the Server capabilities. Refer to [section](#)
3446 [3.1.4 CONNECT Actions](#).

3447

3448 The Reason Codes used for Malformed Packet and Protocol Errors are:

- 3449 • 0x81 Malformed Packet
- 3450 • 0x82 Protocol Error
- 3451 • 0x93 Receive Maximum exceeded
- 3452 • 0x95 Packet too large
- 3453 • 0x9A Retain not supported
- 3454 • 0x9B QoS not supported
- 3455 • 0x9E Shared Subscription not supported
- 3456 • 0xA1 Subscription Identifiers not supported
- 3457 • 0xA2 Wildcard Subscription not supported

3458

3459 When a Client detects a Malformed Packet or Protocol Error, and a Reason Code is given in the
3460 specification, it SHOULD close the Network Connection. In the case of an error in a AUTH packet it MAY
3461 send a DISCONNECT packet containing the reason code, before closing the Network Connection. In the
3462 case of an error in any other packet it SHOULD send a DISCONNECT packet containing the reason code
3463 before closing the Network Connection. Use Reason Code 0x81 (Malformed Packet) or 0x82 (Protocol
3464 Error) unless a more specific Reason Code has been defined in [section 3.14.2.1 Disconnect Reason](#)
3465 [Code](#).

3466

3467 **When a Server detects a Malformed Packet or Protocol Error, and a Reason Code is given in the**
3468 **specification, it MUST close the Network Connection [MQTT-4.13.1-1].** In the case of an error in a
3469 CONNECT packet it MAY send a CONNACK packet containing the Reason Code, before closing the
3470 Network Connection. In the case of an error in any other packet it SHOULD send a DISCONNECT packet
3471 containing the Reason Code before closing the Network Connection. Use Reason Code 0x81 (Malformed
3472 Packet) or 0x82 (Protocol Error) unless a more specific Reason Code has been defined in [section 3.2.2.2](#)

3473 - [Connect Reason Code](#) or in section [3.14.2.1 – Disconnect Reason Code](#). There are no consequences
3474 for other Sessions.

3475

3476 If either the Server or Client omits to check some feature of an MQTT Control Packet, it might fail to
3477 detect an error, consequently it might allow data to be damaged.

3478

3479 **4.13.2 Other errors**

3480 Errors other than Malformed Packet and Protocol Errors cannot be anticipated by the sender because the
3481 receiver might have constraints which it has not communicated to the sender. A receiving Client or Server
3482 might encounter a transient error, such as a shortage of memory, that prevents successful processing of
3483 an individual MQTT Control Packet.

3484

3485 Acknowledgment packets PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK with a
3486 Reason Code of 0x80 or greater indicate that the received packet, identified by a Packet Identifier, was in
3487 error. There are no consequences for other Sessions or other Packets flowing on the same Session.

3488

3489 The CONNACK and DISCONNECT packets allow a Reason Code of 0x80 or greater to indicate that the
3490 Network Connection will be closed. If a Reason Code of 0x80 or greater is specified, then the Network
3491 Connection MUST be closed whether or not the CONNACK or DISCONNECT is sent [MQTT-4.13.2-1].
3492 Sending of one of these Reason Codes does not have consequence for any other Session.

3493

3494 If the Control Packet contains multiple errors the receiver of the Packet can validate the Packet in any
3495 order and take the appropriate action for any of the errors found.

3496

3497 5 Security (non-normative)

3498 5.1 Introduction

3499 It is strongly recommended that Server implementations that offer TLS [\[RFC5246\]](#) should use TCP port
3500 8883 (IANA service name: secure-mqtt).

3501
3502 Security is a fast changing world, so always use the latest recommendations when designing a secure
3503 solution.

3504
3505 There are a number of threats that solution providers should consider. For example:

- 3506 • Devices could be compromised
- 3507 • Data at rest in Clients and Servers might be accessible
- 3508 • Protocol behaviors could have side effects (e.g. “timing attacks”)
- 3509 • Denial of Service (DoS) attacks
- 3510 • Communications could be intercepted, altered, re-routed or disclosed
- 3511 • Injection of spoofed MQTT Control Packets

3512
3513 MQTT solutions are often deployed in hostile communication environments. In such cases,
3514 implementations will often need to provide mechanisms for:

- 3515 • Authentication of users and devices
- 3516 • Authorization of access to Server resources
- 3517 • Integrity of MQTT Control Packets and application data contained therein
- 3518 • Privacy of MQTT Control Packets and application data contained therein

3519
3520 As a transport protocol, MQTT is concerned only with message transmission and it is the implementer’s
3521 responsibility to provide appropriate security features. This is commonly achieved by using TLS
3522 [\[RFC5246\]](#).

3523
3524 In addition to technical security issues there could also be geographic (e.g. U.S.-EU Privacy Shield
3525 Framework [\[USEUPRIVSH\]](#)), industry specific (e.g. PCI DSS [\[PCIDSS\]](#)) and regulatory considerations
3526 (e.g. Sarbanes-Oxley [\[SARBANES\]](#)).

3527

3528 5.2 MQTT solutions: security and certification

3529 An implementation might want to provide conformance with specific industry security standards such as
3530 NIST Cyber Security Framework [\[NISTCSF\]](#), PCI-DSS [\[PCIDSS\]](#), FIPS-140-2 [\[FIPS1402\]](#) and NSA Suite
3531 B [\[NSAB\]](#).

3532
3533 Guidance on using MQTT within the NIST Cyber Security Framework [\[NISTCSF\]](#) can be found in the
3534 MQTT supplemental publication, MQTT and the NIST Framework for Improving Critical Infrastructure
3535 Cybersecurity [\[MQTTNIST\]](#). The use of industry proven, independently verified and certified technologies
3536 will help meet compliance requirements.

3537

3538 **5.3 Lightweight cryptography and constrained devices**

3539 Advanced Encryption Standard [AES] is the most widely adopted encryption algorithm. There is hardware
3540 support for AES in many processors, but not commonly for embedded processors. The encryption
3541 algorithm ChaCha20 [CHACHA20] encrypts and decrypts much faster in software, but is not as widely
3542 available as AES.

3543

3544 ISO 29192 [ISO29192] makes recommendations for cryptographic primitives specifically tuned to perform
3545 on constrained “low end” devices.

3546

3547 **5.4 Implementation notes**

3548 There are many security concerns to consider when implementing or using MQTT. The following section
3549 should not be considered a “check list”.

3550

3551 An implementation might want to achieve some, or all, of the following:

3552

3553 **5.4.1 Authentication of Clients by the Server**

3554 The CONNECT packet contains User Name and Password fields. Implementations can choose how to
3555 make use of the content of these fields. They may provide their own authentication mechanism, use an
3556 external authentication system such as LDAP [RFC4511] or OAuth [RFC6749] tokens, or leverage
3557 operating system authentication mechanisms.

3558

3559 MQTT v5.0 provides an enhanced authentication mechanism as described in [section 4.12](#). Using this
3560 requires support for it in both the Client and Server.

3561

3562 Implementations passing authentication data in clear text, obfuscating such data elements or requiring no
3563 authentication data should be aware this can give rise to Man-in-the-Middle and replay attacks. [Section](#)
3564 [5.4.5](#) introduces approaches to ensure data privacy.

3565

3566 A Virtual Private Network (VPN) between the Clients and Servers can provide confidence that data is only
3567 being received from authorized Clients.

3568

3569 Where TLS [RFC5246] is used, TLS Certificates sent from the Client can be used by the Server to
3570 authenticate the Client.

3571

3572 An implementation might allow for authentication where the credentials are sent in an Application
3573 Message from the Client to the Server.

3574

3575 **5.4.2 Authorization of Clients by the Server**

3576 If a Client has been successfully authenticated, a Server implementation should check that it is authorized
3577 before accepting its connection.

3578

3579 Authorization may be based on information provided by the Client such as User Name, the hostname/IP
3580 address of the Client, or the outcome of authentication mechanisms.

3581

3582 In particular, the implementation should check that the Client is authorized to use the Client Identifier as
3583 this gives access to the MQTT Session State (described in [section 4.1](#)). This authorization check is to
3584 protect against the case where one Client, accidentally or maliciously, provides a Client Identifier that is
3585 already being used by some other Client.

3586

3587 An implementation should provide access controls that take place after CONNECT to restrict the Clients
3588 ability to publish to particular Topics or to subscribe using particular Topic Filters. An implementation
3589 should consider limiting access to Topic Filters that have broad scope, such as the # Topic Filter.

3590

3591 **5.4.3 Authentication of the Server by the Client**

3592 The MQTT protocol is not trust symmetrical. When using basic authentication, there is no mechanism for
3593 the Client to authenticate the Server. Some forms of extended authentication do allow for mutual
3594 authentication.

3595

3596 Where TLS [\[RFC5246\]](#) is used, TLS Certificates sent from the Server can be used by the Client to
3597 authenticate the Server. Implementations providing MQTT service for multiple hostnames from a single IP
3598 address should be aware of the Server Name Indication extension to TLS defined in section 3 of
3599 [\[RFC6066\]](#). This allows a Client to tell the Server the hostname of the Server it is trying to connect to.

3600

3601 An implementation might allow for authentication where the credentials are sent in an Application
3602 Message from the Server to the Client. MQTT v5.0 provides an enhanced authentication mechanism as
3603 described in [section 4.12.](#), which can be used to Authenticate the Server to the Client. Using this requires
3604 support for it in both the Client and Server.

3605

3606 A VPN between Clients and Servers can provide confidence that Clients are connecting to the intended
3607 Server.

3608

3609 **5.4.4 Integrity of Application Messages and MQTT Control Packets**

3610 Applications can independently include hash values in their Application Messages. This can provide
3611 integrity of the contents of Publish packets across the network and at rest.

3612

3613 TLS [\[RFC5246\]](#) provides hash algorithms to verify the integrity of data sent over the network.

3614

3615 The use of VPNs to connect Clients and Servers can provide integrity of data across the section of the
3616 network covered by a VPN.

3617

3618 **5.4.5 Privacy of Application Messages and MQTT Control Packets**

3619 TLS [\[RFC5246\]](#) can provide encryption of data sent over the network. There are valid TLS cipher suites
3620 that include a NULL encryption algorithm that does not encrypt data. To ensure privacy Clients and
3621 Servers should avoid these cipher suites.

3622

3623 An application might independently encrypt the contents of its Application Messages. This could provide
3624 privacy of the Application Message both over the network and at rest. This would not provide privacy for
3625 other Properties of the Application Message such as Topic Name.

3626
3627 Client and Server implementations can provide encrypted storage for data at rest such as Application
3628 Messages stored as part of a Session.

3629
3630 The use of VPNs to connect Clients and Servers can provide privacy of data across the section of the
3631 network covered by a VPN.

3632

3633 **5.4.6 Non-repudiation of message transmission**

3634 Application designers might need to consider appropriate strategies to achieve end to end non-
3635 repudiation.

3636

3637 **5.4.7 Detecting compromise of Clients and Servers**

3638 Client and Server implementations using TLS [\[RFC5246\]](#) should provide capabilities to ensure that any
3639 TLS certificates provided when initiating a TLS connection are associated with the hostname of the Client
3640 connecting or Server being connected to.

3641

3642 Client and Server implementations using TLS can choose to provide capabilities to check Certificate
3643 Revocation Lists (CRLs [\[RFC5280\]](#)) and Online Certificate Status Protocol (OSCP) [\[RFC6960\]](#) to prevent
3644 revoked certificates from being used.

3645

3646 Physical deployments might combine tamper-proof hardware with the transmission of specific data in
3647 Application Messages. For example, a meter might have an embedded GPS to ensure it is not used in an
3648 unauthorized location. [\[IEEE8021AR\]](#) is a standard for implementing mechanisms to authenticate a
3649 device's identity using a cryptographically bound identifier.

3650

3651 **5.4.8 Detecting abnormal behaviors**

3652 Server implementations might monitor Client behavior to detect potential security incidents. For example:

- 3653
- 3654 • Repeated connection attempts
 - 3655 • Repeated authentication attempts
 - 3656 • Abnormal termination of connections
 - 3657 • Topic scanning (attempts to send or subscribe to many topics)
 - 3658 • Sending undeliverable messages (no subscribers to the topics)
 - 3659 • Clients that connect but do not send data

3659

3660 Server implementations might close the Network Connection of Clients that breach its security rules.

3661

3662 Server implementations detecting unwelcome behavior might implement a dynamic block list based on
3663 identifiers such as IP address or Client Identifier.

3664

3665 Deployments might use network-level controls (where available) to implement rate limiting or blocking
3666 based on IP address or other information.

3667

3668 **5.4.9 Other security considerations**

3669 If Client or Server TLS certificates are lost or it is considered that they might be compromised they should
3670 be revoked (utilizing CRLs [\[RFC5280\]](#) and/or OSCP [\[RFC6960\]](#)).

3671
3672 Client or Server authentication credentials, such as User Name and Password, that are lost or considered
3673 compromised should be revoked and/or reissued.

3674
3675 In the case of long lasting connections:

- 3676 • Client and Server implementations using TLS [\[RFC5246\]](#) should allow for session renegotiation to
3677 establish new cryptographic parameters (replace session keys, change cipher suites, change
3678 authentication credentials).
- 3679 • Servers may close the Network Connection of Clients and require them to re-authenticate with new
3680 credentials.
- 3681 • Servers may require their Client to reauthenticate periodically using the mechanism described in
3682 [section 4.12.1](#).

3683
3684 Constrained devices and Clients on constrained networks can make use of TLS [\[RFC5246\]](#) session
3685 resumption, in order to reduce the costs of reconnecting TLS [\[RFC5246\]](#) sessions.

3686
3687 Clients connected to a Server have a transitive trust relationship with other Clients connected to the same
3688 Server and who have authority to publish data on the same topics.

3689

3690 **5.4.10 Use of SOCKS**

3691 Implementations of Clients should be aware that some environments will require the use of SOCKSv5
3692 [\[RFC1928\]](#) proxies to make outbound Network Connections. Some MQTT implementations could make
3693 use of alternative secured tunnels (e.g. SSH) through the use of SOCKS. Where implementations choose
3694 to use SOCKS, they should support both anonymous and User Name, Password authenticating SOCKS
3695 proxies. In the latter case, implementations should be aware that SOCKS authentication might occur in
3696 plain-text and so should avoid using the same credentials for connection to a MQTT Server.

3697

3698 **5.4.11 Security profiles**

3699 Implementers and solution designers might wish to consider security as a set of profiles which can be
3700 applied to the MQTT protocol. An example of a layered security hierarchy is presented below.

3701

3702 **5.4.11.1 Clear communication profile**

3703 When using the clear communication profile, the MQTT protocol runs over an open network with no
3704 additional secure communication mechanisms in place.

3705

3706 **5.4.11.2 Secured network communication profile**

3707 When using the secured network communication profile, the MQTT protocol runs over a physical or virtual
3708 network which has security controls e.g., VPNs or physically secure network.

3709

3710 **5.4.11.3 Secured transport profile**

3711 When using the secured transport profile, the MQTT protocol runs over a physical or virtual network and
3712 using TLS [\[RFC5246\]](#) which provides authentication, integrity and privacy.

3713

3714 TLS [\[RFC5246\]](#) Client authentication can be used in addition to – or in place of – MQTT Client
3715 authentication as provided by the User Name and Password fields.

3716

3717 **5.4.11.4 Industry specific security profiles**

3718 It is anticipated that the MQTT protocol will be designed into industry specific application profiles, each
3719 defining a threat model and the specific security mechanisms to be used to address these threats.

3720 Recommendations for specific security mechanisms will often be taken from existing works including:

3721

3722 [\[NISTCSF\]](#) NIST Cyber Security Framework

3723 [\[NIST7628\]](#) NISTIR 7628 Guidelines for Smart Grid Cyber Security

3724 [\[FIPS1402\]](#) Security Requirements for Cryptographic Modules (FIPS PUB 140-2)

3725 [\[PCIDSS\]](#) PCI-DSS Payment Card Industry Data Security Standard

3726 [\[NSAB\]](#) NSA Suite B Cryptography

3727

3728

6 Using WebSocket as a network transport

3729

If MQTT is transported over a WebSocket [RFC6455] connection, the following conditions apply:

3730
3731

- 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].

3732
3733
3734

- 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].

3735

- The Client MUST include “mqtt” in the list of WebSocket Sub Protocols it offers [MQTT-6.0.0-3].

3736
3737

- The WebSocket Subprotocol name selected and returned by the Server MUST be “mqtt” [MQTT-6.0.0-4].

3738

- The WebSocket URI used to connect the Client and Server has no impact on the MQTT protocol.

3739

3740

6.1 IANA considerations

3741

This specification requests IANA to modify the registration of the WebSocket MQTT sub-protocol under the “WebSocket Subprotocol Name” registry with the following data:

3742

3743

3744

Figure 6.6-1 - IANA WebSocket Identifier

Subprotocol Identifier	mqtt
Subprotocol Common Name	mqtt
Subprotocol Definition	http://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

3745

3746

7 Conformance

3747
3748
3749

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.

3750

7.1 Conformance clauses

3751

7.1.1 MQTT Server conformance clause

3752
3753

Refer to [Server](#) in the Terminology section for a definition of Server.

3754

An MQTT Server conforms to this specification only if it satisfies all the statements below:

3755
3756
3757
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767

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.

3768

7.1.2 MQTT Client conformance clause

3769
3770

Refer to [Client](#) in the Terminology section for a definition of Client.

3771

An MQTT Client conforms to this specification only if it satisfies all the statements below:

3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783

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.

3784

Appendix A. Acknowledgments

3785 The TC owes special thanks to Dr. Andy Stanford-Clark and Arlen Nipper as the original inventors of the
3786 MQTT protocol and for their continued support with the standardization process.

3787

3788 The following individuals were members of the OASIS Technical Committee during the creation of this
3789 specification and their contributions are gratefully acknowledged:

3790

3791 **Participants:**

- 3792 • Senthil Nathan Balasubramaniam (Infiswift)
- 3793 • Dr. Andrew Banks, editor (IBM)
- 3794 • Ken Borgendale, editor (IBM)
- 3795 • Ed Briggs, editor (Microsoft)
- 3796 • Raphael Cohn (Individual)
- 3797 • Richard Coppen, chairman (IBM)
- 3798 • William Cox (Individual)
- 3799 • Ian Craggs , secretary (IBM)
- 3800 • Konstantin Dotchkoff (Microsoft)
- 3801 • Derek Fu (IBM)
- 3802 • Rahul Gupta, editor (IBM)
- 3803 • Stefan Hagen (Individual)
- 3804 • David Horton (Solace Systems)
- 3805 • Alex Kritikos (Software AG, Inc.)
- 3806 • Jonathan Levell (IBM)
- 3807 • Shawn McAllister (Solace Systems)
- 3808 • William McLane (TIBCO Software Inc.)
- 3809 • Peter Niblett (IBM)
- 3810 • Dominik Obermaier (dc-square GmbH)
- 3811 • Nicholas O'Leary (IBM)
- 3812 • Brian Raymor, chairman (Microsoft)
- 3813 • Andrew Schofield (IBM)
- 3814 • Tobias Sommer (Cumulocity)
- 3815 • Joe Speed (IBM)
- 3816 • Dr Andy Stanford-Clark (IBM)
- 3817 • Allan Stockdill-Mander (IBM)
- 3818 • Stehan Vaillant (Cumulocity)

3819

3820 For a list of those who contributed to earlier versions of MQTT refer to Appendix A in the MQTT v3.1.1
3821 specification **[MQTTV311]**.

3822

3823

Appendix B. Mandatory normative statement (non-normative)

3824

3825 This Appendix is non-normative and is provided as a convenient summary of the numbered conformance
 3826 statements found in the main body of this document. Refer to [Chapter 7](#) for a definitive list of
 3827 conformance requirements.

3828

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 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”.

3830

Appendix C. Summary of new features in MQTT v5.0 (non-normative)

3831

3832 The following new features are added to MQTT v5.0

3833

3834

- 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.

3835

3836

3837

3838

3839

3840

3841

- Message expiry
Allow an expiry interval to be set when a message is published.

3842

3843

3844

- 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.

3845

3846

3847

3848

3849

- 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.

3850

3851

3852

3853

- Server disconnect
Allow DISCONNECT to be sent by the Server to indicate the reason the connection is closed.

3854

3855

3856

- 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.

3857

3858

3859

3860

- 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.

3861

3862

3863

3864

3865

3866

- Shared Subscriptions
Add shared subscription support allowing for load balanced consumers of a subscription

3867

3868

3869

- 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.

3870

3871

3872

3873

3874

- 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.

3875

3876

3877

3878

- 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.

3879

3880

3881

- 3882
- 3883 • User properties
- 3884 Add User Properties to most packets. User properties on PUBLISH are included with the message
- 3885 and are defined by the Client applications. The user properties on PUBLISH and Will Properties are
- 3886 forwarded by the Server to the receiver of the message. User properties on the CONNECT,
- 3887 SUBSCRIBE, and UNSUBSCRIBE packets are defined by the Server implementation. The user
- 3888 properties on CONNACK PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK and
- 3889 AUTH packets are defined by the sender, and are unique to the sender implementation. The meaning
- 3890 of user properties is not defined by MQTT.
- 3891
- 3892 • Maximum Packet Size
- 3893 Allow the Client and Server to independently specify the maximum packet size they support. It is an
- 3894 error for the session partner to send a larger packet.
- 3895
- 3896 • Optional Server feature availability
- 3897 Define a set of features which the Server does not allow and provide a mechanism for the Server to
- 3898 specify this to the Client. The features which can be specified in this way are: Maximum QoS, Retain
- 3899 Available, Wildcard Subscription Available, Subscription Identifier Available, and Shared Subscription
- 3900 Available. It is an error for the Client to use features that the Server has declared are not available.
- 3901
- 3902 It is possible in earlier versions of MQTT for a Server to not implement a feature by declaring that the
- 3903 Client is not authorized for that function. This feature allows such optional behavior to be declared
- 3904 and adds specific Reason Codes when the Client uses one of these features anyway.
- 3905
- 3906 • Enhanced authentication
- 3907 Provide a mechanism to enable challenge/response style authentication including mutual
- 3908 authentication. This allows SASL style authentication to be used if supported by both Client and
- 3909 Server, and includes the ability for a Client to re-authenticate within a connection.
- 3910
- 3911 • Subscription options
- 3912 Provide subscription options primarily defined to allow for message bridge applications. These include
- 3913 an option to not send messages originating on this Client (noLocal), and options for handling retained
- 3914 messages on subscribe.
- 3915
- 3916 • Will delay
- 3917 Add the ability to specify a delay between the end of the connection and sending the will message.
- 3918 This is designed so that if a connection to the session is re-established then the will message is not
- 3919 sent. This allows for brief interruptions of the connection without notification to others.
- 3920
- 3921 • Server Keep Alive
- 3922 Allow the Server to specify the value it wishes the Client to use as a keep alive. This allows the
- 3923 Server to set a maximum allowed keepalive and still have the Client honor it.
- 3924
- 3925 • Assigned ClientID
- 3926 In cases where the ClientID is assigned by the Server, return the assigned ClientID. This also lifts the
- 3927 restriction that Server assigned ClientIDs can only be used with Clean Session=1 connections.
- 3928
- 3929 • Server reference
- 3930 Allow the Server to specify an alternate Server to use on CONNACK or DISCONNECT. This can be
- 3931 used as a redirect or to do provisioning.