MQTT Version 5.0

Working Draft 08

18th October 2016

Technical Committee:

[OASIS Message Queuing Telemetry Transport (MQTT) TC](https://www.oasis-open.org/committees/mqtt/)

Chairs:

Brian Raymor ([brian.raymor@microsoft.com](mailto:brian.raymor@microsoft.com)), [Microsoft](http://www.microsoft.com)

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

Editors:

Andrew Banks ([andrew\_banks@uk.ibm.com](mailto:andrew_banks@uk.ibm.com)), [IBM](http://www.ibm.com)

Ed Briggs ([edbriggs@microsoft.com](mailto:edbriggs@microsoft.com)), [Microsoft](https://www.microsoft.com/en-us/)

Ken Borgendale ([kwb@us.ibm.com](mailto:kwb@us.ibm.com)), [IBM](http://www.ibm.com)

Rahul Gupta ([rahul.gupta@us.ibm.com](mailto:rahul.gupta@us.ibm.com)), [IBM](http://www.ibm.com)

Additional artifacts:

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 so as 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 [Working Draft](https://www.oasis-open.org/policies-guidelines/tc-process) (WD) has been produced by one or more TC Members; it has not yet been voted on by the TC or [approved](https://www.oasis-open.org/policies-guidelines/tc-process) as a Committee Draft (Committee Specification Draft or a Committee Note Draft). The OASIS document [Approval Process](https://www.oasis-open.org/policies-guidelines/tc-process) begins officially with a TC vote to approve a WD as a Committee Draft. A TC may approve a Working Draft, revise it, and re-approve it any number of times as a Committee Draft.

URI patterns:

Initial publication URI:  
http://docs.oasis-open.org/mqtt/mqtt/v5.0/csd01/mqtt-v5.0-csd01.docx

Permanent “Latest version” URI:  
http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.docx

(Managed by OASIS TC Administration; please don’t modify.)

Copyright © OASIS Open 2016. 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](https://www.oasis-open.org/policies-guidelines/ipr) 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.

Table of Contents

[1 Introduction 9](#_Toc464563960)

[1.1 Organization of MQTT 9](#_Toc464563961)

[1.2 Terminology 9](#_Toc464563962)

[1.3 Normative references 10](#_Toc464563963)

[1.4 Non normative references 11](#_Toc464563964)

[1.5 Data representation 13](#_Toc464563965)

[1.5.1 Bits 13](#_Toc464563966)

[1.5.2 Two Byte Integer data values 13](#_Toc464563967)

[1.5.3 Four Byte Integer data values 13](#_Toc464563968)

[1.5.4 UTF-8 encoded strings 13](#_Toc464563969)

[1.5.5 Variable Byte Integer 15](#_Toc464563970)

[1.5.6 Binary Data 16](#_Toc464563971)

[1.5.7 UTF-8 String Pair 16](#_Toc464563972)

[1.6 Security 16](#_Toc464563973)

[1.7 Editing convention 16](#_Toc464563974)

[2 MQTT Control Packet format 17](#_Toc464563975)

[2.1 Structure of an MQTT Control Packet 17](#_Toc464563976)

[2.1.1 Fixed header 17](#_Toc464563977)

[2.1.2 MQTT Control Packet type 17](#_Toc464563978)

[2.1.3 Flags 18](#_Toc464563979)

[2.1.4 Remaining Length 19](#_Toc464563980)

[2.2 Variable header 19](#_Toc464563981)

[2.2.1 Packet Identifier 19](#_Toc464563982)

[2.2.2 Return code 21](#_Toc464563983)

[2.2.3 Identifier/Value pairs 21](#_Toc464563984)

[2.2.3.1 Length of Identifier/Value pairs 21](#_Toc464563985)

[2.2.3.2 Identifier/Value pair 21](#_Toc464563986)

[2.3 Payload 22](#_Toc464563987)

[3 MQTT Control Packets 24](#_Toc464563988)

[3.1 CONNECT – Client requests a connection to a Server 24](#_Toc464563989)

[3.1.1 Fixed header 24](#_Toc464563990)

[3.1.2 Variable header 24](#_Toc464563991)

[3.1.2.1 Protocol Name 24](#_Toc464563992)

[3.1.2.2 Protocol version 25](#_Toc464563993)

[3.1.2.3 Connect Flags 25](#_Toc464563994)

[3.1.2.4 Clean Start 26](#_Toc464563995)

[3.1.2.5 Will Flag 26](#_Toc464563996)

[3.1.2.6 Will QoS 27](#_Toc464563997)

[3.1.2.7 Will Retain 27](#_Toc464563998)

[3.1.2.8 User Name Flag 27](#_Toc464563999)

[3.1.2.9 Password Flag 27](#_Toc464564000)

[3.1.2.10 Keep Alive 27](#_Toc464564001)

[3.1.2.11 Length of Identifier/Value pairs. 28](#_Toc464564002)

[3.1.2.12 Session Expiry Interval. 28](#_Toc464564003)

[3.1.2.13 Will Delay Interval. 30](#_Toc464564004)

[3.1.2.14 Receive Maximum. 30](#_Toc464564005)

[3.1.2.15 Topic Alias Maximum. 31](#_Toc464564006)

[3.1.2.16 Request Reply Info 31](#_Toc464564007)

[3.1.2.17 Request Problem Info 32](#_Toc464564008)

[3.1.2.18 User Defined Name-Value Pair 32](#_Toc464564009)

[3.1.2.19 Auth Method 32](#_Toc464564010)

[3.1.2.20 Auth Data 33](#_Toc464564011)

[3.1.2.21 Variable header non normative example 33](#_Toc464564012)

[3.1.3 Payload 34](#_Toc464564013)

[3.1.3.1 Client Identifier 34](#_Toc464564014)

[3.1.3.2 Will Topic 34](#_Toc464564015)

[3.1.3.3 Will Message 35](#_Toc464564016)

[3.1.3.4 User Name 35](#_Toc464564017)

[3.1.3.5 Password 35](#_Toc464564018)

[3.1.4 Response 35](#_Toc464564019)

[3.2 CONNACK – Acknowledge connection request 36](#_Toc464564020)

[3.2.1 Fixed header 36](#_Toc464564021)

[3.2.2 Variable header 37](#_Toc464564022)

[3.2.2.1 Connect Acknowledge Flags 37](#_Toc464564023)

[3.2.2.2 Session Present 37](#_Toc464564024)

[3.2.2.3 Connect Return Code 37](#_Toc464564025)

[3.2.2.4 Length of Identifier/Value pairs 38](#_Toc464564026)

[3.2.2.5 Receive Maximum 39](#_Toc464564027)

[3.2.2.6 Retain Unavailable Advertisement 39](#_Toc464564028)

[3.2.2.7 Assigned Client Identifier. 40](#_Toc464564029)

[3.2.2.8 Topic Alias Maximum. 40](#_Toc464564030)

[3.2.2.9 Reason String 40](#_Toc464564031)

[3.2.2.10 Server Keep Alive 40](#_Toc464564032)

[3.2.2.11 Reply Info 41](#_Toc464564033)

[3.2.2.12 Server Reference 41](#_Toc464564034)

[3.2.2.13 Auth Method 41](#_Toc464564035)

[3.2.2.14 Auth Data 42](#_Toc464564036)

[3.2.3 Payload 42](#_Toc464564037)

[3.3 PUBLISH – Publish message 42](#_Toc464564038)

[3.3.1 Fixed header 42](#_Toc464564039)

[3.3.1.1 DUP 42](#_Toc464564040)

[3.3.1.2 QoS 43](#_Toc464564041)

[3.3.1.3 RETAIN 43](#_Toc464564042)

[3.3.2 Variable header 44](#_Toc464564043)

[3.3.2.1 Topic Name 44](#_Toc464564044)

[3.3.2.2 Packet Identifier 44](#_Toc464564045)

[3.3.2.3 Length of Identifier/Value pairs. 44](#_Toc464564046)

[3.3.2.4 Payload Format Indicator. 44](#_Toc464564047)

[3.3.2.5 Publication Expiry interval. 45](#_Toc464564048)

[3.3.2.6 Topic Alias. 45](#_Toc464564049)

[3.3.2.7 Reply Topic 46](#_Toc464564170)

[3.3.2.8 Correlation Data 47](#_Toc464564171)

[3.3.2.9 User Defined Name-Value Pair 47](#_Toc464564172)

[3.3.3 Payload 47](#_Toc464564173)

[3.3.4 Response 48](#_Toc464564174)

[3.3.5 Actions 48](#_Toc464564175)

[3.4 PUBACK – Publish acknowledgement 49](#_Toc464564176)

[3.4.1 Fixed header 49](#_Toc464564177)

[3.4.2 Variable header 50](#_Toc464564178)

[3.4.2.1 Publish Return code 50](#_Toc464564179)

[3.4.2.2 Length of Identifier/Value pairs. 51](#_Toc464564180)

[3.4.2.3 Reason String 51](#_Toc464564181)

[3.4.3 Payload 51](#_Toc464564183)

[3.4.4 Actions 51](#_Toc464564184)

[3.5 PUBREC – Publish received (QoS 2 publish received, part 1) 51](#_Toc464564185)

[3.5.1 Fixed header 51](#_Toc464564186)

[3.5.2 Variable header 52](#_Toc464564187)

[3.5.2.1 Publish Return codes 52](#_Toc464564188)

[3.5.2.2 Length of Identifier/Value pairs. 53](#_Toc464564189)

[3.5.2.3 Reason String 53](#_Toc464564190)

[3.5.3 Payload 53](#_Toc464564191)

[3.5.4 Actions 53](#_Toc464564192)

[3.6 PUBREL – Publish release (QoS 2 publish received, part 2) 53](#_Toc464564193)

[3.6.1 Fixed header 53](#_Toc464564194)

[3.6.2 Variable header 54](#_Toc464564195)

[PUBREL Return code 54](#_Toc464564196)

[3.6.2.1 Length of Identifier/Value pairs. 54](#_Toc464564197)

[3.6.2.2 Reason String 55](#_Toc464564198)

[3.6.3 Payload 55](#_Toc464564199)

[3.6.4 Actions 55](#_Toc464564200)

[3.7 PUBCOMP – Publish complete (QoS 2 publish received, part 3) 55](#_Toc464564201)

[3.7.1 Fixed header 55](#_Toc464564202)

[3.7.2 Variable header 55](#_Toc464564203)

[3.7.2.1 PUBCOMP Return codes 56](#_Toc464564204)

[3.7.2.2 Length of Identifier/Value pairs. 56](#_Toc464564205)

[3.7.2.3 Reason String 56](#_Toc464564206)

[3.7.3 Payload 56](#_Toc464564207)

[3.7.4 Actions 56](#_Toc464564208)

[3.8 SUBSCRIBE - Subscribe to topics 57](#_Toc464564209)

[3.8.1 Fixed header 57](#_Toc464564210)

[3.8.2 Variable header 57](#_Toc464564211)

[3.8.2.1 Variable header non normative example 57](#_Toc464564212)

[3.8.3 Payload 57](#_Toc464564213)

[3.8.3.1 Subscription Options 58](#_Toc464564214)

[3.8.3.2 Payload non normative example 59](#_Toc464564215)

[3.8.4 Response 60](#_Toc464564216)

[3.9 SUBACK – Subscribe acknowledgement 61](#_Toc464564217)

[3.9.1 Fixed header 61](#_Toc464564218)

[3.9.2 Variable header 62](#_Toc464564219)

[3.9.2.1 Length of Identifier/Value pairs. 62](#_Toc464564220)

[3.9.2.2 Reason String 62](#_Toc464564221)

[3.9.3 Payload 62](#_Toc464564222)

[3.9.3.1 Payload non normative example 63](#_Toc464564223)

[3.10 UNSUBSCRIBE – Unsubscribe from topics 63](#_Toc464564224)

[3.10.1 Fixed header 63](#_Toc464564225)

[3.10.2 Variable header 64](#_Toc464564226)

[3.10.2.1 Length of Identifier/Value pairs. 64](#_Toc464564227)

[3.10.2.2 Reason String 64](#_Toc464564228)

[3.10.3 Payload 64](#_Toc464564229)

[3.10.3.1 Payload non normative example 64](#_Toc464564230)

[3.10.4 Response 65](#_Toc464564231)

[3.11 UNSUBACK – Unsubscribe acknowledgement 66](#_Toc464564232)

[3.11.1 Fixed header 66](#_Toc464564233)

[3.11.2 Variable header 66](#_Toc464564234)

[Payload 66](#_Toc464564235)

[3.12 PINGREQ – PING request 67](#_Toc464564236)

[3.12.1 Fixed header 67](#_Toc464564237)

[3.12.2 Variable header 67](#_Toc464564238)

[3.12.3 Payload 67](#_Toc464564239)

[3.12.4 Response 67](#_Toc464564240)

[3.13 PINGRESP – PING response 68](#_Toc464564241)

[3.13.1 Fixed header 68](#_Toc464564242)

[3.13.2 Variable header 68](#_Toc464564243)

[3.13.3 Payload 68](#_Toc464564244)

[Actions 68](#_Toc464564245)

[3.14 DISCONNECT – Disconnect notification 68](#_Toc464564246)

[3.14.1 Fixed header 68](#_Toc464564247)

[3.14.2 Variable header 69](#_Toc464564248)

[3.14.2.1 Disconnect Return code 69](#_Toc464564249)

[3.14.2.2 Length of Identifier/Value pairs. 70](#_Toc464564250)

[3.14.2.3 Session Expiry Interval. 70](#_Toc464564251)

[3.14.2.4 Reason String 71](#_Toc464564252)

[3.14.2.5 Server Reference 71](#_Toc464564253)

[3.14.3 Payload 72](#_Toc464564254)

[3.14.4 Actions 72](#_Toc464564255)

[3.15 AUTH – Authentication exchange 72](#_Toc464564256)

[3.15.1 Fixed header 72](#_Toc464564257)

[3.15.2 Variable header 72](#_Toc464564258)

[3.15.2.1 Authenticate Return code 73](#_Toc464564259)

[3.15.2.2 Length of Identifier/Value pairs. 73](#_Toc464564260)

[3.15.2.3 Auth Method 73](#_Toc464564261)

[3.15.2.4 Auth Data 73](#_Toc464564262)

[3.15.3 Payload 73](#_Toc464564263)

[3.15.4 Actions 73](#_Toc464564264)

[4 Operational behavior 74](#_Toc464564265)

[4.1 Storing state 74](#_Toc464564266)

[4.1.1 Non normative example 74](#_Toc464564267)

[4.2 Network Connections 75](#_Toc464564268)

[4.3 Quality of Service levels and protocol flows 75](#_Toc464564269)

[4.3.1 QoS 0: At most once delivery 75](#_Toc464564270)

[4.3.2 QoS 1: At least once delivery 76](#_Toc464564271)

[4.3.3 QoS 2: Exactly once delivery 77](#_Toc464564272)

[4.4 Message delivery retry 79](#_Toc464564273)

[4.5 Message receipt 79](#_Toc464564274)

[4.6 Message ordering 79](#_Toc464564275)

[4.7 Topic Names and Topic Filters 80](#_Toc464564276)

[4.7.1 Topic wildcards 80](#_Toc464564277)

[4.7.1.1 Topic level separator 80](#_Toc464564278)

[4.7.1.2 Multi-level wildcard 80](#_Toc464564279)

[4.7.1.3 Single level wildcard 80](#_Toc464564280)

[4.7.2 Topics beginning with $ 81](#_Toc464564281)

[4.7.3 Topic semantic and usage 81](#_Toc464564282)

[4.8 Subscriptions 82](#_Toc464564283)

[4.8.1 Non-Shared Subscriptions 82](#_Toc464564284)

[4.8.2 Shared Subscriptions 82](#_Toc464564285)

[4.9 Flow Control 85](#_Toc464564286)

[4.10 Request / Response 85](#_Toc464564287)

[4.11 Server redirection 85](#_Toc464564288)

[4.12 Enhanced authentication 86](#_Toc464564289)

[4.12.1 Non normative example showing a SCRAM challenge 87](#_Toc464564290)

[4.12.2 Non normative example showing a Kerberos challenge client to server 87](#_Toc464564291)

[4.13 Handling errors 87](#_Toc464564292)

[5 Security 89](#_Toc464564293)

[5.1 Introduction 89](#_Toc464564294)

[5.2 MQTT solutions: security and certification 89](#_Toc464564295)

[5.3 Lightweight cryptography and constrained devices 89](#_Toc464564296)

[5.4 Implementation notes 90](#_Toc464564297)

[5.4.1 Authentication of Clients by the Server 90](#_Toc464564298)

[5.4.2 Authorization of Clients by the Server 90](#_Toc464564299)

[5.4.3 Authentication of the Server by the Client 91](#_Toc464564300)

[5.4.4 Integrity of Application Messages and Control Packets 91](#_Toc464564301)

[5.4.5 Privacy of Application Messages and Control Packets 91](#_Toc464564302)

[5.4.6 Non-repudiation of message transmission 91](#_Toc464564303)

[5.4.7 Detecting compromise of Clients and Servers 91](#_Toc464564304)

[5.4.8 Detecting abnormal behaviors 92](#_Toc464564305)

[5.4.9 Other security considerations 92](#_Toc464564306)

[5.4.10 Use of SOCKS 93](#_Toc464564307)

[5.4.11 Security profiles 93](#_Toc464564308)

[5.4.11.1 Clear communication profile 93](#_Toc464564309)

[5.4.11.2 Secured network communication profile 93](#_Toc464564310)

[5.4.11.3 Secured transport profile 93](#_Toc464564311)

[5.4.11.4 Industry specific security profiles 93](#_Toc464564312)

[6 Using WebSocket as a network transport 94](#_Toc464564313)

[6.1 IANA Considerations 94](#_Toc464564314)

[7 Conformance 95](#_Toc464564315)

[7.1 Conformance Targets 95](#_Toc464564316)

[7.1.1 MQTT Server 95](#_Toc464564317)

[7.1.2 MQTT Client 95](#_Toc464564318)

[Appendix A. Acknowledgments 97](#_Toc464564319)

[Appendix B. Revision History 98](#_Toc464564320)

# Introduction

## Organization of MQTT

The specification is split into seven chapters:

## 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]](#anchor-RFC2119).

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.

For examples see Section 4.2.

Application Message:

The data carried by the MQTT protocol across the network for the application. When Application Messages are transported by MQTT they have an associated Quality of Service and a Topic Name.

**Client:**

A program or device that uses MQTT. A Client always establishes the Network Connection to the Server. It can

* Publish Application Messages that other Clients might be interested in.
* Subscribe to request Application Messages that it is interested in receiving.
* Unsubscribe to remove a request for Application Messages.
* Disconnect from the Server.

**Server:**

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

* Accepts Network Connections from Clients.
* Accepts Application Messages published by Clients.
* Processes Subscribe and Unsubscribe requests from Clients.
* Forwards Application Messages that match Client Subscriptions.

**Subscription:**

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

**Shared Subscription:**

A Shared Subscription comprises a Topic Filter and a maximum QoS. Unlike a regular Subscription, a Shared Subscription can be associated with more than one Session and an Application Message that matches a Shared Subscription is only sent to one of these Sessions' clients. A Session can subscribe to more than one Shared Subscription and can contain Shared Subscriptions alongside regular Shared Subscriptions.

**Topic Name:**

The label attached to an Application Message which is matched against the Subscriptions known to the Server. The Server sends a copy of the Application Message to each Client that has a matching Subscription.

**Topic Filter:**

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

**Session:**

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

**MQTT Control Packet:**

A packet of information that is sent across the Network Connection. The MQTT specification defines fourteen different types of Control Packet, one of which (the PUBLISH packet) is used to convey Application Messages.

## Normative references

**[RFC2119]**

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

<http://www.ietf.org/rfc/rfc2119.txt>

**[RFC3629]**

Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003

<http://www.ietf.org/rfc/rfc3629.txt>

**[RFC5246]**

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

<http://www.ietf.org/rfc/rfc5246.txt>

**[RFC6455]**

Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

<http://www.ietf.org/rfc/rfc6455.txt>

**[Unicode]**

The Unicode Consortium. The Unicode Standard.

<http://www.unicode.org/versions/latest/>

## Non normative references

**[RFC793]**

Postel, J. Transmission Control Protocol. STD 7, IETF RFC 793, September 1981.

<http://www.ietf.org/rfc/rfc793.txt>

**[AES]**

Advanced Encryption Standard (AES) (FIPS PUB 197).

<http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>

**[DES]**

Data Encryption Standard (DES).

<http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>

**[FIPS1402]**

Security Requirements for Cryptographic Modules (FIPS PUB 140-2)

<http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf>

**[IEEE 802.1AR]**

IEEE Standard for Local and metropolitan area networks - Secure Device Identity

<http://standards.ieee.org/findstds/standard/802.1AR-2009.html>

**[ISO29192]**

ISO/IEC 29192-1:2012 Information technology -- Security techniques -- Lightweight cryptography -- Part 1: General

<http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=56425>

**[MQTT NIST]**

MQTT supplemental publication, MQTT and the NIST Framework for Improving Critical Infrastructure Cybersecurity

<http://docs.oasis-open.org/mqtt/mqtt-nist-cybersecurity/v1.0/mqtt-nist-cybersecurity-v1.0.html>

**[MQTTV31]**

MQTT V3.1 Protocol Specification.

<http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html>

**[MQTTV311]**

MQTT V3.1.1 Protocol Specification

<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>

**[NISTCSF]**

Improving Critical Infrastructure Cybersecurity Executive Order 13636 <http://www.nist.gov/itl/upload/preliminary-cybersecurity-framework.pdf>

**[****NIST7628]**

NISTIR 7628 Guidelines for Smart Grid Cyber Security

<http://www.nist.gov/smartgrid/upload/nistir-7628_total.pdf>

**[NSAB]**

NSA Suite B Cryptography

<http://www.nsa.gov/ia/programs/suiteb_cryptography/>

**[PCIDSS]**

PCI-DSS Payment Card Industry Data Security Standard

<https://www.pcisecuritystandards.org/security_standards/>

**[RFC1928]**

Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, March 1996.

<http://www.ietf.org/rfc/rfc1928.txt>

**[RFC4511]**

Sermersheim, J., Ed., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.

<http://www.ietf.org/rfc/rfc4511.txt>

**[RFC5077]**

Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008.

<http://www.ietf.org/rfc/rfc5077.txt>

**[RFC5280]**

Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

<http://www.ietf.org/rfc/rfc5280.txt>

**[RFC6066]**

Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

<http://www.ietf.org/rfc/rfc6066.txt>

**[RFC6749]**

Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

<http://www.ietf.org/rfc/rfc6749.txt>

**[RFC6960]**

Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, June 2013.

<http://www.ietf.org/rfc/rfc6960.txt>

**[SARBANES]**

Sarbanes-Oxley Act of 2002.

<http://www.gpo.gov/fdsys/pkg/PLAW-107publ204/html/PLAW-107publ204.htm>

**[USEUSAFEHARB]**

U.S.-EU Safe Harbor

<http://export.gov/safeharbor/eu/eg_main_018365.asp>

## Data representation

### Bits

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

### Two Byte Integer data values

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

### Four Byte Integer data values

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

### UTF-8 encoded strings

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

Each of these strings is prefixed with a Two Byte Integer length field that gives the number of bytes in a UTF-8 encoded string itself, as illustrated in Figure 1.1 Structure of UTF-8 encoded strings below. Consequently, there is a limit on the size of a string that can be passed in one of these UTF-8 encoded string components; you cannot use a string that would encode to more than 65535 bytes.

Unless stated otherwise all UTF-8 encoded strings can have any length in the range 0 to 65535 bytes.

Figure . Structure of UTF-8 encoded strings

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

The character data in a UTF-8 encoded string MUST be well-formed UTF-8 as defined by the Unicode specification [Unicode] and restated in RFC 3629 [RFC3629]. In particular this data MUST NOT include encodings of code points between U+D800 and U+DFFF. If a Server or Client receives a Control Packet containing ill-formed UTF-8 it MUST close the Network Connection.

A UTF-8 encoded string MUST NOT include an encoding of the null character U+0000. If a receiver (Server or Client) receives a Control Packet containing U+0000 it MUST close the Network Connection.

The data SHOULD NOT include encodings of the Unicode [Unicode] code points listed below. If a receiver (Server or Client) receives a Control Packet containing any of them it MAY close the Network Connection:

U+0001..U+001F control characters

U+007F..U+009F control characters

Code points defined in the Unicode specification [Unicode] to be non-characters (for example U+0FFFF)

A UTF-8 encoded sequence 0xEF 0xBB 0xBF is always to be interpreted to mean 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.

**Non normative example**

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

Figure . 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 |

### Variable Byte Integer

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

Table . Size of Variable Byte Integer

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

**Non normative comment**

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

do

encodedByte = X MOD 128

X = X DIV 128

// if there are more data to encode, set the top bit of this byte

if ( X > 0 )

encodedByte = encodedByte OR 128

endif

'output' encodedByte

while ( X > 0 )

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

**Non normative comment**

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

multiplier = 1

value = 0

do

encodedByte = 'next byte from stream'

value += (encodedByte AND 127) \* multiplier

if (multiplier > 128\*128\*128)

throw Error(Malformed Variable Byte Integer)

multiplier \*= 128

while ((encodedByte AND 128) != 0)

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

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

### Binary Data

Binary Data is represented by a Two Byte Integer length which indicates the number of data bytes, followed by that number of bytes. Thus the length of Binary Data is limited to the range of 0 to 65535 Bytes. The data bytes within the Binary Data can have any value.

### UTF-8 String Pair

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

Both strings MUST comply with the requirements for UTF-8 encoded strings. If a receiver (Client or Server) receives a string which does not meet these requirements, it MUST close the Network Connection.

## Security

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

## Editing convention

Text highlighted in Yellow within this specification identifies conformance statements. Each conformance statement has been assigned a reference in the format.

# MQTT Control Packet format

## Structure of an MQTT Control Packet

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

An MQTT Control Packet consists of up to three parts, always in the following order as illustrated in Figure 2.1 - Structure of an MQTT Control Packet.

Figure . Structure of an MQTT Control Packet

|  |
| --- |
| Fixed header, present in all MQTT Control Packets |
| Variable header, present in some MQTT Control Packets |
| Payload, present in some MQTT Control Packets |

### Fixed header

Each MQTT Control Packet contains a fixed header. Figure 2.2 Fixed header format illustrates the fixed header format.

Figure . 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 | | | | | | | |

### MQTT Control Packet type

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

Represented as a 4-bit unsigned value, the values are listed in Table 2.1 Control packet types

Table . Control packet types

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Value** | **Direction of flow** | **Description** |
| Reserved | 0 | Forbidden | Reserved |
| CONNECT | 1 | Client to Server | Client request to connect to Server |
| 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 |
| PUBREC | 5 | Client to Server or  Server to Client | Publish received (assured delivery part 1) |
| PUBREL | 6 | Client to Server or  Server to Client | Publish release (assured delivery part 2) |
| PUBCOMP | 7 | Client to Server or  Server to Client | Publish complete (assured delivery part 3) |
| SUBSCRIBE | 8 | Client to Server | Client 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 | Client or server is disconnecting |
| AUTH | 15 | Client to Server or Server to Client | Authentication exchange |

### Flags

The remaining bits [3-0] of byte 1 in the fixed header contain flags specific to each MQTT Control Packet type as listed in the [Table 2.2 - Flag Bits](#_Table_2.2_-) below. Where a flag bit is marked as “Reserved” in [Table 2.2 - Flag Bits](#_Table_2.2_-), it is reserved for future use and MUST be set to the value listed in that table. If invalid flags are received, the receiver MUST close the Network Connection. See Section 0 for details about handling errors.

##### Table 2.2 - Flag Bits

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 5.0 | DUP1 | QoS2 | QoS2 | RETAIN3 |
| 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 | 1 |

DUP1 = Duplicate delivery of a PUBLISH Control Packet

QoS2 = PUBLISH Quality of Service

RETAIN3 = PUBLISH Retain flag

See Section 3.3.1 for a description of the DUP, QoS, and RETAIN flags in the PUBLISH Control Packet.

### Remaining Length

**Position:** starts at byte 2.

The Remaining Length is a Variable Byte Integer which represents the number of bytes remaining within the current packet, including data in the variable header and the payload. The Remaining Length does not include the bytes used to encode the Remaining Length.

## Variable header

Some types of MQTT Control Packets contain a variable header component. It resides between the fixed header and the payload. The content of the variable header varies depending on the Packet type. The Packet Identifier field of variable header is common in several packet types.

### Packet Identifier

##### Figure 2.3 - Packet Identifier bytes

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

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

SUBSCRIBE, UNSUBSCRIBE, and PUBLISH (in cases where QoS > 0) Control Packets MUST contain a non-zero Two Byte Integer Packet Identifier. Each time a Client sends a new packet of one of these types it MUST assign it a currently unused Packet Identifier. If a Client re-sends a particular Control Packet, then it MUST use the same Packet Identifier in subsequent re-sends of that packet. The Packet Identifier becomes available for reuse after the Client has processed the corresponding acknowledgement packet. In the case of a QoS 1 PUBLISH this is the corresponding PUBACK; in the case of QoS 2 it is PUBCOMP. For SUBSCRIBE or UNSUBSCRIBE it is the corresponding SUBACK or UNSUBACK. The same conditions apply to a Server when it sends a PUBLISH with QoS > 0.

Packet identifiers used with PUBLISH, SUBSCRIBE and UNSUBSCRIBE commands form a single, unified set of identifiers. A packet identifier cannot be used by more than one command at any time.

A PUBLISH Packet MUST NOT contain a Packet Identifier if its QoS value is set to 0.

A PUBACK, PUBREC or PUBREL Packet MUST contain the same Packet Identifier as the PUBLISH Packet that was originally sent. Similarly, SUBACK and UNSUBACK MUST contain the Packet Identifier that was used in the corresponding SUBSCRIBE and UNSUBSCRIBE Packet respectively.

Control Packets that require a Packet Identifier are listed in Table 2.5 - Control Packets that contain a Packet Identifier.

##### Table 2.5 - Control Packets that contain a Packet Identifier

|  |  |
| --- | --- |
| **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 |

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

**Non normative comment**

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

Client Server

PUBLISH Packet Identifier=0x1234---🡪

🡨--PUBLISH Packet Identifier=0x1234

PUBACK Packet Identifier=0x1234---🡪

🡨--PUBACK Packet Identifier=0x1234

### Return code

The Return code is an 8 bit unsigned integer value indicates the result of an operation. Return codes less than 128 indicate successful completion. The normal Return code for success is 0. Return code values of 128 or greater indicate failure.

The CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, DISCONNECT, and AUTH Packets have a Return code as part of the Variable header. The SUBACK and UNSUBACK Packets have a list of Return codes as the Payload.

### Identifier/Value pairs

The CONNECT, CONNACK, PUBLISH, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK, DISCONNECT, and AUTH Packet variable header ends with a set of Identifier/Value pairs. This is composed of a length field followed by the Identifier/Value pairs. There is no significance in the order of Identifier/Value pairs with different Identifiers.

#### Length of Identifier/Value pairs

The length of Identifier/Value pairs in the variable header is encoded as a Variable Byte Integer. The length of the Identifier/Value fields does not include the bytes used to encode the length. A length of zero indicates that there are no Identifier/Value fields in the packet.

#### Identifier/Value pair

An Identifier/Value pair consists of an Identifier which defines the usage and data type, followed by the value itself. The Identifier is encoded as a Variable Byte Integer. On receipt of an Identifier or Value that is not defined for the Packet, the receiver MUST close the network connection.

Table 2.6 – Identifier/Value pairs (non normative)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dec** | **ID** | **Name** | **Type** | **Packet** |
| 1 | 0x01 | Payload Format | Byte | PUBLISH |
| 2 | 0x02 | Publication Expiry | Four Byte Integer | PUBLISH |
| 8 | 0x08 | Reply Topic | UTF8 encoded string | PUBLISH |
| 9 | 0x09 | Correlation Data | Binary Data | PUBLISH |
| 17 | 0x11 | Session Expiry Interval | Four Byte Integer | CONNECT, DISCONNECT |
| 18 | 0x12 | Assigned Client Identifier | UTF8 encoded string | CONNACK |
| 19 | 0x13 | Server Keep Alive | Two Byte Integer | CONNACK |
| 21 | 0x15 | Auth Method | UTF8 encoded string | CONNECT, CONNACK, AUTH |
| 22 | 0x16 | Auth Data | Binary Data | CONNECT, CONNACK, AUTH |
| 23 | 0x17 | Request Problem Info | Byte | CONNECT |
| 24 | 0x18 | Will Delay Interval | Four Byte Integer | CONNECT |
| 25 | 0x19 | Request Reply Info | Byte | CONNECT |
| 26 | 0x1A | Reply Info | UTF8 encoded string | CONNACK |
| 28 | 0x1C | Server Reference | UTF8 encoded string | CONNACK, DISCONNECT |
| 31 | 0x1F | Reason String | UTF8 encoded string | CONNACK, DISCONNECT |
| 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 | CONNECT, CONNACK |
| 37 | 0x25 | Retain Unavailable | Zero Length | CONNACK |
| 38 | 0x26 | User Defined Name-Value Pair | UTF-8 String Pair | CONNECT, PUBLISH |

**Non normative comment**   
In this specification, only one-byte Identifiers are used.

## Payload

Some MQTT Control Packets contain a payload as the final part of the packet, as described in Chapter **Error! Reference source not found.**. In the case of the PUBLISH packet this is the Application Message. [Table 2.6 - Control Packets that contain a Payload](#_Table_2.6_-) lists the Control Packets that require a Payload.

##### Table 2.6 - Control Packets that contain a Payload

|  |  |
| --- | --- |
| **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 |

# MQTT Control Packets

## CONNECT – Client requests a connection to a Server

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.

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 disconnect the Client. See section 4.13 for information about handling errors.

The payload contains one or more encoded fields. They specify a unique Client identifier for the Client, a Will topic, Will Message, User Name and Password. All but the Client identifier are optional and their presence is determined based on flags in the variable header.

### Fixed header

Figure . - CONNECT Packet fixed header

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

**Remaining Length field**

Remaining Length is the length of the variable header plus the length of the Payload. It is encoded as a Variable Byte Integer in the manner described in section **Error! Reference source not found.**.

### Variable header

The variable header for the CONNECT Packet consists of fields in the following order: Protocol Name, Protocol Level, Connect Flags, Keep Alive, Length in bytes of Identifier/Value pairs, and the Identifier/Value pairs. The rules for encoding Identifier/Value pairs are described in 2.2.2.

#### Protocol Name

Figure . Protocol Name bytes

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Description** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Protocol Name | | | | | | | | | |
| byte 1 | Length MSB (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 2 | Length LSB (4) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| byte 3 | ‘M’ | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| byte 4 | ‘Q’ | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| byte 5 | ‘T’ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| byte 6 | ‘T’ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |

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

If the Protocol Name is incorrect the Server MAY continue processing the CONNECT packet in accordance with some other specification, but in that case, the Server MUST NOT continue to process the CONNECT packet according to this specification. Otherwise the Server MAY respond to the CONNECT Packet with a Connect Return code of 0x84 (Unsupported protocol version) and MUST close the Network Connection

**Non normative comment**

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

#### Protocol version

Figure . - Protocol version byte

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Description** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Protocol Level | | | | | | | | | |
| byte 7 | Level(5) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

The 8 bit unsigned value that represents the revision level of the protocol used by the Client. The value of the Protocol Level field for the version 5.0 of the protocol is 5 (0x05).

If this value is not 5, the Server MAY continue processing the CONNECT packet in accordance with some other specification, but in that case the Server MUST NOT continue to process the CONNECT according to this specification. Otherwise the Server MAY respond to the CONNECT Packet with a Connect Return code 0x84 (Unsupported protocol version) and MUST close the Network Connection.

#### Connect Flags

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

Figure . - Connect Flag bits

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

The Server MUST validate that the reserved flag in the CONNECT Control Packet is set to zero and disconnect the Client if it is not zero.

#### Clean Start

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

This bit specifies whether the Session state is kept when a connection is made, see section 3.1.2.12 for a definition of the Session state.

If Clean Start is set to 1, the Client and Server MUST discard any previous Session and start a new one. Consequently, the Session Present flag in CONNACK is always set to 0 if Clean Start is set to 1.

**Non normative comment**

To ensure consistent state in the event of a failure, the Client should repeat its attempts to connect with a Clean Start set to 1, until it connects successfully.

#### Will Flag

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

If the Will Flag is set to 1 this indicates that, if the Connect request is accepted, a Will Message MUST be stored on the Server and associated with the Session. The Will Message MUST be published after the Network Connection is subsequently closed unless the Will Message has been deleted by the Server on receipt of a DISCONNECT Packet containing a return code of 0.

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

* An I/O error or network failure detected by the Server.
* The Client fails to communicate within the Keep Alive time.
* The Client closes the Network Connection without first sending a DISCONNECT Packet with a Return code less than 128.
* The Server closes the Network Connection without first receiving a DISCONNECT Packet with a Return code less than 128.

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 Topic and Will Message fields MUST be present in the payload. 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 Return code of 0 from the Client.

If the Will Flag is set to 0, the Will QoS and Will Retain fields in the Connect Flags MUST be set to zero and the Will Topic and Will Message fields MUST NOT be present in the payload. If the Will Flag is set to 0, the Server MUST NOT publish a Will Message.

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

**Non normative comment**

There is no notification of session expiry after the Client sends a DISCONNECT Packet with a Return code of 0. The Will Message is effectively a Session Expiry message if the Will Delay Interval is set to be longer than the Session Expiry and then the client ends the network connection or sends a DISCONNECT with a non-zero Return code.

#### 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).

If the Will Flag is set to 1, the value of Will QoS can be 0 (0x00), 1 (0x01), or 2 (0x02). It value of 3 (0x03) is a malformed Packet and the Server MUST close the Network Connection.

#### 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 the Will Retain Flag MUST be set to 0.

If the Will Flag is set to 1:

* If Will Retain is set to 0, the Server MUST publish the Will Message as a non-retained message.
* If Will Retain is set to 1, the Server MUST publish the Will Message as a retained message.

#### 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.

If the User Name Flag is set to 1, a user name MUST be present in the payload.

#### Password Flag

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

If the Password Flag is set to 0, a password MUST NOT be present in the payload.

If the Password Flag is set to 1, a password MUST be present in the payload.

#### Keep Alive

Figure . - Keep Alive bytes

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

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

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.

The Client can send PINGREQ at any time, irrespective of the Keep Alive value, and use the PINGRESP to determine that the network and the Server are working.

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

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

A Keep Alive value of zero (0) has the effect of turning off the keep alive mechanism. This means that, in this case, the Server is not required to disconnect the Client on the grounds of inactivity.

Note that a Server is permitted to disconnect a Client that it determines to be inactive or non-responsive at any time, regardless of the Keep Alive value provided by that Client.

**Non normative comment**

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

#### Length of Identifier/Value pairs

The length of Identifier/Value pairs in the CONNECT Packet variable header encoded as a Variable Byte Integer.

#### Session Expiry Interval

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

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

The Client and Server can store Session state to enable reliable messaging to continue across a sequence of Network Connections. The Session Expiry is a time interval measured in seconds. After a network disconnection and the Session Expiry interval has elapsed, without a new connection being made the Client and Server each delete the Session state they hold.

If the Session Expiry Interval is absent, the Session does not expire. If it is set to zero, the Session ends when the Network Connection is closed.

If a new Network Connection is made before the Session has expired, the Server MUST resume communications with the Client based on state from the current Session (as identified by the Client identifier). If there is no Session associated with the Client identifier the Server MUST create a new Session. The Client and Server MUST store the Session after the Client and Server are disconnected.

After the disconnection of a Session and before the Session Expiry Interval has passed, the Server MUST store further QoS 1 and QoS 2 messages that match any subscriptions that the Client had at the time of disconnection as part of the Session state.

It MAY also store QoS 0 messages that meet the same criteria. State data associated with this Session MUST NOT be reused in any subsequent Session.

After the disconnection of a Session and after the Session Expiry Interval has passed, the Client and Server discard any Session state.

After reconnection the Session lasts as long as the Network Connection plus the new Session Expiry interval.

The Session state in the Client consists of:

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

QoS 2 messages which have been received from the Server, but have not been completely acknowledged.

The Session state in the Server consists of:

The existence of a Session, even if the rest of the Session state is empty.

The Client’s subscriptions.

QoS 1 and QoS 2 messages which have been sent to the Client, but have not been completely acknowledged.

QoS 1 and QoS 2 messages pending transmission to the Client.

QoS 2 messages which have been received from the Client, but have not been completely acknowledged.

Optionally, QoS 0 messages pending transmission to the Client.

If the Session is currently disconnected, the time at which the Session state will be deleted.

Retained messages do not form part of the Session state in the Server, they MUST NOT be deleted when the Session ends.

See Section 4.1 for details and limitations of stored state.

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

**Non Normative comment**

Setting a Session Expiry Interval of 0 and Clean Start to 1, is equivalent to setting CleanSession to 1 in the MQTT Specification Version 3.1.1.

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

**Non normative comment**

Typically, a Client will always connect using the same Session Expiry interval. The choice will depend on the application. A Client using Session Expiry set to 0 will not receive old Application Messages and has to subscribe afresh to any topics that it is interested in each time it connects. A Client using a non zero Session Expiry Interval will receive all QoS 1 or QoS 2 messages that were published while it was disconnected. Hence, to ensure that you do not loose messages while disconnected, use QoS 1 or QoS 2 with no Session Expiry or a Session Expiry greater than zero.

**Non normative comment**

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

**Non normative comment**

A Client might be connecting to a server using a network that provides intermittent connectivity. This Client can use a short Session Expiry Interval so that it can re-connect when the network is available again and continue reliable message delivery. If the Client does not re-connect, allowing the Session state to expire, then application messages may be lost.

**Non normative comment**

If the Client connects using this protocol, then re connects using the MQTT V3.1.1 protocol using CleanStart 0 before the session has expired, the Session state is kept indefinitely.

**Non normative comment**

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

**Non normative comment**

The Client and Server clocks might drift and not measure time intervals accurately. The Client should always rely on the Session Present flag in the CONNACK packet rather than try to calculate whether the Server actually did keep its Session state.

#### Will Delay Interval

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

Followed by the Four Byte Integer representing the Will Delay Interval in seconds. It is a protocol error to include the Will Delay Interval more than once.

If the Will Delay Interval is absent, then there is no delay before the will message is published.

**Non normative comment**

The Server delays publishing the Client’s Will message until the Will Delay Interval has passed or the Session expires. One use of this is to avoid publishing will messages if there is a temporary network disconnection and the Client succeeds in reconnecting and continuing its Session before the Will message is published.

#### Receive Maximum

33 (0x21) Byte Identifier of the Receive Maximum. Followed by the Two Byte Integer representing the Receive Maximum value. It is a protocol error to include the Receive Maximum value more than once, or to be a value less than 1.

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

The Server MUST NOT send more than Receive Maximum QoS 1 and QoS 2 PUBLISH Packets for which it has not received PUBACK or PUBCOMP from the Client. The Client MAY close the Network Connection if it receives more than Receive Maximum QoS 1 and QoS 2 PUBLISH Packets where it has not sent a PUBACK or PUBCOMP in response.

The Server 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. This might result in deadlock. The value of Receive Maximum applies only to the current Network Connection.

If the Receive Maximum value is absent, then the Client MUST NOT close the Network Connection because it has received too many unacknowledged PUBLISH packets.

See section 4.8 Flow Control for details of how the Receive Maximum is used.

**Non normative comment**

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

**Non normative comment**

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

**Non normative comment**

Not setting the Receive Maximum value is the same as setting it to 65535.

#### Topic Alias Maximum

34 (0x22) Byte Identifier of the Topic Alias Maximum. Followed by the Two Byte Integer representing the Topic Alias Maximum value. It is a protocol error to include the Topic Alias Maximum value more than once.

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

#### Request Reply Info

25 (0x19) Byte, Identifier of the Request Reply Info. Followed by a Byte with a value of either 0 or 1. It is protocol error to include the Request Reply Info more than once, or to have a value other than 0 or 1. If the Request Reply Info is absent, the value of 0 is used.

The Client uses this value to indicate whether the Server should return Reply Info in the CONNACK. A value of 0 indicates that the Server SHOULD NOT return Reply Info. If the value is 1 the Server MAY return Reply Info in the CONNACK packet.

**Non normative comment**

If the Server does not support Reply Info, or the Server is not configured to return Reply Info for this Client, it should not return Reply Info in the CONNACK.

See section 4.10 for more information about how request /response works.

#### Request Problem Info

23 (0x17) Byte, Identifier of the Request Problem Info. Followed by a Byte with a value of either 0 or 1. It is a protocol error to include Request Problem Info more than once, or to have a value other than 0 or 1. If the Request Problem Info is absent, the Server uses its configured default.

The Client uses this value to indicate whether it can use a human readable Reason String in the case of failures. If this value is 0, the Server MAY return a Reason String on a CONNACK or DISCONNECT Packet, but MUST NOT send on Reason String on any other Packet. If this value is 1, the Server MAY return a Reason String on any Packet where it is allowed.

#### User Defined Name-Value Pair

38 (0x26) Byte Identifier of the User Defined Name-Value Pair. This is followed by a UTF-8 string pair. The first string represents a user defined name, and the second string contains the corresponding value. Both strings MUST comply with restrictions for UTF-8 encoded strings.

The User Defined Name-Value Pair may appear zero or more times in a CONNECT command packet, and may contain the same name field more than once. T

**Non normative comment**

User Defined Name-Value Pairs on the CONNECT packet are used to send connection related properties from the Client to the Server. The meaning of these properties is not defined by this specification.

#### Auth Method

21 (0x15) Byte, Identifier of the Auth Method. Followed by a UTF8 encoded string containing the name of the authentication method. See section 4.12 to understand how extended authentication works.

If a Client sets an Auth Method on the CONNECT, the Client MUST NOT send any Packets other than AUTH Packets until it has received a CONNACK Packet.

#### Auth Data

22 (0x16) Byte, Identifier of the Auth Data. Followed by BinaryData containing authentication data. The contents of this data are defined by the authentication method and the state of already exchanged authentication data. See section 4.12 to understand how extended authentication works.

#### Variable header non normative example

Figure . - Variable header non normative 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 Level | | | | | | | | | |
|  | **Description** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| byte 7 | Level (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)  Will Flag (1)  Clean Start(1)  *Reserved* (0) | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 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 |
| Identifier/Value fields | | | | | | | | | |
| byte 11 | Length (5) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| byte 12 | Session Expiry Identifier (17) | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| byte 13 | Session Expiry (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 |

### Payload

The payload of the CONNECT Packet contains one or more length-prefixed fields, whose presence is determined by the flags in the variable header. These fields, if present, MUST appear in the order Client Identifier, Will Topic, Will Message, User Name, Password.

#### Client Identifier

The Client Identifier (ClientId) identifies the Client to the Server. Each Client connecting to the Server has a unique ClientId. The ClientId MUST be used by Clients and by Servers to identify state that they hold relating to this MQTT Session between the Client and the Server.

The Client Identifier (ClientId) MUST be present and MUST be the first field in the CONNECT packet payload.

The ClientId MUST be a UTF-8 encoded string as defined in Section **Error! Reference source not found.**.   
  
The Server MUST allow ClientIds which are between 1 and 23 UTF-8 encoded bytes in length, and that contain only the characters

"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ".

The Server MAY allow ClientIds that contain more than 23 encoded bytes. The Server MAY allow ClientIds that contain characters not included in the list given above.   
  
A Server MAY allow a Client to supply a ClientId that has a length of zero bytes, however if it does so the Server MUST treat this as a special case and assign a unique ClientId to that Client. 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.  
  
If the Server rejects the ClientId it MAY respond to the CONNECT Packet with a CONNACK return code 0x85 (Identifier not valid) and then MUST close the Network Connection.

**Non normative comment**

A Client implementation could provide a convenience method to generate a random ClientId. Use of such a method should be actively discouraged when the Session Expiry is long, causing the Server to remember a ClientId which the Client itself may forget.

#### Will Topic

If the Will Flag is set to 1, the Will Topic is the next field in the payload. The Will Topic MUST be a UTF-8 encoded string as defined in Section **Error! Reference source not found.**.

#### Will Message

If the Will Flag is set to 1 the Will Message is the next field in the payload. The Will Message defines the Application Message that is to be published to the Will Topic as described in Section 3.1.2.5. This field consists of Binary Data which has a Two Byte Integer length followed by the payload for the Will Message expressed as a sequence of zero or more bytes. The length gives the number of bytes in the data that follows and does not include the 2 bytes taken up by the length itself.

When the Will Message is published to the Will Topic its payload consists only of the data portion of this field, not the first two length bytes.

#### User Name

If the User Name Flag is set to 1, this is the next field in the payload. The User Name MUST be a UTF-8 encoded string as defined in Section **Error! Reference source not found.**. It can be used by the Server for authentication and authorization.

#### Password

If the Password Flag is set to 1, this is the next field in the payload. The Password field is Binary Data which has a Two Byte Integer length field which indicates the number of bytes of binary data. Although this field is called Password, it can be used to carry any credential information.

### Response

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

1. If the Server does not receive a CONNECT Packet within a reasonable amount of time after the Network Connection is established, the Server SHOULD close the connection.
2. The Server MUST validate that the CONNECT Packet conforms to section 3.1 and close the Network Connection if it does not conform. The Server MAY send a DISCONNECT with a Return code of 128 or greater before closing the Network Connection.
3. The Server MAY check that the contents of the CONNECT Packet meet any further restrictions and SHOULD perform authentication and authorization checks. If any of these checks fail, it MUST close the Network Connection. Before closing the connection, it MAY send an appropriate CONNACK response with a Return code of 128 or greater as described in section 3.2.

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

1. If the ClientId represents a Client already connected to the Server, the Server MAY send a DISCONNECT to the existing Client with a Return code of 0x8E (Session taken over) and MUST close the Network Connection of the existing Client. If the existing Client has a Will Message, that Will Message is sent if the Will Delay Interval of the existing Client is 0, or if the session is discarded (the existing Client has a Session Expiry of 0, or the is new Client is CleanStart 1).
2. The Server MUST perform the processing of CleanStart that is described in section **Error! Reference source not found.**.
3. The Server MUST acknowledge the CONNECT Packet with a CONNACK Packet containing a 0x00 (Success) Return code.  
     
   **Non-normative comment**   
     
   It is recommended that authentication and authorization checks be performed if the server is being used to serve any form of sensitive data. If these tests succeed, the server responds by sending a CONNACK with a return code of zero. If they fail, the server is advised not to send a CONNACK at all, as this could alert a potential attacker to the presence of the MQTT server and encourage such an attacker to launch a denial of service or password-guessing attack.
4. Start message delivery and keep alive monitoring.

Clients are allowed to send further Control Packets immediately after sending a CONNECT Packet; Clients need not wait for a CONNACK Packet to arrive from the Server. If the Server rejects the CONNECT, it MUST NOT process any data sent by the Client after the CONNECT Packet.

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

## CONNACK – Acknowledge connection request

The CONNACK Packet is the packet sent by the Server in response to a CONNECT Packet received from a Client. The first packet sent from the Server to the Client MUST be a CONNACK Packet.

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

### Fixed header

The fixed header format is illustrated in Figure 3.7 – CONNACK Packet fixed header.

Figure . – 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 (2) | | | | | | | |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

**Remaining Length field**

This is the length of the variable header, encoded as a Variable Byte Integer in the manner described in section **Error! Reference source not found.**. For the CONNACK Packet this has the value 2.

### Variable header

The variable header contains the following fields in the order: Connect Acknowledge Flags, Connect Return Code, Length in bytes of Identifier/Value pairs, the Identifier/Value pairs. The rules for encoding Identifier/Value pairs are described in 2.2.2.

#### Connect Acknowledge Flags

Byte 1 is the "Connect Acknowledge Flags". Bits 7-1 are reserved and MUST be set to 0.   
  
Bit 0 (SP1) is the Session Present Flag.

#### Session Present

Position: bit 0 of the Connect Acknowledge Flags.   
  
If the Server accepts a connection with Session Expiry set to 0, the Server MUST set Session Present to 0 in the CONNACK packet in addition to setting a zero return code in the CONNACK packet.  
  
If the Server accepts a connection with non zero Session Expiry, the value set in Session Present depends on whether the Server has already has stored Session state for the supplied client ID. If the Server has stored Session state, it MUST set Session Present to 1 in the CONNACK packet. If the Server has not already stored Session state, it MUST set Session Present to 0 in the CONNACK packet. This is in addition to setting a zero return code in the CONNACK packet.  
  
The Session Present flag enables a Client to establish whether the Client and Server have a consistent view about whether there is already stored Session state.   
  
Once the initial setup of a Session is complete, a Client with stored Session state will expect the Server to maintain its stored Session state. In the event that the value of Session Present received by the Client from the Server is not as expected, the Client can choose whether to proceed with the Session or to disconnect. The Client can discard the Session state on both Client and Server by disconnecting with Session Expiry set to 0.   
  
If a server sends a CONNACK packet containing a non-zero return code, it MUST set Session Present to 0.

**Non normative comment**

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

#### Connect Return Code

Byte 2 in the Variable header.

The values the Connect Return Code are listed in Table 3.1 - Connect Return code values. If a well formed CONNECT Packet is received by the Server, but the Server is unable to complete the Connection the Server MAY send a CONNACK packet containing the appropriate Connect Return Code from this table. If a server sends a CONNACK packet containing a Return Code of 128 or greater it MUST then close the Network Connection.

Table . - Connect Return code values

|  |  |  |  |
| --- | --- | --- | --- |
| **Value** | **Hex** | **Return 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 Return codes apply. |
| 129 | 0x81 | Malformed control packet | Data within the CONNECT Packet was not consistent with 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 level of the MQTT protocol requested by the Client |
| 133 | 0x85 | 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 username 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 | The authentication method is not supported or does not match the authentication method currently in use |
| 144 | 0x90 | Topic Invalid | The Will Topic Is Invalid |
| 149 | 0x95 | Packet Too Large | The Connect Packet exceeded the maximum permissible size |
| 156 | 0x9C | Use another server | The Client should temporarily use another server |
| 157 | 0x9D | Server moved | The Client should permanently use another server |

The Server MUST use one of the Return code values in Table 3.1 - Connect Return code values.

**Non normative comment**

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

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

#### Length of Identifier/Value pairs

The length of Identifier/Value pairs in the CONNACK Packet variable header encoded as a Variable Byte Integer.

#### Receive Maximum

33 (0x21) Byte. Identifier of the Receive Maximum. Followed by the Two Byte Integer representing the Receive Maximum value. It is a protocol error to include the Receive Maximum value more than once.

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

The Client MUST NOT send more than Receive Maximum QoS 1 and QoS 2 PUBLISH Packets for which it has not received PUBACK or PUBCOMP from the Server. The Server MAY close the Network Connection if it receives more than Receive Maximum QoS 1 and QoS 2 PUBLISH Packets where it has not sent a PUBACK or PUBCOMP in response.

The Client 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. This might result in deadlock. The value of Receive Maximum applies only to the current Network Connection.

If the Receive Maximum value is absent, then the Server MUST NOT close the Network Connection because it has received too many unacknowledged PUBLISH packets.

See section 4.8 Flow Control for details of how the Receive Maximum is used.

**Non normative comment**

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

**Non normative comment**

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

**Non normative comment**

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

**Non normative comment**

Leaving the Receive Maximum value absent is equivalent to setting it to 65535.

#### Retain Unavailable Advertisement

Identifier 37 (0x25) followed by a zero-length payload signifies that the Server will not honor the RETAIN flag if it is set on a PUBLISH message received from the Client. If a Server does not support RETAIN, it MUST send this advertisement. The absence of this advertisement in a CONNACK signifies the Server supports RETAIN.

#### Assigned Client Identifier

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

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

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

If the Client connects using a zero length Client Identifier, the Server MUST respond with a CONNACK containing an Assigned Client Identifier.

#### Topic Alias Maximum

34 (0x22) Byte Identifier of the Topic Alias Maximum. Followed by the Two Byte Integer representing the Topic Alias Maximum value. It is a protocol error to include the Topic Alias Maximum value more than once.

This value indicates the highest value that the Server will accept as a Topic Alias sent by the Client. The Server uses this value to limit the number of topic aliases that it is willing to hold on this Connection. The Client MUST NOT send a Topic Alias in a PUBLISH Packet to the Server greater than this. A value of 0 indicates that the Server does not accept any Topic Aliases on this connection. If Topic Alias Maximum is absent, the Client MUST NOT send any topic aliases on to the Server.

#### Reason String

31 (0x1F) Byte Identifier of the Reason String. Followed by the UTF8 encoded string representing the reason for the connection failure. This Reason String is a human readable string designed for diagnostics and should not be parsed by the receiver.

The Server uses this value to give additional information to the client. The Server MUST NOT use this Identifier/Value pair if it would increase the size of the CONNACK packet to beyond the Maximum Packet Size specified by the Client. It is a protocol error to include the Reason String more than once.

**Non normative comment**

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

#### Server Keep Alive

19 (0x13) Byte Identifier of the Server Keep Alive. Followed by a Two Byte Integer with the keep alive time assigned by the server. If the Server sends a Server Keep Alive on the CONNACK packet, the Client MUST use this value instead of the Keep Alive value the Client sent on CONNECT. If the Server does not send the Server Keep Alive, the Server MUST use the Keep Alive value set by the Client on CONNECT. It is a protocol error to include the Server Keep Alive more than once.

**Non normative comment**

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

#### Reply Info

26 (0x1A) Byte Identifier of the Reply Info. Followed by a UTF8 encoded string which is used as the basis for creating a Reply Topic. The way in which the Client creates a Reply Topic from the Reply Info is not defined by this standard. It is a protocol error to include the Reply Info more than once.

If the Client sends a Request Reply Info with a value 1, the Server MAY send the Reply Info in the CONNACK.

**Non normative comment**

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

See section 4.10 for more information about how request / reply works.

#### Server Reference

28 (0x1C) Byte Identifier of the Server Reference. Followed by a UTF8 encoded string which can be used by the Client to identify another Server to use. It is a protocol error to include the Server Reference more than once.

The Server MUST only use send a Server Reference along with a Return Code of 0x9C – Use another server or 0x9D – Server moved.

See section 4.11 Server Redirection for information about how Server Reference is used.

#### Auth Method

21 (0x15) Byte, Identifier of the Auth Method. Followed by a UTF8 encoded string containing the name of the authentication method. See section 4.12 to understand how extended authentication works.

#### Auth Data

22 (0x16) Byte, Identifier of the Auth Data. Followed by BinaryData containing authentication data. The contents of this data are defined by the authentication method and the state of already exchanged authentication data. See section 4.12 to understand how extended authentication works.

### Payload

The CONNACK Packet has no payload.

## PUBLISH – Publish message

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

### Fixed header

**Error! Reference source not found.** illustrates the fixed header format:

Figure . – 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 | | | | | | | |

#### DUP

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

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

The DUP flag MUST be set to 1 by the Client or Server when it attempts to re-deliver a PUBLISH Packet. The DUP flag MUST be set to 0 for all QoS 0 messages.

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

**Non normative comment**

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

**Non normative comment**

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

#### QoS

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

This field indicates the level of assurance for delivery of an Application Message. The QoS levels are listed in the Table 3.2 - QoS definitions, below.

Table . - 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 |

A PUBLISH Packet MUST NOT have both QoS bits set to 1. If a Server or Client receives a PUBLISH Packet which has both QoS bits set to 1 it MUST close the Network Connection.

#### RETAIN

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

If the RETAIN flag is set to 1, in a PUBLISH Packet sent by a Client to a Server, the Server MUST store the Application Message and its QoS, so that it can be delivered to future subscribers whose subscriptions match its topic name. When a new Non-Shared subscription is established, the last retained message, if any, on each matching topic name MUST be sent to the subscriber if the Retain Handling option of the subscription is 0, or if the Retain Handling option is 1 and the subscription did not already exist. If the Server receives a QoS 0 message with the RETAIN flag set to 1 it MUST discard any message previously retained for that topic. It SHOULD store the new QoS 0 message as the new retained message for that topic, but MAY choose to discard it at any time - if this happens there will be no retained message for that topic. See Section 4.1 for more information on storing state.

If the Retain As Published subscription option is 0, when sending a PUBLISH Packet to a Client the Server MUST set the RETAIN flag to 1 if a message is sent as a result of a new Non-Shared subscription being made by a Client. It MUST set the RETAIN flag to 0 when a PUBLISH Packet is sent to a Client because it matches an established subscription regardless of how the flag was set in the message it received.

If the Retain As Published subscription option is 1, when sending a PUBLISH Packet to a Client the Server MUST set the RETAIN flag equal to the RETAIN flag of the message it received.

A PUBLISH Packet with a RETAIN flag set to 1 and a payload containing zero bytes will be processed as normal by the Server and sent to Clients with a subscription matching the topic name. Additionally, any existing retained message with the same topic name MUST be removed and any future subscribers for the topic will not receive a retained message. “As normal” means that the RETAIN flag is not set in the message received by existing Clients. A zero byte retained message MUST NOT be stored as a retained message on the Server.

If the RETAIN flag is 0, in a PUBLISH Packet sent by a Client to a Server, the Server MUST NOT store the message and MUST NOT remove or replace any existing retained message.

**Non normative comment**

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

**Remaining Length field**

This is the length of variable header plus the length of the payload, encoded as a Variable Byte Integer in the manner described in section **Error! Reference source not found.**.

### Variable header

The variable header contains the following fields in the order: Topic Name, Packet Identifier, Length in bytes of Identifier/Value pairs, the Identifier/Value pairs. The rules for encoding Identifier/Value pairs are described in 2.2.2.

#### Topic Name

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

The Topic Name MUST be present as the first field in the PUBLISH Packet Variable header. It MUST be a UTF-8 encoded string as defined in section **Error! Reference source not found.**.

The Topic Name in the PUBLISH Packet MUST NOT contain wildcard characters.

The Topic Name in a PUBLISH Packet sent by a Server to a subscribing Client MUST match the Subscription’s Topic Filter according to the matching process defined in Section 4.7. However, since the Server is permitted to override the Topic Name, it might not be the same as the Topic Name in the original PUBLISH Packet.

A Topic Name might be long, to reduce the size of the PUBLISH packet the sender can use a shorter Topic Alias instead. The Topic Alias is described in section 3.3.2.6 and in general it is shorter than the full Topic Name. If it provides a Topic Alias the sender is permitted to supply a zero-length String, consisting of two zero bytes only, for the Topic Name.

#### Packet Identifier

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

#### Length of Identifier/Value pairs

The length of Identifier/Value pairs in the PUBLISH Packet variable header encoded as a Variable Byte Integer followed by the Identifier/Value pairs themselves.

#### Payload Format Indicator

If absent the PUBLISH payload consists of unspecified bytes.

1 (0x01) Byte, Identifier of the Payload Format Indicator. Followed by the Value of the Payload Format Indicator, either of:

0 (0x00) Byte Indicates that the payload is unspecified bytes, this is equivalent to not sending a Payload Format Indicator.

1 (0x01) Byte Indicates that the payload is UTF-8 Encoded Character Data.  Note that the UTF-8 Data in the payload does not include a length prefix, nor is it subject to the restrictions described in section 1.5.3.

A server MUST send the Payload Format Indicator unaltered to all subscribers receiving the publication, using this version of MQTT. The receiver is not required to validate that the payload is of the format indicated, however it MAY close the network connection if it discovers that the payload does not match the Payload Format Indicator.

The Server MUST send the publication to an MQTT Client using the MQTT V3.1.1 protocol with a matching subscription, in this case no Payload Format Indicator is included.

#### Publication Expiry Interval

If absent, the publication does not expire.

2 (0x02) Byte, Identifier of the Publication Expiry interval. Followed by the Four Byte Integer representing the Publication Expiry interval.

If present, the Four Byte value is the lifetime of the publication in seconds. If Publication 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.

The PUBLISH packet sent to a Client by the Server MUST contain a Publication Expiry Interval set to the received value minus the time that the publication has been waiting in the server.

See Section 4.1 for details and limitations of stored state.

#### Topic Alias.

35 (0x23) Byte Identifier of the Topic Alias. Followed by the Two Byte Integer representing the Topic Alias value. It is a protocol error to include the Topic Alias value more than once.

A Topic Alias is an integer value that can be used to identify the Topic, instead of including the full Topic Name. This reduces the size of the PUBLISH packet, and is of use if Topic Names are lengthy and the same Topic Names are used repetitively within a Connection.

If a Topic Alias has been established at the receiver a sender is permitted to send a PUBLISH packet that contains that Topic Alias and supplying a zero-length String in place of the Topic Name. The receiver then treats the incoming PUBLISH as if it had contained the full Topic Name corresponding to the alias.   
  
The sender decides whether to use an alias and chooses the alias value. It establishes an alias by including a non-zero length Topic Name and a Topic Alias value in the PUBLISH packet. When it receives a packet like this the receiver processes the packet as normal, but it also records the mapping of Topic Alias to a full Topic Name.

A sender is permitted to send a subsequent PUBLISH in the same Network Connection that has the same Topic Alias value and a different Topic Name. If it does this the Topic Alias is remapped to the Topic Name in this subsequent packet.

Topic Alias mappings are scoped to the Network Connection and last only for the lifetime of that Network Connection. A receiver MUST NOT carry forward any Topic Alias mappings from one Network Connection to another.

A Topic Alias of 0 is not permitted. A Sender MUST NOT send a PUBLISH packet containing a Topic Alias which has the value 0.

A Client MUST NOT send a PUBLISH packet with a Topic Alias whose value exceeds the Maximum Alias value returned by the Server in the CONNACK packet.

A Server MUST NOT send a PUBLISH packet with a Topic Alias whose value exceeds the Maximum Alias value sent by the Client in the CONNECT packet.

A Server MUST accept all Topic Alias values greater than 0 and less than or equal to the Maximum Alias value that it returned in the CONNACK packet.

A Client MUST accept all Topic Alias values greater than 0 and less than or equal to the Maximum Alias value that it sent in the CONNECT packet.

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

#### Reply Topic

8 (0x08) Byte, Identifier of the Reply Topic, followed by a UTF-8 string which is used as the Topic Name for a reply message. The Reply Topic MUST be a UTF-8 encoded string as defined in section 1.5.3. The Reply Topic MUST NOT contain wildcard characters.  It is a protocol error to include the Reply Topic more than once. If the Reply Topic is not present, this Application Message is not a request.

The Server MUST send the Reply Topic unaltered to all subscribers receiving the publication which use this version of MQTT.

**Non normative comment:**   
The receiver of an Application Message with a Reply Topic sends a response by using the Reply Topic as the Topic Name of a PUBLISH. If the request message contains a Correlation Data, the receiver of the request should also include this Correlation Data as an Identifier/Value pair in the PUBLISH packet of the response.

See section 4.10 for more information about how request / response works.

#### Correlation Data

9 (0x09) Byte, Identifier of the Correlation Data, followed by Binary Data which is a Two Byte Integer length followed by that many byes of binary data. The Correlation Data is used by the sender of the request to identify which request the response is for when the response is received. It is a protocol error to include Correlation Data more than once. If the Correlation Data is not present, the requestor does not require any correlation data.

The Server MUST send the Correlation Data unaltered to all subscribers receiving the publication which use this version of MQTT.

The Value of the Correlation Data only has meaning to the sender of the request and receiver of the response.

**Non normative comment:**

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

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

See section 4.10 for more information about how request / response works.

#### User Defined Name-Value Pair

38 (0x26) Byte Identifier of the User Defined Name-Value Pair. This is followed by a UTF-8 string pair. The first string represents a name, and the second string contains the corresponding value. Both strings MUST comply with restrictions for UTF-8 encoded strings.

The User Defined Name-Value Pair may appear zero or more times in a PUBLISH packet, and may contain the same name more than once.

The Server MUST send all User Defined Name-Value Pairs unaltered in a PUBLISH packet when forwarding the Application Message to a Client using this version of MQTT. The Server MUST maintain the order of User Defined Name-Value Pairs when forwarding the Application Message.

**Non normative comment**

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

### Payload

The Payload contains the Application Message that is being published. The content and format of the data is application specific. The length of the payload can be calculated by subtracting the length of the variable header from the Remaining Length field that is in the Fixed Header. It is valid for a PUBLISH Packet to contain a zero length payload.

[Figure 3.11 - Publish Packet variable header non normative example illustrates an example variable header for the PUBLISH Packet briefly described in Table 3.3 - Publish Packet non normative example.](#_Figure_3.11_-)

Table . Publish Packet non normative example

|  |  |
| --- | --- |
| **Field** | **Value** |
| Topic Name | a/b |
| Packet Identifier | 10 |
| Identifier/Value pairs | None |

Figure . - 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 |
| Byte 8 | No Identifier/Value pairs | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### Response

The receiver of a PUBLISH Packet MUST respond according to Table 3.4 - Expected Publish Packet response as determined by the QoS in the PUBLISH Packet.

Table . Expected Publish Packet response

|  |  |
| --- | --- |
| **QoS Level** | **Expected Response** |
| QoS 0 | None |
| QoS 1 | PUBACK Packet |
| QoS 2 | PUBREC Packet |

### Actions

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

The Server uses a PUBLISH Packet to send an Application Message to each Client which has a matching subscription.

When Clients make subscriptions with Topic Filters that include wildcards, it is possible for a Client’s subscriptions to overlap so that a published message might match multiple filters. In this case the Server MUST deliver the message to the Client respecting the maximum QoS of all the matching subscriptions. In addition, the Server MAY deliver further copies of the message, one for each additional matching subscription and respecting the subscription’s QoS in each case.

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

The receiver MUST reject a PUBLISH packet if it has a zero-length String in the Topic Name field and does not contain a Topic Alias. In the case of a QoS 1 or QoS 2 PUBLISH the receiver sends a 0x9B (No Topic) return code on the PUBACK or PUBREL response.

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

* 1. A Topic Alias value of zero is treated as a Protocol Error.
  2. If the receiver has already established a mapping for the Topic Alias, then
     1. If the packet has a zero-length String in the Topic Name field, the receiver processes it using the Topic Name that corresponds to the Topic Alias
     2. If the packet contains a non-zero length Topic Name, the receiver processes the packet using that Topic Name and updates its mapping for the Topic Alias to point to the Topic Name from the incoming packet
  3. If the Receiver doesn't already have a mapping for this Topic Alias
     1. If the packet has a zero-length String in the Topic Name field, the receiver MUST reject it. The receiver sends a No Topic return code in a DISCONNECT packet and then closes the network connection.
     2. If the packet contains a non-zero length Topic Name, the receiver processes the packet using that Topic Name and in addition it adds the Topic Alias to its list of established Topic Alias mappings, provided that it is less than or equal to the Maximum Alias supported by the receiver for this connection. The Alias mapping is added even if publishing of the Application Message itself fails.
     3. If the Topic Alias exceeds the Maximum Alias, the receiver MUST NOT process the packet. The receiver sends an Alias Not Accepted return code in a DISCONNECT packet and then closes the network connection.

## PUBACK – Publish acknowledgement

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

### Fixed header

Figure . - PUBACK Packet fixed header

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

**Remaining Length field**

This is the length of the variable header, encoded as a Variable Byte Integer in the manner described in section **Error! Reference source not found.**. For the PUBACK Packet this has the value 3 if a Return code is specified and 2 if it a Return code is not specified.

### Variable header

The variable header format is illustrated in Figure 3.13 – PUBACK Packet variable header. The variable header contains the following fields in order, the Packet Identifier from the PUBLISH Packet that is being acknowledged, an optional Return code, and optional Length of Identifier/Value pairs, and the Identifier/Value pairs.

Figure . – 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 Return code | | | | | | | |

#### Publish Return code

Byte 3 in the Variable header is the Publish Return code. If the Remaining Length is less than 3, there is no Return code and the value of 0x00 (Success) is used.

The values for the Publish Return Code field are listed in Table 3.5 - PUBACK Return codes

Table . - PUBACK Return codes

|  |  |  |  |
| --- | --- | --- | --- |
| **Value** | **Hex** | **Return 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 does not know if there are any matching subscribers, it SHOULD use the 0x00 (Success) Return code. |
| 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 Invalid | The topic is invalid. |
| 149 | 0x95 | Packet too large | The packet size exceeded the permissible value. |
| 154 | 0x9A | QoS level not supported | The QoS level is not supported by this system |

The Client or Server sending the PUBACK MUST use one of the PUBACK Return codes shown in Table 3.5 - PUBACK Return codes.

#### Length of Identifier/Value pairs

The length of Identifier/Value pairs in the PUBACK Packet variable header encoded as a Variable Byte Integer. If the Remaining Length is less than 4 there is no Length of Identifier/Value pairs and the value of 0 is used.

#### Reason String

31 (0x1F) Byte Identifier of the Reason String. Followed by the UTF8 encoded string representing the reason for the connection failure. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the receiver.

The sender uses this value to give additional information to the receiver. The sender MUST NOT use this Identifier/Value pair if it would increase the size of the PUBACK packet to beyond the Maximum Packet Size specified by the Client. It is a protocol error to include the Reason String more than once.

### Payload

The PUBACK Packet has no payload.

### Actions

This is fully described in Section 4.3.2.

## PUBREC – Publish received (QoS 2 publish received, part 1)

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

### Fixed header

Figure . - PUBREC Packet fixed header

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

**Remaining Length field**

This is the length of the variable header, encoded as a Variable Byte Integer in the manner described in section **Error! Reference source not found.**. For the PUBREC Packet this has the value 2 if no Return code is specified, and 3 if it a Return code is specified.

### Variable header

The variable header format is illustrated in Figure 3.15 – PUBREC Packet variable header. The variable header contains the following fields in order, the Packet Identifier from the PUBLISH Packet that is being acknowledged, an optional Return code, and optional Length of Identifier/Value pairs, and the Identifier/Value pairs.

Figure . - 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 | PUBREL Return code | | | | | | | |

#### Publish Return codes

Byte 3 in the Variable header.

The values for the one byte unsigned Publish Return code field are listed in Table 3.6 – PUBREC Return cod. If the Remaining Length is 2, then the Publish Return code has the value 0x00 (Success).

Table . – PUBREC Return codes

|  |  |  |  |
| --- | --- | --- | --- |
| **Value** | **Hex** | **Return 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 is does not know if there are any matching subscribers, it should use the 0x00 (Success) Return code. |
| 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 |
| 139 | 0x8B | Topic Invalid | The topic is invalid |
| 149 | 0x85 | Packet too large | The packet size exceeded the permissible value for this system |
| 154 | 0x9A | QoS level not supported | The QoS level is not supported by this system. |

The Client or Server sending the PUBREC MUST use one of the return codes in Table 3.6 – PUBREC Return codes.

When a PUBREL with a Return code value of 128 or greater, the QoS 2 flow is ended as if a PUBCOMP has been received.

#### Length of Identifier/Value pairs

The length of Identifier/Value pairs in the PUBREC Packet variable header encoded as a Variable Byte Integer. If the Remaining Length is less than 4 there are no Length of Identifier/Value pairs and the value of 0 is used.

#### Reason String

31 (0x1F) Byte Identifier of the Reason String. Followed by the UTF8 encoded string representing the reason for the connection failure. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the receiver.

The sender uses this value to give additional information to the receiver. The sender MUST NOT use this Identifier/Value pair if it would increase the size of the PUBREC packet to beyond the Maximum Packet Size specified by the Client. It is a protocol error to include the Reason String more than once.

### Payload

The PUBREC Packet has no payload.

### Actions

This is fully described in Section 4.3.3.

## PUBREL – Publish release (QoS 2 publish received, part 2)

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

### Fixed header

Figure . – PUBREL Packet fixed header

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

Bits 3,2,1 and 0 of the fixed header in the PUBREL Control 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.

**Remaining Length field**

This is the length of the variable header, encoded as a Variable Byte Integer in the manner described in section **Error! Reference source not found.**. For the PUBREL Packet this has the value 3 if the variable header contains a PUBREL Return code, and 2 otherwise.

### Variable header

The variable header contains the following fields in order, the Packet Identifier from the PUBREC Packet that is being acknowledged, an optional Return code, and optional Length of Identifier/Value pairs, and the Identifier/Value pairs.

Figure . – 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 Return code | | | | | | | |

#### PUBREL Return code

Byte 3 in the Variable header.

The values for the one byte unsigned PUBCOMP Return code field are listed in Table 3.7 - PUBREL Return code values

Table . - PUBREL Return code values

|  |  |  |  |
| --- | --- | --- | --- |
| **Value** | **Hex** | **Return code name** | **Description** |
| 0 | 0x00 | Success | Message released. Publication of QoS 2 message is complete. |
| 146 | 0x92 | PacketID not found | The PacketID is not known. This is not an error during recovery, but at other times indicates a mismatch between the Session state on the Client and Server. |

The Client or Server sending the PUBREL MUST use one of the return codes in Table 3.8 – PUBCOMP Return code v.

#### Length of Identifier/Value pairs

The length of Identifier/Value pairs in the PUBREL Packet variable header encoded as a Variable Byte Integer. If the Remaining Length is less than 4 there is no Length of Identifier/Value pairs and the value of 0 is used.

#### Reason String

31 (0x1F) Byte Identifier of the Reason String. Followed by the UTF8 encoded string representing the reason for the connection failure. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the receiver.

The sender uses this value to give additional information to the receiver. The sender MUST NOT use this Identifier/Value pair if it would increase the size of the PUBREL packet to beyond the Maximum Packet Size specified by the Client. It is a protocol error to include the Reason String more than once.

### Payload

The PUBREL Packet has no payload.

### Actions

This is fully described in Section 4.3.3.

## PUBCOMP – Publish complete (QoS 2 publish received, part 3)

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

### Fixed header

Figure . – PUBCOMP Packet fixed header

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

**Remaining Length field**

This is the length of the variable header, encoded as a Variable Byte Integer in the manner described in section **Error! Reference source not found.**. For the PUBCOMP Packet this has the value 3 if the variable header contains a PUBCOMP Return code, and a value of 2 if it does not.

### Variable header

The variable header format is illustrated in Figure 3.19 – PUBCOMP Packet variable header. The variable header contains the following fields in order, the Packet Identifier from the PUBREL Packet that is being acknowledged, an optional Return code, and optional Length of Identifier/Value pairs, and the Identifier/Value pairs.

Figure . - 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 Return code | | | | | | | |

#### PUBCOMP Return codes

Byte 3 in the Variable header.

The values for the one byte unsigned PUBCOMP Return code field are listed in Table 3.8 – PUBCOMP Return code v.

Table . – PUBCOMP Return code values.

|  |  |  |  |
| --- | --- | --- | --- |
| **Value** | **Hex** | **Return code name** | **Description** |
| 0 | 0x00 | Success | Message released. Publication of QoS 2 message is complete. |
| 146 | 0x92 | PacketID not found | The PacketID is not known. This is not an error during recovery, but at other times indicates a mismatch between the Session state on the Client and Server. |

The Client or Server sending the PUBCOMP MUST use one of the return codes in Table 3.8 – PUBCOMP Return code v.

#### Length of Identifier/Value pairs

The length of Identifier/Value pairs in the PUBCOMP Packet variable header encoded as a Variable Byte Integer. If the Remaining Length is less than 4 there is no Length of Identifier/Value pairs and the value of 0 is used.

#### Reason String

31 (0x1F) Byte Identifier of the Reason String. Followed by the UTF8 encoded string representing the reason for the connection failure. This Reason String is a human readable string designed for diagnostics and should not be parsed by the receiver.

The sender uses this value to give additional information to the receiver. The sender MUST NOT use this Identifier/Value pair if it would increase the size of the PUBCOMP packet to beyond the Maximum Packet Size specified by the Client. It is a protocol error to include the Reason String more than once.

### Payload

The PUBCOMP Packet has no payload.

### Actions

This is fully described in Section 4.3.3.

## SUBSCRIBE - Subscribe to topics

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

### Fixed header

##### 

Figure . SUBSCRIBE Packet fixed header

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

Bits 3,2,1 and 0 of the fixed header of the SUBSCRIBE Control 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.

**Remaining Length field**

This is the length of variable header (2 bytes) plus the length of the payload, encoded as a Variable Byte Integer in the manner described in section **Error! Reference source not found.**.

### Variable header

The variable header contains a Packet Identifier. Section 2.2.1 provides more information about Packet Identifiers.

#### Variable header non normative example

[Figure 3.21 shows a variable header with Packet Identifier set to 10.](#_Figure_3.21_-)

##### 

Figure .- Variable header with a Packet Identifier of 10, Non normative 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 |

### Payload

The payload of a SUBSCRIBE Packet contains a list of Topic Filters indicating the Topics to which the Client wants to subscribe. The Topic Filters in a SUBSCRIBE packet payload MUST be UTF-8 encoded strings as defined in Section **Error! Reference source not found.**. A Server SHOULD support Topic filters that contain the wildcard characters defined in Section [4.7.1](#_Topic_wildcards). It MAY support Shared Subscription Topic Filters defined in Section 4.8. If it chooses not to support Shared Subscription Topic Filters or Topic Filters that contain wildcard characters, it MUST reject any Subscription request whose filter contains them. Each filter is followed by a Subscription Options byte.

The payload of a SUBSCRIBE packet MUST contain at least one Topic Filter / Subscription Options pair. A SUBSCRIBE packet with no payload is a protocol error. See section 0 for information about handling errors.

#### Subscription Options

Bits 0 an 1 of the Subscription Options represent the Maximum QoS field. This gives the maximum QoS level at which the Server can send Application Messages to the Client. It is a protocol error is the Maximum QoS field has the value 3.

Bit 2 of the Subscription Options represents the NoLocal (NL) 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. It is a protocol error to set the NoLocal bit to 1 on a shared subscription and the Server MUST close the Network Connection.

Bit 3 of the Subscription Options represents the Retain As Published (RAP) option. If the value is 0, Application Messages forwarded using this subscription keep the RETAIN flag with which they were published, rather than changing the RETAIN flag to indicate if the Application Message came from a retained source.

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

0 = Send retained messages at the time of the subscribe

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

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

It is a protocol error to send a Retain Handling value of 3.

Bits 6 and 7 of the Subscription Options byte are not used, and are reserved for future use. The Server MUST treat a SUBSCRIBE packet as malformed and close the Network Connection if any of Reserved bits in the payload are non-zero.

**Non normative comment**

The NoLocal and Retain As Published options can be used to implement bridging where the Client is sending the message on to another Server.

Not sending retained messages for an existing subscription is useful when a reconnect is done and the Client is not certain if the subscriptions were complete in the previous connection to the session.

Not sending retained messages on a new subscription is useful for a Client that only wants to see change notifications and does not need to know the initial state.

Figure .– 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 |

#### Payload non normative example

[Figure 3.23 - Payload byte format non normative example shows the payload for the SUBSCRIBE Packet briefly described in Table 3.5 - Payload non normative example.](#_Figure_3.23_-)

##### 

Table . - Payload non normative example

|  |  |
| --- | --- |
| Topic Name | “a/b” |
| Subscription Options | 0x01 |
| Topic Name | “c/d” |
| Subscription Options | 0x02 |

Figure . - 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 |

### Response

When the Server receives a SUBSCRIBE Packet from a Client, the Server MUST respond with a SUBACK Packet. The SUBACK Packet MUST have the same Packet Identifier as the SUBSCRIBE Packet that it is acknowledging.

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

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 completely replace that existing Subscription with a new Subscription. The Topic Filter in the new Subscription will be identical to that in the previous Subscription, although its maximum QoS value could be different. If the Retain Handling option is 0, any existing retained messages matching the Topic Filter MUST be re-sent, but the flow of publications MUST NOT be interrupted [MQTT-3.8.4-3].

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

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

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

See section 4.8 for more details on Shared Subscriptions.

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.

The SUBACK Packet sent by the Server to the Client MUST contain a return code for each Topic Filter/Subscription Option pair. This return code MUST either show the maximum QoS that was granted for that Subscription or indicate that the subscription failed. The Server might grant a lower Maximum QoS than the subscriber requested. 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. The server is permitted to send duplicate copies of a message to a subscriber in the case where the original message was published with QoS 1 and the maximum QoS granted was QoS 0.

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

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

**Non normative comment**

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

## SUBACK – Subscribe acknowledgement

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

A SUBACK Packet contains a list of Return codes, that specify the maximum QoS level that was granted or the error which was found for each Subscription that was requested by the SUBSCRIBE.

### Fixed header

##### 

Figure . - 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 | | | | | | | |

**Remaining Length field**

This is the length of variable header (2 bytes) plus the length of the payload, encoded as a Variable Byte Integer in the manner described in section **Error! Reference source not found.**.

### Variable header

The variable header contains the following fields in order, The Packet Identifier from the SUBSCRIBE Packet that is being acknowledged, the Length of Identifier/Value pairs, and the Identifier/Value pairs. Figure 3.25 - variable header format below illustrates the format of the variable header.

#### Length of Identifier/Value pairs

The length of Identifier/Value pairs in the SUBACK Packet variable header encoded as a Variable Byte Integer. If the Remaining Length is less than 4 there is no Length of Identifier/Value pairs and the value of 0 is used.

#### Reason String

31 (0x1F) Byte Identifier of the Reason String. Followed by the UTF8 encoded string representing the reason for the connection failure. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the Client.

The Server uses this value to give additional information to the Client. The Server MUST NOT use this Identifier/Value pair if it would increase the size of the SUBACK packet to beyond the Maximum Packet Size specified by the Client. It is a protocol error to include the Reason String more than once.

Figure . SUBACK Packet variable header

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

### Payload

The payload contains a list of Return Codes. Each Return code corresponds to a Topic Filter in the SUBSCRIBE Packet being acknowledged. The order of return codes in the SUBACK Packet MUST match the order of Topic Filters in the SUBSCRIBE Packet.

Table . - Subscribe Return codes

|  |  |  |  |
| --- | --- | --- | --- |
| **Value** | **Hex** | **Return code name** | **Description** |
| 0 | 0x00 | Maximum 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 | Maximum 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 | Maximum 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 Return 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 |
| 144 | 0x90 | Topic Filter not valid | The topic filter is valid MQTT but is not allowed for this Client. |
| 145 | 0x91 | PacketID in use | The specified packet identifier is already in use |
| 151 | 0x97 | Shared Subscription not supported | The Server does not support shared subscriptions for this Client |

The Server MUST send one of the return codes listed in Table 3.10 - Subscribe Return codes.

#### Payload non normative example

[Figure 3.27 - Payload byte format non normative example shows the payload for the SUBACK Packet briefly described in](#_Figure_3.27_-)  Table 3.11 Payload non normative example.

Table . Payload non normative example

|  |  |
| --- | --- |
| Success - Maximum QoS 0 | 0 |
| Success - Maximum QoS 2 | 2 |
| Failure | 128 |

##### Figure 3.27 - Payload byte format non normative example

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Description** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| byte 1 | Success - Maximum QoS 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 2 | Success - Maximum QoS 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| byte 3 | Failure | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## UNSUBSCRIBE – Unsubscribe from topics

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

### Fixed header

##### Figure 3.28 – UNSUBSCRIBE Packet Fixed header

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

Bits 3,2,1 and 0 of the fixed header of the UNSUBSCRIBE Control 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.

**Remaining Length field**

This is the length of variable header (2 bytes) plus the length of the payload, encoded as a Variable Byte Integer in the manner described in section **Error! Reference source not found.**.

### Variable header

The variable header contains a Packet Identifier. Section 2.2.1 provides more information about Packet Identifiers.

##### Figure 3.29 – UNSUBSCRIBE Packet variable header

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

#### Length of Identifier/Value pairs

The length of Identifier/Value pairs in the UNSUBACK Packet variable header encoded as a Variable Byte Integer. If the Remaining Length is less than 4 there is no Length of Identifier/Value pairs, and the value of 0 is used.

#### Reason String

31 (0x1F) Byte Identifier of the Reason String. Followed by the UTF8 encoded string representing the reason for the connection failure. This Reason String is a human readable string designed for diagnostics and SHOULD NOT be parsed by the Client.

The Server uses this value to give additional information to the Client. The Server MUST NOT use this Identifier/Value pair if it would increase the size of the UNSUBACK packet to beyond the Maximum Packet Size specified by the Client. It is a protocol error to include the Reason String more than once.

### Payload

The payload for the UNSUBSCRIBE Packet contains the list of Topic Filters that the Client wishes to unsubscribe from. The Topic Filters in an UNSUBSCRIBE packet MUST be UTF-8 encoded strings as defined in Section **Error! Reference source not found.**, packed contiguously.

The Payload of an UNSUBSCRIBE packet MUST contain at least one Topic Filter. An UNSUBSCRIBE packet with no payload is a protocol violation. See section 0 for information about handling errors.

#### Payload non normative example

[Figure 3.30 - Payload byte format non normative example show the payload for the UNSUBSCRIBE Packet briefly described in Table3.7 - Payload non normative example.](#_Figure_3.30_-)

##### Table3.7 - Payload non normative example

|  |  |
| --- | --- |
| Topic Filter | “a/b” |
| Topic Filter | “c/d” |

##### Figure 3.30 - Payload byte format non normative example

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Description** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| Topic Filter | | | | | | | | | |
| byte 1 | Length MSB (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 2 | Length LSB (3) | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| byte 3 | ‘a’ (0x61) | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| byte 4 | ‘/’ (0x2F) | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| byte 5 | ‘b’ (0x62) | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| Topic Filter | | | | | | | | | |
| byte 6 | Length MSB (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 7 | Length LSB (3) | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| byte 8 | ‘c’ (0x63) | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| byte 9 | ‘/’ (0x2F) | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| byte 10 | ‘d’ (0x64) | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |

### Response

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 is deleted, otherwise no additional processing occurs. 

If a Server deletes a Subscription:

* It MUST stop adding any new messages for delivery to the Client.
* It MUST complete the delivery of any QoS 1 or QoS 2 messages which it has started to send to the Client.
* It MAY continue to deliver any existing messages buffered for delivery to the Client.

The Server MUST respond to an UNSUBSCRIBE request by sending an UNSUBACK packet. 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.

If a Server receives an UNSUBSCRIBE packet that contains multiple Topic Filters, it MUST handle that packet as if it had received a sequence of multiple UNSUBSCRIBE packets, except that it sends just one UNSUBACK response.

## UNSUBACK – Unsubscribe acknowledgement

The UNSUBACK Packet is sent by the Server to the Client to confirm receipt of an UNSUBSCRIBE Packet.

### Fixed header

##### Figure 3.31 – UNSUBACK Packet fixed header

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

**Remaining Length field**

This is the length of the variable header, encoded as a Variable Byte Integer in the manner described in section **Error! Reference source not found.**. For the UNSUBACK Packet this has the value 3.

### Variable header

The variable header format is illustrated in Figure 3.32 – UNSUBACK Packet variable header. The variable header contains the same Packet Identifier as the UNSUBSCRIBE Packet that is being acknowledged and a Return code.

##### 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 | | | | | | | |

### Payload

The payload contains a list of Return codes. Each Return code corresponds to a Topic Filter in the UNSUBSCRIBE Packet being acknowledged. The order of return codes in the UNSUBACK Packet MUST match the order of Topic Filters in the UNSUBSCRIBE Packet.

The values for the one byte unsigned Unsubscribe Return codes are listed in Table 3.12 - Unsubscribe Return codes. The Server MUST use one of the Return code values from this table.

Table . - Unsubscribe Return codes

|  |  |  |  |
| --- | --- | --- | --- |
| **Value** | **Hex** | **Return code name** | **Description** |
| 0 | 0x00 | 0x00 Success | The subscription is deleted |
| 17 | 0x11 | No subscription existed | No matching subscription existed |
| 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 Return 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 |
| 144 | 0x90 | Topic filter not valid | The topic filter is valid but is not allowed for this Client. |
| 145 | 0x91 | PacketID in use | The specified packet identifier is already in use |

The Server MUST use one of the Return code values in Table 3.12 - Unsubscribe Return codes.

## PINGREQ – PING request

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

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

This Packet is used in Keep Alive processing, see Section 0 for more details.

### Fixed header

##### Figure 3.33 – PINGREQ Packet fixed header

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

### Variable header

The PINGREQ Packet has no variable header.

### Payload

The PINGREQ Packet has no payload.

### Response

The Server MUST send a PINGRESP Packet in response to a PINGREQ Packet.

## PINGRESP – PING response

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

This Packet is used in Keep Alive processing, see Section 0 for more details.

### Fixed header

##### Figure 3.34 – PINGRESP Packet fixed header

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

### Variable header

The PINGRESP Packet has no variable header.

### Payload

The PINGRESP Packet has no payload.

### Actions

The Client takes no action on receiving this Packet

## DISCONNECT – Disconnect notification

The DISCONNECT Packet is the final Control Packet sent from the Client or the Server. It indicates the reason the Network Connection is closed. The Client or Server MAY send a DISCONNECT Packet before closing the Network Connection. If the Client closes the Network Connection without first sending a DISCONNECT and the Connection has a Will Message, the Will Message is sent.

A Server MUST NOT send a DISCONNECT until after it has sent a CONNECT with Return code of less than 128.

### Fixed header

##### Figure 3.35 – DISCONNECT Packet fixed header

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

The Client or Server MUST validate that reserved bits are set to zero and close the Network Connection if they are not zero.

**Remaining Length field**

This is the length of variable header encoded as a Variable Byte Integer in the manner described in section **Error! Reference source not found.**.

### Variable header

The variable header contains the following fields in the order: Disconnect Return code, Length in bytes of Identifier/Value pairs, the Identifier/Value pairs. The rules for encoding Identifier/Value pairs are described in 2.2.2.

#### Disconnect Return code

Byte 1 in the Variable header.

The values for the one byte unsigned Disconnect Return code field are listed Table 3.13 – Disconnect Return code value.

Table . – Disconnect Return code value

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Value** | **Hex** | **Return code name** | **Sent by** | **Description** |
| 0 | 0x00 | Normal completion | Client | Close the connection normally. Do not send the 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 Return codes apply. |
| 129 | 0x81 | Malformed control 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 this Client. |
| 139 | 0x8B | Server shutting down | Server | The Server is shutting down. |
| 140 | 0x8C | Maximum connect time | Server | The maximum connection time authorized for this user has been exceeded. |
| 142 | 0x8E | Session taken over | Server | Another Connection using the same ClientId has connected causing this Connection to be closed. |
| 143 | 0x8F | Keep alive timeout | Server | The Connection is closed because no Packet has been received for 1.5 times the Keep alive time. |
| 144 | 0x90 | Topic name or filter not valid | Client or Server | The topic name or filter is valid but is not accepted by this Client or Server |
| 149 | 0x95 | Packet too large | Client or Server | The packet size is too large for this Client or Server |
| 150 | 0x96 | Message rate too high | Client or Server | The rate of publish is too high |
| 151 | 0x97 | Quota exceeded | Client or Server | An implementation imposed limit has been exceeded |
| 152 | 0x98 | Administrative action | Client or Server | The Connection is closed due to an administrative action. |
| 153 | 0x99 | Disconnect with Will message | Client | The client wishes to disconnect but requires that the Server also publishes its will message. |
| 154 | 0x9A | Alias Not Accepted | Client or Server | The requested Topic Alias was too large. |
| 155 | 0x9B | No Topic | Client or Server | No Topic Name given, or Topic Alias was invalid. |
| 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. |

**Non normative comment**

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

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

#### Length of Identifier/Value pairs

The length of Identifier/Value pairs in the DISCONNECT Packet variable header encoded as a Variable Byte Integer.

#### Session Expiry Interval.

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

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

If the Session Expiry interval is absent the Session Expiry in the CONNECT packet is used.

If the Session Expiry in the CONNECT packet was zero it is a protocol error to set a non-zero Session Expiry in the DISCONNECT packet.

The Session Expiry interval MUST NOT be sent on a DISCONNECT by the Server.

#### Reason String

31 (0x1F) Byte Identifier of the Reason String. Followed by the UTF8 encoded string representing the reason for the disconnect. This Reason String is a human readable string designed for diagnostics and should not be parsed by the receiver.

The sender MUST NOT use this Identifier/Value pair if it would increase the size of the DISCONNECT packet to beyond the Maximum Packet Size specified by the receiver. It is a protocol error to include the Reason String more than once.

**Non normative comment**

Proper uses for the reason string include putting this information into an exception message or writing this string to a log.

#### Server Reference

28 (0x1C) Byte Identifier of the Server Reference. Followed by a UTF8 encoded string which can be used by the Client to identify another Server to use. It is a protocol error to include the Server Reference more than once.

The Server MUST only use send a Server Reference along with a Return Code of 0x9C – Use another server or 0x9D – Server moved.

See section 4.11 Server Redirection for information about how Server Reference is used.

Figure . DISCONNECT Packet variable header non normative example

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Description** | **7** | | **6** | **5** | | **4** | | **3** | | **2** | | **1** | | **0** | |
| Connect Return code | | | | | | | | | | | | | | | | |
| byte 1 |  | 0 | | 0 | 0 | | 0 | | 0 | | 0 | | 0 | | 0 | |
| Identifier/Value fields | | | | | | | | | | | | | | | | |
| byte 2 | Length (2) | | 0 | 0 | | 0 | | 0 | | 0 | | 0 | | 1 | | 0 |
| byte 3 | Session Expiry Identifier (17) | | 0 | 0 | | 0 | | 1 | | 0 | | 0 | | 0 | | 1 |
| byte 4 | Session Expiry (0) | | 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 | 0 | | 0 | | | 0 | 0 | |

### Payload

The DISCONNECT Packet has no payload.

### Actions

After sending a DISCONNECT Packet the sender:

* MUST close the Network Connection.
* MUST NOT send any more Control Packets on that Network Connection.

On receipt of DISCONNECT with a Return code less than 128 by the Server it:

* MUST discard any Will Message associated with the current Connection without publishing it, as described in Section 3.1.2.5.

On receipt of DISCONNECT by the Client or Server they:

* SHOULD close the Network Connection

## AUTH – Authentication exchange

An AUTH Packet is sent from Client to Server or Server to Client as part of the authentication exchange. This allows for challenge / response authentication. The AUTH packet MUST be sent after a CONNECT Packet and before a CONNACK Packet.

### Fixed header

##### Figure 3.35 – AUTH Packet fixed header

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

Bits 3,2,1 and 0 of the fixed header of the AUTH Control Packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Client or Server MUST treat any other value as malformed and close the Network Connection.

**Remaining Length field**

This is the length of variable header encoded as a Variable Byte Integer in the manner described in section **Error! Reference source not found.**.

### Variable header

The variable header contains the following fields in the order: Authenticate Return code, Length in bytes of Identifier/Value pairs, the Identifier/Value pairs. The rules for encoding Identifier/Value pairs are described in 2.2.2.

#### Authenticate Return code

Byte 1 in the Variable header. The values for the one byte unsigned Disconnect Return code field are listed Table 3.13 – Disconnect Return code value.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Value** | **Hex** | **Return code name** | **Sent by** | **Description** |
| 0 | 0x00 | Success | Client | Authentication step accepted by Client |
| 34 | 0x18 | Continue authentication | Server | Continue the authentication with another step |

#### Length of Identifier/Value pairs.

The length of Identifier/Value pairs in the DISCONNECT Packet variable header encoded as a Variable Byte Integer.

#### Auth Method

21 (0x15) Byte, Identifier of the Auth Method. Followed by a UTF8 encoded string containing the name of the authentication method. See section 4.12 to understand how extended authentication works.

#### Auth Data

22 (0x16) Byte, Identifier of the Auth Data. Followed by BinaryData containing authentication data. The contents of this data are defined by the authentication method and the state of already exchanged authentication data. See section 4.12 to understand how extended authentication works.

### Payload

The AUTH Packet has no payload.

### Actions

See section 4.12 for a description of how extended authentication works.

# Operational behavior

## Storing state

It is necessary for the Client and Server to store Session state in order to provide Quality of Service guarantees. The Client and Server MUST store Session state for the entire duration of the Session. A Session MUST last at least as long it has an active Network Connection.

Retained messages do not form part of the Session state in the Server. The Server SHOULD retain such messages until deleted by a Client.

**Non normative comment**

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

**Non normative comment**

It is possible that hardware or software failures may result in loss or corruption of Session state stored by the Client or Server.

**Non normative comment**

Normal operation of the Client of Server could mean that stored state is lost or corrupted because of administrator action, hardware failure or software failure. An administrator action could be an automated response to defined conditions. These actions might be prompted by resource constraints or for other operational reasons. For example, the server might determine that based on external knowledge, a message or messages can no longer be delivered to any current or future client.

**Non normative comment**

An MQTT user should evaluate the storage capabilities of the MQTT Client and Server implementations to ensure that they are sufficient for their needs.

### Non normative example

For example, a user wishing to gather electricity meter readings may decide that they need to use QoS 1 messages because they need to protect the readings against loss over the network, however they may have determined that the power supply is sufficiently reliable that the data in the Client and Server can be stored in volatile memory without too much risk of its loss.

Conversely a parking meter payment application provider might decide that there are no circumstances where a payment message can be lost so they require that all data are force written to non-volatile memory before it is transmitted across the network.

## Network Connections

The MQTT protocol requires an underlying transport that provides an ordered, lossless, stream of bytes from the Client to Server and Server to Client.

**Non normative comment**

The transport protocol used to carry MQTT 3.1 was TCP/IP as defined in [[RFC793](#RFC793)]. TCP/IP can be used for MQTT 3.1.1. **[RFC793]**

following are also suitable: following are also suitable:

* TLS [RFC5246]
* WebSocket [RFC6455]

**Non normative comment**

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

Connectionless network transports such as [User Datagram Protocol](http://en.wikipedia.org/wiki/User_Datagram_Protocol) (UDP) are not suitable on their own because they might lose or reorder data.

## Quality of Service levels and protocol flows

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

The non-normative flow diagrams in the following sections are intended to show possible implementation approaches.

### QoS 0: At most once delivery

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

In the QoS 0 delivery protocol, the Sender

* MUST send a PUBLISH packet with QoS=0, DUP=0.

In the QoS 0 delivery protocol, the Receiver

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

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

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

### QoS 1: At least once delivery

This quality of service ensures that the message arrives at the receiver at least once. A QoS 1 PUBLISH Packet has a Packet Identifier in its variable header and is acknowledged by a PUBACK Packet. Section 2.2.1 provides more information about Packet Identifiers.

In the QoS 1 delivery protocol, the Sender

* MUST assign an unused Packet Identifier each time it has a new Application Message to publish.
* MUST send a PUBLISH Packet containing this Packet Identifier with QoS=1, DUP=0.
* MUST treat the PUBLISH Packet as “unacknowledged” until it has received the corresponding PUBACK packet from the receiver. See Section 4.4 for a discussion of unacknowledged messages.

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

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

In the QoS 1 delivery protocol, the Receiver

* MUST respond with a PUBACK Packet containing the Packet Identifier from the incoming PUBLISH Packet, having accepted ownership of the Application Message
* 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 publication, irrespective of the setting of its DUP flag.

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

|  |  |  |
| --- | --- | --- |
| **Sender Action** | **Control Packet** | **Receiver action** |
| Store message |  |  |
| Send PUBLISH QoS 1, DUP 0,  <Packet Identifier> | ----------> |  |
|  |  | Initiate onward delivery of the Application Message1 |
|  | <---------- | Send PUBACK <Packet Identifier> |
| Discard message |  |  |

1 The receiver is not required 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.

### QoS 2: Exactly once delivery

This is the highest quality of service, for use when neither loss nor duplication of messages are acceptable. There is an increased overhead associated with this quality of service.

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.
* MUST send a PUBLISH packet containing this Packet Identifier with QoS=2, DUP=0.
* MUST treat the PUBLISH packet as “unacknowledged” until it has received the corresponding PUBREC packet from the receiver. See 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 Return code value less than 128. This PUBREL packet MUST contain the same Packet Identifier as the original PUBLISH packet.
* MUST treat the PUBREL packet as “unacknowledged” until it has received the corresponding PUBCOMP packet from the receiver.
* MUST NOT re-send the PUBLISH once it has sent the corresponding PUBREL packet.
* MUST NOT apply Publication Expiry if a Publish Packet has been sent.

The Packet Identifier becomes available for reuse once the Sender has received the PUBCOMP Packet or a PUBREL with a Return code of 128 or greater.

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

In the QoS 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.
* After it has sent a PUBREC with a Return code of 128 or greater, the receiver MUST treat any subsequent PUBLISH packet that contains that Packet Identifier as being a new publication.
* 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.
* MUST respond to a PUBREL packet by sending a PUBCOMP packet containing the same Packet Identifier as the PUBREL.
* After it has sent a PUBCOMP, the receiver MUST treat any subsequent PUBLISH packet that contains that Packet Identifier as being a new publication.
* MUST NOT apply Publication Expiry until after PUBCOMP has been sent.

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

|  |  |  |
| --- | --- | --- |
| **Sender Action** | **Control Packet** | **Receiver Action** |
| Store message |  |  |
| PUBLISH QoS 2, DUP 0  <Packet Identifier> |  |  |
|  | ----------> |  |
|  |  | Method A, Store message  or Method B, Store <Packet Identifier> then Initiate onward delivery of the Application Message1 |
|  |  | PUBREC <Packet Identifier> |
|  | <---------- |  |
| Discard message, Store PUBREC received <Packet Identifier> |  |  |
| PUBREL <Packet Identifier> |  |  |
|  | ----------> |  |
|  |  | Method A, Initiate onward delivery of the Application Message1  then discard message  or Method B, Discard <Packet Identifier> |
|  |  | Send PUBCOMP <Packet Identifier> |
|  | <---------- |  |
| Discard stored state |  |  |

1 The receiver is not required 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.

[Figure 4.3 shows that there are two methods by which QoS 2 can be handled by the receiver. They differ in the point within the flow at which the message is made available for onward delivery. The choice of Method A or Method B is implementation specific. As long as an implementation chooses exactly one of these approaches, this does not affect the guarantees of a QoS 2 flow.](#_Figure_4.3_–)

## Message delivery retry

When a Client reconnects with CleanSession set to 0, 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.

**Non normative comment**  
Historically retransmission of Control Packets was required to overcome data loss on some older TCP networks. This might remain a concern where MQTT 3.1.1 implementations are to be deployed in such environments.

## Message receipt

When a Server takes ownership of an incoming Application Message it MUST add it to the Session state of those clients that have matching Subscriptions. Matching rules are defined in Section 4.7.

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

## Message ordering

A Client MUST follow these rules when implementing the protocol flows defined elsewhere in this chapter:

* When it 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)
* It MUST send PUBACK packets in the order in which the corresponding PUBLISH packets were received (QoS 1 messages)
* It MUST send PUBREC packets in the order in which the corresponding PUBLISH packets were received (QoS 2 messages)
* It MUST send PUBREL packets in the order in which the corresponding PUBREC packets were received (QoS 2 messages)

A Server MUST by default treat each Topic as an "Ordered Topic". It MAY provide an administrative or other mechanism to allow one or more Topics to be treated as an "Unordered Topic".

When a Server processes a message that has been published to an Ordered Topic, it MUST follow the rules listed above when delivering messages to each of its subscribers. In addition, it MUST send PUBLISH packets to consumers (for the same Topic and QoS) in the order that they were received from any given Client.

**Non normative comment**

The rules listed above ensure that when a stream of messages is published and subscribed to with QoS 1, the final copy of each message received by the subscribers will be in the order that they were originally published in, but the possibility of message duplication could result in a re-send of an earlier message being received after one of its successor messages. For example, a publisher might send messages in the order 1,2,3,4 and the subscriber might receive them in the order 1,2,3,2,3,4.

If both Client and Server make sure that no more than one message is “in-flight” at any one time (by not sending a message until its predecessor has been acknowledged), then no QoS 1 message will be received after any later one - for example a subscriber might receive them in the order 1,2,3,3,4 but not 1,2,3,2,3,4. Setting an in-flight window of 1 also means that order will be preserved even if the publisher sends a sequence of messages with different QoS levels on the same topic.

## Topic Names and Topic Filters

### Topic wildcards

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

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

The wildcard characters can be used in Topic Filters, but MUST NOT be used within a Topic Name.

#### Topic level separator

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

#### Multi-level wildcard

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

**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

#### 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. It can be used at more than one level in the Topic Filter and can be used in conjunction with the multilevel 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/”.

**Non normative comment**

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

### Topics beginning with $

The Server MUST NOT match Topic Filters starting with a wildcard character (# or +) with Topic Names beginning with a $ character. The Server SHOULD prevent Clients from using such Topic Names to exchange messages with other Clients. Server implementations MAY use Topic Names that start with a leading $ character for other purposes.

**Non normative comment**

* $SYS/ has been widely adopted as a prefix to topics that contain Server-specific information or control APIs
* Applications cannot use a topic with a leading $ character for their own purposes

**Non normative comment**

* A subscription to “#” will not receive any messages published to a topic beginning with a $
* A subscription to “+/monitor/Clients” will not receive any messages published to “$SYS/monitor/Clients”
* A subscription to “$SYS/#” will receive messages published to topics beginning with “$SYS/”
* A subscription to “$SYS/monitor/+” will receive messages published to “$SYS/monitor/Clients”
* For a Client to receive messages from topics that begin with $SYS/ and from topics that don’t begin with a $, it has to subscribe to both “#” and “$SYS/#”

### Topic semantic and usage

The following rules apply to Topic Names and Topic Filters:

* All Topic Names and Topic Filters MUST be at least one character long
* Topic Names and Topic Filters are case sensitive
* Topic Names and Topic Filters can include the space character
* A leading or trailing ‘/’ creates a distinct Topic Name or Topic Filter
* A Topic Name or Topic Filter consisting only of the ‘/’ character is valid
* Topic Names and Topic Filters MUST NOT include the null character (Unicode U+0000) [Unicode]
* Topic Names and Topic Filters are UTF-8 encoded strings; they MUST NOT encode to more than 65535 bytes. See Section **Error! Reference source not found.**

There is no limit to the number of levels in a Topic Name or Topic Filter, other than that imposed by the overall length of a UTF-8 encoded string.

When it performs subscription matching the Server MUST NOT perform any normalization of Topic Names or Topic Filters, or any modification or substitution of unrecognized characters. Each non-wildcarded level in the Topic Filter has to match the corresponding level in the Topic Name character for character for the match to succeed.

**Non normative comment**

The UTF-8 encoding rules mean that the comparison of Topic Filter and Topic Name could be performed either by comparing the encoded UTF-8 bytes, or by comparing decoded Unicode characters

**Non normative comment**

* “ACCOUNTS” and “Accounts” are two different topic names
* “Accounts payable” is a valid topic name
* “/finance” is different from “finance”

An Application Message is sent to each Client Subscription whose Topic Filter matches the Topic Name attached to an Application Message. The topic resource MAY be either predefined in the Server by an administrator or it MAY be dynamically created by the Server when it receives the first subscription or an Application Message with that Topic Name. The Server MAY also use a security component to selectively authorize actions on the topic resource for a given Client.

## Subscriptions

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

### Non-Shared Subscriptions

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

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

If there are multiple clients, each with its own Non-Shared Subscription to the same Topic, each Client gets its own copy of the Application Messages that are published on that Topic. This means that Non-Shared Subscriptions cannot be used to load-balance Application Messages across multiple consuming clients as in such cases you would want each message to be delivered to exactly one Client.

### Shared Subscriptions

A Shared Subscription can be associated with multiple subscribing MQTT Sessions. Like a Non-Shared Subscription it has a Topic Filter and a Subscription Options, however a publication that matches its Topic Filter is only sent to one of its subscribing Sessions. This makes it suitable for the load-balancing case.

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

$share/{ShareName}/{filter} 

* $share is a literal string that marks the Topic Filter as being a Shared Subscription Topic Filter.
* {ShareName} is a character string that does not include "/", "+" or "#"
* {filter} The remainder of the string has the same syntax and semantics as a Topic Filter in a non-shared subscription - see section **4.7.**

A Shared Subscription's Topic Filter MUST start with $share/ and MUST contain a ShareName that is at least one character long. The ShareName MUST NOT contain the characters "/", "+" or "#", but MUST be followed by a "/" character. This "/" character MUST be followed by a Topic Filter as described in section 4.7.

**Non normative comment**

Shared Subscriptions are defined at the scope of the MQTT Server, rather than of a Session. A ShareName is included in the Shared Subscription's Topic Filter so that there can be more than one Shared Subscription on a Server that has the same {filter} component. An application could choose to use the ShareName to represent the group of subscribing Sessions that are sharing the subscription, but it is not obliged to do this.   
  
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 them having the same ShareName.

A Shared Subscription is created by using a Shared Subscription Topic Filter in a SUBSCRIBE request. So long as only one Session subscribes to a particular Shared Subscription, the shared subscription behaves like a non-shared subscription, except that

* The $share and {ShareName}portions of the Topic Filter are not taken into account when matching against publications.
* No Retained Messages are sent to the Session when it first subscribes. It will be sent other matching messages as they are published.

Once a Shared Subscription exists it is possible for other Sessions to subscribe with the same Shared Subscription Topic Filter. The new Session is associated with the Shared Subscription as an additional subscriber. Retained messages are not sent to this new subscriber. Each subsequent Application Message that matches the Shared Subscription is now sent to one and only one of the Sessions that are subscribed to the Shared Subscription.   
  
A Session can explicitly detach itself from a Shared Subscription by sending an UNSUBSCRIBE Packet that contains the full Shared Subscription Topic Filter. Sessions are also detached from the Shared Subscription when they terminate.   
  
A Shared Subscription lasts for as long as it is associated with at least one Session (i.e. a Session that has issued a successful SUBSCRIBE request to its Topic Filter and that has not completed a corresponding UNSUBSCRIBE). A Shared Subscription survives when the Session that originally created it unsubscribes, unless there are no other Sessions left when this happens. A Shared Subscription ends, and any undelivered messages associated with it are deleted, when there are no longer any Sessions subscribed to it.

Notes on Shared Subscriptions

* If there's more than one Session subscribed to the Shared Subscription, the server implementation is free to choose, on a message by message basis, which Session to use and what criteria it uses to make this selection.
* Different subscribing clients are permitted to ask for different Requested QoS levels in their SUBSCRIBE request packets. The server decides which Maximum QoS to grant to each client, and it is permitted to grant different Maximum QoS levels to different subscribers. When sending an Application Message to a Client, the Server MUST respect the granted QoS for the Client's subscription, in the same that it does when sending a message to a Non-Shared subscriber.
* If the Server is in the process of sending a QoS 2 message to its chosen subscribing Client and the connection to that Client breaks before delivery is complete the Server MUST complete the delivery of the message to that Client when it reconnects as described in section 4.3.3. If the client's Session terminates before the Client reconnects, the Server MUST NOT send the Application Message to any other subscribed client.
* If the Server is in the process of sending a QoS 1 message to its chosen subscribing Client and the connection to that Client breaks before the Server has received an acknowledgement from the Client, the Server MAY wait for the Client to reconnect and retransmit the message to that Client when that happens. If the Client's Session terminates before the Client reconnects, the Server SHOULD send the Application Message to another Client that is subscribed to the same Shared Subscription. It MAY attempt to send the message to another client as soon as it loses its connection to the first client.
* If a Client responds with a negative acknowledgement to a PUBLISH packet from the Server, the Server MUST discard the Application Message and not attempt to send it to any other Subscriber.
* A Client is permitted to submit a second SUBSCRIBE request to a Shared Subscription on a Session that's already subscribed to that Shared Subscription - it might for example do this in order to change the Requested QoS for its subscription. This does not increase the number of times that that Session is associated with the Shared Subscription, so the Session will leave the Shared Subscription on its first UNSUBSCRIBE.
* Each Shared Subscription is independent from any other. It is possible to have two Shared Subscriptions with overlapping filters. In such cases a message that matches both Shared Subscriptions will be processed separately by both of them. If a Client has a Shared Subscription and a Non-Shared Subscription and a message matches both of them, the Client will receive a copy of the message by virtue of it having the Non-Shared Subscription. A second copy of the message will be delivered to one of the subscribers to the Shared Subscription, and this could result in a second copy being sent to this client.

## Flow Control

Both Clients and Servers may control arrival bursts by advertising a Receive Maximum value as described in section 3.1.2.14 and 3.2.2.5. The Receive Maximum establishes a quota which is used to limit the number of PUBLISH QoS > 0 packets which can be sent without receiving an PUBACK (for QoS 1) or PUBCOMP (for QoS 2). The PUBACK and PUBCOMP replenish the quota in the manner described below.

When a Client or Server receives a Receive Maximum advertisement value, it sets its send quota to that value. It may use a smaller value, but the value chosen must always be in range of [1...Receive Maximum]. If the Receive Maximum value is missing from the CONNECT or CONNACK packets, the maximum value of 65535 is used. It is a protocol error to send a Receive Maximum value of zero.

Each time the Client or Server sends a PUBLISH packet at QoS > 0, it decrements the quota. If the quota becomes zero, the client or server cannot send any more PUBLISH commands at QoS > 0. It MAY continue to send PUBLISH packets at QoS 0, or MAY choose to suspend sending these as well. The Client and Server MUST continue to process and respond to all other command packets even if the quota is zero.

Each time a PUBACK or PUBCOMP are received, the quota is incremented by 1. The increment is applied regardless of whether the PUBACK or PUBCOMP carried an error code. The quota must not be allowed to grow larger than the original quota or Receive Maximum. In the event of QoS 2 retransmission, it is possible for a PUBREL to be sent, causing a duplicate PUBCOMP to be received. If applying the quota update would cause the quota to exceed the original value, it is not applied. In this way, the maximum level is 'clamped'.

The quota and Receive Maximum value are not preserved across transport connections, and are re-initialized with each new transport connection as described above. They are not part of the session state.

## Request / Response

{Not yet complete]

## Server redirection

A Server can request the Client to use another Server by sending the Return codes 0x9C (Use another server), or 0x9D (Server moved) on a CONNACK or DISCONNECT Packet. When sending one of these Return codes, the Server MAY also send a Server Reference Identifier/Value pair to indicate the location of the Server or Servers the Client should use.

The Return code 0x9C (Use another server) specifies that the Client should temporarily change to using another server. Which other Server to use can be either be known to the Client or specified using a Server Reference.

The Return code 0x9D (Server moved) specifies that the Client should permanently change to using another server. Which other Server to use can either be known to the Client, or specified using a Server Reference.

The Server Reference is an Identifier/Value pair which is a UTF8 encoded string. The value of this string is a space separated list of references. Each reference consists of a name optionally followed by a colon and a port number. If the name contains a colon the name string can be enclosed within square brackets (“[“ and ‘]”). A name enclosed by square brackets MUST NOT contain the right square bracket (“]”) character. This is used to represent an IPv6 literal address which uses colon separators.

The name within a Server Reference commonly represents a host name, DNS name, SRV name, or literal IP address.

The value following the colon separator is commonly a set of digits representing the port. This is not needed where this information comes from the name resolution (such as with SRV) or is defaulted.

**Non normative comments**

The format of the Server Reference is designed to be a simplified version of an URI authority field.

The Server is not required to give a Server Reference, and the Client is not required to follow a Server Reference. This feature can be used to allow for load balancing, server relocation, and client provisioning to a server.

## Enhanced authentication

The CONNECT Packet provides data for basic authentication of a Network Connection using the Username and Password fields. While these fields are named for a simple password authentication, they can be used to carry other forms of authentication such as passing a token as the password.

Enhanced authentication extends this basic authentication to include challenge / response style authentication. This enhanced authentication is done by exchanging AUTH Packets between the Client and the Server between the CONNECT and CONNACK.

The implementation of enhanced authentication is optional for both Clients and Servers. If the Client does not include an Auth Method in the CONNECT, the Server MUST NOT send an AUTH Packet, and it MUST NOT send an Auth Method in the CONNACK Packet. If the Server does not send an Auth Method on an AUTH or CONNACK Packet, the Client MUST NOT send an AUTH Packet to the Server.

To begin an enhanced authentication, the Client includes an Auth Method in the CONNECT Packet. This specifies the authentication method to use. The Client needs to know the authentication methods that the Server supports and choose one of them. It also needs to possess a set of credentials that the Server will accept. If the Server does not support the Auth Method supplied by the Client, it MAY send a CONNACK with a Return code of 0x8C (Bad authentication method) or 0x87 (Not Authorized), and MUST close the Network Connection.

If the Client does not include an Auth Method in the CONNECT Packet, the Server SHOULD authenticate using the information in the CONNECT packet, TLS session, and Network Connection.

The Auth Method is an agreement between the Client and Server about the meaning of the data sent in the Auth Data and any of the other fields on CONNECT, and of the processing required by the Client and Server to complete the authentication.

**Non normative comment**

The Auth Method is commonly a SASL Mechanism, and using such a registered name aids interchange. However, the Auth Method is not constrained to using registered SASL Mechanisms.

If the Auth Method selected by the Client specifies that the Client sends data first, the Client should include an Auth Data field in the CONNECT packet. For contents of the Auth Data is defined by the Auth Method.

If the Server requires additional information to complete the authorization, it sends an AUTH Packet with a Return code of 0x18 – Continue authentication. It MUST also set the Auth Method to the same value sent by the Client. If the authentication method requires the Server to send authentication data to the Client it is sent in the Auth Data field. The Server can fail the authentication at any time and MAY send a CONNACK with a Return code of 128 or above and MUST close the Network Connection.

The Client responds to the Server sending an AUTH Packet with a Return code of 0x18 – Continue authentication by sending an AUTH Packet with a Return code of 0. The Client MUST set the Auth Method to the same value sent to it, and if the authentication mechanism requires more data, sends that as Auth Data. The Client can close the connection at any point, and can sent a DISCONNECT Packet before doing so.

The Client and Server can exchange AUTH Packets as required until the Server accepts the authentication by sending a CONNACK with a Return code of 0. The Server MUST set the Auth Method in the CONNACK equal to the Auth Method the Client set on the CONNECT.

### Non normative example showing a SCRAM challenge

Client to Server: CONNECT Auth Method="SCRAM-SHA-1" Auth Data=client-first-data   
Server to Client: AUTH rc=0x18 Auth Method="SCRAM-SHA-1" Auth Data=server-first-data   
Client to Server AUTH rc=0 Auth Method="SCRAM-SHA-1" Auth Data=client-final-data   
Server to Client CONNACK rc=0 Auth Method="SCRAM-SHA-1" Auth Data=server-final-data

### Non normative example showing a Kerberos challenge client to server

Client to Server CONNECT Auth Method="GS2-KRB5"   
Server to Client AUTH rc=0x18 auth" Auth Method="GS2-KRB5"   
Client to Server AUTH rc=0 Auth Method="GS2-KRB5" Auth Data=inital context token   
Server to Client AUTH rc=0x18 Auth Method="GS2-KRB5" Auth Data=reply context token   
Client to Server AUTH rc=0 Auth Method="GS2-KRB5"   
Server to Client CONNACK rc=0 Auth Method="GS2-KRB5" Auth Data=outcome of authentication

## Handling errors

If either the Server or Client encounters a malformed packet or protocol error, it MUST close the Network Connection on which it received that Control Packet. It MAY send a CONNACK or DISCONNECT Packet with a return code of 128 or greater before closing the Network Connection.

A Client or Server implementation might encounter a Transient Error (for example an internal buffer full condition) that prevents successful processing of an MQTT packet. If the Control Packet being processed has an acknowledge, the Client or Server MAY send a Return code of 128 or greater to indicate an error, otherwise the Client or Server which encounters a Transient Error while processing an inbound Control Packet MUST close the Network Connection on which it received that Control Packet. If a Server detects a Transient Error, it SHOULD NOT disconnect or have any other effect on its interactions with any other Client.

# Security

## Introduction

This Chapter is provided for guidance only and is **Non Normative**. However, it is strongly recommended that Server implementations that offer TLS [RFC5246] SHOULD use TCP port 8883 (IANA service name: secure-mqtt).

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

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

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

* Authentication of users and devices
* Authorization of access to Server resources
* Integrity of MQTT Control Packets and application data contained therein
* Privacy of MQTT Control Packets and application data contained therein

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

In addition to technical security issues there could also be geographic (e.g. U.S.-EU SafeHarbor [USEUSAFEHARB]), industry specific (e.g. PCI DSS [PCIDSS]) and regulatory considerations (e.g. Sarbanes-Oxley [SARBANES]).

## MQTT solutions: security and certification

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

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

## Lightweight cryptography and constrained devices

Advanced Encryption Standard [AES] and Data Encryption Standard [DES] are widely adopted.

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

## Implementation notes

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

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

### Authentication of Clients by the Server

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

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

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

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

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

### Authorization of Clients by the Server

If a client has been successfully authenticated, a server implementation should check that it is authorized before accepting its connection.

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

In particular, the implementation should check that the client is authorized to use the Client Identifier as this gives access to the MQTT Session state (described in section 3.1.2.12). This authorization check is to protect against the case where one client, accidentally or maliciously, provides a Client Identifier that is already being used by some other client.

An implementation should provide access controls that take place after CONNECT to restrict the client's ability to publish to particular Topics or to subscribe using particular Topic Filters. In particular, an implementation should consider limiting access to Topic Filters that have broad scope, such as the # Topic Filter.

### Authentication of the Server by the Client

The MQTT protocol is not trust symmetrical: it provides no mechanism for the Client to authenticate the Server.

Where TLS [RFC5246] is used, SSL Certificates sent from the Server can be used by the Client to authenticate the Server. Implementations providing MQTT service for multiple hostnames from a single IP address should be aware of the Server Name Indication extension to TLS defined in section 3 of RFC 6066 [RFC6066]. This allows a Client to tell the Server the hostname of the Server it is trying to connect to.

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

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

### Integrity of Application Messages and Control Packets

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

TLS [RFC5246] provides hash algorithms to verify the integrity of data sent over the network.

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

### Privacy of Application Messages and Control Packets

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

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

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

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

### Non-repudiation of message transmission

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

### Detecting compromise of Clients and Servers

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

Client and Server implementations using TLS [RFC5246] can choose to provide capabilities to check Certificate Revocation Lists (CRLs [RFC5280]) and Online Certificate Status Protocol (OSCP) [RFC6960] to prevent revoked certificates from being used.

Physical deployments might combine tamper-proof hardware with the transmission of specific data in Application Messages. For example, a meter might have an embedded GPS to ensure it is not used in an unauthorized location. [IEEE 802.1AR] is a standard for implementing mechanisms to authenticate a device’s identity using a cryptographically bound identifier.

### Detecting abnormal behaviors

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

* Repeated connection attempts
* Repeated authentication attempts
* Abnormal termination of connections
* Topic scanning (attempts to send or subscribe to many topics)
* Sending undeliverable messages (no subscribers to the topics)
* Clients that connect but do not send data

Server implementations might disconnect Clients that breach its security rules.

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

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

### Other security considerations

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

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

In the case of long lasting connections:

* Client and Server implementations using TLS [RFC5246] should allow for session renegotiation to establish new cryptographic parameters (replace session keys, change cipher suites, change authentication credentials).
* Servers may disconnect Clients and require them to re-authenticate with new credentials.

Constrained devices and Clients on constrained networks can make use of TLS session resumption [RFC5077], in order to reduce the costs of reconnecting TLS [RFC5246] sessions.

Clients connected to a Server have a transitive trust relationship with other Clients connected to the same Server and who have authority to publish data on the same topics.

### Use of SOCKS

Implementations of Clients should be aware that some environments will require the use of SOCKSv5 [RFC1928] proxies to make outbound Network Connections. Some MQTT implementations could make use of alternative secured tunnels (e.g. SSH) through the use of SOCKS. Where implementations choose to use SOCKS, they should support both anonymous and user-name password authenticating SOCKS proxies. In the latter case, implementations should be aware that SOCKS authentication might occur in plain-text and so should avoid using the same credentials for connection to a MQTT Server.

### Security profiles

Implementers and solution designers might wish to consider security as a set of profiles which can be applied to the MQTT protocol. An example of a layered security hierarchy is presented below.

#### Clear communication profile

When using the clear communication profile, the MQTT protocol runs over an open network with no additional secure communication mechanisms in place.

#### Secured network communication profile

When using the secured network communication profile, the MQTT protocol runs over a physical or virtual network which has security controls e.g., VPNs or physically secure network.

#### Secured transport profile

When using the secured transport profile, the MQTT protocol runs over a physical or virtual network and using TLS [RFC5246] which provides authentication, integrity and privacy.

TLS [RFC5246] Client authentication can be used in addition to – or in place of – MQTT Client authentication as provided by the Username and Password fields.

#### Industry specific security profiles

It is anticipated that the MQTT protocol will be designed into industry specific application profiles, each defining a threat model and the specific security mechanisms to be used to address these threats. Recommendations for specific security mechanisms will often be taken from existing works including:

[[NISTCSF] NIST Cyber Security Framework  
[NIST7628] NISTIR 7628 Guidelines for Smart Grid Cyber Security  
[FIPS1402] Security Requirements for Cryptographic Modules (FIPS PUB 140-2)  
[PCIDSS] PCI-DSS Payment Card Industry Data Security Standard  
[NSAB] NSA Suite B Cryptography](#NISTCSF)

# Using WebSocket as a network transport

If MQTT is transported over a WebSocket [[RFC6455]](#RFC6455) connection, the following conditions apply:

* MQTT Control Packets MUST be sent in WebSocket binary data frames. If any other type of data frame is received the recipient MUST close the Network Connection.
* 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.
* The client MUST include “mqtt” in the list of WebSocket Sub Protocols it offers.
* The WebSocket Sub Protocol name selected and returned by the server MUST be “mqtt”.
* The WebSocket URI used to connect the client and server has no impact on the MQTT protocol.

## IANA Considerations

This specification requests IANA to register the WebSocket MQTT sub-protocol under the “WebSocket Subprotocol Name” registry with the following data:

##### Figure 6.6.1 - IANA WebSocket Identifier

|  |  |
| --- | --- |
| Subprotocol Identifier | mqtt |
| Subprotocol Common Name | mqtt |
| Subprotocol Definition | http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html |

# Conformance

The MQTT specification defines conformance for MQTT Client implementations and MQTT Server implementations.

An MQTT implementation MAY conform as both an MQTT Client and MQTT Server implementation. A Server that both accepts inbound connections and establishes outbound connections to other Servers MUST conform as both an MQTT Client and MQTT Server.

Conformant implementations MUST NOT require the use of any extensions defined outside of this specification in order to interoperate with any other conformant implementation.

## Conformance Targets

### MQTT Server

An MQTT Server conforms to this specification only if it satisfies all the statements below:

1. The format of all 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.

3. It satisfies all of 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 - (if MQTT is transported over a WebSocket connection)

- Chapter 7 - Conformance Targets

A conformant 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. However, conformance does not depend on it supporting any specific transport protocols. A Server MAY support any of the transport protocols listed in Section 4.2, or any other transport protocol that meets the requirements of.

### MQTT Client

An MQTT Client conforms to this specification only if it satisfies all the statements below:

1. The format of all Control Packets that the Client sends matches the format described in Chapter 2 and Chapter 3.

2. It satisfies all of 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 - (if MQTT is transported over a WebSocket connection)

- Chapter 7 - Conformance Targets

A conformant Client 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. However, conformance does not depend on it supporting any specific transport protocols. A Client MAY support any of the transport protocols listed in Section 4.2, or any other transport protocol that meets the requirements of.

1. Acknowledgments

The TC owes special thanks to Dr. Andy Stanford-Clark and Arlen Nipper as the original inventors of the MQTT protocol and for their continued support with the standardization process.

The following individuals were members of the OASIS Technical Committee during the creation of this specification and their contributions are gratefully acknowledged:

Participants:

[Participant Name, Affiliation | Individual Member]

[Participant Name, Affiliation | Individual Member]

1. Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Editor** | **Changes Made** |
| [1] | [18th July 2016] | [Andrew Banks] | * [MQTT-249] Add expiry capabilities to MQTT * [MQTT-256] Message Format indication and message metadata in general.TC accepted proposal * [MQTT-269] MQTT-SN Feature: Topic Registration * [MQTT-270] SN Feature: server initiated disconnects * Rename Remaining length datatype to Variable Byte Integer * Introduce two and four-byte integer data types |
| [2] | [10th August 2016] | [Andrew Banks]  [Rahul Gupta] | * [MQTT-249] Add expiry capabilities to MQTT. * [MQTT-263] Simplified State Management. TC accepted proposals. * [MQTT-289] Update the working draft to the new template for MQTT v5 from OASIS |
| [3] | [25th August 2016] | [Rahul Gupta]  [Ken Borgendale] | * [MQTT-236] Consolidate acknowledgements, enable negative acknowledgements * [MQTT-270] Server initiated disconnects * [MQTT-294] Incorrect version number in section 3.1.2.2 Protocol Level |
| [4] | [6th September 2016] | [Andrew Banks] | * [MQTT-257 Flow Control] |
| [5] | [22nd September 2016] | [Andrew Banks] | * [MQTT-249 Session Expiry] * [MQTT-302 WD4: Minor suggestions in sections 2.3.3.X] |
| [6] | [23nd September 2016] | [Andrew Banks] | * Accept all changes, remove markup. |
| [7] | [26th September 2016] | [Ed Briggs] | * [MQTT-295] Modified 4.4 to prohibit retransmission during a transport connection * [MQTT-257] Flow Control algorithm added |
| [7] | [28th September 2016] | [Andrew Banks] | * [MQTT-251 Return server assigned client id to client] * [MQTT-303 Missing reference to Receive Maximum in Appendix B] * [MQTT-290 Session Expiry Will message.] * [MQTT-269 MQTT-SN Feature: Topic Registration] |
| [7] | [3rd October 2016] | [Ed Briggs] | * [MQTT-236] Added CONNACK Banned Error Code * Added QoS Not Supported to PUBACK and PUBREC. * Added Invalid Topic to CONNACK to signify invalid Will Topic * Changed ‘Message Too Long” to “Packet too long based on TC agreement to use packet size, not payload size. |
| [7] | [3 October 2016] | Ken Borgendale | * [MQTT=197] Request / response (mechanism, section 4.9 not complete) * [MQTT-235] NoLocal * [MQTT-278] Server Keep Alive * [MQTT-284] Enhanced problem determination |
| [7] | [4 October 2016] | [Ed Briggs] | * [MQTT-301] Added Identifier definition for Retain Unavailable Advertisement * [MQTT-300] Added Identifier definition for Maximum QoS * [MQTT-296] Added sentence requiring minimum size encoded value for variable length integer in section 1.5.5. * [MQTT-287] Added text for single unified packet identifier space |
| [7] | [5 October 2016] | [Ken Borgendale] | * [MQTT-234] Shared Subscriptions * [MQTT-293] Recommendations for securing an MQTT server |
| [7] | [6 October 2016] | [Ed Briggs] | * [MQTT-304] User Defined CONNECT Tags * [MQTT-305] User Defined PUBLISH Tags * Defined new UTF-8 String Pair Data Type * Added Identifier 38 (0x26) for User Defined Name-Value Pair |
| [8] | [7 October 2016] | [Andrew Banks] | * [MQTT-310] Treat invalid topic alias as a protocol error |
| [8] | [8 October 2016] | [Ken Borgendale] | * Fix section numbering and TOC issues, along with other formatting issues. |
| [8] | [18 October 2016] | [Ken Borgendale} | * [MQTT-260] Try another server * [MQTT-255] Alternate authentication * [MQTT-285] Subscribe options * [MQTT-309] Enhanced Problem determination for ACKS * Editorial changes |
| [8] | [18 October 2016] | [Ed Briggs] | * [MQTT-314] Simplified String Pair Type |