[](https://www.oasis-open.org/)

MQTT 协议 5.0中文版

委员会规范草案 02 /  
公开评审草案 02

2017年10月26日

规范链接

当前版本：

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

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

<http://docs.oasis-open.org/mqtt/mqtt/v5.0/csprd02/mqtt-v5.0-csprd02.pdf>

以前的版本：

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

<http://docs.oasis-open.org/mqtt/mqtt/v5.0/csprd01/mqtt-v5.0-csprd01.html>

<http://docs.oasis-open.org/mqtt/mqtt/v5.0/csprd01/mqtt-v5.0-csprd01.pdf>

最新版本：

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

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

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

技术委员会：

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

主席：

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)

编辑：

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](http://www.microsoft.com)

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)

相关文档：

本规范代替：

* MQTT协议3.1.1版本。编辑是Andrew Banks和Rahul Gupta，发布于2014年10月29日，OASIS标准：<http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>.

本规范与此有关：

* MQTT 和 NIST 网络安全框架 1.0 版。 编辑是Geoff Brown和Louis-Philippe Lamoureux。最新版本：<http://docs.oasis-open.org/mqtt/mqtt-nist-cybersecurity/v1.0/mqtt-nist-cybersecurity-v1.0.html>.

摘要：

MQTT 是一个客户端服务端架构的发布/订阅模式的消息传输协议。它的设计思想是轻巧、开放、简单、规范，因此易于实现。这些特点使得它对很多场景来说都是很好的选择，包括受限的环境如机器与机器的通信（M2M）以及物联网环境（IoT），这些场景要求很小的代码封装或者网络带宽非常昂贵。

本协议运行在TCP/IP，或其他提供了有序、可靠、双向连接的网络连接上。它有以下特点：

* 使用发布/订阅消息模式，提供了一对多的消息分发和应用之间的解耦。
* 消息传输不需要知道负载内容。
* 提供三种等级的服务质量：
* “最多一次”，尽操作环境所能提供的最大努力分发消息。消息可能会丢失。例如，这个等级可用于环境传感器数据，单次的数据丢失没关系，因为不久之后会再次发送。
* “至少一次”，保证消息可以到达，但是可能会重复。
* “仅一次”，保证消息只到达一次。例如，这个等级可用在一个计费系统中，这里如果消息重复或丢失会导致不正确的收费。
* 很小的传输消耗和协议数据交换，最大限度减少网络流量。
* 异常连接断开发生时，能通知到相关各方。

状态：

本文档最后由OASIS成员在上面标示的日期最终修订或批准。批准的级别也在上面列出了。如果要查看本文档最新的修订版请检查上面的 *最新版本* 位置。技术委员会产生的其他修订版和其他技术文档都列在这里：<https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt#technical>.

技术委员会成员对本规范的评论应该发送到技术委员会的邮件列表。其他人应该发送评论到技术委员会的公共评论列表，方法是点击技术委员会网站的[发送评论](https://www.oasis-open.org/committees/comments/index.php?wg_abbrev=mqtt)按钮，网页地址是<https://www.oasis-open.org/committees/mqtt/>.

本规范草案的发布基于[OASIS 知识产权政策](https://www.oasis-open.org/policies-guidelines/ipr)的[Non-Assertion](https://www.oasis-open.org/policies-guidelines/ipr#Non-Assertion-Mode)模式。关于实现本规范必不可少的任何专利是否已公开，以及其他的专利许可条款相关的信息，请参考技术委员会网站的知识产权部分 (<https://www.oasis-open.org/committees/mqtt/ipr.php>).

请注意，为本产品声明的任何机器可读内容（[计算机语言定义](https://www.oasis-open.org/policies-guidelines/tc-process#wpComponentsCompLang)）单独提供纯文本文件。对于产品叙述文档中出现的任何不一致，以纯文本文档为准。

引用格式：

引用此规范时应该使用下面的引文格式：

[mqtt-v5.0]

*MQTT Version 5.0*. Edited by Andrew Banks, Ed Briggs, Ken Borgendale, and Rahul Gupta. 26 October 2017. OASIS Committee Specification Draft 02 / Public Review Draft 02. <http://docs.oasis-open.org/mqtt/mqtt/v5.0/csprd02/mqtt-v5.0-csprd02.html>. Latest version: <http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>.

注意

Copyright © OASIS Open 2017. All Rights Reserved.

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

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

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

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

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

目录

[1 概述 11](#_Toc513542087)

[1.0 知识产权政策 11](#_Toc513542088)

[1.1 MQTT协议的组织结构 11](#_Toc513542089)

[1.2 术语 11](#_Toc513542090)

[1.3 规范引用 13](#_Toc513542091)

[1.4 非规范引用 13](#_Toc513542092)

[1.5 数据表示 16](#_Toc513542093)

[1.5.1 二进制位 16](#_Toc513542094)

[1.5.2 双字节整数 16](#_Toc513542095)

[1.5.3 四字节整数 16](#_Toc513542096)

[1.5.4 UTF-8 编码字符串 16](#_Toc513542097)

[1.5.5 变长字节整数 18](#_Toc513542098)

[1.5.6 二进制数据 19](#_Toc513542099)

[1.5.7 UTF-8字符串对 19](#_Toc513542100)

[1.6 安全 19](#_Toc513542101)

[1.7 编辑约定 19](#_Toc513542102)

[1.8 变更历史 19](#_Toc513542103)

[1.8.1 MQTT v3.1.1 19](#_Toc513542104)

[1.8.2 MQTT v5.0 19](#_Toc513542105)

[2 MQTT控制报文格式 21](#_Toc513542106)

[2.1 MQTT控制报文结构 21](#_Toc513542107)

[2.1.1 固定报头 21](#_Toc513542108)

[2.1.2 MQTT控制报文的类型 21](#_Toc513542109)

[2.1.3 标志 22](#_Toc513542110)

[2.1.4 剩余长度 23](#_Toc513542111)

[2.2 可变报头 23](#_Toc513542112)

[2.2.1 报文标识符 23](#_Toc513542113)

[2.2.2 内容 25](#_Toc513542114)

[2.2.2.1 Property Length 25](#_Toc513542115)

[2.2.2.2 Property 25](#_Toc513542116)

[2.3 Payload 26](#_Toc513542117)

[2.4 Reason Code 27](#_Toc513542118)

[3 MQTT Control Packets 30](#_Toc513542119)

[3.1 CONNECT – Connection Request 30](#_Toc513542120)

[3.1.1 CONNECT Fixed Header 30](#_Toc513542121)

[3.1.2 CONNECT Variable Header 30](#_Toc513542122)

[3.1.2.1 Protocol Name 30](#_Toc513542123)

[3.1.2.2 Protocol Version 31](#_Toc513542124)

[3.1.2.3 Connect Flags 31](#_Toc513542125)

[3.1.2.4 Clean Start 32](#_Toc513542126)

[3.1.2.5 Will Flag 32](#_Toc513542127)

[3.1.2.6 Will QoS 33](#_Toc513542128)

[3.1.2.7 Will Retain 33](#_Toc513542129)

[3.1.2.8 User Name Flag 33](#_Toc513542130)

[3.1.2.9 Password Flag 33](#_Toc513542131)

[3.1.2.10 Keep Alive 34](#_Toc513542132)

[3.1.2.11 CONNECT Properties 34](#_Toc513542133)

[3.1.2.11.1 Property Length 34](#_Toc513542134)

[3.1.2.11.2 Session Expiry Interval 34](#_Toc513542135)

[3.1.2.11.3 Receive Maximum 36](#_Toc513542136)

[3.1.2.11.4 Maximum Packet Size 36](#_Toc513542137)

[3.1.2.11.5 Topic Alias Maximum 37](#_Toc513542138)

[3.1.2.11.6 Request Response Information 37](#_Toc513542139)

[3.1.2.11.7 Request Problem Information 37](#_Toc513542140)

[3.1.2.11.8 User Property 38](#_Toc513542141)

[3.1.2.11.9 Authentication Method 38](#_Toc513542142)

[3.1.2.11.10 Authentication Data 38](#_Toc513542143)

[3.1.2.12 Variable Header non-normative example 38](#_Toc513542144)

[3.1.3 CONNECT Payload 39](#_Toc513542145)

[3.1.3.1 Client Identifier (ClientID) 39](#_Toc513542146)

[3.1.3.2 Will Properties 40](#_Toc513542147)

[3.1.3.2.1 Property Length 40](#_Toc513542148)

[3.1.3.2.2 Will Delay Interval 40](#_Toc513542149)

[3.1.3.2.3 Payload Format Indicator 41](#_Toc513542150)

[3.1.3.2.4 Message Expiry Interval 41](#_Toc513542151)

[3.1.3.2.5 Content Type 41](#_Toc513542152)

[3.1.3.2.6 Response Topic 41](#_Toc513542153)

[3.1.3.2.7 Correlation Data 42](#_Toc513542154)

[3.1.3.2.8 User Property 42](#_Toc513542155)

[3.1.3.3 Will Topic 42](#_Toc513542156)

[3.1.3.4 Will Payload 42](#_Toc513542157)

[3.1.3.5 User Name 42](#_Toc513542158)

[3.1.3.6 Password 43](#_Toc513542159)

[3.1.4 CONNECT Actions 43](#_Toc513542160)

[3.2 CONNACK – Connect acknowledgement 44](#_Toc513542161)

[3.2.1 CONNACK Fixed Header 44](#_Toc513542162)

[3.2.2 CONNACK Variable Header 45](#_Toc513542163)

[3.2.2.1 Connect Acknowledge Flags 45](#_Toc513542164)

[3.2.2.1.1 Session Present 45](#_Toc513542165)

[3.2.2.2 Connect Reason Code 45](#_Toc513542166)

[3.2.2.3 CONNACK Properties 47](#_Toc513542167)

[3.2.2.3.1 Property Length 47](#_Toc513542168)

[3.2.2.3.2 Session Expiry Interval 47](#_Toc513542169)

[3.2.2.3.3 Receive Maximum 47](#_Toc513542170)

[3.2.2.3.4 Maximum QoS 47](#_Toc513542171)

[3.2.2.3.5 Retain Available 48](#_Toc513542172)

[3.2.2.3.6 Maximum Packet Size 48](#_Toc513542173)

[3.2.2.3.7 Assigned Client Identifier 48](#_Toc513542174)

[3.2.2.3.8 Topic Alias Maximum 49](#_Toc513542175)

[3.2.2.3.9 Reason String 49](#_Toc513542176)

[3.2.2.3.10 User Property 49](#_Toc513542177)

[3.2.2.3.11 Wildcard Subscription Available 50](#_Toc513542178)

[3.2.2.3.12 Subscription Identifiers Available 50](#_Toc513542179)

[3.2.2.3.13 Shared Subscription Available 50](#_Toc513542180)

[3.2.2.3.14 Server Keep Alive 50](#_Toc513542181)

[3.2.2.3.15 Response Information 51](#_Toc513542182)

[3.2.2.3.16 Server Reference 51](#_Toc513542183)

[3.2.2.3.17 Authentication Method 51](#_Toc513542184)

[3.2.2.3.18 Authentication Data 51](#_Toc513542185)

[3.2.3 CONNACK Payload 52](#_Toc513542186)

[3.3 PUBLISH – Publish message 52](#_Toc513542187)

[3.3.1 PUBLISH Fixed Header 52](#_Toc513542188)

[3.3.1.1 DUP 52](#_Toc513542189)

[3.3.1.2 QoS 53](#_Toc513542190)

[3.3.1.3 RETAIN 53](#_Toc513542191)

[3.3.1.4 Remaining Length 54](#_Toc513542192)

[3.3.2 PUBLISH Variable Header 54](#_Toc513542193)

[3.3.2.1 Topic Name 54](#_Toc513542194)

[3.3.2.2 Packet Identifier 55](#_Toc513542195)

[3.3.2.3 PUBLISH Properties 55](#_Toc513542196)

[3.3.2.3.1 Property Length 55](#_Toc513542197)

[3.3.2.3.2 Payload Format Indicator 55](#_Toc513542198)

[3.3.2.3.3 Message Expiry Interval` 55](#_Toc513542199)

[3.3.2.3.4 Topic Alias 56](#_Toc513542200)

[3.3.2.3.5 Response Topic 57](#_Toc513542201)

[3.3.2.3.6 Correlation Data 57](#_Toc513542202)

[3.3.2.3.7 User Property 57](#_Toc513542203)

[3.3.2.3.8 Subscription Identifier 58](#_Toc513542204)

[3.3.2.3.9 Content Type 58](#_Toc513542205)

[3.3.3 PUBLISH Payload 59](#_Toc513542206)

[3.3.4 PUBLISH Actions 59](#_Toc513542207)

[3.4 PUBACK – Publish acknowledgement 61](#_Toc513542208)

[3.4.1 PUBACK Fixed Header 61](#_Toc513542209)

[3.4.2 PUBACK Variable Header 62](#_Toc513542210)

[3.4.2.1 PUBACK Reason Code 62](#_Toc513542211)

[3.4.2.2 PUBACK Properties 63](#_Toc513542212)

[3.4.2.2.1 Property Length 63](#_Toc513542213)

[3.4.2.2.2 Reason String 63](#_Toc513542214)

[3.4.2.2.3 User Property 63](#_Toc513542215)

[3.4.3 PUBACK Payload 63](#_Toc513542216)

[3.4.4 PUBACK Actions 63](#_Toc513542217)

[3.5 PUBREC – Publish received (QoS 2 delivery part 1) 63](#_Toc513542218)

[3.5.1 PUBREC Fixed Header 64](#_Toc513542219)

[3.5.2 PUBREC Variable Header 64](#_Toc513542220)

[3.5.2.1 PUBREC Reason Code 64](#_Toc513542221)

[3.5.2.2 PUBREC Properties 65](#_Toc513542222)

[3.5.2.2.1 Property Length 65](#_Toc513542223)

[3.5.2.2.2 Reason String 65](#_Toc513542224)

[3.5.2.2.3 User Property 65](#_Toc513542225)

[3.5.3 PUBREC Payload 65](#_Toc513542226)

[3.5.4 PUBREC Actions 66](#_Toc513542227)

[3.6 PUBREL – Publish release (QoS 2 delivery part 2) 66](#_Toc513542228)

[3.6.1 PUBREL Fixed Header 66](#_Toc513542229)

[3.6.2 PUBREL Variable Header 66](#_Toc513542230)

[3.6.2.1 PUBREL Reason Code 66](#_Toc513542231)

[3.6.2.2 PUBREL Properties 67](#_Toc513542232)

[3.6.2.2.1 Property Length 67](#_Toc513542233)

[3.6.2.2.2 Reason String 67](#_Toc513542234)

[3.6.2.2.3 User Property 67](#_Toc513542235)

[3.6.3 PUBREL Payload 67](#_Toc513542236)

[3.6.4 PUBREL Actions 67](#_Toc513542237)

[3.7 PUBCOMP – Publish complete (QoS 2 delivery part 3) 68](#_Toc513542238)

[3.7.1 PUBCOMP Fixed Header 68](#_Toc513542239)

[3.7.2 PUBCOMP Variable Header 68](#_Toc513542240)

[3.7.2.1 PUBCOMP Reason Code 68](#_Toc513542241)

[3.7.2.2 PUBCOMP Properties 69](#_Toc513542242)

[3.7.2.2.1 Property Length 69](#_Toc513542243)

[3.7.2.2.2 Reason String 69](#_Toc513542244)

[3.7.2.2.3 User Property 69](#_Toc513542245)

[3.7.3 PUBCOMP Payload 69](#_Toc513542246)

[3.7.4 PUBCOMP Actions 69](#_Toc513542247)

[3.8 SUBSCRIBE - Subscribe request 69](#_Toc513542248)

[3.8.1 SUBSCRIBE Fixed Header 70](#_Toc513542249)

[3.8.2 SUBSCRIBE Variable Header 70](#_Toc513542250)

[3.8.2.1 SUBSCRIBE Properties 70](#_Toc513542251)

[3.8.2.1.1 Property Length 70](#_Toc513542252)

[3.8.2.1.2 Subscription Identifier 70](#_Toc513542253)

[3.8.2.1.3 User Property 71](#_Toc513542254)

[3.8.3 SUBSCRIBE Payload 71](#_Toc513542255)

[3.8.3.1 Subscription Options 71](#_Toc513542256)

[3.8.4 SUBSCRIBE Actions 73](#_Toc513542257)

[3.9 SUBACK – Subscribe acknowledgement 75](#_Toc513542258)

[3.9.1 SUBACK Fixed Header 75](#_Toc513542259)

[3.9.2 SUBACK Variable Header 76](#_Toc513542260)

[3.9.2.1 SUBACK Properties 76](#_Toc513542261)

[3.9.2.1.1 Property Length 76](#_Toc513542262)

[3.9.2.1.2 Reason String 76](#_Toc513542263)

[3.9.2.1.3 User Property 76](#_Toc513542264)

[3.9.3 SUBACK Payload 77](#_Toc513542265)

[3.10 UNSUBSCRIBE – Unsubscribe request 77](#_Toc513542266)

[3.10.1 UNSUBSCRIBE Fixed Header 78](#_Toc513542267)

[3.10.2 UNSUBSCRIBE Variable Header 78](#_Toc513542268)

[3.10.2.1 UNSUBSCRIBE Properties 78](#_Toc513542269)

[3.10.2.1.1 Property Length 78](#_Toc513542270)

[3.10.2.1.2 User Property 78](#_Toc513542271)

[3.10.3 UNSUBSCRIBE Payload 78](#_Toc513542272)

[3.10.4 UNSUBSCRIBE Actions 79](#_Toc513542273)

[3.11 UNSUBACK – Unsubscribe acknowledgement 80](#_Toc513542274)

[3.11.1 UNSUBACK Fixed Header 80](#_Toc513542275)

[3.11.2 UNSUBACK Variable Header 80](#_Toc513542276)

[3.11.2.1 UNSUBACK Properties 80](#_Toc513542277)

[3.11.2.1.1 Property Length 80](#_Toc513542278)

[3.11.2.1.2 Reason String 81](#_Toc513542279)

[3.11.2.1.3 User Property 81](#_Toc513542280)

[3.11.3 UNSUBACK Payload 81](#_Toc513542281)

[3.12 PINGREQ – PING request 82](#_Toc513542282)

[3.12.1 PINGREQ Fixed Header 82](#_Toc513542283)

[3.12.2 PINGREQ Variable Header 82](#_Toc513542284)

[3.12.3 PINGREQ Payload 82](#_Toc513542285)

[3.12.4 PINGREQ Actions 82](#_Toc513542286)

[3.13 PINGRESP – PING response 82](#_Toc513542287)

[3.13.1 PINGRESP Fixed Header 83](#_Toc513542288)

[3.13.2 PINGRESP Variable Header 83](#_Toc513542289)

[3.13.3 PINGRESP Payload 83](#_Toc513542290)

[3.13.4 PINGRESP Actions 83](#_Toc513542291)

[3.14 DISCONNECT – Disconnect notification 83](#_Toc513542292)

[3.14.1 DISCONNECT Fixed Header 83](#_Toc513542293)

[3.14.2 DISCONNECT Variable Header 84](#_Toc513542294)

[3.14.2.1 Disconnect Reason Code 84](#_Toc513542295)

[3.14.2.2 DISCONNECT Properties 86](#_Toc513542296)

[3.14.2.2.1 Property Length 86](#_Toc513542297)

[3.14.2.2.2 Session Expiry Interval 86](#_Toc513542298)

[3.14.2.2.3 Reason String 86](#_Toc513542299)

[3.14.2.2.4 User Property 86](#_Toc513542300)

[3.14.2.2.5 Server Reference 87](#_Toc513542301)

[3.14.3 DISCONNECT Payload 87](#_Toc513542302)

[3.14.4 DISCONNECT Actions 87](#_Toc513542303)

[3.15 AUTH – Authentication exchange 88](#_Toc513542304)

[3.15.1 AUTH Fixed Header 88](#_Toc513542305)

[3.15.2 AUTH Variable Header 88](#_Toc513542306)

[3.15.2.1 Authenticate Reason Code 88](#_Toc513542307)

[3.15.2.2 AUTH Properties 89](#_Toc513542308)

[3.15.2.2.1 Property Length 89](#_Toc513542309)

[3.15.2.2.2 Authentication Method 89](#_Toc513542310)

[3.15.2.2.3 Authentication Data 89](#_Toc513542311)

[3.15.2.2.4 Reason String 89](#_Toc513542312)

[3.15.2.2.5 User Property 89](#_Toc513542313)

[3.15.3 AUTH Payload 89](#_Toc513542314)

[3.15.4 AUTH Actions 89](#_Toc513542315)

[4 Operational behavior 90](#_Toc513542316)

[4.1 Session State 90](#_Toc513542317)

[4.1.1 Storing Session State 90](#_Toc513542318)

[4.1.2 Session State non-normative examples 91](#_Toc513542319)

[4.2 Network Connections 91](#_Toc513542320)

[4.3 Quality of Service levels and protocol flows 91](#_Toc513542321)

[4.3.1 QoS 0: At most once delivery 91](#_Toc513542322)

[4.3.2 QoS 1: At least once delivery 92](#_Toc513542323)

[4.3.3 QoS 2: Exactly once delivery 93](#_Toc513542324)

[4.4 Message delivery retry 94](#_Toc513542325)

[4.5 Message receipt 95](#_Toc513542326)

[4.6 Message ordering 95](#_Toc513542327)

[4.7 Topic Names and Topic Filters 96](#_Toc513542328)

[4.7.1 Topic wildcards 96](#_Toc513542329)

[4.7.1.1 Topic level separator 96](#_Toc513542330)

[4.7.1.2 Multi-level wildcard 96](#_Toc513542331)

[4.7.1.3 Single-level wildcard 97](#_Toc513542332)

[4.7.2 Topics beginning with $ 97](#_Toc513542333)

[4.7.3 Topic semantic and usage 97](#_Toc513542334)

[4.8 Subscriptions 98](#_Toc513542335)

[4.8.1 Non‑shared Subscriptions 98](#_Toc513542336)

[4.8.2 Shared Subscriptions 99](#_Toc513542337)

[4.9 Flow Control 101](#_Toc513542338)

[4.10 Request / Response 101](#_Toc513542339)

[4.10.1 Basic Request Response (non-normative) 102](#_Toc513542340)

[4.10.2 Determining a Response Topic value (non-normative) 103](#_Toc513542341)

[4.11 Server redirection 103](#_Toc513542342)

[4.12 Enhanced authentication 104](#_Toc513542343)

[4.12.1 Re-authentication 105](#_Toc513542344)

[4.13 Handling errors 106](#_Toc513542345)

[4.13.1 Malformed Packet and Protocol Errors 106](#_Toc513542346)

[4.13.2 Other errors 107](#_Toc513542347)

[5 Security (non-normative) 108](#_Toc513542348)

[5.1 Introduction 108](#_Toc513542349)

[5.2 MQTT solutions: security and certification 108](#_Toc513542350)

[5.3 Lightweight crytography and constrained devices 109](#_Toc513542351)

[5.4 Implementation notes 109](#_Toc513542352)

[5.4.1 Authentication of Clients by the Server 109](#_Toc513542353)

[5.4.2 Authorization of Clients by the Server 109](#_Toc513542354)

[5.4.3 Authentication of the Server by the Client 110](#_Toc513542355)

[5.4.4 Integrity of Application Messages and MQTT Control Packets 110](#_Toc513542356)

[5.4.5 Privacy of Application Messages and MQTT Control Packets 110](#_Toc513542357)

[5.4.6 Non-repudiation of message transmission 111](#_Toc513542358)

[5.4.7 Detecting compromise of Clients and Servers 111](#_Toc513542359)

[5.4.8 Detecting abnormal behaviors 111](#_Toc513542360)

[5.4.9 Other security considerations 111](#_Toc513542361)

[5.4.10 Use of SOCKS 112](#_Toc513542362)

[5.4.11 Security profiles 112](#_Toc513542363)

[5.4.11.1 Clear communication profile 112](#_Toc513542364)

[5.4.11.2 Secured network communication profile 112](#_Toc513542365)

[5.4.11.3 Secured transport profile 112](#_Toc513542366)

[5.4.11.4 Industry specific security profiles 113](#_Toc513542367)

[6 Using WebSocket as a network transport 114](#_Toc513542368)

[6.1 IANA considerations 114](#_Toc513542369)

[7 Conformance 115](#_Toc513542370)

[7.1 Conformance clauses 115](#_Toc513542371)

[7.1.1 MQTT Server conformance clause 115](#_Toc513542372)

[7.1.2 MQTT Client conformance clause 115](#_Toc513542373)

[Appendix A. Acknowledgments 116](#_Toc513542374)

[Appendix B. Mandatory normative statement (non-normative) 117](#_Toc513542375)

[Appendix C. Summary of new features in MQTT v5.0 (non-normative) 132](#_Toc513542376)

# 概述

## 知识产权政策

此公开评审草案的发布基于[OASIS IPR Policy](https://www.oasis-open.org/policies-guidelines/ipr)的[Non-Assertion](https://www.oasis-open.org/policies-guidelines/ipr#Non-Assertion-Mode)模式。

关于实现本规范必不可少的任何专利是否已公开，以及其他的专利许可条款相关的信息，请参考技术委员会网站的知识产权部分(<https://www.oasis-open.org/committees/mqtt/ipr.php>).

## MQTT协议的组织结构

本规范分为七个章节：

* [第一章 - 介绍](#_Introduction)

* [第二章 - MQTT控制报文格式](#_MQTT_Control_Packet)

* [第三章 - MQTT控制报文](#_MQTT_Control_Packets)

* [第四章 - 操作行为](#_Operational_behavior)
* [第五章 - 安全](#_Security)
* [第六章 - 使用Websocket作为网络传输层](#_Using_WebSocket_as)
* [第七章 - 一致性目标](#_Conformance)

## 术语

本规范中用到的关键字 **必须** MUST，**不能** MUST NOT，**要求** REQUIRED，**将会** SHALL，**不会** SHALL NOT，**应该** SHOULD，**不应该** SHOULD NOT，**推荐** RECOMMENDED，**可以** MAY，**可选** OPTIONAL都是按照IETF RFC 2119 [[RFC2119]](#RFC2119)中的描述解释。

网络连接（Network Connection）：

MQTT使用的底层传输协议基础设施。

* 客户端使用它连接服务端。
* 它提供有序的、可靠的、双向字节流传输。

例子见[4.2](#_Network_Connections)节。

应用消息（Application Message）：

MQTT协议通过网络传输应用数据。应用消息通过MQTT传输时，它们有关联的服务质量（QoS）和主题（Topic）。

**客户端（Client）：**

使用MQTT的程序或设备。客户端总是通过网络连接到服务端。它可以：

* 打开连接到服务端的网络连接
* 发布应用消息给其他相关的客户端
* 订阅以请求接受相关的应用消息
* 取消订阅以移除接受相应消息的请求
* 关闭连接到服务端的网络连接

**服务端（Server）：**

一个程序或设备，作为发送消息的客户端和请求订阅的客户端之间的中介。服务端：

* 接受来自客户端的网络连接
* 接受客户端发布的应用消息
* 处理客户端的订阅和取消订阅请求
* 转发应用消息给符合条件的客户端订阅
* 关闭来自客户端的网络连接

**会话（Session）：**

客户端和服务端之间的状态交互。一些会话持续时长与网络连接一样，另一些可以在客户端和服务端的多个连续网络连接间扩展。

**订阅（Subscription）：**

订阅包含一个主题过滤器（Topic Filter）和一个最大的服务质量（QoS）等级。订阅与单个会话（Session）关联。会话可以包含多于一个的订阅。会话的每个订阅都有一个不同的主题过滤器。

**共享订阅（Shared Subscription）：**

一个共享订阅包含一个主题过滤器（Topic Filter）和一个最大的服务质量（QoS）等级。一个共享订阅可以与多个订阅会话相关联，便于支持大范围消息交换模式。一条主题匹配的应用消息只发送给关联到此共享订阅的多个会话中的一个会话。一个会话可以包括多个共享订阅，可以同时包含共享订阅与非共享订阅。

**通配符订阅（Wildcard Subscription）：**

通配符订阅是指主题过滤器（Topic Filter）包含一个或多个通配符的订阅。通配符订阅使得一次订阅匹配多个主题名（Topic Name）。[4.7](#_Topic_Names_and)节描述了主题过滤器中的通配符。

**主题名（Topic Name）：**

附加在应用消息上的一个标签，服务端已知且与订阅匹配。服务端发送应用消息的一个副本给每一个匹配的客户端订阅。

**主题过滤器（Topic Filter）：**

订阅中包含的一个表达式，用于表示相关的一个或多个主题。主题过滤器可以使用通配符。

**控制报文（MQTT Control Packet）：**

通过网络连接发送的信息数据包。MQTT 规范定义了十四种不同类型的控制报文，其中一个（PUBLISH 报文）用于传输应用消息。

**错误报文：**

根据规范不能被正确解析的控制报文。[4.13](#_Handling_errors)节描述了如何进行相应的错误处理。

**协议错误（Protocol Error）：**

在报文解析之后发现包含协议不允许或与客户端或服务端当前状态不一致的数据的错误。[4.13](#S4_13_Errors)节描述了如何进行相应的错误处理。

**遗嘱消息（Will Message）：**

在网络连接非正常关闭的情况下，由服务端发布的应用消息。[3.1.2.5](#_Toc479576982)节描述了遗嘱消息。

## 规范引用

**[RFC2119]**

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997,

<http://www.rfc-editor.org/info/rfc2119>

**[RFC3629]**

Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003,

<http://www.rfc-editor.org/info/rfc3629>

**[RFC6455]**

Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011,

<http://www.rfc-editor.org/info/rfc6455>

**[Unicode]**

The Unicode Consortium. The Unicode Standard,

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

## 非规范引用

**[RFC0793]**

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

**[RFC5246]**

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

<http://www.rfc-editor.org/info/rfc5246>

**[AES]**

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

<https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf>

**[CHACHA20]**

ChaCha20 and Poly1305 for IETF Protocols

<https://tools.ietf.org/html/rfc7539>

**[FIPS1402]**

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

<https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/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

<https://www.iso.org/standard/56425.html>

**[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>

**[MQTTV311]**

MQTT V3.1.1 Protocol Specification

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

**[ISO20922]**

MQTT V3.1.1 ISO Standard (ISO/IEC 20922:2016)

<https://www.iso.org/standard/69466.html>

**[NISTCSF]**

Improving Critical Infrastructure Cybersecurity Executive Order 13636

<https://www.nist.gov/sites/default/files/documents/itl/preliminary-cybersecurity-framework.pdf>

**[****NIST7628]**

NISTIR 7628 Guidelines for Smart Grid Cyber Security Catalogue

<https://www.nist.gov/sites/default/files/documents/smartgrid/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/pci_security/>

**[RFC1928]**

Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, DOI 10.17487/RFC1928, March 1996,

<http://www.rfc-editor.org/info/rfc1928>

**[RFC4511]**

Sermersheim, J., Ed., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, DOI 10.17487/RFC4511, June 2006,

<http://www.rfc-editor.org/info/rfc4511>

**[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, DOI 10.17487/RFC5280, May 2008,

<http://www.rfc-editor.org/info/rfc5280>

**[RFC6066]**

Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011,

<http://www.rfc-editor.org/info/rfc6066>

**[RFC6749]**

Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012,

<http://www.rfc-editor.org/info/rfc6749>

**[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, DOI 10.17487/RFC6960, June 2013,

<http://www.rfc-editor.org/info/rfc6960>

**[SARBANES]**

Sarbanes-Oxley Act of 2002.

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

**[USEUPRIVSH]**

U.S.-EU Privacy Shield Framework

<https://www.privacyshield.gov>

**[RFC39****86]**

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

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

**[RFC1035]**

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

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

**[RFC2782]**

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

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

## 数据表示

### 二进制位

字节中的位从0到7。第7位是最高有效位，第0位是最低有效位。

### 双字节整数

双字节整数是 16 位，使用大端序（big-endian，高位字节在低位字节前面）。这意味着一个 16 位的字在网络上表示为最高有效字节（MSB），后面跟着最低有效字节（LSB）。

### 四字节整数

四字节整数是32位，使用大端序（big-endian，高位字节在低位字节前面）。这意味着一个32位的字在网络上表示为第一个最高有效字节（MSB）后面跟着第一个最低有效字节（LSB），再后面为第二个最高有效字节（MSB）后面跟着第二个最低有效字节（LSB）。

### UTF-8 编码字符串

后面会描述的控制报文中的文本字段编码为 UTF-8 格式的字符串。UTF-8 [[RFC3629]](#RFC3629) 是一个高效的

Unicode 字符编码格式，为了支持基于文本的通信，它对 ASCII 字符的编码做了优化。

每一个字符串都有一个两字节的长度字段作为前缀，它给出这个字符串 UTF-8 编码的字节数，它们在图例1.1 UTF-8编码字符串的结构中描述。因此可以传送的UTF-8编码的字符串大小有一个限制，不能超过65535字节。

除非另有说明，所有的 UTF-8 编码字符串的长度都必须在 0 到 65535 字节这个范围内。

图例 1‑1 UTF-8编码字符串的结构

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **二进制位** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| byte 1 | 字符串长度的最高有效字节（MSB） | | | | | | | |
| byte 2 | 字符串长度的最低有效字节（LSB） | | | | | | | |
| byte 3 …. | 如果长度大于0，这里是UTF-8编码的字符数据 | | | | | | | |

UTF-8 编码字符串中的字符数据**必须**是按照Unicode 规范[[Unicode]](#Unicode) 定义的和在RFC3629 [[RFC3629]](#RFC3629)中重申的有效的UTF-8格式。特别需要指出的是，这些数据**不能**包含字符码在 U+D800 和 U+DFFF 之间的

数据。如果服务端或客户端收到了一个包含无效 UTF-8 字符的控制报文，它**必须**关闭网络连接[MQTT-1.5.4-1]。

UTF-8 编码的字符串**不能**包含空字符 U+0000。如果客户端或服务端收到了一个包含 U+0000 的控制报

文，它**必须**关闭网络连接[MQTT-1.5.4-2]。

数据中不应该包含下面这些 Unicode 代码点的编码。如果一个接收者（服务端或客户端）收到了包含下列

任意字符的控制报文，它可以关闭网络连接：

* U+0001 和 U+001F 之间的控制字符
* U+007F 和 U+009F 之间的控制字符
* [[Unicode]](#Unicode) 规范定义的非字符代码点(例如U+0FFFF)

UTF-8 编码序列 0XEF 0xBB 0xBF 总是被解释为 U+FEFF（零宽度非换行空白字符），无论它出现在字符

串的什么位置，报文接收者都不能跳过或者剥离它 [MQTT-1.5.4-3]。

**非规范示例**

例如，字符串 A 𪛔 是一个大写拉丁字母 A 后面跟着一个代码点 U+2A6D4(它表示一个中日韩统一表意文字扩展 B 中的字符)，这个字符串编码如下：

图例 1‑2 UTF-8编码字符串非规范示例

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| byte 1 | 字符串长度MSB (0x00) | | | | | | | |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 2 | 字符串长度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 |

### 变长字节整数

剩余长度字段使用一个变长字节编码方案，对小于 128 的值它使用单字节编码。更大的值按下面的方式处理。低 7 位有效位用于编码数据，最高有效位用于指示是否有更多的字节。因此每个字节可以编码 128 个数值和一个延续位（continuation bit）。剩余长度字段最大 4 个字节[MQTT-1.5.5-1]。如表 1‑1 变长字节整数。

表 1‑1 变长字节整数大小

|  |  |  |
| --- | --- | --- |
| **字节数** | **最小值** | **最大值** |
| 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) |

**非规范评注**

非负整数 X 使用变长编码方案的算法如下：

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)

MOD是模运算，DIV是整数除法，OR是位操作或（C语言中分别是%，/，|）。

**非规范评注**

剩余长度字段的解码算法如下：

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)

AND 是位操作与（C 语言中的&）

这个算法终止时，value 包含的就是剩余长度的值。

### 二进制数据

二进制数据由一个双字节整数指示其数据长度，因此，二进制数据的长度被限制为0到65,535字节。

### UTF-8字符串对

UTF-8字符串对由两个UTF-8编码的字符串组成，用来表示名字-值对，第一个字符串表示名字，第二个字符串表示值。

所有的字符串**必须**遵循UTF-8字符串编码规范 [MQTT-1.5.7-1]。如果接受者（客户端或者服务端）接受到一个字符串对，然而其编码并不遵循规范，则此报文为无效报文。[4.13](#S4_13_Errors)节描述了错误处理的信息。

## 安全

MQTT客户端和服务端必须实现认证、授权和安全通信功能，如第5章所描述。强烈建议任何关注于个人身份信息或敏感信息的应用使用这些安全设施。

## 编辑约定

本规范用黄色高亮的文本标识一致性声明，每个一致性声明都分配了一个这种格式的引用：[MQTT-x.x.x-y]。

## 变更历史

### MQTT v3.1.1

MQTT v3.1.1 是首个OASIS标准版本MQTT **[MQTTV311]**。

MQTT v3.1.1也是ISO/IEC 20922:2016 [**[ISO20922]**](#ISO20922) 标准。

### MQTT v5.0

MQTT v5.0 在保持MQTT核心不变的基础上添加了大量的新功能。这些功能的主要目标如下：

* 进一步支持大规模可扩展系统
* 改进的错误报告Improved error reporting
* 规范化包括容量探索和请求响应在内的通用模式
* 包括用户属性在内的可扩展机制
* 改进性能并支持小型客户端

[附录C](#AppendixC)对MQTT v5.0的改进做出了总结。

# MQTT控制报文格式

## MQTT控制报文结构

MQTT 协议通过交换预定义的 MQTT 控制报文来通信。这一节描述这些报文的格式。

MQTT 控制报文由三部分组成，按照下图描述的顺序。

图 2‑1 - MQTT控制报文的结构

|  |
| --- |
| Fixed Header固定报头，所有控制报文都包含 |
| Variable Header 可变报头，部分控制报文包含 |
| Payload 有效载荷，部分控制报文包含 |

### 固定报头

如下图所示，每个 MQTT 控制报文都包含一个固定报头。

图 2‑2 固定报头的格式

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| byte 1 | MQTT控制报文的类型 | | | | 用于指定控制报文类型的标志位 | | | |
| byte 2… | 剩余长度 | | | | | | | |

### MQTT控制报文的类型

**位置：**第 1 个字节，二进制位 7-4。

表示为 4 位无符号值，这些值的定义见下表。

表 2‑1 - 控制报文的类型

|  |  |  |  |
| --- | --- | --- | --- |
| **名字** | **值** | **报文流动方向** | **描述** |
| Reserved | 0 | 禁止 | 保留 |
| CONNECT | 1 | 客户端到服务端 | 客户端请求连接服务端 |
| CONNACK | 2 | 服务端到客户端 | 连接报文确认 |
| PUBLISH | 3 | 两个方向都允许 | 发布消息 |
| PUBACK | 4 | 两个方向都允许 | QoS 1 消息发布收到确认 |
| PUBREC | 5 | 两个方向都允许 | 发布收到（保证交付第一步） |
| PUBREL | 6 | 两个方向都允许 | 发布释放（保证交付第二步） |
| PUBCOMP | 7 | 两个方向都允许 | QoS 2 消息发布完成（保证交付第三步） |
| SUBSCRIBE | 8 | 客户端到服务端 | 客户端订阅请求 |
| SUBACK | 9 | 服务端到客户端 | 订阅请求报文确认 |
| UNSUBSCRIBE | 10 | 客户端到服务端 | 客户端取消订阅请求 |
| UNSUBACK | 11 | 服务端到客户端 | 取消订阅报文确认 |
| PINGREQ | 12 | 客户端到服务端 | 心跳请求 |
| PINGRESP | 13 | 服务端到客户端 | 心跳响应 |
| DISCONNECT | 14 | 客户端到服务端 | 客户端断开连接 |
| AUTH | 15 | 两个方向都允许 | 认证信息交换 |

### 标志

固定报头第 1 个字节的剩余的 4 位 [3-0]包含每个 MQTT 控制报文类型特定的标志如下表所示。表格中任何标记为“保留”的标志位，都是保留给以后使用的，必须设置为表格中列出的值 [MQTT-2.1.3-1]。如果收到非法的标志，接收者必须关闭网络连接。有关错误处理的详细信息见[4.13](#S4_13_Errors)节。

表格 2‑2 - 标志位

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **MQTT控制报文** | **固定报头标志** | **Bit 3** | **Bit 2** | **Bit 1** | **Bit 0** |
| CONNECT | Reserved | 0 | 0 | 0 | 0 |
| CONNACK | Reserved | 0 | 0 | 0 | 0 |
| PUBLISH | Used in MQTT v5.0 | DUP | QoS | | RETAIN |
| PUBACK | Reserved | 0 | 0 | 0 | 0 |
| PUBREC | Reserved | 0 | 0 | 0 | 0 |
| PUBREL | Reserved | 0 | 0 | 1 | 0 |
| PUBCOMP | Reserved | 0 | 0 | 0 | 0 |
| SUBSCRIBE | Reserved | 0 | 0 | 1 | 0 |
| SUBACK | Reserved | 0 | 0 | 0 | 0 |
| UNSUBSCRIBE | Reserved | 0 | 0 | 1 | 0 |
| UNSUBACK | Reserved | 0 | 0 | 0 | 0 |
| PINGREQ | Reserved | 0 | 0 | 0 | 0 |
| PINGRESP | Reserved | 0 | 0 | 0 | 0 |
| DISCONNECT | Reserved | 0 | 0 | 0 | 0 |
| AUTH | Reserved | 0 | 0 | 0 | 0 |

DUP =控制报文的重复分发标志

QoS = PUBLISH 报文的服务质量等级

RETAIN = PUBLISH 报文的保留标志

PUBLISH控制报文中的DUP、QoS和RETAIN标志的描述见[3.3.1](#_CONNECT_Fixed_Header)节。

### 剩余长度

**位置：**从第 2 个字节开始。

剩余长度（Remaining Length）是一个可变字节整数，用来表示当前控制报文剩余部分的字节数，包括可变报头和负载的数据。剩余长度不包括用于编码剩余长度字段本身的字节数。MQTT控制报文总长度等于固定报头的长度加上剩余长度。

## 可变报头

某些 MQTT 控制报文包含一个可变报头部分。它在固定报头和负载之间。可变报头的内容根据报文类型的

不同而不同。可变报头的报文标识符（Packet Identifier）字段存在于在多个类型的报文里。

### 报文标识符

部分类型MQTT控制报文的可变报头部分包含了2个字节的报文标识符字段。这些MQTT控制报文类型为：PUBLISH报文（当QoS>0时），PUBACK，PUBREC，PUBREC，PUBREL，PUBCOMP，SUBSCRIBE，SUBACK，UNSUBSCRIBE，UNSUBACK。

需要报文标识符的MQTT控制报文如下表所示。

表 2‑3 - 包含报文标识符的控制报文

|  |  |
| --- | --- |
| **控制报文** | **报文标识符字段** |
| CONNECT | 不需要 |
| CONNACK | 不需要 |
| PUBLISH | 需要（如果QoS > 0） |
| PUBACK | 需要 |
| PUBREC | 需要 |
| PUBREL | 需要 |
| PUBCOMP | 需要 |
| SUBSCRIBE | 需要 |
| SUBACK | 需要 |
| UNSUBSCRIBE | 需要 |
| UNSUBACK | 需要 |
| PINGREQ | 不需要 |
| PINGRESP | 不需要 |
| DISCONNECT | 不需要 |
| AUTH | 不需要 |

QoS 设置为 0 的 PUBLISH 报文**不能**包含报文标识符[MQTT-2.2.1-2]。

客户端每次发送一个新的SUBSCRIBE，UNSUBSCRIBE或者PUBLISH（当QoS>0时）控制报文时都**必须**分配一个当前未使用的非零报文标识符 [MQTT-2.2.1-3]。

服务端每次发送一个新的PUBLISH（当QoS>0）控制报文时都**必须**分配一个当前未使用的非零报文标识符 [MQTT-2.2.1-4]。

当客户端处理完这个报文对应的确认后，这个报文标识符就释放可重用。QoS 1 的 PUBLISH 对应的是 PUBACK，QoS 2 的 PUBLISH 对应的是包含原因码128以上的 PUBCOMP或PUBREC，与 SUBSCRIBE 或UNSUBSCRIBE 对应的分别是 SUBACK 或 UNSUBACK。

PUBLISH，SUBSCRIBE和UNSUBSCRIBE的报文标识符，在一次会话中对于客户端和服务端来说分属于不同的组。某个报文标识符在某一时刻**不能**被多个命令所使用。

PUBACK，PUBREC和PUBREL 报文**必须**包含与最初发送的 PUBLISH 报文相同的报文标识符 [MQTT-2.2.1-5]。类似地，SUBACK 和 UNSUBACK **必须**包含在对应的 SUBSCRIBE 和 UNSUBSCRIBE 报文中使用的报文标识符 [MQTT-2.2.1-6]。

客户端和服务端彼此独立地分配报文标识符。因此，客户端服务端组合使用相同的报文标识符可以实现并

发的消息交换。

**非规范评注**

客户端发送标识符为 0x1234 的 PUBLISH 报文，它有可能会在收到那个报文的 PUBACK 之前，先

收到服务端发送的另一个不同的但是报文标识符也为 0x1234 的 PUBLISH 报文。

客户端 服务端

PUBLISH报文标识符 = 0x1234 ---->

<---- PUBLISH报文标识符 = 0x1234

PUBACK报文标识符 = 0x1234 ---->

<---- PUBACK报文标识符 = 0x1234

### 属性

CONNECT，CONNACK，PUBLISH，PUBACK，PUBREC，PUBREL，PUBCOMP，SUBSCRIBE，SUBACK，UNSUBACK，DISCONNECT和AUTH报文可变报头的最后一部分是一组属性。CONNECT报文的遗嘱（Will）属性字段中也包含了一组可选的属性。

属性字段由属性长度和所有属性组成。

#### 属性长度

属性长度被编码为变长字节整数。属性长度不包含用于编码属性长度自身的字节数，但包含所有属性的长度。如果没有任何属性，**必须**由属性长度为零的字段来指示 [MQTT-2.2.2-1]。

#### 属性

一个属性包含一段数据和一个定义了属性用途和数据类型的标识符。标识符被编码为变长字节整数。任何控制报文，如果包含了：对于该控制报文无效的标识符，或者错误类型的数据，都是无效报文。收到无效报文时，服务端或客户端使用包含原因码0x81（无效报文）CONNACK或DISCONNECT报文进行错误处理，如[4.13](#S4_13_Errors)节所述。标识符排序不分先后。

表 2‑4 - 属性

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **标识符** | | **属性名 （用途）** | **数据类型** | **报文/遗嘱属性** |
| **Dec** | **Hex** |
| 1 | 0x01 | 载荷格式说明 | 字节 | PUBLISH, Will Properties |
| 2 | 0x02 | 消息过期时间 | 四字节整数 | PUBLISH, Will Properties |
| 3 | 0x03 | 内容类型 | UTF-8编码字符串 | PUBLISH, Will Properties |
| 8 | 0x08 | 响应主题 | UTF-8编码字符串 | PUBLISH, Will Properties |
| 9 | 0x09 | 相关数据 | 二进制数据 | PUBLISH, Will Properties |
| 11 | 0x0B | 定义标识符 | 变长字节整数 | PUBLISH, SUBSCRIBE |
| 17 | 0x11 | 会话过期间隔 | 四字节整数 | CONNECT, CONNACK, DISCONNECT |
| 18 | 0x12 | 分配客户端标识符 | UTF-8编码字符串 | CONNACK |
| 19 | 0x13 | 服务端保活时间 | 双字节整数 | CONNACK |
| 21 | 0x15 | 认证方法 | UTF-8编码字符串 | CONNECT, CONNACK, AUTH |
| 22 | 0x16 | 认证数据 | 二进制数据 | CONNECT, CONNACK, AUTH |
| 23 | 0x17 | 请求问题信息 | 字节 | CONNECT |
| 24 | 0x18 | 遗嘱延时间隔 | 四字节整数 | Will Properties |
| 25 | 0x19 | 请求响应信息 | 字节 | CONNECT |
| 26 | 0x1A | 请求信息 | UTF-8编码字符串 | CONNACK |
| 28 | 0x1C | 服务端参考 | UTF-8编码字符串 | CONNACK, DISCONNECT |
| 31 | 0x1F | 原因字符串 | UTF-8编码字符串 | CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK, DISCONNECT, AUTH |
| 33 | 0x21 | 接收最大数量 | 双字节整数 | CONNECT, CONNACK |
| 34 | 0x22 | 主题别名最大长度 | 双字节整数 | CONNECT, CONNACK |
| 35 | 0x23 | 主题别名 | 双字节整数 | PUBLISH |
| 36 | 0x24 | 最大QoS | 字节 | CONNACK |
| 37 | 0x25 | 保留属性可用性 | 字节 | CONNACK |
| 38 | 0x26 | 用户属性 | UTF-8字符串对 | CONNECT, CONNACK, PUBLISH, Will Properties, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK, DISCONNECT, AUTH |
| 39 | 0x27 | 最大报文长度 | 四字节整数 | CONNECT, CONNACK |
| 40 | 0x28 | 通配符订阅可用性 | 字节 | CONNACK |
| 41 | 0x29 | 订阅标识符可用性 | 字节 | CONNACK |
| 42 | 0x2A | 共享订阅可用性 | 字节 | CONNACK |

**非规范评注**

尽管属性标识符用变长字节整数来表示，但在此版本协议中，所有的标识符均由一个字节来表示。

## 载荷

某些 MQTT 控制报文在报文的最后部分包含一个有效载荷，这将在第三章论述。对于 PUBLISH 来说有效

载荷就是应用消息。

表 2‑5 -包含有效载荷的控制报文

|  |  |
| --- | --- |
| **控制报文** | **载荷** |
| CONNECT | 需要 |
| CONNACK | 不需要 |
| PUBLISH | 可选 |
| PUBACK | 不需要 |
| PUBREC | 不需要 |
| PUBREL | 不需要 |
| PUBCOMP | 不需要 |
| SUBSCRIBE | 需要 |
| SUBACK | 需要 |
| UNSUBSCRIBE | 需要 |
| UNSUBACK | 需要 |
| PINGREQ | 不需要 |
| PINGRESP | 不需要 |
| DISCONNECT | 不需要 |
| AUTH | 不需要 |

## 原因码

原因码是一个单字节无符号数，用来指示一次操作的结果。小于0x80的原因码指示某次操作成功完成，通常用0来表示。大于等于0x80的原因码用来指示操作失败。

CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, DISCONNECT和AUTH控制报文的可变报头有一个单字节的原因码。SUBACK和UNSUBACK报文的载荷字段包含一个或多个原因码。

原因码如下表所示。

表 2‑6 - 原因码

|  |  |  |  |
| --- | --- | --- | --- |
| **Reason Code** | | **Name** | **Packets** |
| **Decimal** | **Hex** |
| 0 | 0x00 | 成功 | CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, UNSUBACK, AUTH |
| 0 | 0x00 | 正常断开 | DISCONNECT |
| 0 | 0x00 | 授权的QoS 0 | SUBACK |
| 1 | 0x01 | 授权的QoS 1 | SUBACK |
| 2 | 0x02 | 授权的QoS 2 | SUBACK |
| 4 | 0x04 | 包含遗嘱的断开 | DISCONNECT |
| 16 | 0x10 | 无匹配订阅 | PUBACK, PUBREC |
| 17 | 0x11 | 订阅不存在 | UNSUBACK |
| 24 | 0x18 | 继续认证 | AUTH |
| 25 | 0x19 | 重新认证 | AUTH |
| 128 | 0x80 | 未指明的错误 | CONNACK, PUBACK, PUBREC, SUBACK, UNSUBACK, DISCONNECT |
| 129 | 0x81 | 无效报文 | CONNACK, DISCONNECT |
| 130 | 0x82 | 协议错误 | CONNACK, DISCONNECT |
| 131 | 0x83 | 实现错误 | CONNACK, PUBACK, PUBREC, SUBACK, UNSUBACK, DISCONNECT |
| 132 | 0x84 | 协议版本不支持 | CONNACK |
| 133 | 0x85 | 客户端标识符无效 | CONNACK |
| 134 | 0x86 | 用户名密码错误 | CONNACK |
| 135 | 0x87 | 未授权 | CONNACK, PUBACK, PUBREC, SUBACK, UNSUBACK, DISCONNECT |
| 136 | 0x88 | 服务端不可用 | CONNACK |
| 137 | 0x89 | 服务端正忙 | CONNACK, DISCONNECT |
| 138 | 0x8A | 禁止 | CONNACK |
| 139 | 0x8B | 服务端关闭中 | DISCONNECT |
| 140 | 0x8C | 无效的认证方法 | CONNACK, DISCONNECT |
| 141 | 0x8D | 保活超时 | DISCONNECT |
| 142 | 0x8E | 会话被接管 | DISCONNECT |
| 143 | 0x8F | 主题过滤器无效 | SUBACK, UNSUBACK, DISCONNECT |
| 144 | 0x90 | 主题名无效 | CONNACK, PUBACK, PUBREC, DISCONNECT |
| 145 | 0x91 | 报文标识符已被占用 | PUBACK, PUBREC, SUBACK, UNSUBACK |
| 146 | 0x92 | 报文标识符无效 | PUBREL, PUBCOMP |
| 147 | 0x93 | 接收超出最大数量 | DISCONNECT |
| 148 | 0x94 | 主题别名无效 | DISCONNECT |
| 149 | 0x95 | 报文过长 | CONNACK, DISCONNECT |
| 150 | 0x96 | 消息太过频繁 | DISCONNECT |
| 151 | 0x97 | 超出配额 | CONNACK, PUBACK, PUBREC, SUBACK, DISCONNECT |
| 152 | 0x98 | 管理行为 | DISCONNECT |
| 153 | 0x99 | 载荷格式无效 | CONNACK, PUBACK, PUBREC, DISCONNECT |
| 154 | 0x9A | 不支持保留 | CONNACK, DISCONNECT |
| 155 | 0x9B | 不支持QoS | CONNACK, DISCONNECT |
| 156 | 0x9C | （临时）使用其他服务端 | CONNACK, DISCONNECT |
| 157 | 0x9D | 服务端已（永久）移动 | CONNACK, DISCONNECT |
| 158 | 0x9E | 不支持共享订阅 | SUBACK, DISCONNECT |
| 159 | 0x9F | 超出连接速率限制 | CONNACK, DISCONNECT |
| 160 | 0xA0 | 最大连接时间 | DISCONNECT |
| 161 | 0xA1 | 不支持订阅标识符 | SUBACK, DISCONNECT |
| 162 | 0xA2 | 不支持通配符订阅 | SUBACK, DISCONNECT |

**非规范评注**

对于原因码0x91（报文标识符已被占用）的处理可以为尝试修复会话、以会话清除标志为1重置会话或者判定客户端或服务端实现有缺陷。

# 控制报文

## CONNECT – 连接请求

客户端到服务端的网络连接建立后，客户端发给服务端的第一个报文**必须**是CONNECT报文 [MQTT-3.1.0-1]。

在一个网络连接上，客户端只能发送一次 CONNECT 报文。服务端必须将客户端发送的第二个 CONNECT报文当作协议违规处理并断开客户端的连接[MQTT-3.1.0-2]. 有关错误处理的信息请查看[4.13](#S4_13_Errors)节。

有效载荷包含一个或多个编码的字段。包括客户端的唯一标识符，Will 主题，Will 消息，用户名和密码。除了客户端标识之外，其它的字段都是可选的，基于标志位来决定可变报头中是否需要包含这些字段。

### CONNECT固定报头

图 3‑1 - CONNECT报文固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| byte 1 | MQTT 报文类型 (1) | | | | 保留位 | | | |
|  | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| byte 2… | 剩余长度值 | | | | | | | |

**剩余长度字段**

剩余长度等于可变报头的长度加上有效载荷的长度。编码方式为可变字节整数。

### CONNECT可变报头

CONNECT 报文的可变报头按下列次序包含四个字段：协议名（Protocol Name），协议级别（Protocol

Level），连接标志（Connect Flags），保持连接（Keep Alive）和属性（Properties）。[2.2.2](#_Properties)节描述了属性（Properties）编码规则。

#### 协议名

图 3‑2 - 协议名字节

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **说明** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 协议名 | | | | | | | | | |
| byte 1 | 长度MSB (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 2 | 长度 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 |

协议名是表示协议名 *MQTT* 的 UTF-8 编码的字符串。MQTT 规范的后续版本不会改变这个字符串的偏移和长度。

支持多种协议的服务端使用协议名字段判断数据是否为MQTT报文。协议名**必须**是UTF-8字符串“MQTT”。如果服务端不愿意接受CONNECT但希望表明其MQTT服务端身份，**可以**发送包含原因码为0x84（不支持的协议版本）的CONNACK报文，然后**必须**关闭网络连接。 [MQTT-3.1.2-1].

**非规范评注**

数据包检测工具，例如防火墙，可以使用协议名来识别 MQTT 流量。

#### 协议版本

图 3‑3 - 协议版本字节

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **描述** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 协议级别 | | | | | | | | | |
| byte 7 | 版本(5) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

客户端使用一个字节无符号数表示协议修订级别。MQTT v5.0的协议版本字段为5（0x05）。

支持多版本MQTT协议的服务端使用*协议版本* 字段判定客户端正使用的MQTT协议版本。如果协议版本不是5且服务端不愿意接受此CONNECT报文，可以发送包含原因码0x84（不支持的协议版本）的CONNACK报文，然后必须关闭网络连接。 [MQTT-3.1.2-2]。

#### 连接标志

连接标志字节包含一些用于指定 MQTT 连接行为的参数。它还指出有效载荷中的字段是否存在。

图 3‑4 - 连接标志位

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

服务端必须验证 CONNECT 控制报文的保留标志位（第 0 位）是否为 0 [MQTT-3.1.2-3]，如果不为 0 必须断开客户端连接。[4.13](#S4_13_Errors)节给出了错误处理信息。

#### 新开始

**位置：**连接标志字节的第 1 位

这个二进制位表明此次连接是一个新的会话还是一个已存在的会话的延续。[4.1](#_Session_State)节定义了会话状态。

如果收到新开始（Clean Start）为1的CONNECT报文，客户端和服务端**必须**丢弃任何已存在的会话，并开始一个新的会话 [MQTT-3.1.2-4]。相应的，CONNACK报文中的当前会话标志设置为0。

如果收到新开始（Clean Start）为0的CONNECT报文，并且存在一个关联此客户端标识符的会话，服务端**必须**基于此会话的状态恢复与客户端的通信。 [MQTT-3.1.2-5]。 如果收到新开始（Clean Start）为0的CONNECT报文，并且不存在任何关联此客户端标识符的会话，服务端**必须**创建一个新的会话 [MQTT-3.1.2-6]。

#### 遗嘱标志

**位置：**连接标志字节的第2位

如果遗嘱标志（Will Flag）被设置为1，表示遗嘱消息**必须**已存储在服务端与此客户端标识符相关的会话中。[MQTT-3.1.2-7]。遗嘱消息（Will Message）包含遗嘱属性，遗嘱主题和遗嘱载荷字段。遗嘱**必须**在网络连接被关闭、遗嘱延时间隔到期或者会话结束之后被发布，除非服务端收到包含原因码为0x00（正常关闭）的DISCONNECT报文之后删除了遗嘱消息（Will Message），或者一个关于此客户端标识符的新的网络连接在遗嘱迟发时间（Will Delay Interval）超时之前被创建。 [MQTT-3.1.2-8]。

遗嘱发布的条件，包括但不限于：

* 服务端检测到了一个 I/O 错误或者网络故障
* 客户端在保持连接（Keep Alive）的时间内未能通讯
* 客户端在没有发送包含原因码0x00（正常关闭）的情况下关闭了网络连接
* 服务端在没有收到包含原因码0x00（正常关闭）的情况下关闭了网络连接

如果遗嘱标志（Will Flag）被设置为1，遗嘱属性（Will Property）、遗嘱主题（Will Topic）和遗嘱载荷（Will Payload）字段**必须**存在于报文有效载荷中 [MQTT-3.1.2-9]。一旦遗嘱消息（Will Message）被发布或者服务端收到包含原因码为0x00（正常关闭）的DISCONNECT报文，遗嘱消息（Will Message）**必须**从服务端的会话中删除 [MQTT-3.1.2-10]。

服务端**必须**在网络连接断开并且遗嘱迟发时间（Will Delay Interval）到期，或者会话结束之后立即发布遗嘱消息。服务端关闭或出错的情况下，**可以**在服务重新启动之后发布遗嘱消息（Will Message）。这种情况下从服务端出错到遗嘱发布之间存在一定的延迟。

Refer to 关于遗嘱迟发时间（Will Delay Interval）的详细信息，请参考[3.1.3.2](#_Will_Delay_Interval_1)节。

**非规范评注**

通过设置晚于会话过期间隔（Session Expiry Interval）的遗嘱迟发时间（Will Delay Interval）并发送包含原因码0x04（包含遗嘱的断开连接），客户端得以发出会话过期（Session Expiry）通告。

#### 遗嘱QoS

**位置：**连接标志字节的第3、4位

这两个比特指定了发布遗嘱消息（Will Message）时的服务质量（QoS）。

如果遗嘱标志（Will Flag）设置为0，遗嘱服务质量（Will QoS）**必须**也设置为0 (0x00) [MQTT-3.1.2-11]。

如果遗嘱标志（Will Flag）设置为1，遗嘱服务质量（Will QoS）**可以**被设置为0（0x00），1（0x01）或2（0x02） [MQTT-3.1.2-12]。设置为3（0x03）的报文是无效报文。[4.13](#S4_13_Errors)节描述了错误处理信息。

#### 遗嘱保留

**位置：**连接标志字节的第5位

此位指定遗嘱消息（Will Message）在发布时是否会被保留。

如果遗嘱标志被设置为0，遗嘱保留（Will Retain）标志也**必须**设置为0 [MQTT-3.1.2-13]。如果遗嘱标志（Will Flag）被设置为1时，如果遗嘱保留（Will Retain）被设置为0，则服务端**必须**将遗嘱消息当做非保留消息发布 [MQTT-3.1.2-14]。如果遗嘱保留（Will Retain）被设置为1，则服务端**必须**将遗嘱消息当做保留消息发布 [MQTT-3.1.2-15]。

#### 用户名标志

**位置：**连接标志字节的第7位

如果用户名标志（User Name Flag）被设置为 0，有效载荷中**不能**包含用户名字段 [MQTT-3.1.2-16]。如果用户名标志（User Name Flag）被设置为 0，有效载荷中**必须**包含用户名字段 [MQTT-3.1.2-17]。

#### 密码标志

**位置：**连接标志字节的第6位

如果密码标志（Password Flag）被设置为 0，有效载荷中不能包含密码字段 [MQTT-3.1.2-18]。如果密码标志（Password Flag）被设置为 1，有效载荷中必须包含密码字段 [MQTT-3.1.2-19]。

**非规范评注**

相比MQTT v3.1.1，此版本协议允许在没有用户名的情况下发送密码。这表明密码除了作为口令之外还可以有其他用途。

#### 保持连接

图 3‑5 - 保持连接字节

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

保持连接（Keep Alive）使用双字节整数来表示以秒为单位的时间间隔。它是指在客户端传输完成

一个控制报文的时刻到发送下一个报文的时刻，两者之间允许空闲的最大时间间隔。客户端负责保证控制

报文发送的时间间隔不超过保持连接的值。如果没有任何其它的控制报文可以发送，客户端**必须**发送一个

PINGREQ 报文 [MQTT-3.1.2-20]。

如果服务端返回的CONNACK报文中包含服务端保持连接（Server Keep Alive），客户端必须使用此值代替其发送的保持连接（Keep Alive） [MQTT-3.1.2-21]。

不管保持连接的值是多少，客户端任何时候都可以发送 PINGREQ 报文，并且使用 PINGRESP 报文判断网络和服务端的活动状态。

如果保持连接的值非零，并且服务端在1.5倍的保持连接时间内没有收到客户端的控制报文，它**必须**断开客户端的网络连接，并判定网络连接已断开 [MQTT-3.1.2-22]。

客户端发送了 PINGREQ 报文之后，如果在合理的时间内仍没有收到 PINGRESP 报文，它应该关闭到服务端的网络连接。

保持连接（Keep Alive）值为零的结果是关闭保持连接（Keep Alive）机制。如果保持连接（Keep Alive）值为零，客户端不必按照任何特定的时间发送控制报文。

**非规范评注**

服务端可能因为其他原因断开客户端连接，比如服务端将要关闭服务。设置保持连接（Keep Alive）不保证客户端将一直保持连接状态。

**非规范评注**

保持连接的实际值是由应用指定的，一般是几分钟。允许的最大值是 18 小时 12 分 15 秒。

#### CONNECT属性

##### 属性长度

CONNECT报文可变报头中的属性（Properties）长度被编码为可变字节整数。

##### 会话过期间隔

**17 (0x11)，**会话过期间隔（Session Expiry Interval）标识符。

跟随其后的是用四字节整数表示的以秒为单位的会话过期间隔（Session Expiry Interval）。包含多个会话过期间隔（Session Expiry Interval）将造成协议错误（Protocol Error）。

如果会话过期间隔（Session Expiry Interval）值未指定，则使用0。如果设置为0或者未指定，会话将在网络连接（Network Connection）关闭时结束。

如果会话过期间隔（Session Expiry Interval）为0xFFFFFFFF (UINT\_MAX)，则会话永不过期。

如果网络连接关闭时会话过期间隔（Session Expiry Interval）大于0，则客户端与服务端必须存储会话状态 [MQTT-3.1.2-23]。

**非规范评注**

客户端或服务端可能会因为中断运行导致会话时钟某些时间未运行。这将导致会话的删除被延迟。

更多关于会话的信息参考[4.1](#_Session_State)节。关于会话存储的状态的详细和限制参考[4.1.1](#_Storing_Session_State)节。

当会话过期时，客户端和服务端无需以原子操作的方式删除会话状态。

**非规范评注**

把新开始（Clean Start）设置为1且会话过期间隔（Session Expiry Interval）设置为0，等同于在MQTT v3.1.1中把会话清除（CleanSession）设置为1。把会话清除（Clean Start）设置为0且不设置会话过期间隔（Session Expiry Interval），等同于在MQTT v3.1.1中把会话清除（CleanSession）设置为0。

**非规范评注**

当希望只处理连接上服务端之后才发布的消息，客户端应该把新开始（Clean Start）设置为1且会话过期间隔（Session Expiry Interval）设置为0，这样客户端就不会收到它连接之前被服务端所发布的消息，并且需要每次连接上服务端时重新订阅其感兴趣的主题。

**非规范评注**

某些客户端使用的网络可能只能提供断断续续的连接，这种客户端可以使用较短的会话过期间隔（Session Expiry Interval）以便在网络再次可用后重新连接到服务端时获得持续的消息交付。如果客户端不再重新连接，且允许会话过期，应用消息将会丢失。

**非规范评注**

某个客户端设置较长的会话过期间隔（Session Expiry Interval）或设置会话不过期，即要求服务端为其保持会话到其下一次连接上服务端之后。只有打算在一段时间之后将会重连服务端时，客户端才应该设置较长的会话过期间隔（Session Expiry Interval）。当客户端认定其将来不会使用本次会话时，应该在断开时把会话过期间隔（Session Expiry Interval）设置为0。

**非规范评注**

客户端应当使用CONNACK报文中的会话存在（Session Present）来判定服务端是否存储了其会话。

**非规范评注**

客户端应当以服务端返回的会话存在（Session Present）标志来判定会话是否已过期，而不是客户端自己实现的会话过期状态。如果客户端自己实现会话过期状态，则需要将会话应当被删除的时间作为会话状态的一部分而存储。

##### 接收最大值

**33 (0x21)，**接收最大值（Receive Maximum）标识符。

跟随其后的是由双字节整数表示的最大接收值。包含多个接收最大值或接收最大值为0将造成协议错误（Protocol Error）。

客户端使用此值限制客户端愿意同时处理的QoS为1和QoS为2的发布消息最大数量。没有机制可以限制服务端试图发送的QoS为0的发布消息。

接收最大值只将被应用在当前网络连接。如果没有设置最大接收值，将使用默认值65535。

关于接收最大值的详细使用，参考[4.9](#_Flow_Control)节 流控。

##### 最大报文长度

**39 (0x27)**，最大报文长度（Maximum Packet Size）标识符。

跟随其后的是由四字节整数表示的客户端愿意接收的最大报文长度（Maximum Packet Size），如果没有设置最大报文长度（Maximum Packet Size），则按照协议由固定报头中的剩余长度可编码最大值和协议报头对数据包的大小做限制。

包含多个最大报文长度（Maximum Packet Size）或者最大报文长度（Maximum Packet Size）值为0将造成协议错误。

**非规范评注**

客户端如果选择了限制最大报文长度，应该为最大报文长度设置一个合理的值。

如[2.1.4](#_Remaining_Length)节 所述，最大报文长度是控制报文的总长度。客户端使用最大报文长度通知服务端其所能处理的单个报文长度限制。

服务端**不能**发送超过最大报文长度（Maximum Packet Size）的报文给客户端 [MQTT-3.1.2-24]。收到长度超过限制的报文将导致协议错误，此时客户端应该发送包含原因码0x95（报文过大）的DISCONNECT报文给服务端，详见[4.13](#S4_13_Errors) 节。

当报文过大而不能发送时，服务端**必须**丢弃这些报文，然后当做应用消息发送已完成处理 [MQTT-3.1.2-25]。

共享订阅的情况下，如果一条消息对于部分客户端来说太长而不能发送，服务端可以选择丢弃此消息或者把消息发送给剩余能够接收此消息的客户端。

**非规范评注**

服务端可以把那些没有发送就被丢弃的报文放在*死信队列* 上，或者执行其他诊断操作。具体的操作超出了本规范的范围。

##### 主题别名最大值

**34 (0x22)，**主题别名最大值（Topic Alias Maximum）标识符。

跟随其后的是用双字节整数表示的主题别名最大值（Topic Alias Maximum）。包含多个主题别名最大值（Topic Alias Maximum）将造成协议错误（Protocol Error）。没有设置主题别名最大值属性的情况下，主题别名最大值默认为零。

此值指示了客户端能够接收的来自服务端的主题别名（Topic Alias）最大数量。客户端使用此值来限制本次连接可以拥有的主题别名的数量。服务端在一个PUBLISH报文中发送的主题别名**不能**超过客户端设置的主题别名最大值（Topic Alias Maximum） [MQTT-3.1.2-26]。值为零表示本次连接客户端不接受任何主题别名（Topic Alias）。如果主题别名最大值（Topic Alias）没有设置，或者设置为零，则服务端**不能**向此客户端发送任何主题别名（Topic Alias）。 [MQTT-3.1.2-27]。

##### 请求响应信息

**25 (0x19)，**请求响应信息（Request Response Information）标识符。

跟随其后的是用一个字节表示的0或1。包含多个请求响应信息（Request Response Information），或者请求响应信息（Request Response Information）的值既不为0也不为1会造成协议错误（Protocol Error）。如果没有请求响应信息（Request Response Information），则请求响应默认值为0。

客户端使用此值向服务端请求CONNACK报文中的响应信息（Response Information）。值为0，表示服务端**不能**返回响应信息 [MQTT-3.1.2-28]。值为1，表示服务端**可以**在CONNACK报文中返回响应信息。

非规范评注

即使客户端请求响应信息（Response Information），服务端也可以选择不发送响应信息（Response Information）。

更多关于请求/响应信息的内容，请参考[4.10](#_Request_/_Response)节。

##### 请求问题信息

**23 (0x17)，**请求问题信息（Request Problem Information）标识符。

跟随其后的是用一个字节表示的0或1。包含多个请求问题信息（Request Problem Information），或者请求问题信息（Request Problem Information）的值既不为0也不为1会造成协议错误（Protocol Error）。如果没有请求问题信息（Request Problem Information），则请求问题默认值为1。

客户端使用此值指示遇到错误时是否发送原因字符串（Reason String）或用户属性（User Properties）。

如果请求问题信息的值为0，服务端**可以**选择在CONNACK或DISCONNECT报文中返回原因字符串（Reason String）或用户属性（User Properties），但**不能**在除PUBLISH，CONNACK或DISCONNECT之外的报文中发送原因字符串（Reason String）或用户属性（User Properties） [MQTT-3.1.2-29]。如果此值为0，并且在除PUBLISH，CONNACK或DISCONNECT之外的报文中收到了原因字符串（Reason String）或用户属性（User Properties），客户端将发送一个包含原因码0x82（协议错误）的DISCONNECT报文给服务端，如[4.13](#S4_13_Errors)节 所述。

如果此值为1，服务端可以在任何被允许的报文中返回原因字符串（Reason String）或用户属性（User Properties）。

##### 用户属性

**38 (0x26)，**用户属性（User Property）标识符。

跟随其后的是一个UTF-8字符串对。

用户属性（User Property）可以出现多次，表示多个名字/值对。相同的名字可以出现多次。

**非规范评注**

CONNECT报文中的用户属性可以被用来发送客户端到服务端的连接相关的属性。这些属性的意义本规范不做定义。

##### 认证方法

**21 (0x15)，**认证方法（Authentication Method）标识符。

跟随其后的是一个UTF-8编码的字符串，包含了扩展认证的认证方法（Authentication Method）名称。包含多个认证方法将造成协议错误（协议错误）。

如果没有认证方法，则不进行扩展验证。参考[4.12](#_Enhanced_authentication)节。

如果客户端在CONNECT报文中设置了认证方法，则客户端在收到CONNACK报文之前**不能**发送除AUTH或DISCONNECT之外的报文 [MQTT-3.1.2-30]。

##### 认证数据

**22 (0x16)，**认证数据（Authentication Data）标识符。

跟随其后的是二进制的认证数据。没有认证方法却包含了认证数据（Authentication Data），或者包含多个认证数据（Authentication Data）将造成协议错误（Protocol Error）。

认证数据的内容由认证方法定义，关于扩展认证的更多信息，请参考[4.12](#_Enhanced_authentication)节。

#### 可变报头非规范示例

图 3‑6 - 可变报头示例

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **说明** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 协议名Protocol Name | | | | | | | | | |
| byte 1 | 长度Length MSB (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 2 | 长度Length LSB (4) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| byte 3 | ‘M’ | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| byte 4 | ‘Q’ | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| byte 5 | ‘T’ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| byte 6 | ‘T’ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| Protocol Version | | | | | | | | | |
|  | **说明** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| byte 7 | 版本Version (5) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 连接标志Connect Flags | | | | | | | | | |
| byte 8 | 用户名标志User Name Flag (1)  密码标志Password Flag (1)  遗嘱保留标志Will Retain (0)  遗嘱服务质量Will QoS (01)  遗嘱标志Will Flag (1)  新开始Clean Start(1)  *保留Reserved* (0) | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 保持连接Keep Alive | | | | | | | | | |
| byte 9 | 保持连接Keep Alive MSB (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 10 | 保持连接Keep Alive LSB (10) | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 属性Properties | | | | | | | | | |
| byte 11 | 长度Length (5) | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| byte 12 | 会话过期间隔标识符 (17) | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| byte 13 | 会话过期间隔  Session Expiry Interval (10) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 16 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

### CONNECT载荷

CONNECT报文的载荷中包含由可变报头（Variable Header）中的标志确定的一个或多个以长度为前缀的字段。这些字段若存在，**必须**按照客户端标识符（Client Identifier）、遗嘱属性（Will Properties）、遗嘱主题（Will Topic）、遗嘱载荷（Will Payload）、用户名（User Name）、密码（Password）的顺序出现 [MQTT-3.1.3-1]。

#### 客户端标识符（ClientID）

服务端使用客户端标识符（ClientID）识别客户端。连接服务端的每个客户端都有唯一的客户端标识符

（ClientID）。客户端和服务端都**必须**使用客户端标识符（ClientID）识别两者之间的 MQTT 会话相关的状态 [MQTT-3.1.3-2]。更多关于会话状态的信息请参考[4.1](#_Session_State)节。

客户端标识符**必须**存在，且**必须**作为CONNECT报文载荷的第一个字段出现 [MQTT-3.1.3-3]。

客户端标识符**必须**被编码为[1.5.4](#_UTF-8_Encoded_String)节 中所定义的UTF-8字符串 [MQTT-3.1.3-4]。

服务端**必须**允许 1 到 23 个字节长的 UTF-8 编码的客户端标识符，客户端标识符只能包含这些字符：

"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"（大写字母、小写字母和数字） [MQTT-3.1.3-5]。

服务端**可以**允许编码后超过 23 个字节的客户端标识符 (ClientID)。服务端**可以**允许包含不是上面列表字符

的客户端标识符 (ClientID)。

服务端**可以**允许客户端提供一个零字节的客户端标识符 (ClientID) ，如果这样做了，服务端**必须**将这看作特

殊情况并分配唯一的客户端标识符给那个客户端 [MQTT-3.1.3-6]。然后它**必须**假设客户端提供了那个唯一的客户端标识符，正常处理这个 CONNECT 报文 [MQTT-3.1.3-7]。

如果服务端拒绝了某个客户端标识符（ClientID），它**可以**发送包含原因码0x85（客户端标识符无效）的CONNACK报文作为对客户端的CONNECT报文的回应 ，如[4.13](#S4_13_Errors)节 所述。之后**必须**关闭网络连接 [MQTT-3.1.3-8]。

**非规范评注**

客户端在实现时可以提供一个便于生成随机客户端标识符的算法。使用此算法时，客户端需要注意避免创建长期孤儿会话。

#### 遗嘱属性

如果遗嘱标志（Will Flag）被设置为1，有效载荷的下一个字段是遗嘱属性（Will Properties）。遗嘱属性字段定义了遗嘱消息（Will Message）将何时被发布，以及被发布时的应用消息（Application Message）属性。遗嘱属性包括属性长度和属性。

##### 属性长度

遗嘱属性（Will Properties）中的属性长度被编码为可变长字节整数。

##### 遗嘱延时间隔

**24 (0x18)，**遗嘱延时间隔（Will Delay Interval）标识符。

跟随其后的是由四字节整数表示的以秒为单位的遗嘱延时间隔（Will Delay Interval）。包含多个遗嘱延时间隔将造成协议错误（Protocol Error）。如果没有设置遗嘱延时间隔，遗嘱延时间隔默认值将为0，即不用延时发布遗嘱消息（Will Message）。

服务端将在遗嘱延时间隔（Will Delay Interval）到期或者会话（Session）结束时发布客户端的遗嘱消息（Will Message），取决于两者谁先发生。如果某个会话在遗嘱延时间隔到期之前创建了新的网络连接，则服务端**不能**发送遗嘱消息 [MQTT-3.1.3-9]。

**非规范评注**

遗嘱时间间隔的一个用途是避免在频繁的网络连接临时断开时发布遗嘱消息，因为客户端往往会很快重新连上网络并继续之前的会话。

**非规范评注**

如果某个连接到服务端的网络连接使用已存在的客户标识符，此已存在的网络连接的遗嘱消息将会被发布，除非新的网络连接设置了新开始（Clean Start）为0并且遗嘱延时大于0。如果遗嘱延时为0，遗嘱消息将在网络连接断开时发布。如果新开始为1，遗嘱消息也将被发布，因为此会话已结束。

##### 载荷格式指示

**1 (0x01)，**载荷格式指示（Payload Format Indicator）标识符。

跟随载荷格式指示（Payload Format Indicator ）之后的可能是：

* + 0 (0x00)，表示遗嘱消息（Will Message）是未指定的字节，等同于不发送载荷格式指示。
  + 1 (0x01)，表示遗嘱消息（Will Message）是UTF-8编码的字符数据。载荷中的UTF-8数据**必须**按照Unicode规范[[Unicode]](#Unicode) 和RFC 3629 [[RFC3629]](#RFC3629)中的申明进行编码。

包含多个载荷格式指示（Payload Format Indicator）将造成协议错误（Protocol Error）。服务端可以按照格式指示对遗嘱消息（Will Message）进行验证，如果验证失败发送一条包含原因码0x99（载荷格式无效）的CONNACK报文。如4.13节所述。

##### 消息过期间隔

**2 (0x02)，**消息过期间隔（Message Expiry Interval）标识符。

跟随其后的是表示消息过期间隔（Message Expiry Interval）的四字节整数。包含多个消息过期间隔将导致协议错误（Protocol Error）。

如果设定了消息过期间隔（Message Expiry Interval），四字节整数描述了遗嘱消息的生命周期（秒），并在服务端发布遗嘱消息时被当做发布过期间隔（Publication Expiry Interval）。

如果没有设定消息过期间隔，服务端发布遗嘱消息时将不发送消息过期间隔（Message Expiry Interval）。

##### 内容类型

**3 (0x03)**，内容类型（Content Type）标识符。

跟随其后的是一个以UTF-8格式编码的字符串，用来描述遗嘱消息（Will Message）的内容。包含多个内容类型（Content Type）将造成协议错误（Protocol Error）。内容类型的值由发送应用程序和接收应用程序确定。

##### 响应主题

**8 (0x08)**，响应主题（Response Topic）标识符。

跟随其后的是一个以UTF-8格式编码的字符串，用来表示响应消息的主题名（Topic Name）。包含多个响应主题（Response Topic）将造成协议错误。响应主题的存在将遗嘱消息（Will Message）标识为一个请求报文。

更多关于请求/响应的内容，参考[4.10](#_Request_/_Response)节。

##### 对比数据

**9 (0x09)，**对比数据（Correlation Data）标识符。

跟随其后的是二进制数据。对比数据被请求消息发送方在收到响应消息时用来标识相应的请求。包含多个对比数据将造成协议错误（Protocol Error）。如果没有设置对比数据，则请求方（Requester）不需要任何对比数据。

对比数据只对请求消息（Request Message）的发送方和响应消息（Response Message）的接收方有意义。

更多关于请求/响应的内容，参考[4.10](#_Request_/_Response)节。

##### 用户属性

**38 (0x26)**，用户属性（User Property）标识符。

跟随其后的是一个UTF-8字符串对。用户属性（User Property）可以出现多次，表示多个名字/值对。相同的名字可以出现多次。

服务端在发布遗嘱消息（Will Message）时**必须**维护用户属性（User Properties）的顺序 [MQTT-3.1.3-10]。

**非规范评注**

此属性旨在提供一种传递应用层名称-值标签的方法，其含义和解释仅由负责发送和接收它们的应用程序所有。

#### 遗嘱主题

如果遗嘱标志（Will Flag）被设置为1，遗嘱主题（Will Topic）为载荷中下一个字段。遗嘱主题（Will Topic）必须为UTF-8编码的字符串，如[1.5.4](#_UTF-8_Encoded_String)节 所定义[MQTT-3.1.3-11]。

#### 遗嘱载荷

如果遗嘱标志（Will Flag）被设置为1，遗嘱载荷（Will Payload）为载荷中下一个字段。遗嘱载荷定义了将要发布到遗嘱主题（Will Topic）的应用消息载荷，如[3.1.2.5](#_Toc479576982)节 所定义。此字段为二进制数据。

#### 用户名

如果用户名标志（User Name Flag）被设置为1，用户名（User Name）为载荷中下一个字段。用户名必须是 [1.5.4](#_UTF-8_Encoded_String)节 定义的UTF-8 编码字符串 [MQTT-3.1.3-12]。服务端可以将它用于身份验证和授权。

#### 密码

如果密码标志（Password Flag）被设置为1，密码（Password）为载荷中下一个字段。密码字段是二进制数据，尽管被称为密码，但可以被用来承载任何认证信息。

### CONNECT响应

注意：服务端**可以**在同一个 TCP 端口或其他网络端点上支持多种协议（包括MQTT协议的早期版本）。如果服务端确定协议是 MQTT v5.0，那么它按照下面的方法验证连接请求。

1. 网络连接建立后，如果服务端在合理的时间内没有收到 CONNECT 报文，服务端**应该**关闭这个连接。
2. 服务端**必须**按照[3.1](#_CONNECT_–_Connection)节 的要求验证CONNECT报文，如果报文不符合规范，服务端关闭网络连接 [MQTT-3.1.4-1]。服务端**可以**在关闭网络连接之前发送包含4.13节所述的0x80及以上原因码的CONNACK报文。
3. 服务端**可以**检查 CONNECT 报文的内容是不是满足任何进一步的限制，**应该**执行身份验证和授权检查。如果任何一项检查没通过，服务端**必须**关闭网络连接 [MQTT-3.1.4-2]。在关闭网络连接之前，服务端**可以**发送一个合适的包含如[3.2](#_CONNACK_–_Connect)节和[4.13](#S4_13_Errors)节所述的0x80及以上原因码的CONNACK报文。

如果验证成功，服务端会执行下列步骤。

1. 如果客户标识符（ClientID）所代表的客户端已经连接到此服务端，那么向原有的客户端发送一个包含原因码为0x8E（会话被接管）的DISCONNECT报文，并且必须关闭原有的网络连接 [MQTT-3.1.4-3]。如果原有客户端存在遗嘱消息（Will Message），遗嘱消息按照3.1.2.5节所描述的方式发布。

**非规范评注**

如果原有网络连接包含遗嘱消息，且遗嘱延时间隔为0，则遗嘱消息会在此网络连接被关闭时发送。如果原有网络连接会话过期间隔为0，或者新网络连接新开始标志设置为1且原有网络连接包含遗嘱消息，则遗嘱消息会被发送，因为原有会话已结束。

1. 服务端**必须**按照 [3.1.2.4](#_Clean_Start)节 所描述的方式对新开始标志进行处理[MQTT-3.1.4-4]。
2. 服务端**必须**使用包含原因码为0x00（成功）的CONNACK报文对客户端的CONNECT报文进行确认 [MQTT-3.1.4-5]。

**非规范评注**

如果服务端被用来处理商业关键数据，推荐对网络连接进行认证和授权。如果认证和授权成功，服务端可通过发送包含原因码为0x00（成功）的CONNACK报文进行响应，否则建议服务端根本不要发送CONNACK报文，因为这是一种潜在的对MQTT服务端的攻击，可以被用来进行拒绝服务攻击或密码猜测攻击。

1. 开始消息分发和保持连接状态监视。

允许客户端在发送CONNECT报文之后立即发送其它的控制报文；客户端不需要等待服务端的CONNACK报文。如果服务端拒绝了CONNECT报文，它**不能**处理客户端在CONNECT报文之后发送的任何除AUTH以外的报文 [MQTT-3.1.4-6]。

**非规范评注**

客户端通常会等待CONNACK报文。然而，如果在收到CONNACK报文之前就自由的发送其它控制报文将会简化客户端的实现，因为它不必监督连接的状态。如果连接被拒绝了，客户端在接收CONNACK报文之前发送的任何数据将不会被服务端所处理。

**非规范评注**

选择在收到CONNACK报文之前就发送控制报文的客户端将不知道服务端所存在的约束以及会话是否被使用。

**非规范评注**

服务端在对某个客户端完成认证之前，可以选择限制读取该客户端的网络数据或者关闭该客户端的网络连接。这是一种避免拒绝服务攻击的方法。

## CONNACK – 确认连接请求

CONNACK报文由服务端所发送，作为对来自客户端的CONNECT报文的响应。服务端在发送任何除AUTH以外的报文之前**必须**先发送包含原因码为0x00（成功）的CONNACK报文 [MQTT-3.2.0-1]。服务端在一次网络连接中**不能**发送多个CONNACK报文 [MQTT-3.2.0-2]。

如果客户端在合理的时间内没有收到服务端的CONNACK报文，客户端**应该**关闭网络连接。*合理* 的时间取决于应用的类型和通信基础设施。

### CONNACK固定报头

固定报头的格式见图 3-7 的描述。

图 3‑7 – CONNACK报文固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| byte 1 | MQTT控制报文类型 (2) | | | | Reserved 保留位 | | | |
|  | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| byte 2 | 剩余长度 | | | | | | | |

**剩余长度字段**

用可变字节整数来编码，表示可变报头的长度。

### CONNACK可变报头

CONNACK报文的可变报头按顺序包含以下字段：连接确认标志（Connect Acknowledge Flags），连接原因码（Reason Code），属性（Properties）。属性的编码规则如[2.2.2](#_Properties)节 所描述。

#### 连接确认标志

第1个字节是连接确认标志，位7-1是保留位且必须设置为0 [MQTT-3.2.2-1] 。

第0（SP）位是当前会话标志（Session Present Flag）。

#### 当前会话

**位置：**连接确认标志（Connect Acknowledge Flags）的第0位。

当前会话（Session Present）标志通知客户端，服务端是否正在使用此客户端标识符之前连接的会话状态（Session State）。当前会话标志使服务端和客户端在是否有已存储的会话状态上保持一致。

如果服务端接受一个新开始（Clean Start）为1的连接，服务端在CONNACK报文中除了把原因码设置为0x00（成功）之外，还**必须**把当前会话标志设置为0 [MQTT-3.2.2-2]。

如果服务端接受一个新开始（Clean Start）为0的连接，并且服务端已经保存了此客户标识符（ClientID）的会话状态（Session State），服务端在CONNACK报文中**必须**把当前会话标志设置为1。否则，服务端**必须**把当前会话标志设置为0。无论如何，服务端在CONNACK报文中**必须**把原因码设置为0x00（成功） [MQTT-3.2.2-3]。

如果客户端从服务端接收到的当前会话标志值与预期的不同，客户端做如下处理：

* 如果客户端没有保存的会话状态，但收到会话标志为1，客户端**必须**关闭网络连接 [MQTT-3.2.2-4]。 如果希望重新开始一个新的会话，客户端可以使用新开始（Clean Start）为1并重新连接服务端。
* 如果客户端保存了会话状态，但收到的会话标志为0，客户端若要继续此网络连接，它**必须**丢弃其保存的会话状态 [MQTT-3.2.2-5]。

如果服务端发送的CONNACK报文中原因码非0，它**必须**把当前会话标志设置为0 [MQTT-3.2.2-6]。

#### 连接原因码

可变报头中第2个字节是连接原因码（Reason Code）。

连接原因码（Reason Code）的值如下所示。如果服务端收到一个格式正确的CONNECT报文，但服务端无法完成连接的创建，服务端**可以**发送一个包含适当的连接原因码的CONNACK报文。如果服务端发送了一个包含原因码大于等于128的CONNACK报文，它随后**必须**关闭网络连接 [MQTT-3.2.2-7]。

表 3‑1 - 连接原因码

|  |  |  |  |
| --- | --- | --- | --- |
| **值** | **16进制** | **原因码名称** | **说明** |
| 0 | 0x00 | 成功 | 连接被接受。 |
| 128 | 0x80 | 未指明的错误 | 服务端不愿透露的错误，或者没有适用的原因码。 |
| 129 | 0x81 | 无效报文 | CONNECT报文内容不能被正确的解析。 |
| 130 | 0x82 | 协议错误 | CONNECT报文内容不符合本规范。 |
| 131 | 0x83 | 实现错误 | CONNECT有效，但不被服务端所接受。 |
| 132 | 0x84 | 协议版本不支持 | 服务端不支持客户端所请求的MQTT协议版本。 |
| 133 | 0x85 | 客户端标识符无效 | 客户端标识符有效，但未被服务端所接受。 |
| 134 | 0x86 | 用户名密码错误 | 客户端指定的用户名密码未被服务端所接受。 |
| 135 | 0x87 | 未授权 | 客户端未被授权连接。 |
| 136 | 0x88 | 服务端不可用 | MQTT服务端不可用。 |
| 137 | 0x89 | 服务端正忙 | 服务端正忙，请重试。 |
| 138 | 0x8A | 禁止 | 客户端被禁止，请联系服务端管理员。 |
| 140 | 0x8C | 无效的认证方法 | 认证方法未被支持，或者不匹配当前使用的认证方法。 |
| 144 | 0x90 | 主题名无效 | 遗嘱主题格式正确，但未被服务端所接受。 |
| 149 | 0x95 | 报文过长 | CONNECT报文超过最大允许长度。 |
| 151 | 0x97 | 超出配额 | 已超出实现限制或管理限制。 |
| 153 | 0x99 | 载荷格式无效 | 遗嘱载荷数据与载荷格式指示符不匹配。 |
| 154 | 0x9A | 不支持保留 | 遗嘱保留标志被设置为1，但服务端不支持保留消息。 |
| 155 | 0x9B | 不支持QoS | 服务端不支持遗嘱中设置的QoS。 |
| 156 | 0x9C | 使用其他服务端 | 客户端应该临时使用其他服务端。 |
| 157 | 0x9D | 服务端已移动 | 客户端应该永久使用其他服务端 |
| 159 | 0x9F | 超出连接速率限制 | 超出了所能接受的连接速率限制。 |

服务端发送的CONNACK报文必须使用一种原因码 [MQTT-3.2.2-8]。

**非规范评注**

原因码0x80（未指明的错误）可以被用作：服务器知道失败的原因但是并不希望透露给客户端，或者没有其他适用的原因码。

出于安全考虑，发现CONNECT出错时服务端可以选择不发送CONNACK报文而关闭网络连接。例如，在公网中向未被授权的网络连接告知自身MQTT服务端身份并不明智。

#### CONNACK属性

##### 属性长度

CONNACK报文可变报头中的属性长度，编码为可变字节整数。

##### 会话过期间隔

**17 (0x11)，**会话过期间隔（Session Expiry Interval）标识符。

跟随其后的是用四字节整数表示的以秒为单位的会话过期间隔（Session Expiry Interval）。包含多个会话过期间隔（Session Expiry Interval）将造成协议错误（Protocol Error）。

如果会话过期间隔（Session Expiry Interval）值未指定，则使用CONNECT报文中指定的会话过期时间间隔。服务端使用此属性通知客户端它使用的会话过期时间间隔与客户端在CONNECT中发送的值不同。更详细的关于会话过期时间的描述，请参考3.1.2.11.2节。

##### 接收最大值

**33 (0x21) ，**接收最大值（Receive Maximum）描述符。

跟随其后的是由双字节整数表示的最大接收值。包含多个接收最大值或接收最大值为0将造成协议错误（Protocol Error）。

服务端使用此值限制服务端愿意为该客户端同时处理的QoS为1和QoS为2的发布消息最大数量。没有机制可以限制客户端试图发送的QoS为0的发布消息。

如果没有设置最大接收值，将使用默认值65535。

关于接收最大值的详细使用，参考[4.9](#_Flow_Control)节 流控部分。

##### 最大服务质量

**36，**最大服务质量（Maximum QoS）标识符。

跟随其后的是用一个字节表示的0或1。包含多个最大服务质量（Maximum QoS）或最大服务质量既不为0也不为1将造成协议错误。如果没有设置最大服务质量，客户端可使用最大QoS为2。

如果服务端不支持Qos为1或2的PUBLISH报文，服务端**必须**在CONNACK报文中发送最大服务质量以指定其支持的最大QoS值。 [MQTT-3.2.2-9]。即使不支持QoS为1或2的PUBLISH报文，服务端也**必须**接受请求QoS为0、1或2的SUBSCRIBE报文 [MQTT-3.2.2-10]。

如果从服务端接收到了最大服务质量，则客户端**不能**发送超过最大服务质量所指定的QoS级别的PUBLISH报文。 [MQTT-3.2.2-11]。服务端接收到超过其指定的最大服务质量的PUBLISH报文将造成协议错误（Protocol Error）。这种情况下应使用包含原因码为0x9B（不支持的服务质量）的DISCONNECT报文进行处理，如[4.13](#S4_13_Errors)节 所述。

如果服务端收到包含遗嘱的QoS超过服务端处理能力的CONNECT报文，服务端**必须**拒绝此连接。服务端**必须**使用包含原因码为0x9B（不支持的服务质量）的CONNACK报文进行错误处理，随后**必须**关闭网络连接。[4.13](#S4_13_Errors)节 所述 [MQTT-3.2.2-12]。

**非规范评注**

客户端不必支持QoS为1和2的PUBLISH报文。客户端只需将其发送的任何SUBSCRIBE报文中的QoS字段限制在其支持的最大服务质量以内即可。

##### 保留可用

**37 (0x25)，**保留可用（Retain Available）标识符。

跟随其后的是一个单字节字段，用来声明服务端是否支持保留消息。值为0表示不支持保留消息，为1表示支持保留消息。如果没有设置保留可用字段，表示支持保留消息。包含多个保留可用字段或保留可用字段值不为0也不为1将造成协议错误（Protocol Error）。

如果服务端收到一个包含保留标志位1的遗嘱消息的CONNECT报文且服务端不支持保留消息，服务端**必须**拒绝此连接请求，且**必须**发送包含原因码为0x9A（不支持保留）的CONNACK报文，随后**必须**关闭网络连接 [MQTT-3.2.2-13]。

从服务端接收到的保留可用标志为0时，客户端**不能**发送保留标志设置为1的PUBLISH报文 [MQTT-3.2.2-14]。如果服务端收到这种PUBLISH报文，将造成协议错误（Protocol Error），此时服务端必须发送包含原因码为0x9A（不支持保留）的DISCONNECT报文，如[4.13](#S4_13_Errors)节 所述。

##### 最大报文长度

**39 (0x27)，**最大报文长度（Maximum Packet Size）标识符。

跟随其后的是由四字节整数表示的服务端愿意接收的最大报文长度（Maximum Packet Size）。如果没有设置最大报文长度，则按照协议由固定报头中的剩余长度可编码最大值和协议报头对数据包的大小做限制。

包含多个最大报文长度（Maximum Packet Size），或最大报文长度为0将造成协议错误（Protocol Error）。

如[2.1.4](#_Dup)节 所述，最大报文长度是控制报文的总长度。服务端使用最大报文长度通知客户端其所能处理的单个报文长度限制。

客户端不能发送超过最大报文长度（Maximum Packet Size）的报文给服务端 [MQTT-3.2.2-15]。收到长度超过限制的报文将导致协议错误，此时服务端应该发送包含原因码0x95（报文过长）的DISCONNECT报文给客户端，详见[4.13](#S4_13_Errors)节。

##### 分配客户端标识符

**18 (0x12)，**分配客户端标识符（Assigned Client Identifier）标识符。

跟随其后的是UTF-8编码的分配客户端标识符（Assigned Client Identifier）字符串。包含多个分配客户端标识符将造成协议错误（Protocol Error）。

服务端分配客户端标识符的原因是CONNECT报文中的客户端标识符长度为0。

如果客户端使用长度为0的客户端标识符（ClientID），服务端必须回复包含分配客户端标识符（Assigned Client Identifier）的CONNACK报文。分配客户端标识符必须是没有被服务端的其他会话所使用的新客户端标识符 [MQTT-3.2.2-16]。

##### 主题别名最大值

**34 (0x22)，**主题别名最大值（Topic Alias Maximum）标识符。

跟随其后的是用双字节整数表示的主题别名最大值（Topic Alias Maximum）。包含多个主题别名最大值（Topic Alias Maximum）将造成协议错误（Protocol Error）。没有设置主题别名最大值属性的情况下，主题别名最大值默认为零。

此值指示了服务端能够接收的来自客户端的主题别名（Topic Alias）最大值。服务端使用此值来限制本次连接可以拥有的主题别名的值。客户端在一个PUBLISH报文中发送的主题别名值**不能**超过服务端设置的主题别名最大值（Topic Alias Maximum） [MQTT-3.2.2-17]。值为0表示本次连接服务端不接受任何主题别名（Topic Alias）。如果主题别名最大值（Topic Alias）没有设置，或者设置为0，则客户端**不能**向此服务端发送任何主题别名（Topic Alias）。 [MQTT-3.2.2-18]。

##### 原因字符串

**31 (0x1F)，**原因字符串（Reason String）标识符。

跟随其后的是UTF-8编码的字符串，表示此次响应相关的原因。此原因字符串（Reason String）是为诊断而设计的可读字符串，**不能**被客户端所解析。

服务端使用此值向客户端提供附加信息。如果加上原因字符串之后的CONNACK报文长度超出了客户端指定的最大报文长度，则服务端**不能**发送此原因字符串 [MQTT-3.2.2-19]。包含多个原因字符串将造成协议错误（Protocol Error）。

**非规范评注**

客户端对原因字符串的恰当使用包括：抛出异常时使用此字符串，或者将此字符串写入日志。

##### 用户属性

**38 (0x26)，**用户属性（User Property）标识符。

跟随其后的是一个UTF-8字符串对。此属性可用于向客户端提供包括诊断信息在内的附加信息。如果加上原因字符串之后的CONNACK报文长度超出了客户端指定的最大报文长度，则服务端**不能**发送此原因字符串 [MQTT-3.2.2-20]。用户属性（User Property）允许出现多次，以表示多个名字/值对，且相同的名字可以多次出现。

用户属性的内容和意义本规范不做定义。CONNACK报文的接收方可以选择忽略此属性。

##### 通配符订阅可用

**40 (0x28)，**通配符订阅可用（Wildcard Subscription Available）标识符。

跟随其后的是一个单字节字段，用来声明服务器是否支持通配符订阅（Wildcard Subscriptions）。值为0表示不支持通配符订阅，值为1表示支持通配符订阅。如果没有设置此值，则表示支持通配符订阅。包含多个通配符订阅可用属性，或通配符订阅可用属性值不为0也不为1将造成协议错误（Protocol Error）。

如果服务端在不支持通配符订阅（Wildcard Subscription）的情况下收到了包含通配符订阅的SUBSCRIBE报文，将造成协议错误（Protocol Error）。此时服务端将发送包含原因码为0xA2（通配符订阅不支持）的DISCONNECT报文，如[4.13](#S4_13_Errors)节所述。

服务端在支持通配符订阅的情况下仍然可以拒绝特定的包含通配符订阅的订阅请求。这种情况下，服务端**可以**发送一个包含原因码为0xA2（通配符订阅不支持）的SUBACK报文。

##### 订阅标识符可用

**41 (0x29)，**订阅标识符可用（Subscription Identifier Available）标识符。

跟随其后的是一个单字节字段，用来声明服务端是否支持订阅标识符（Subscription Identifiers）。值为0表示不支持订阅标识符，值为1表示支持订阅标识符。如果没有设置此值，则表示支持订阅标识符。包含多个订阅标识符可用属性，或订阅标识符可用属性值不为0也不为1将造成协议错误（Protocol Error）。

如果服务端在不支持订阅标识符（Subscription Identifier）的情况下收到了包含订阅标识符的SUBSCRIBE报文，将造成协议错误（Protocol Error）。此时服务端将发送包含原因码为0xA1（订阅标识符不支持）的DISCONNECT报文，如[4.13](#S4_13_Errors)节 所述。

##### 共享订阅可用

**42 (0x2A)，**共享订阅可用（Shared Subscription Available）标识符。

跟随其后的是一个单字节字段，用来声明服务端是否支持共享订阅（Shared Subscription）。值为0表示不支持共享订阅，值为1表示支持共享订阅。如果没有设置此值，则表示支持共享订阅。包含多个共享订阅可用（Shared Subscription Available），或共享订阅可用属性值不为0也不为1将造成协议错误（Protocol Error）。

如果服务端在不支持共享订阅（Shared Subscription）的情况下收到了包含共享订阅的SUBSCRIBE报文，将造成协议错误（Protocol Error）。此时服务端将发送包含原因码为0x9E（共享订阅不支持）的DISCONNECT报文，如[4.13](#S4_13_Errors)节 所述。

##### 服务端保持连接

**19 (0x13)，**服务端保持连接（Server Keep Alive）标识符。

跟随其后的是由服务端分配的双字节整数表示的保持连接（Keep Alive）时间。如果服务端发送了服务端保持连接（Server Keep Alive）属性，客户端必须使用此值代替其在CONNECT报文中发送的保持连接时间值 [MQTT-3.2.2-21]。如果服务端没有发送服务端保持连接属性，服务端必须使用客户端在CONNECT报文中设置的保持连接时间值 [MQTT-3.2.2-22]。包含多个服务端保持连接属性将造成协议错误（Protocol Error）。

**非规范评注**

服务端保持连接属性的主要作用是通知客户端它将会比客户端指定的保持连接更快的断开非活动的客户端。

##### 响应信息

**26 (0x1A)，**响应信息（Response Information）标识符。

跟随其后的是一个以UTF-8编码的字符串，作为创建响应主题（Response Topic）的基本信息。关于客户端如何根据响应信息（Response Information）创建响应主题不在本规范的定义范围内。包含多个响应信息将造成协议错误（Protocol Error）。

如果客户端发送的请求响应信息（Request Response Information）值为1，则服务端在CONNACK报文中发送响应信息（Response Information）为**可选**项。

**非规范评注**

响应信息通常被用来传递主题订阅树的一个全局唯一分支，此分支至少在该客户端的会话生命周期内为该客户端所保留。请求客户端和响应客户端的授权需要使用它，所以它通常不能仅仅是一个随机字符串。一般把此分支作为特定客户端的订阅树根节点。通常此信息需要正确配置，以使得服务器能返回信息。使用此机制时，具体的信息一般由服务端来进行统一配置，而非由各个客户端自己配置。

更多关于请求/响应的信息，请参考[4.10](#_Request_/_Response)节 。

##### 服务端参考

**28 (0x1C)，**服务端参考（Server Reference）标识符。

跟随其后的是一个以UTF-8编码的字符串，可以被客户端用来标识其他可用的服务端。包含多个服务端参考（Server Reference）将造成协议错误（Protocol Error）。

服务端在包含了原因码为0x9C（使用其他服务端）或0x9D（服务端移动）的CONNACK报文或DISCONNECT报文中设置服务端参考，如[4.13](#S4_13_Errors)节 所述。

关于如何使用服务端参考，请参考[4.11](#_Server_redirection)节 服务端重定向信息。

##### 认证方法

**21 (0x15)，认证方法**（Authentication Method）标识符。

跟随其后的是一个以UTF-8编码的字符串，包含了认证方法（Authentication Method）名。包含多个认证方法将造成协议错误（Protocol Error）。更多关于扩展认证的信息，请参考[4.12](#_Enhanced_authentication)节 。

##### 认证数据

**22 (0x16) Byte，**认证数据（Authentication Data）标识符。

跟随其后的是包含认证数据（Authentication Data）的二进制数据。此数据的内容由认证方法和已交换的认证数据状态定义。包含多个认证数据将造成协议错误（Protocol Error）。更多关于扩展认证的信息，请参考[4.12](#_Enhanced_authentication)节 。

### CONNACK载荷

CONNACK报文不包含有效载荷。

## PUBLISH – 发布消息

PUBLISH控制报文是指从客户端向服务端或者服务端向客户端传输一个应用消息。

### PUBLISH固定报头

图 3‑8 – PUBLISH报文固定报头

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Bit** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| byte 1 | MQTT控制报文类型 (3) | | | | DUP | QoS等级 | | RETAIN |
|  | 0 | 0 | 1 | 1 | X | X | X | X |
| byte 2… | 剩余长度 | | | | | | | |

#### 重发标志

**位置：**第1个字节，第3位

如果DUP标志被设置为0，表示这是客户端或服务端第一次请求发送这个 PUBLISH报文。如果DUP标志

被设置为1，表示这可能是一个早前报文请求的重发。

客户端或服务端请求重发一个PUBLISH报文时，必须将DUP标志设置为1 [MQTT-3.3.1-1]。对于QoS为0的消息，DUP标志**必须**设置为0 [MQTT-3.3.1-2]。

服务端发送PUBLISH报文给订阅者时，收到（入站）的PUBLISH报文的DUP标志的值不会被传播。发

送（出站）的PUBLISH报文与收到（入站）的PUBLISH报文中的DUP标志是独立设置的，它的值**必须**

单独的根据发送（出站）的PUBLISH报文是否是一个重发来确定 [MQTT-3.3.1-3]。

**非规范评注**

接收者收到一个DUP标志位1的控制报文时，不能假设它看到了一个这个报文之前的一个副本。

**非规范评注**

需要特别指出的是，DUP标志关注的是控制报文本身，与它包含的应用消息无关。当使用QoS 1时，客户端可能会收到一个DUP标志为0的PUBLISH 报文，这个报文包含一个它之前收到过的应用消息的副本，但是用的是不同的报文标识符。 [2.2.1](#_Toc358219870)节 提供了有关报文标识符的更多信息。

#### 服务质量等级

**位置：**第1个字节，第2-1位。

这个字段表示应用消息分发的服务质量等级保证。服务质量等级在下表中列出。

表 3‑2 - 服务质量定义

|  |  |  |  |
| --- | --- | --- | --- |
| **QoS值** | **Bit 2** | **Bit 1** | **说明** |
| 0 | 0 | 0 | 最多分发一次 |
| 1 | 0 | 1 | 至少分发一次 |
| 2 | 1 | 0 | 仅分发一次 |
| - | 1 | 1 | 保留 – 不能使用 |

如果服务端在对客户端响应的CONNACK报文中包含了最大服务质量（Maximum QoS）且服务端收到的PUBLISH报文的QoS大于此最大服务质量，服务端发送包含原因码为0x9B（不支持的服务质量）的DISCONNECT报文，如[4.13](#S4_13_Errors)节 所述。

PUBLISH报文的2个QoS比特位不能同时设置为1 [MQTT-3.3.1-4]。如果服务端或客户端收到QoS 2个比特位都为1的无效PUBLISH报文，使用包含原因码为0x81（无效报文）的DISCONNECT报文关闭网络连接，如[4.13](#S4_13_Errors)节 所述。

#### 保留标志

**位置：**第1个字节，第0位。

如果客户端发给服务端的PUBLISH报文的保留（Retain）标志被设置为1，服务端**必须**存储此应用消息，并用其替换此话题下任何已存在的消息 [MQTT-3.3.1-5]，以便它可以被分发给未来的匹配此主题名（Topic Name）的订阅者。如果载荷为空，消息可以正常被服务端所处理，但是此话题下的任何保留消息**必须**被丢弃，并且此话题未来的订阅者将不会收到保留消息 [MQTT-3.3.1-6]。载荷为空的保留消息将**不能**被存储在服务端 [MQTT-3.3.1-7]。

如果客户端发给服务端的PUBLISH报文的保留标志位为0，服务器**不能**把此消息存储为保留消息，也不能丢弃或替换任何已存在的保留消息 [MQTT-3.3.1-8]。

如果服务端发送给客户端的CONNACK报文中包含保留可用属性，且属性值为0，但收到的PUBLISH报文中保留标志位为1，服务端使用包含原因码为0x9A（保留不支持）的DISCONNECT报文断开网络连接，如[4.13](#S4_13_Errors)节 所述。

当一个新的非共享订阅（Non-shared Subscription）被创建时，每个匹配的话题下的最新保留消息如果存在，将根据保留消息订阅选项（Retain Handling Subscription Option）发送给客户端。这些消息在发送时保留标志被设置为1。保留消息的发送由保留消息处理订阅选项控制，收到订阅时：

* 如果保留消息处理属性被设置为0，服务端**必须**发送主题与客户端订阅的主题过滤器（Topic Filter）相匹配的所有保留消息 [MQTT-3.3.1-9]。
* 如果保留消息处理属性被设置为1，如果尚不存在匹配的订阅，服务端**必须**发送主题与客户端订阅的主题过滤器相匹配的所有保留消息。如果已存在相匹配的订阅，服务器**不能**发送这些保留消息 [MQTT-3.3.1-10]。
* 如果保留消息处理属性被设置为2，服务器**不能**发送这些保留消息 [MQTT-3.3.1-11]。

订阅选项（Subscription Options）的定义，参考[3.8.3.1](#_Subscription_Options)节 。

如果服务端收到保留标志设置为1且QoS设置为0的PUBLISH报文，服务端必须把此QoS为0的消息存储为其主题下最新的保留消息，但服务端可以选择在任何时间丢弃此消息。如果发生丢弃，该主题下将不存在任何保留消息。

如果某个主题当前的保留消息过期，该主题下将不存在任何保留消息。

服务端转发应用消息时，保留标志位的设置由保留即发布（Retain As Published）订阅选项决定。订阅选项的定义，请参考[3.8.3.1](#_Subscription_Options)节 。

* 如果保留即发布（Retain As Published）订阅选项被设置为0，服务端在转发应用消息时**必须**将保留标志设置为0，而不管收到的PUBLISH报文中保留标志位如何设置的。 [MQTT-3.3.1-12]。
* 如果保留即发布（Retain As Published）订阅选项被设置为1，服务端在转发应用消息时**必须**将保留标志设置为与收到的PUBLISH消息中的保留标志位相同 [MQTT-3.3.1-13]。

**非规范评注**

对于发布者不定期发送状态消息这个场景，保留消息很有用。新的非共享订阅者将会收到最近的状态。

#### 剩余长度

等于可变报头的长度加上有效载荷的长度，被编码为可变字节整数。

### PUBLISH可变报头

PUBLISH报文可变报头按顺序包含：主题名（Topic Name），报文标识符（Packet Identifier），属性（Properties）。属性的编码规则如[2.2.2](#_Properties)节 所述。

#### 主题名

主题名（Topic Name）用于识别有效载荷数据应该被发布到哪一个信息通道。

主题名**必须**是PUBLISH报文可变报头的第一个字段。它**必须**是 [1.5.4](#_UTF-8_Encoded_String)节 定义的UTF-8编码的字符串 [MQTT-3.3.2-1]。

PUBLISH报文中的主题名**不能**包含通配符 [MQTT-3.3.2-2]。

服务端发送给订阅客户端的PUBLISH报文中的主题名**必须**匹配该订阅的主题过滤器（Topic Filter）， 如 [4.7](#_Topic_Names_and)节 所定义的匹配过程 [MQTT-3.3.2-3]。然而，由于服务端允许将主题名映射为其他名字，主题名可能与原始PUBLISH报文中的主题名不同。

发送方可以使用主题别名（Topic Alias）以便减少PUBLISH报文的长度。主题别名如[3.3.2.3.4](#_Topic_Alias)节 所述。主题名长度为0且没有主题别名，将造成协议错误（Protocol Error）。

#### 报文标识符

只有当QoS等级是1或2时，报文标识符（Packet Identifier）字段才能出现在PUBLISH报文中。[2.2.1](#_Toc358219870)节 提供了有关报文标识符的更多信息。

#### PUBLISH属性

##### 属性长度

PUBLISH报文可变报头中的属性长度被编码为可变字节整数。

##### 载荷格式指示

**1 (0x01)，**载荷格式指示（Payload Format Indicator）标识符。

跟随其后的是单字节的载荷格式指示值，可以是：

* + 0 (0x00)，说明载荷是未指定格式的字节，相当于没有发送载荷格式指示。
  + 1 (0x01)，说明载荷是UTF-8编码的字符数据。载荷中的UTF-8数据必须是按照Unicode [[Unicode](#Unicode)] 的规范和RFC 3629 [[RFC3629]](#RFC3629) 的重申进行编码。

服务端发送给订阅者的载荷格式指示**必须**与接收到的应用消息（Application Message）一致 [MQTT-3.3.2-4]。接收者**可以**验证载荷数据与所指示的格式一致，如果不一致，发送包含原因码为0x99（载荷格式无效）的PUBACK，PUBREC或DISCONNECT报文，如[4.13](#S4_13_Errors)节 所述。

##### 消息过期间隔

**2 (0x02)，**消息过期间隔（Message Expiry Interval）标识符。

跟随其后的是四字节整数表示的消息过期间隔（Message Expiry Interval）。

如果消息过期间隔存在，四字节整数表示以秒为单位的应用消息（Application Message）生命周期。如果消息过期间隔（Message Expiry Interval）已过期，服务端还没开始向匹配的订阅者交付该消息，服务端**必须**删除该订阅者的消息副本 [MQTT-3.3.2-5]。

如果消息过期间隔不存在，应用消息不会过期。

服务端在将PUBLISH报文发送给客户端时所包含的消息过期间隔（Message Expiry Interval）**必须**被设置为减去应用消息在服务端的等待时间 [MQTT-3.3.2-6]。关于状态存储的细节和限制，参考[4.1](#_Session_State)节。

##### 主题别名

**35 (0x23)，**主题别名（Topic Alias）标识符。

跟随其后的是表示主题别名（Topic Alias）值的双字节整数。包含多个主题别名值将造成协议错误（Protocol Error）。

主题别名是一个整数，用来代替主题名对主题进行识别。主题别名可以减小PUBLISH报文的长度，这对某个网络连接中发送的很长且反复使用的主题名来说很有用。

发送方决定是否使用主题别名及别名值如何选取。发送方通过在PUBLISH报文中包含的非0长度主题名和主题别名来设置主题别名映射。接收方正常处理该PUBLISH报文，但同样将指定的主题别名映射到主题名。

如果接收方已经设置了某个主题别名映射，发送方可以发送包含主题别名和长度为0的主题名的PUBLISH报文。接收方把此PUBLISH报文的主题名当做其包含的主题别名所映射的主题名。

发送方可以通过在同一个网络连接中发送另一个包含同样主题别名和不同非0长度主题名的PUBLISH报文来修改主题别名映射关系。

主题别名映射仅作用于某个网络连接及其生命周期内。接收方**不能**将任何主题别名映射从一个网络连接转发到另一个网络连接 [MQTT-3.3.2-7]。

主题别名不允许为0。发送方**不能**发送包含主题别名值为0的PUBLISH报文 [MQTT-3.3.2-8]。

客户端**不能**发送主题别名值大于服务端的CONNACK报文中指定的主题别名最大值（Topic Alias Maximum）的PUBLISH报文 [MQTT-3.3.2-9]。客户端**必须**接受所有值大于0且小于等于其发送的CONNECT报文中的主题别名最大值的主题别名 [MQTT-3.3.2-10]。

服务端**不能**发送包含主题别名值大于客户端在CONNECT报文中指定的主题别名最大值（Topic Alias Maximum）的PUBLISH报文 [MQTT-3.3.2-11]。服务端**必须**接受所有值大于0且小于等于其发送的CONNACK报文中的主题别名最大值的主题别名 [MQTT-3.3.2-12]。

客户端和服务端使用的主题别名映射相互独立。因此一般来说，客户端发送给服务端的主题别名值为1的PUBLISH报文和服务端发送给客户端的主题别名值为1的PUBLISH报文，将被映射到不同的主题。

##### 响应主题

**8 (0x08)，**响应主题（Response Topic）标识符。

跟随其后的是一个UTF-8编码的字符串，用作响应消息的主题名。响应主题**必须**是按照[1.5.4](#_UTF-8_Encoded_String)节 所定义的UTF-8字符串 [MQTT-3.3.2-13]。响应主题不能包含通配符 [MQTT-3.3.2-14]。包含多个响应主题将造成协议错误（Protocol Error）。响应主题的存在将消息标识为请求报文。

更多关于请求/响应的信息，参考[4.10](#_Request_/_Response)节 。

服务端在收到应用消息时**必须**将响应主题原封不动的发送给所有的订阅者 [MQTT-3.3.2-15]。

**非规范评注：**

包含响应主题的应用消息接收方使用响应主题作为主题名，发送作为响应消息的PUBLISH报文。如果请求消息中包含对比数据，接收方应当在发送作为对此请求消息进行响应的PUBLISH报文中包含此对比数据。

##### 对比数据

**9 (0x09)，**对比数据（Correlation Data）标识符。

跟随其后的是二进制数据。对比数据被请求消息发送方在收到响应消息时用来标识相应的请求。包含多个对比数据将造成协议错误（Protocol Error）。如果没有设置对比数据，则请求方（Requester）不需要任何对比数据。

服务端在收到应用消息时**必须**原封不动的把对比数据发送给所有的订阅者 [MQTT-3.3.2-16]。对比数据只对请求消息（Request Message）的发送方和响应消息（Response Message）的接收方有意义。

**非规范评注**

接收方收到包含响应主题和对比数据的应用消息时，发送以响应主题为主题名的PUBLISH报文作为响应消息。客户端在响应消息中应将对比数据作为PUBLISH报文的一部分原封不动的发送出去。

**非规范评注**

如果对客户端响应消息中的对比数据所做的任何更改会造成应用程序错误，则应当对对比数据进行加密/哈希，以便接收方能检测到对比数据是否被更改。

更多关于请求/响应的信息，请参考[4.10](#_Request_/_Response)节。

##### 用户属性

**38 (0x26)，**用户属性（User Property）。

跟随其后的是一个UTF-8字符串对。用户属性（User Property）允许出现多次，以表示多个名字/值对，且相同的名字可以多次出现。

服务端在转发应用消息到客户端时**必须**原封不动的把所有的用户属性放在PUBLISH报文中 [MQTT-3.3.2-17]。服务端在转发应用消息时**必须**保持所有用户属性的先后顺序 [MQTT-3.3.2-18]。

**非规范评注**

此属性旨在提供一种传递应用层名称-值标签的方法，其含义和解释仅由负责发送和接收它们的应用程序所有。

##### 订阅标识符

**11 (0x0B)**，订阅标识符（Subscription Identifier）标识符。

跟随其后的是一个可变字节整数表示的订阅标识符。

订阅标识符值的范围从1到268,435,455。订阅标识符的值为0将造成协议错误。如果某条发布消息匹配了多个订阅，则将包含多个订阅标识符。这种情况下他们的顺序并不重要。

##### 内容类型

**3 (0x03)，** 内容类型（Content Type）标识符。

跟随其后的是一个以UTF-8格式编码的字符串，用来描述应用消息的内容。内容类型**必须**是UTF-8编码的字符串， 如[section 1.5.4](#_UTF-8_Encoded_String)节 所定义[MQTT-3.3.2-19]。

包含多个内容类型将造成协议错误（Protocol Error）。内容类型的值由发送应用程序和接收应用程序确定。

服务端**必须**把收到的应用消息中的内容类型原封不动的发送给所有的订阅者 [MQTT-3.3.2-20]。

**非规范评注**

UTF-8编码字符串可以使用一个MIME内容类型字符串来描述应用消息的内容。由于发送程序和接收程序负责内容类型字符串的定义和解释，因此MQTT服务端只确保内容类型是有效的UTF-8编码的字符串，不会做其他方面的验证。

**非规范评注**

图 3-9 是一个PUBLISH示例报文，其中主题名为*a/b*，报文标识符为10，没有属性。

图 3‑9 - PUBLISH报文可变报头非规范示例

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **说明** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| 主题名 | | | | | | | | | |
| byte 1 | 长度MSB (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 2 | 长度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 |
| 报文标识符 | | | | | | | | | |
| byte 6 | 报文标识符MSB (0) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| byte 7 | 报文标识符LSB (10) | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 属性长度 | | | | | | | | | |
| byte 8 | 无属性 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### PUBLISH载荷

载荷包含将被发布的应用消息。载荷的内容和格式由应用程序指定。有效载荷的长度这样计算：用固定

报头中的剩余长度字段的值减去可变报头的长度。包含零长度有效载荷的PUBLISH报文是合法的。

### PUBLISH响应

PUBLISH报文的接收方**必须**按照PUBLISH报文中的QoS等级发送响应报文 [MQTT-3.3.4-1]。

表 3‑3 PUBLISH报文的预期响应

|  |  |
| --- | --- |
| **服务质量等级** | **预期响应** |
| QoS 0 | 无响应 |
| QoS 1 | PUBACK报文 |
| QoS 2 | PUBREC报文 |

客户端使用PUBLISH报文发送应用消息给服务端，目的是分发到其他订阅匹配的客户端。

服务端使用PUBLISH报文发送应用消息给每一个订阅匹配的客户端。PUBLISH报文包含SUBSCRIBE报文中承载的订阅标识符--如果存在的话。

客户端使用带通配符的主题过滤器请求订阅时，客户端的订阅可能会重叠，因此发布的消息可能会匹配多个主题过滤器。这种情况下，服务端**必须**按照所有匹配的订阅中最大的QoS等级把消息发送给客户端 [MQTT-3.3.4-2]。此外，服务端可以为每一个匹配的订阅按照订阅时的QoS等级，把消息副本分发给客户端。

如果客户端收到一个未经请求的应用消息（没有匹配任何订阅），且QoS大于客户端指定的最大服务质量（Maximum QoS），客户端使用包含原因码为0x9B（QoS不支持）的DISCONNECT报文断开连接，如[4.13](#S4_13_Errors)节 所述。

如果客户端在这些重叠的订阅中指定了订阅标识符，服务端在发布这些订阅相匹配的消息时**必须**包含这些订阅标识符 [MQTT-3.3.4-3]。如果服务端对这些重叠的订阅只发送一条相匹配的消息，服务端**必须**在PUBLISH报文中包含所有的相匹配的订阅标识符（如果存在），但没有顺序要求 [MQTT-3.3.4-4]。如果服务端对这些重叠的订阅分别发送相匹配的消息，则每个PUBLISH报文中必须包含与订阅相匹配的订阅标识符（如果存在） [MQTT-3.3.4-5]。

可能存在客户端对同一个发布消息做了多次订阅，并且这些订阅中有多个订阅使用了相同的订阅标识符，这种情况下PUBLISH报文将携带多个相同的订阅标识符。

PUBLISH报文中若包含服务端收到的SUBSCRIBE报文以外的订阅标识符，将造成协议错误（Protocol Error）。从客户端发送给服务端的PUBLISH报文**不能**包含订阅标识符 [MQTT-3.3.4-6]。

对于共享订阅，发送给某个客户端的PUBLISH报文中将只包含该客户端的SUBSCRIBE报文中发送的订阅标识符。

收到PUBLISH报文时，接收方的行为取决于报文的QoS等级，如 [4.3节](#_Quality_of_Service) 所述。

如果PUBLISH报文包含主题别名，接收方按照以下方式进行处理：

1. 主题别名为0或大于最大主题别名（Maximum Topic Alias），将造成协议错误（Protocol Error），接收方使用包含原因码为0x94（主题别名无效）的DISCONNECT报文断开网络连接，如[4.13节](#S4_13_Errors) 所述。

1. 如果接收方已创建此主题别名的映射，
   1. 如果报文包含的主题名长度为0，接收方使用主题别名对应的主题名处理此报文
   2. 如果报文包含的主题名长度不为0，接收方使用此主题名处理此报文，并更新此主题别名映射到此主题名
2. 如果接收方还没有创建此主题别名的映射，
   1. 如果报文包含的主题名长度为0，将造成协议错误，接收方使用包含原因码为0x82（协议错误）的DISCONNECT报文断开网络连接，如[4.13](#S4_13_Errors) 节所述。
   2. 如果报文包含的主题名长度不为0，接收方使用此主题名处理此报文，并为此报文中的主题别名和主题名创建映射关系

**非规范评注**

如果服务端向客户端分发应用消息时使用了不同的协议级别（比如MQTT v3.1.1）--不支持属性或本规范提供的其他功能，应用消息中的某些信息将丢失，依赖于这些信息的应用程序可能无法正常工作。

客户端在收到服务端的PUBACK，PUBCOMP或包含原因码大于128的PUBREC报文之前，**不能**发送数量超过服务端的接收最大值（Receive Maximum）的QoS为1和2的PUBLISH报文 [MQTT-3.3.4-7]。客户端在发送PUBACK或PUBCOMP响应之前，如果收到数量超过客户端的接收最大值的QoS为1和2的PUBLISH报文，客户端使用包含原因码为0x93（超出接收最大值）的DISCONNECT报文断开网络连接， 如[4.13](#S4_13_Errors)节 所述。更多关于流量控制的信息，参考[4.9](#_Flow_Control)节。

客户端**不能**延迟发送任何报文，除了PUBLISH报文--如果已发送且没有收到确认的PUBLISH报文数量已达到服务端的接收最大值（Receive Maximum） [MQTT-3.3.4-8]。接收最大值只应用于当前网络连接。

**非规范评注**

客户端可以选择发送少于服务端接收最大值的未经确认的PUBLISH报文，尽管它可以发送更多数量的消息。

**非规范评注**

客户端可以选择暂停发送QoS为0的PUBLISH报文，当客户端暂停发送QoS为1和2的PUBLISH报文时。

**非规范评注**

如果客户端在收到CONNACK之前发送QoS为1或QoS为2的PUBLISH报文，客户端有可能被服务器断开连接，因为它发送了超过服务端接收最大值数量的发布报文。

服务端The Server MUST NOT send more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets for which it has not received PUBACK, PUBCOMP, or PUBREC with a Reason Code of 128 or greater from the Client [MQTT-3.3.4-9]. If it receives more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets where it has not sent a PUBACK or PUBCOMP in response, the Client uses DISCONNECT with Reason Code 0x93 (Receive Maximum exceeded) as described in [section 4.13](#S4_13_Errors) Handling errors. Refer to [section 4.9](#_Flow_Control) for more information about flow control.

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

**Non-normative comment**

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

**Non-normative comment**

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

## PUBACK – Publish acknowledgement

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

### PUBACK Fixed Header

Figure 3‑10 - PUBACK packet Fixed Header

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

**Remaining Length field**

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

### PUBACK Variable Header

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

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 Reason Code | | | | | | | |
| byte 4 | Property Length | | | | | | | |

#### PUBACK Reason Code

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

Table ‑4 - PUBACK Reason Codes

|  |  |  |  |
| --- | --- | --- | --- |
| **Value** | **Hex** | **Reason Code name** | **Description** |
| 0 | 0x00 | Success | The message is accepted. Publication of the QoS 1 message proceeds. |
| 16 | 0x10 | No matching subscribers. | The message is accepted but there are no subscribers. This is sent only by the Server. If the Server knows that there are no matching subscribers, it MAY use this Reason Code instead of 0x00 (Success). |
| 128 | 0x80 | Unspecified error | The receiver does not accept the publish but either does not want to reveal the reason, or it does not match one of the other values. |
| 131 | 0x83 | Implementation specific error | The PUBLISH is valid but the receiver is not willing to accept it. |
| 135 | 0x87 | Not authorized | The PUBLISH is not authorized. |
| 144 | 0x90 | Topic Name invalid | The Topic Name is not malformed, but is not accepted by this Client or Server. |
| 145 | 0x91 | Packet identifier in use | The Packet Identifier is already in use. This might indicate a mismatch in the Session State between the Client and Server. |
| 151 | 0x97 | Quota exceeded | An implementation or administrative imposed limit has been exceeded. |
| 153 | 0x99 | Payload format invalid | The payload format does not match the specified Payload Format Indicator. |

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

#### PUBACK Properties

##### Property Length

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

##### Reason String

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

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

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

##### User Property

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

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

### PUBACK Payload

The PUBACK packet has no Payload.

### PUBACK Actions

This is described in [section 4.3.2](#_QoS_1:_At).

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

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

### PUBREC Fixed Header

Figure 3‑12 - PUBREC packet Fixed Header

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

**Remaining Length field**

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

### PUBREC Variable Header

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

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 | PUBREC Reason Code | | | | | | | |
| byte 4 | Property Length | | | | | | | |

#### PUBREC Reason Code

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

Table ‑5 – PUBREC Reason Codes

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

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

#### PUBREC Properties

##### Property Length

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

##### Reason String

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

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

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

##### User Property

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

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

### PUBREC Payload

The PUBREC packet has no Payload.

### PUBREC Actions

This is described in [section 4.3.3](#_QoS_2:_Exactly).

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

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

### PUBREL Fixed Header

Figure 3‑14 – PUBREL packet Fixed Header

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

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

**Remaining Length field**

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

### PUBREL Variable Header

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

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 Reason Code | | | | | | | |
| byte 4 | Property Length | | | | | | | |

#### PUBREL Reason Code

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

Table ‑6 - PUBREL Reason Codes

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

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

#### PUBREL Properties

##### Property Length

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

##### Reason String

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

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

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

##### User Property

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

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

### PUBREL Payload

The PUBREL packet has no Payload.

### PUBREL Actions

This is described in [section 4.3.3](#_QoS_2:_Exactly).

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

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

### PUBCOMP Fixed Header

Figure 3‑16 – PUBCOMP packet Fixed Header

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

**Remaining Length field**

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

### PUBCOMP Variable Header

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

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 Reason Code | | | | | | | |
| byte 4 | Property Length | | | | | | | |

#### PUBCOMP Reason Code

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

Table ‑7 – PUBCOMP Reason Codes

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

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

#### PUBCOMP Properties

##### Property Length

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

##### Reason String

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

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

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

##### User Property

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

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

### PUBCOMP Payload

The PUBCOMP packet has no Payload.

### PUBCOMP Actions

This is described in [section 4.3.3](#_QoS_2:_Exactly).

## SUBSCRIBE - Subscribe request

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

### SUBSCRIBE Fixed Header

Figure 3‑18 SUBSCRIBE packet Fixed Header

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

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

**Remaining Length field**

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

### SUBSCRIBE Variable Header

The Variable Header of the SUBSCRIBE Packet contains the following fields in the order: Packet Identifier, and Properties. [Section 2.2.1](#_Toc358219870) provides more information about Packet Identifiers. The rules for encoding Properties are described in [section 2.2.2](#_Properties).

**Non-normative example**

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

Figure ‑ – SUBSCRIBE Variable Header example

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

#### SUBSCRIBE Properties

##### Property Length

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

##### Subscription Identifier

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

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

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

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

##### User Property

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

Followed by a UTF-8 String Pair.

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

**Non-normative comment**

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

### SUBSCRIBE 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 MUST be a UTF-8 Encoded String [MQTT-3.8.3-1]. Each Topic Filter is followed by a Subscription Options byte.

The Payload MUST contain at least one Topic Filter and Subscription Options pair [MQTT-3.8.3-2]. A SUBSCRIBE packet with no Payload is a Protocol Error. Refer to [section 4.13](#S4_13_Errors) for information about handling errors.

#### Subscription Options

Bits 0 and 1 of the Subscription Options represent 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 if the Maximum QoS field has the value 3.

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

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

Bits 4 and 5 of the Subscription Options represent the Retain Handling option. This option 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 reserved for future use. The Server MUST treat a SUBSCRIBE packet as malformed if any of Reserved bits in the Payload are non-zero [MQTT-3.8.3-5].

**Non-normative comment**

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

**Non-normative comment**

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

**Non-normative comment**

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

**Non-normative comment**

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

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 |

RAP means Retain as Published.

NL means No Local.

**Non-normative example**

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

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 |

### SUBSCRIBE Actions

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

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 replace that existing Subscription with a new Subscription [MQTT-3.8.4-3]. The Topic Filter in the new Subscription will be identical to that in the previous Subscription, although its Subscription Options could be different. If the Retain Handling option is 0, any existing retained messages matching the Topic Filter MUST be re-sent, but Applicaton Messages MUST NOT be lost due to replacing the Subscription [MQTT-3.8.4-4].

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.

Refer to [section 4.8](#_Shared_Subscriptions) 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 [MQTT-3.8.4-5].

The SUBACK packet sent by the Server to the Client MUST contain a Reason Code for each Topic Filter/Subscription Option pair [MQTT-3.8.4-6]. This Reason Code MUST either show the maximum QoS that was granted for that Subscription or indicate that the subscription failed [MQTT-3.8.4-7]. The Server might grant a lower Maximum QoS than the subscriber requested. The QoS of Application Messages sent in response to a Subscription MUST be the minimum of the QoS of the originally published message and the Maximum QoS granted by the Server [MQTT-3.8.4-8]. 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 comment**  
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.

**Non-normative comment**

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.

The Subscription Identifiers are part of the Session State in the Server and are returned to the Client receiving a matching PUBLISH packet. They are removed from the Server’s Session State when the Server receives an UNSUBSCRIBE packet, when the Server receives a SUBSCRIBE packet from the Client for the same Topic Filter but with a different Subscription Identifier or with no Subscription Identifier, or when the Server sends Session Present 0 in a CONNACK packet.   
  
The Subscription Identifiers do not form part of the Client’s Session State in the Client. In a useful implementation, a Client will associate the Subscription Identifiers with other Client side state, this state is typically removed when the Client unsubscribes, when the Client subscribes for the same Topic Filter with a different identifier or no identifier, or when the Client receives Session Present 0 in a CONNACK packet.   
  
The Server need not use the same set of Subscription Identifiers in the retransmitted PUBLISH packet. The Client can remake a Subscription by sending a SUBSCRIBE packet containing a Topic Filter that is identical to the Topic Filter of an existing Subscription in the current Session. If the Client remade a subscription after the initial transmission of a PUBLISH packet and used a different Subscription Identifier, then the Server is allowed to use the identifiers from the first transmission in any retransmission. Alternatively, the Server is allowed to use the new identifiers during a retransmission. The Server is not allowed to revert to the old identifier after it has sent a PUBLISH packet containing the new one.

**Non-normative comment**

Usage scenarios, for illustration of Subscription Identifiers.

* The Client implementation indicates via its programming interface that a publication matched more than one subscription. The Client implementation generates a new identifier each time a subscription is made. If the returned publication carries more than one Subscription Identifier, then the publication matched more than one subscription.
* The Client implementation allows the subscriber to direct messages to a callback associated with the subscription. The Client implementation generates an identifier which uniquely maps the identifier to the callback. When a publication is received it uses the Subscription Identifier to determine which callback is driven.
* The Client implementation returns the topic string used to make the subscription to the application when it delivers the published message. To achieve this the Client generates an identifier which uniquely identifies the Topic Filter. When a publication is received the Client implementation uses the identifiers to look up the original Topic Filters and return them to the Client application.
* A gateway forwards publications received from a Server to Clients that have made subscriptions to the gateway. The gateway implementation maintains a map of each unique Topic Filter it receives to the set of ClientID, Subscription Identifier pairs that it also received. It generates a unique identifier for each Topic Filter that it forwards to the Server. When a publication is received, the gateway uses the Subscription Identifiers it received from the Server to look up the Client Identifier, Subscription Identifier pairs associated with them. It adds these to the PUBLISH packets it sends to the Clients. If the upstream Server sent multiple PUBLISH packets because the message matched multiple subscriptions, then this behavior is mirrored to the Clients.

## 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 Reason 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.

### SUBACK Fixed Header

Figure 3‑22 - SUBACK Packet Fixed Header

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

**Remaining Length field**

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

### SUBACK Variable Header

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

#### SUBACK Properties

##### Property Length

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

##### Reason String

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

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

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

##### User Property

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

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

Figure ‑ SUBACK packet Variable Header

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

### SUBACK Payload

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

Table ‑8 - Subscribe Reason Codes

|  |  |  |  |
| --- | --- | --- | --- |
| **Value** | **Hex** | **Reason Code name** | **Description** |
| 0 | 0x00 | Granted QoS 0 | The subscription is accepted and the maximum QoS sent will be QoS 0. This might be a lower QoS than was requested. |
| 1 | 0x01 | Granted QoS 1 | The subscription is accepted and the maximum QoS sent will be QoS 1. This might be a lower QoS than was requested. |
| 2 | 0x02 | Granted QoS 2 | The subscription is accepted and any received QoS will be sent to this subscription. |
| 128 | 0x80 | Unspecified error | The subscription is not accepted and the Server either does not wish to reveal the reason or none of the other Reason Codes apply. |
| 131 | 0x83 | Implementation specific error | The SUBSCRIBE is valid but the Server does not accept it. |
| 135 | 0x87 | Not authorized | The Client is not authorized to make this subscription. |
| 143 | 0x8F | Topic Filter invalid | The Topic Filter is correctly formed but is not allowed for this Client. |
| 145 | 0x91 | Packet Identifier in use | The specified Packet Identifier is already in use. |
| 151 | 0x97 | Quota exceeded | An implementation or administrative imposed limit has been exceeded. |
| 158 | 0x9E | Shared Subscription not supported | The Server does not support Shared Subscriptions for this Client. |
| 161 | 0xA1 | Subscription Identifiers not supported | The Server does not support Subscription Identifiers; the subscription is not accepted. |
| 162 | 0xA2 | Wildcard subscriptions not supported | The Server does not support Wildcard subscription; the subscription is not accepted. |

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

**Non-normative comment**

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

## UNSUBSCRIBE – Unsubscribe request

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

### UNSUBSCRIBE Fixed Header

Figure 3.28 – UNSUBSCRIBE packet Fixed Header

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

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

**Remaining Length field**

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

### UNSUBSCRIBE Variable Header

The Variable Header of the UNSUBSCRIBE Packet contains the following fields in the order: Packet Identifier, and Properties. [Section 2.2.1](#_Toc358219870) provides more information about Packet Identifiers. The rules for encoding Properties are described in [section 2.2.2](#_Properties).

#### UNSUBSCRIBE Properties

##### Property Length

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

##### User Property

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

Followed by a UTF-8 String Pair.

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

**Non-normative comment**

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

### UNSUBSCRIBE Payload

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

The Payload of an UNSUBSCRIBE packet MUST contain at least one Topic Filter [MQTT-3.10.3-2]. An UNSUBSCRIBE packet with no Payload is a Protocol Error. Refer to [section 4.13](#S4_13_Errors) for information about handling errors.

**Non-normative example**

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

Figure 3.30 - Payload byte format non-normative example

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

### UNSUBSCRIBE Actions

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

When a Server receives UNSUBSCRIBE :

* It MUST stop adding any new messages which match the Topic Filters, for delivery to the Client [MQTT-3.10.4-2].
* It MUST complete the delivery of any QoS 1 or QoS 2 messages which match the Topic Filters and it has started to send to the Client [MQTT-3.10.4-3].
* 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 [MQTT-3.10.4-4]. The UNSUBACK packet MUST have the same Packet Identifier as the UNSUBSCRIBE packet. Even where no Topic Subscriptions are deleted, the Server MUST respond with an UNSUBACK [MQTT-3.10.4-5].

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

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

## UNSUBACK – Unsubscribe acknowledgement

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

### UNSUBACK 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 plus the length of the Payload, encoded as a Variable Byte Integer.

### UNSUBACK Variable Header

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

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 | | | | | | | |

#### UNSUBACK Properties

##### Property Length

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

##### Reason String

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

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

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

##### User Property

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

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

### UNSUBACK Payload

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

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

Table 3‑9 - Unsubscribe Reason Codes

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

**Non-normative comment**

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

## PINGREQ – PING request

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

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

This packet is used in Keep Alive processing. Refer to [section 3.1.2.10](#_Keep_Alive_1) for more details.

### PINGREQ Fixed Header

Figure 3.33 – PINGREQ packet Fixed Header

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

### PINGREQ Variable Header

The PINGREQ packet has no Variable Header.

### PINGREQ Payload

The PINGREQ packet has no Payload.

### PINGREQ Actions

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

## PINGRESP – PING response

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

This packet is used in Keep Alive processing. Refer to [section 3.1.2.10](#_Keep_Alive_1) for more details.

### PINGRESP Fixed Header

Figure 3.34 – PINGRESP packet Fixed Header

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

### PINGRESP Variable Header

The PINGRESP packet has no Variable Header.

### PINGRESP Payload

The PINGRESP packet has no Payload.

### PINGRESP Actions

The Client takes no action on receiving this packet

## DISCONNECT – Disconnect notification

The DISCONNECT packet is the final MQTT Control Packet sent from the Client or the Server. It indicates the reason why the Network Connection is being closed. The Client or Server MAY send a DISCONNECT packet before closing the Network Connection. If the Network Connection is closed without the Client first sending a DISCONNECT packet with Reason Code 0x00 (Normal disconnection) and the Connection has a Will Message, the Will Message is published. Refer to [section 3.1.2.5](#_Toc479576982) for further details.

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

### DISCONNECT Fixed Header

Figure 3.35 – DISCONNECT packet Fixed Header

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

The Client or Server MUST validate that reserved bits are set to 0. If they are not zero it sends a DISCONNECT packet with a Reason code of 0x81 (Malformed Packet) as described in [section 4.13](#S4_13_Errors) [MQTT-3.14.1-1].

**Remaining Length field**

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

### DISCONNECT Variable Header

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

#### Disconnect Reason Code

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

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

Table ‑10 – Disconnect Reason Code values

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Value** | **Hex** | **Reason Code name** | **Sent by** | **Description** |
| 0 | 0x00 | Normal disconnection | Client or Server | Close the connection normally. Do not send the Will Message. |
| 4 | 0x04 | Disconnect with Will Message | Client | The Client wishes to disconnect but requires that the Server also publishes its Will Message. |
| 128 | 0x80 | Unspecified error | Client or Server | The Connection is closed but the sender either does not wish to reveal the reason, or none of the other Reason Codes apply. |
| 129 | 0x81 | Malformed Packet | Client or Server | The received packet does not conform to this specification. |
| 130 | 0x82 | Protocol Error | Client or Server | An unexpected or out of order packet was received. |
| 131 | 0x83 | Implementation specific error | Client or Server | The packet received is valid but cannot be processed by this implementation. |
| 135 | 0x87 | Not authorized | Server | The request is not authorized. |
| 137 | 0x89 | Server busy | Server | The Server is busy and cannot continue processing requests from this Client. |
| 139 | 0x8B | Server shutting down | Server | The Server is shutting down. |
| 141 | 0x8D | Keep Alive timeout | Server | The Connection is closed because no packet has been received for 1.5 times the Keepalive time. |
| 142 | 0x8E | Session taken over | Server | Another Connection using the same ClientID has connected causing this Connection to be closed. |
| 143 | 0x8F | Topic Filter invalid | Server | The Topic Filter is correctly formed, but is not accepted by this Sever. |
| 144 | 0x90 | Topic Name invalid | Client or Server | The Topic Name is correctly formed, but is not accepted by this Client or Server. |
| 147 | 0x93 | Receive Maximum exceeded | Client or Server | The Client or Server has received more than Receive Maximum publication for which it has not sent PUBACK or PUBCOMP. |
| 148 | 0x94 | Topic Alias invalid | Client or Server | The Client or Server has received a PUBLISH packet containing a Topic Alias which is greater than the Maximum Topic Alias it sent in the CONNECT or CONNACK packet. |
| 149 | 0x95 | Packet too large | Client or Server | The packet size is greater than Maximum Packet Size for this Client or Server. |
| 150 | 0x96 | Message rate too high | Client or Server | The received data rate is too high. |
| 151 | 0x97 | Quota exceeded | Client or Server | An implementation or administrative imposed limit has been exceeded. |
| 152 | 0x98 | Administrative action | Client or Server | The Connection is closed due to an administrative action. |
| 153 | 0x99 | Payload format invalid | Client or Server | The payload format does not match the one specified by the Payload Format Indicator. |
| 154 | 0x9A | Retain not supported | Server | The Server has does not support retained messages. |
| 155 | 0x9B | QoS not supported | Server | The Client specified a QoS greater than the QoS specified in a Maximum QoS in the CONNACK. |
| 156 | 0x9C | Use another server | Server | The Client should temporarily change its Server. |
| 157 | 0x9D | Server moved | Server | The Server is moved and the Client should permanently change its server location. |
| 158 | 0x9E | Shared Subscription not supported | Server | The Server does not support Shared Subscriptions. |
| 159 | 0x9F | Connection rate exceeded | Server | This connection is closed because the connection rate is too high. |
| 160 | 0xA0 | Maximum connect time | Server | The maximum connection time authorized for this connection has been exceeded. |
| 161 | 0xA1 | Subscription Identifiers not supported | Server | The Server does not support Subscription Identifiers; the subscription is not accepted. |
| 162 | 0xA2 | Wildcard subscriptions not supported | Server | The Server does not support Wildcard subscription; the subscription is not accepted. |

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

**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.

**Non-normative comment**

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

#### DISCONNECT Properties

##### Property Length

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

##### 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 Interval in the CONNECT packet is used.

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

If the Session Expiry Interval in the CONNECT packet was zero, then it is a Protocol Error to set a non-zero Session Expiry Interval in the DISCONNECT packet sent by the Client. If such a non-zero Session Expiry Interval is received by the Server, it does not treat it as a valid DISCONNECT packet. The Server uses DISCONNECT with Reason Code 0x82 (Protocol Error) as described in [section 4.13](#S4_13_Errors).

##### Reason String

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

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

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

##### User Property

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

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

##### Server Reference

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

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

The Server sends DISCONNECT including a Server Reference and Reason Code 0x9C (Use another server) or 0x9D (Server moved) as described in [section 4.13](#S4_13_Errors).

Refer to [section 4.11](#_Server_redirection) 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** | |
| Disconnect Reason Code | | | | | | | | | | | | | | | | |
| byte 1 |  | 0 | | 0 | 0 | | 0 | | 0 | | 0 | | 0 | | 0 | |
| Properties | | | | | | | | | | | | | | | | |
| byte 2 | Length (5) | | 0 | 0 | | 0 | | 0 | | 0 | | 1 | | 1 | | 1 |
| byte 3 | Session Expiry Interval identifier (17) | | 0 | 0 | | 0 | | 1 | | 0 | | 0 | | 0 | | 1 |
| byte 4 | Session Expiry Interval (0) | | 0 | 0 | | 0 | | 0 | | 0 | | 0 | | 0 | | 0 |
| byte 5 | 0 | 0 | | 0 | | 0 | 0 | | 0 | | | 0 | 0 | |
| byte 6 | 0 | 0 | | 0 | | 0 | 0 | | 0 | | | 0 | 0 | |
| byte 7 | 0 | 0 | | 0 | | 0 | 0 | | 0 | | | 0 | 0 | |

### DISCONNECT Payload

The DISCONNECT packet has no Payload.

### DISCONNECT Actions

After sending a DISCONNECT packet the sender:

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

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

* MUST discard any Will Message associated with the current Connection without publishing it [MQTT-3.14.4-3], as described in [section 3.1.2.5](#_Toc479576982).

On receipt of DISCONNECT, the receiver:

* SHOULD close the Network Connection.

## AUTH – Authentication exchange

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

### AUTH Fixed Header

Figure 3.35 – AUTH packet Fixed Header

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

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

**Remaining Length field**

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

### AUTH Variable Header

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

#### Authenticate Reason Code

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

Table ‑11 Authenticate Reason Codes

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

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

#### AUTH Properties

##### Property Length

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

##### Authentication Method

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

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

##### Authentication Data

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

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

##### Reason String

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

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

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

##### User Property

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

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

### AUTH Payload

The AUTH packet has no Payload.

### AUTH Actions

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

# Operational behavior

## Session State

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

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

The Session State in the Client consists of:

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

The Session State in the Server consists of:

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

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

### Storing Session State

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

**Non-normative comment**

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

### Session State non-normative examples

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

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

## Network Connections

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

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

**Non-normative comment**

TCP/IP as defined in [[RFC0793]](#RFC793) can be used for MQTT v5.0. The following transport protocols are also suitable:

* TLS [[RFC5246]](#RFC5246)
* WebSocket [[RFC6455]](#RFC6455)

**Non-normative comment**

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

**Non-normative comment**

Connectionless network transports such as [User Datagram Protocol](https://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 in the following sections. The delivery protocol is symmetric, in the description below the Client and Server can each take the role of either sender or receiver. The delivery protocol is concerned solely with the delivery of an application message from a single sender to a single receiver. When the Server is delivering an Application Message to more than one Client, each Client is treated independently. The QoS level used to deliver an Application Message outbound to the Client could differ from that of the inbound Application Message.

### QoS 0: At most once delivery

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

In the QoS 0 delivery protocol, the sender

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

In the QoS 0 delivery protocol, the receiver

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

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

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

### QoS 1: At least once delivery

This Quality of Service level ensures that the message arrives at the receiver at least once. A QoS 1 PUBLISH packet has a Packet Identifier in its Variable Header and is acknowledged by a PUBACK packet. [Section 2.2.1](#_Toc358219870) provides more information about Packet Identifiers.

In the QoS 1 delivery protocol, the sender

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

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

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

In the QoS 1 delivery protocol, the receiver

* MUST respond with a PUBACK packet containing the Packet Identifier from the incoming PUBLISH packet, having accepted ownership of the Application Message [MQTT-4.3.2-4].
* After it has sent a PUBACK packet the receiver MUST treat any incoming PUBLISH packet that contains the same Packet Identifier as being a new Application Message, irrespective of the setting of its DUP flag [MQTT-4.3.2-5].

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

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

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

### QoS 2: Exactly once delivery

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

A QoS 2 message has a Packet Identifier in its Variable Header. [Section 2.2.1](#_Toc358219870) provides more information about Packet Identifiers. The receiver of a QoS 2 PUBLISH packet acknowledges receipt with a two-step acknowledgement process.

In the QoS 2 delivery protocol, the sender:

* MUST assign an unused Packet Identifier when it has a new Application Message to publish [MQTT-4.3.3-1].
* MUST send a PUBLISH packet containing this Packet Identifier with QoS 2 and DUP flag set to 0 [MQTT-4.3.3-2].
* MUST treat the PUBLISH packet as “unacknowledged” until it has received the corresponding PUBREC packet from the receiver [MQTT-4.3.3-3]. Refer to [section 4.4](#_Figure_4.3_–) for a discussion of unacknowledged messages.
* MUST send a PUBREL packet when it receives a PUBREC packet from the receiver with a Reason Code value less than 0x80. This PUBREL packet MUST contain the same Packet Identifier as the original PUBLISH packet [MQTT-4.3.3-4].
* MUST treat the PUBREL packet as “unacknowledged” until it has received the corresponding PUBCOMP packet from the receiver [MQTT-4.3.3-5].
* MUST NOT re-send the PUBLISH once it has sent the corresponding PUBREL packet [MQTT-4.3.3-6].
* MUST NOT apply Message expiry if a PUBLISH packet has been sent [MQTT-4.3.3-7].

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

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

In the QoS 2 delivery protocol, the receiver:

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

## Message delivery retry

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

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

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

|  |  |  |
| --- | --- | --- |
| **Sender Action** | **MQTT Control Packet** | **Receiver Action** |
| Store message |  |  |
| PUBLISH QoS 2, DUP=0  <Packet Identifier> |  |  |
|  | ----------> |  |
|  |  | Store <Packet Identifier> then Initiate onward delivery of the Application Message1 |
|  |  | PUBREC <Packet Identifier><Reason Code> |
|  | <---------- |  |
| Discard message, Store PUBREC received <Packet Identifier> |  |  |
| PUBREL <Packet Identifier> |  |  |
|  | ----------> |  |
|  |  | Discard <Packet Identifier> |
|  |  | Send PUBCOMP <Packet Identifier> |
|  | <---------- |  |
| Discard stored state |  |  |

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

## Message receipt

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

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

## Message ordering

The following these rules apply to the Client when implementing the protocol flows defined in [section 4.3.](#_Quality_of_Service)

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

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

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

**Non-normative comment**

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

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

## 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 a Client to subscribe to multiple topics at once.

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

#### 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 [MQTT-4.7.1-1].

**Non-normative comment**

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

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

**Non-normative comment**

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

#### Single-level wildcard

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

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

**Non-normative comment**

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

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

### Topics beginning with $

The Server MUST NOT match Topic Filters starting with a wildcard character (# or +) with Topic Names beginning with a $ character [MQTT-4.7.2-1]. The Server SHOULD prevent Clients from using such Topic Names to exchange messages with other Clients. Server implementations MAY use Topic Names that start with a leading $ character for other purposes.

**Non-normative comment**

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

**Non-normative comment**

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

### Topic semantic and usage

The following rules apply to Topic Names and Topic Filters:

* All Topic Names and Topic Filters MUST be at least one character long [MQTT-4.7.3-1]
* Topic Names and Topic Filters are case sensitive
* Topic Names and Topic Filters can include the space character
* A leading or trailing ‘/’ creates a distinct Topic Name or Topic Filter
* A Topic Name or Topic Filter consisting only of the ‘/’ character is valid
* Topic Names and Topic Filters MUST NOT include the null character (Unicode U+0000) [[Unicode]](#Unicode) [MQTT-4.7.3-2]
* Topic Names and Topic Filters are UTF-8 Encoded Strings; they MUST NOT encode to more than 65,535 bytes [MQTT-4.7.3-3]. Refer to [section 1.5.4](#_UTF-8_Encoded_String).

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

When it performs subscription matching the Server MUST NOT perform any normalization of Topic Names or Topic Filters, or any modification or substitution of unrecognized characters [MQTT-4.7.3-4]. Each non-wildcarded level in the Topic Filter has to match the corresponding level in the Topic Name character for character for the match to succeed.

**Non-normative comment**

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

**Non-normative comment**

* “ACCOUNTS” and “Accounts” are two different Topic Names
* “Accounts payable” is a valid Topic Name
* “/finance” is different from “finance”

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

## Subscriptions

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

**Non-normative comment**

In earlier versions of MQTT all Subscriptions are 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 Subscription Options. 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 the Non‑shared Subscriptions cannot be used to load-balance Application Messages across multiple consuming Clients as in such cases every message is delivered to every subscribing 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 Subscription Options; however, a publication that matches its Topic Filter is only sent to one of its subscribing Sessions. Shared Subscriptions are useful where several consuming Clients share the processing of the publications in parallel.

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. Refer to [section 4.7](#_Topic_Names_and).

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

**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. Typically, applications use the ShareName to represent the group of subscribing Sessions that are sharing the subscription.  
  
Examples:

* + - Shared subscriptions "$share/consumer1/sport/tennis/+" and "$share/consumer2/sport/tennis/+" are distinct shared subscriptions and so can be associated with different groups of Sessions. Both of them match the same topics as a non-shared subscription to sport/tennis/+ .

If a message were to be published that matches sport/tennis/+ then a copy would be sent to exactly one of the Sessions subscribed to $share/consumer1/sport/tennis/+ , a separate copy of the message would be sent to exactly one of the Sessions subscribed to $share/consumer2/sport/tennis/+ and further copies would be sent to any Clients with non-shared subscriptions to sport/tennis/+

* + - Shared subscription "$share/consumer1//finance" matches the same topics as a non-shared subscription to /finance.   
        
      Note that "$share/consumer1//finance" and "$share/consumer1/sport/tennis/+" are distinct shared subscriptions, even though they have the same ShareName. While they might be related in some way, no specific relationship between them is implied by them having the same ShareName.

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

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

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

Notes on Shared Subscriptions

* If there's more than one Session subscribed to the Shared Subscription, the Server implementation is free to choose, on a message by message basis, which Session to use and what criteria it uses to make this selection.
* Different subscribing Clients are permitted to ask for different Requested QoS levels in their SUBSCRIBE packets. The Server decides which Maximum QoS to grant to each Client, and it is permitted to grant different Maximum QoS levels to different subscribers. When sending an Application Message to a Client, the Server MUST respect the granted QoS for the Client's subscription [MQTT-4.8.2-3], in the same that it does when sending a message to a ‑Subscriber.
* If the Server is in the process of sending a QoS 2 message to its chosen subscribing Client and the connection to the Client breaks before delivery is complete, the Server MUST complete the delivery of the message to that Client when it reconnects [MQTT-4.8.2-4] as described in [section 4.3.3](#_QoS_2:_Exactly). If the Client's Session terminates before the Client reconnects, the Server MUST NOT send the Application Message to any other subscribed Client [MQTT-4.8.2-5].
* If the Server is in the process of sending a QoS 1 message to its chosen subscribing Client and the connection to that Client breaks before the Server has received an acknowledgement from the Client, the Server MAY wait for the Client to reconnect and retransmit the message to that Client. If the Client'sSession terminates before the Client reconnects, the Server SHOULD send the Application Message to another Client that is subscribed to the same Shared Subscription. It MAY attempt to send the message to another Client as soon as it loses its connection to the first Client.
* If a Client responds with a PUBACK or PUBREC containing a Reason Code of 0x80 or greater to a PUBLISH packet from the Server, the Server MUST discard the Application Message and not attempt to send it to any other Subscriber [MQTT-4.8.2-6].
* A Client is permitted to submit a second SUBSCRIBE request to a Shared Subscription on a Session that's already subscribed to that Shared Subscription. For example, it might do this to change the Requested QoS for its subscription or because it was uncertain that the previous subscribe completed before the previous connection was closed. This does not increase the number of times that the Session is associated with the Shared Subscription, so the Session will leave the Shared Subscription on its first UNSUBSCRIBE.
* Each Shared Subscription is independent from any other. It is possible to have two Shared Subscriptions with overlapping filters. In such cases a message that matches both Shared Subscriptions will be processed separately by both of them. If a Client has a Shared Subscription and a Non‑shared Subscription and a message matches both of them, the Client will receive a copy of the message by virtue of it having the Non‑shared Subscription. A second copy of the message will be delivered to one of the subscribers to the Shared Subscription, and this could result in a second copy being sent to this Client.

## Flow Control

Clients and Servers control the number of unacknowledged PUBLISH packets they receive by using a Receive Maximum value as described in [section 3.1.2.11.4](#_Receive_Maximum) and [section 3.2.2.3.2](#_Receive_Maximum_1). The Receive Maximum establishes a send quota which is used to limit the number of PUBLISH QOS > 0 packets which can be sent without receiving an PUBACK (for QoS 1) or PUBCOMP (for QoS 2). The PUBACK and PUBCOMP replenish the quota in the manner described below.

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

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

The send quota is incremented by 1:

* Each time a PUBACK or PUBCOMP packet is received, regardless of whether the PUBACK or PUBCOMP carried an error code.
* Each time a PUBREC packet is received with a Return Code of 0x80 or greater.

The send quota is not incremented if it is already equal to the initial send quota. The attempt to increment above the initial send quota might be caused by the re-transmission of a PUBREL packet after a new Network Connection is established.

Refer to [section 3.3.4](#_PUBLISH_Actions) for a description of how Clients and Servers react if they are sent more PUBLISH packets than the Receive Maximum allows.

The send quota and Receive Maximum value are not preserved across Network Connections, and are re-initialized with each new Network Connection as described above. They are not part of the session state.

## Request / Response

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

* Response Topic, described in [section 3.3.2.3.5](#_Response_Topic)
* Correlation Data, described in [section 3.3.2.3.6](#_Correlation_Data)
* Request Response Information, described in [section 3.1.2.11.7](#_Request_Response_Information)
* Response Information, described in [section 3.2.2.3.14](#_Response_Information)

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

A Client sends a Request Message by publishing an Application Message which has a Response Topic set as described in [section 3.3.2.3.5](#_Toc463595993). The Request can include a Correlation Data property as described in [section 3.3.2.3.6](#_Correlation_Data).

## Basic Request Response (non-normative)

Request/Response interaction proceeds as follows:

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

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

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

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

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

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

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

## Determining a Response Topic value (non-normative)

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

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

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

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

## Server redirection

A Server can request that the Client uses another Server by sending CONNACK or DISCONNECT with Reason Codes 0x9C (Use another server), or 0x9D (Server moved) as described in [section 4.13](#S4_13_Errors). When sending one of these Reason Codes, the Server MAY also include a Server Reference property to indicate the location of the Server or Servers the Client SHOULD use.

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

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

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

**Non-normative comment**

It is recommended that 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 cannot contain the right square bracket (“]”) character. This is used to represent an IPv6 literal address which uses colon separators. This is a simplified version of an URI authority as described in [[RFC3986]](#RFC3986).

**Non-normative comment**

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

**Non-normative comment**

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

**Non-normative comment**

Examples of the Server Reference are:

myserver.xyz.org

myserver.xyz.org:8883

10.10.151.22:8883 [fe80::9610:3eff:fe1c]:1883

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

## Enhanced authentication

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

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

To begin an enhanced authentication, the Client includes an Authentication Method in the CONNECT packet. This specifies the authentication method to use. If the Server does not support the Authentication Method supplied by the Client, it MAY send a CONNACK with a Reason Code of 0x8C (Bad authentication method) or 0x87 (Not Authorized) as described in [section 4.13](#S4_13_Errors) and MUST close the Network Connection [MQTT-4.12.0-1].

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

**Non-normative comment**

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

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

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

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

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

The Client can close the connection at any point in this process. It MAY send a DISCONNECT packet before doing so. The Server can reject the authentication at any point in this process. It MAY send a CONNACK with a Reason Code of 0x80 or above as described in [section 4.13](#S4_13_Errors), and MUST close the Network Connection [MQTT-4.12.0-4].

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

The implementation of enhanced authentication is OPTIONAL for both Clients and Servers. If the Client does not include an Authentication Method in the CONNECT, the Server MUST NOT send an AUTH packet, and it MUST NOT send an Authentication Method in the CONNACK packet [MQTT-4.12.0-6]. If the Client does not include an Authentication Method in the CONNECT, the Client MUST NOT send an AUTH packet to the Server [MQTT-4.12.0-7].

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

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

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

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

* Client to Server CONNECT Authentication Method="GS2-KRB5"
* Server to Client AUTH rc=0x18 Authentication Method="GS2-KRB5"
* Client to Server AUTH rc=0x18 Authentication Method="GS2-KRB5" Authentication Data=initial context token
* Server to Client AUTH rc=0x18 Authentication Method="GS2-KRB5" Authentication Data=reply context token
* Client to Server AUTH rc=0x18 Authentication Method="GS2-KRB5"
* Server to Client CONNACK rc=0 Authentication Method="GS2-KRB5" Authentication Data=outcome of authentication

### Re-authentication

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

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

If the re-authentication fails, the Client or Server SHOULD send DISCONNECT with an appropriate Reason Code as described in [section 4.13](#S4_13_Errors), and MUST close the Network Connection [MQTT-4.12.1-2].

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

**Non-normative comment**

The Server might limit the scope of the changes the Client can attempt in a re-authentication by rejecting the re-authentication. For instance, if the Server does not allow the User Name to be changed it can fail any re-authentication attempt which changes the User Name.

## Handling errors

### Malformed Packet and Protocol Errors

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

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

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

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

* 0x81 Malformed Packet
* 0x82 Protocol Error
* 0x93 Receive Maximum exceeded
* 0x95 Packet too large
* 0x9A Retain not supported
* 0x9B QoS not supported
* 0x9E Shared Subscription not supported
* 0xA1 Subscription Identifiers not supported
* 0xA2 Wildcard Subscription not supported

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

When a Server detects a Malformed Packet or Protocol Error, and a Reason Code is given in the specification, it MUST close the Network Connection [MQTT-4.13.1-1]. In the case of an error in a CONNECT packet it MAY send a CONNACK packet containing the Reason Code, before closing the Network Connection. In the case of an error in any other packet it SHOULD send a DISCONNECT packet containing the Reason Code before closing the Network Connection. Use Reason Code 0x81 (Malformed Packet) or 0x82 (Protocol Error) unless a more specific Reason Code has been defined in [section 3.2.2.2 - Connect Reason Code](#_Connect_Reason_Code) or in section [3.14.2.1 – Disconnect Reason Code](#_Disconnect_Return_codes). There are no consequences for other Sessions.

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

### Other errors

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

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

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

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

# Security (non-normative)

## Introduction

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

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

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

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

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

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

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]](#RFC5246).

In addition to technical security issues there could also be geographic (e.g. U.S.-EU Privacy Shield Framework [[USEUPRIVSH]](#USEUPRIVSH)), industry specific (e.g. PCI DSS [[PCIDSS]](#PCIDSS)) and regulatory considerations (e.g. Sarbanes-Oxley [[SARBANES]](#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]](#NISTCSF), PCI-DSS [[PCIDSS]](#PCIDSS)), FIPS-140-2 [[FIPS1402]](#FIPS1402) and NSA Suite B [[NSAB]](#NSAB).

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

## Lightweight crytography and constrained devices

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

ISO 29192 [[ISO29192]](#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 User Name 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]](#RFC4511) or OAuth [[RFC6749]](#RFC6749) tokens, or leverage operating system authentication mechanisms.

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

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](#_Privacy_of_Application) 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]](#RFC5246) is used, TLS 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 4.1](#_Session_State)). This authorization check is to protect against the case where one Client, accidentally or maliciously, provides a Client Identifier that is already being used by some other Client.

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

### Authentication of the Server by the Client

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

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

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

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

### Integrity of Application Messages and MQTT Control Packets

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

TLS [[RFC5246]](#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 MQTT Control Packets

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

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

Physical deployments might combine tamper-proof hardware with the transmission of specific data in Application Messages. For example, a meter might have an embedded GPS to ensure it is not used in an unauthorized location. [[IEEE8021AR]](#IEEE8021AR) 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 close the Network Connection of Clients that breach its security rules.

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

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 TLS certificates are lost or it is considered that they might be compromised they should be revoked (utilizing CRLs [[RFC5280]](#RFC5280) and/or OSCP [[RFC6960]](#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]](#RFC5246) should allow for session renegotiation to establish new cryptographic parameters (replace session keys, change cipher suites, change authentication credentials).
* Servers may close the Network Connection of Clients and require them to re-authenticate with new credentials.
* Servers may require their Client to reauthenticate periodically using the mechanism described in [section 4.12.1](#_Re-authentication).

Constrained devices and Clients on constrained networks can make use of TLS [[RFC5246]](#RFC5246) session resumption, in order to reduce the costs of reconnecting TLS [[RFC5246]](#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]](#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]](#RFC5246) which provides authentication, integrity and privacy.

TLS [[RFC5246]](#RFC5246) Client authentication can be used in addition to – or in place of – MQTT Client authentication as provided by the User Name 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](#NSAB)

# 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 [MQTT-6.0.0-1].
* A single WebSocket data frame can contain multiple or partial MQTT Control Packets. The receiver MUST NOT assume that MQTT Control Packets are aligned on WebSocket frame boundaries [MQTT-6.0.0-2].
* The Client MUST include “mqtt” in the list of WebSocket Sub Protocols it offers [MQTT-6.0.0-3].
* The WebSocket Subprotocol name selected and returned by the Server MUST be “mqtt” [MQTT-6.0.0-4].
* The WebSocket URI used to connect the Client and Server has no impact on the MQTT protocol.

## IANA considerations

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

Figure 6.‑ - IANA WebSocket Identifier

|  |  |
| --- | --- |
| Subprotocol Identifier | mqtt |
| Subprotocol Common Name | mqtt |
| Subprotocol Definition | http://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html |

# Conformance

The MQTT specification defines conformance for MQTT Client implementations and MQTT Server implementations. An MQTT implementation can conform as both an MQTT Client and an MQTT Server.

## Conformance clauses

### MQTT Server conformance clause

Refer to [Server](#Server) in the Terminology section for a definition of Server.

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

1. The format of all MQTT Control Packets that the Server sends matches the format described in [Chapter 2](#_MQTT_Control_Packet) and [Chapter 3](#_MQTT_Control_Packets).
2. It follows the Topic matching rules described in [section 4.7](#_Topic_Names_and) and the Subscription rules in [section 4.8](#_Subscriptions).
3. It satisfies the MUST level requirements in the following chapters that are identified except for those that only apply to the Client:

* [Chapter 1 - Introduction](#_Introduction)
* [Chapter 2 - MQTT Control Packet format](#_MQTT_Control_Packet)
* [Chapter 3 - MQTT Control Packets](#_MQTT_Control_Packets)
* [Chapter 4 - Operational behavior](#_Operational_behavior)
* [Chapter 6 - Using WebSocket as a network transport](#_Using_WebSocket_as)

1. It does not require the use of any extensions defined outside of the specification in order to interoperate with any other conformant implementation.

### MQTT Client conformance clause

Refer to [Client](#Client) in the Terminology section for a definition of Client.

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

1. The format of all MQTT Control Packets that the Client sends matches the format described in [Chapter 2](#_MQTT_Control_Packet) and [Chapter 3](#_MQTT_Control_Packets).
2. It satisfies the MUST level requirements in the following chapters that are identified except for those that only apply to the Server:

* [Chapter 1 - Introduction](#_Introduction)
* [Chapter 2 - MQTT Control Packet format](#_MQTT_Control_Packet)
* [Chapter 3 - MQTT Control Packets](#_MQTT_Control_Packets)
* [Chapter 4 - Operational behavior](#_Operational_behavior)
* [Chapter 6 - Using WebSocket as a network](#_Using_WebSocket_as) transport

1. It does not require the use of any extensions defined outside of the specification in order to interoperate with any other conformant implementation.
2. 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:**

* Senthil Nathan Balasubramaniam (Infiswift)
* Dr. Andrew Banks, editor (IBM)
* Ken Borgendale, editor (IBM)
* Ed Briggs, editor (Microsoft)
* Raphael Cohn (Individual)
* Richard Coppen, chairman (IBM)
* William Cox (Individual)
* Ian Craggs , secretary (IBM)
* Konstantin Dotchkoff (Microsoft)
* Derek Fu (IBM)
* Rahul Gupta, editor (IBM)
* Stefan Hagen (Individual)
* David Horton (Solace Systems)
* Alex Kritikos (Software AG, Inc.)
* Jonathan Levell (IBM)
* Shawn McAllister (Solace Systems)
* William McLane (TIBCO Software Inc.)
* Peter Niblett (IBM)
* Dominik Obermaier (dc-square GmbH)
* Nicholas O'Leary (IBM)
* Brian Raymor, chairman (Microsoft)
* Andrew Schofield (IBM)
* Tobias Sommer (Cumulocity)
* Joe Speed (IBM)
* Dr Andy Stanford-Clark (IBM)
* Allan Stockdill-Mander (IBM)
* Stehan Vaillant (Cumulocity)

For a list of those who contributed to earlier versions of MQTT refer to Appendix A in the MQTT v3.1.1 specification **[MQTTV311]**.

1. Mandatory normative statement (non-normative)

This Appendix is non-normative and is provided as a convenient summary of the numbered conformance statements found in the main body of this document. Refer to [Chapter 7](#_Conformance) for a definitive list of conformance requirements.

|  |  |
| --- | --- |
| **Normative Statement Number** | **Normative Statement** |
| [MQTT-1.5.4-1] | The character data in a UTF-8 Encoded String MUST be well-formed UTF-8 as defined by the Unicode specification [Unicode] and restated in RFC 3629 [RFC3629]. In particular, the character data MUST NOT include encodings of code points between U+D800 and U+DFFF. |
| [MQTT-1.5.4-2] | A UTF-8 Encoded String MUST NOT include an encoding of the null character U+0000. |
| [MQTT-1.5.4-3] | A UTF-8 encoded sequence 0xEF 0xBB 0xBF is always interpreted as U+FEFF ("ZERO WIDTH NO-BREAK SPACE") wherever it appears in a string and MUST NOT be skipped over or stripped off by a packet receiver. |
| [MQTT-1.5.5-1] | The encoded value MUST use the minimum number of bytes necessary to represent the value. |
| [MQTT-1.5.7-1] | Both strings MUST comply with the requirements for UTF-8 Encoded Strings. |
| [MQTT-2.1.3-1] | Where a flag bit is marked as “Reserved” it is reserved for future use and MUST be set to the value listed. |
| [MQTT-2.2.1-2] | A PUBLISH packet MUST NOT contain a Packet Identifier if its QoS value is set to 0. |
| [MQTT-2.2.1-3] | Each time a Client sends a new SUBSCRIBE, UNSUBSCRIBE,or PUBLISH (where QoS > 0) MQTT Control Packet it MUST assign it a non-zero Packet Identifier that is currently unused. |
| [MQTT-2.2.1-4] | Each time a Server sends a new PUBLISH (with QoS > 0) MQTT Control Packet it MUST assign it a non zero Packet Identifier that is currently unused. |
| [MQTT-2.2.1-5] | A PUBACK, PUBREC, PUBREL, or PUBCOMP packet MUST contain the same Packet Identifier as the PUBLISH packet that was originally sent. |
| [MQTT-2.2.1-6] | A SUBACK and UNSUBACK MUST contain the Packet Identifier that was used in the corresponding SUBSCRIBE and UNSUBSCRIBE packet respectively. |
| [MQTT-2.2.2-1] | If there are no properties, this MUST be indicated by including a Property Length of zero. |
| [MQTT-3.1.0-1] | After a Network Connection is established by a Client to a Server, the first packet sent from the Client to the Server MUST be a CONNECT packet. |
| [MQTT-3.1.0-2] | The Server MUST process a second CONNECT packet sent from a Client as a Protocol Error and close the Network Connection. |
| [MQTT-3.1.2-1] | The protocol name MUST be the UTF-8 String "MQTT". If the Server does not want to accept the CONNECT, and wishes to reveal that it is an MQTT Server it MAY send a CONNACK packet with Reason Code of 0x84 (Unsupported Protocol Version), and then it MUST close the Network Connection. |
| [MQTT-3.1.2-2] | If the Protocol Version is not 5 and the Server does not want to accept the CONNECT packet, the Server MAY send a CONNACK packet with Reason Code 0x84 (Unsupported Protocol Version) and then MUST close the Network Connection |
| [MQTT-3.1.2-3] | The Server MUST validate that the reserved flag in the CONNECT packet is set to 0. |
| [MQTT-3.1.2-4] | If a CONNECT packet is received with Clean Start is set to 1, the Client and Server MUST discard any existing Session and start a new Session. |
| [MQTT-3.1.2-5] | If a CONNECT packet is received with Clean Start set to 0 and there is a Session associated with the Client Identifier, the Server MUST resume communications with the Client based on state from the existing Session. |
| [MQTT-3.1.2-6] | If a CONNECT packet is received with Clean Start set to 0 and there is no Session associated with the Client Identifier, the Server MUST create a new Session. |
| [MQTT-3.1.2-7] | If the Will Flag is set to 1 this indicates that, a Will Message MUST be stored on the Server and associated with the Session. |
| [MQTT-3.1.2-8] | The Will Message MUST be published after the Network Connection is subsequently closed and either the Will Delay Interval has elapsed or the Session ends, unless the Will Message has been deleted by the Server on receipt of a DISCONNECT packet with Reason Code 0x00 (Normal disconnection) or a new Network Connection for the ClientID is opened before the Will Delay Interval has elapsed. |
| [MQTT-3.1.2-9] | If the Will Flag is set to 1, the Will QoS and Will Retain fields in the Connect Flags will be used by the Server, and the Will Properties, Will Topic and Will Message fields MUST be present in the Payload. |
| [MQTT-3.1.2-10] | The Will Message MUST be removed from the stored Session State in the Server once it has been published or the Server has received a DISCONNECT packet with a Reason Code of 0x00 (Normal disconnection) from the Client. |
| [MQTT-3.1.2-11] | If the Will Flag is set to 0, then the Will QoS MUST be set to 0 (0x00). |
| [MQTT-3.1.2-12] | If the Will Flag is set to 1, the value of Will QoS can be 0 (0x00), 1 (0x01), or 2 (0x02). |
| [MQTT-3.1.2-13] | If the Will Flag is set to 0, then Will Retain MUST be set to 0. |
| [MQTT-3.1.2-14] | If the Will Flag is set to 1 and Will Retain is set to 0, the Server MUST publish the Will Message as a non-retained message. |
| [MQTT-3.1.2-15] | If the Will Flag is set to 1 and Will Retain is set to 1, the Server MUST publish the Will Message as a retained message. |
| [MQTT-3.1.2-16] | If the User Name Flag is set to 0, a User Name MUST NOT be present in the Payload. |
| [MQTT-3.1.2-17] | If the User Name Flag is set to 1, a User Name MUST be present in the Payload. |
| [MQTT-3.1.2-18] | If the Password Flag is set to 0, a Password MUST NOT be present in the Payload. |
| [MQTT-3.1.2-19] | If the Password Flag is set to 1, a Password MUST be present in the Payload. |
| [MQTT-3.1.2-20] | If Keep Alive is non-zero and in the absence of sending any other MQTT Control Packets, the Client MUST send a PINGREQ packet. |
| [MQTT-3.1.2-21] | If the Server returns a Server Keep Alive on the CONNACK packet, the Client MUST use that value instead of the value it sent as the Keep Alive. |
| [MQTT-3.1.2-22] | If the Keep Alive value is non-zero and the Server does not receive an MQTT Control Packet from the Client within one and a half times the Keep Alive time period, it MUST close the Network Connection to the Client as if the network had failed. |
| [MQTT-3.1.2-23] | The Client and Server MUST store the Session State after the Network Connection is closed if the Session Expiry Interval is greater than 0. |
| [MQTT-3.1.2-24] | The Server MUST NOT send packets exceeding Maximum Packet Size to the Client. |
| [MQTT-3.1.2-25] | Where a Packet is too large to send, the Server MUST discard it without sending it and then behave as if it had completed sending that Application Message. |
| [MQTT-3.1.2-26] | The Server MUST NOT send a Topic Alias in a PUBLISH packet to the Client greater than Topic Alias Maximum. |
| [MQTT-3.1.2-27] | If Topic Alias Maximum is absent or zero, the Server MUST NOT send any Topic Aliases to the. |
| [MQTT-3.1.2-28] | A value of 0 indicates that the Server MUST NOT return Response Information. |
| [MQTT-3.1.2-29] | If the value of Request Problem Information is 0, the Server MAY return a Reason String or User Properties on a CONNACK or DISCONNECT packet, but MUST NOT send a Reason String or User Properties on any packet other than PUBLISH, CONNACK, or DISCONNECT. |
| [MQTT-3.1.2-30] | If a Client sets an Authentication Method in the CONNECT, the Client MUST NOT send any packets other than AUTH or DISCONNECT packets until it has received a CONNACK packet. |
| [MQTT-3.1.3-1] | The Payload of the CONNECT packet contains one or more length-prefixed fields, whose presence is determined by the flags in the Variable Header. These fields, if present, MUST appear in the order Client Identifier, Will Topic, Will Message, User Name, Password. |
| [MQTT-3.1.3-2] | The ClientID MUST be used by Clients and by Servers to identify state that they hold relating to this MQTT Session between the Client and the Server. |
| [MQTT-3.1.3-3] | The ClientID MUST be present and is the first field in the CONNECT packet Payload. |
| [MQTT-3.1.3-4] | The ClientID MUST be a UTF-8 Encoded String. |
| [MQTT-3.1.3-5] | The Server MUST allow ClientID’s which are between 1 and 23 UTF-8 encoded bytes in length, and that contain only the characters "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ". |
| [MQTT-3.1.3-6] | A Server MAY allow a Client to supply a ClientID that has a length of zero bytes, however if it does so the Server MUST treat this as a special case and assign a unique ClientID to that Client. |
| [MQTT-3.1.3-7] | It MUST then process the CONNECT packet as if the Client had provided that unique ClientID, and MUST return the Assigned Client Identifier in the CONNACK packet. |
| [MQTT-3.1.3-8] | If the Server rejects the ClientID it MAY respond to the CONNECT packet with a CONNACK using Reason Code 0x85 (Client Identifier not valid) as described in section 4.13 Handling errors, and then it MUST close the Network Connection. |
| [MQTT-3.1.3-9] | If a new Network Connection to this Session is made before the Will Delay Interval has passed, the Server MUST NOT send the Will Message. |
| [MQTT-3.1.3-10] | The Server MUST maintain the order of User Properties when forwarding the Application Message. |
| [MQTT-3.1.3-11] | The Will Topic MUST be a UTF-8 Encoded String. |
| [MQTT-3.1.3-12] | If the User Name Flag is set to 1, the User Name is the next field in the Payload. The User Name MUST be a UTF-8 Encoded String. |
| [MQTT-3.1.4-1] | The Server MUST validate that the CONNECT packet matches the format described in section 3.1 and close the Network Connection if it does not match. |
| [MQTT-3.1.4-2] | The Server MAY check that the contents of the CONNECT packet meet any further restrictions and SHOULD perform authentication and authorization checks. If any of these checks fail, it MUST close the Network Connection. |
| [MQTT-3.1.4-3] | If the ClientID represents a Client already connected to the Server, the Server sends a DISCONNECT packet to the existing Client with Reason Code of 0x8E (Session taken over) as described in section 4.13 and MUST close the Network Connection of the existing Client. |
| [MQTT-3.1.4-4] | The Server MUST perform the processing of Clean Start. |
| [MQTT-3.1.4-5] | The Server MUST acknowledge the CONNECT packet with a CONNACK packet containing a 0x00 (Success) Reason Code. |
| [MQTT-3.1.4-6] | If the Server rejects the CONNECT, it MUST NOT process any data sent by the Client after the CONNECT packet except AUTH packets. |
| [MQTT-3.2.0-1] | The Server MUST send a CONNACK with a 0x00 (Success) Reason Code before sending any Packet other than AUTH. |
| [MQTT-3.2.0-2] | The Server MUST NOT send more than one CONNACK in a Network Connection. |
| [MQTT-3.2.2-1] | Byte 1 is the "Connect Acknowledge Flags". Bits 7-1 are reserved and MUST be set to 0. |
| [MQTT-3.2.2-2] | If the Server accepts a connection with Clean Start set to 1, the Server MUST set Session Present to 0 in the CONNACK packet in addition to setting a 0x00 (Success) Reason Code in the CONNACK packet. |
| [MQTT-3.2.2-3] | If the Server accepts a connection with Clean Start set to 0 and the Server has Session State for the ClientID, it MUST set Session Present to 1 in the CONNACK packet, otherwise it MUST set Session Present to 0 in the CONNACK packet. In both cases it MUST set a 0x00 (Success) Reason Code in the CONNACK packet. |
| [MQTT-3.2.2-4] | If the Client does not have Session State and receives Session Present set to 1 it MUST close the Network Connection. |
| [MQTT-3.2.2-5] | If the Client does have Session State and receives Session Present set to 0 it MUST discard its Session State if it continues with the Network Connection. |
| [MQTT-3.2.2-6] | If a Server sends a CONNACK packet containing a non-zero Reason Code it MUST set Session Present to 0. |
| [MQTT-3.2.2-7] | If a Server sends a CONNACK packet containing a Reason code of 0x80 or greater it MUST then close the Network Connection. |
| [MQTT-3.2.2-8] | The Server sending the CONNACK packet MUST use one of the Connect Reason Code values. |
| [MQTT-3.2.2-9] | If a Server does not support QoS 1 or QoS 2 PUBLISH packets it MUST send a Maximum QoS in the CONNACK packet specifying the highest QoS it supports. |
| [MQTT-3.2.2-10] | A Server that does not support QoS 1 or QoS 2 PUBLISH packets MUST still accept SUBSCRIBE packets containing a Requested QoS of 0, 1 or 2. |
| [MQTT-3.2.2-11] | If a Client receives a Maximum QoS from a Server, it MUST NOT send PUBLISH packets at a QoS level exceeding the Maximum QoS level specified. |
| [MQTT-3.2.2-12] | If a Server receives a CONNECT packet containing a Will QoS that exceeds its capabilities, it MUST reject the connection. It SHOULD use a CONNACK packet with Reason Code 0x9B (QoS not supported) as described in section 4.13 Handling errors, and MUST close the Network Connection. |
| [MQTT-3.2.2-13] | If a Server receives a CONNECT packet containing a Will Message with the Will Retain 1, and it does not support retained messages, the Server MUST reject the connection request. It SHOULD send CONNACK with Reason Code 0x9A (Retain not supported) and then it MUST close the Network Connection. |
| [MQTT-3.2.2-14] | A Client receiving Retain Available from the Server MUST NOT send a PUBLISH packet with the RETAIN flag set to 1. |
| [MQTT-3.2.2-15] | The Client MUST NOT send packets exceeding Maximum Packet Size to the Server. |
| [MQTT-3.2.2-16] | If the Client connects using a zero length Client Identifier, the Server MUST respond with a CONNACK containing an Assigned Client Identifier. The Assigned Client Identifier MUST be a new Client Identifier not used by any other Session currently in the Server. |
| [MQTT-3.2.2-17] | The Client MUST NOT send a Topic Alias in a PUBLISH packet to the Server greater than this value. |
| [MQTT-3.2.2-18] | Topic Alias Maximum is absent, the Client MUST NOT send any Topic Aliases on to the Server. |
| [MQTT-3.2.2-19] | The Server MUST NOT send this property if it would increase the size of the CONNACK packet beyond the Maximum Packet Size specified by the Client. |
| [MQTT-3.2.2-20] | The Server MUST NOT send this property if it would increase the size of the CONNACK packet beyond the Maximum Packet Size specified by the Client. |
| [MQTT-3.2.2-21] | If the Server sends a Server Keep Alive on the CONNACK packet, the Client MUST use this value instead of the Keep Alive value the Client sent on CONNECT. |
| [MQTT-3.2.2-22] | If the Server does not send the Server Keep Alive, the Server MUST use the Keep Alive value set by the Client on CONNECT. |
| [MQTT-3.3.1-1] | The DUP flag MUST be set to 1 by the Client or Server when it attempts to re-deliver a PUBLISH packet. |
| [MQTT-3.3.1-2] | The DUP flag MUST be set to 0 for all QoS 0 messages. |
| [MQTT-3.3.1-3] | The DUP flag in the outgoing PUBLISH packet is set independently to the incoming PUBLISH packet, its value MUST be determined solely by whether the outgoing PUBLISH packet is a retransmission. |
| [MQTT-3.3.1-4] | A PUBLISH Packet MUST NOT have both QoS bits set to 1. |
| [MQTT-3.3.1-5] | If the RETAIN flag is set to 1 in a PUBLISH packet sent by a Client to a Server, the Server MUST replace any existing retained message for this topic and store the Application Message. |
| [MQTT-3.3.1-6] | If the Payload contains zero bytes it is processed normally by the Server but any retained message with the same topic name MUST be removed and any future subscribers for the topic will not receive a retained message. |
| [MQTT-3.3.1-7] | A retained message with a Payload containing zero bytes MUST NOT be stored as a retained message on the Server. |
| [MQTT-3.3.1-8] | If the RETAIN flag is 0 in a PUBLISH packet sent by a Client to a Server, the Server MUST NOT store the message as a retained message and MUST NOT remove or replace any existing retained message. |
| [MQTT-3.3.1-9] | If Retain Handling is set to 0 the Server MUST send the retained messages matching the Topic Filter of the subscription to the Client. |
| [MQTT-3.3.1-10] | If Retain Handling is set to 1 then if the subscription did already exist, the Server MUST send all retained message matching the Topic Filter of the subscription to the Client, and if the subscription did not exist, the Server MUST NOT send the retained messages. |
| [MQTT-3.3.1-11] | If Retain Handling is set to 2, the Server MUST NOT send the retained |
| [MQTT-3.3.1-12] | If the value of Retain As Published subscription option is set to 0, the Server MUST set the RETAIN flag to 0 when forwarding an Application Message regardless of how the RETAIN flag was set in the received PUBLISH packet. |
| [MQTT-3.3.1-13] | If the value of Retain As Published subscription option is set to 1, the Server MUST set the RETAIN flag equal to the RETAIN flag in the received PUBLISH packet. |
| [MQTT-3.3.2-1] | The Topic Name MUST be present as the first field in the PUBLISH packet Variable Header. It MUST be a UTF-8 Encoded String. |
| [MQTT-3.3.2-2] | The Topic Name in the PUBLISH packet MUST NOT contain wildcard characters. |
| [MQTT-3.3.2-3] | The Topic Name in a PUBLISH packet sent by a Server to a subscribing Client MUST match the Subscription’s Topic Filter. |
| [MQTT-3.3.2-4] | A Server MUST send the Payload Format Indicator unaltered to all subscribers receiving the message. |
| [MQTT-3.3.2-5] | If the Message Expiry Interval has passed and the Server has not managed to start onward delivery to a matching subscriber, then it MUST delete the copy of the message for that subscriber. |
| [MQTT-3.3.2-6] | The PUBLISH packet sent to a Client by the Server MUST contain a Message Expiry Interval set to the received value minus the time that the message has been waiting in the Server. |
| [MQTT-3.3.2-7] | A receiver MUST NOT carry forward any Topic Alias mappings from one Network Connection to another. |
| [MQTT-3.3.2-8] | A sender MUST NOT send a PUBLISH packet containing a Topic Alias which has the value 0. |
| [MQTT-3.3.2-9] | A Client MUST NOT send a PUBLISH packet with a Topic Alias greater than the Topic Alias Maximum value returned by the Server in the CONNACK packet. |
| [MQTT-3.3.2-10] | A Client MUST accept all Topic Alias values greater than 0 and less than or equal to the Topic Alias Maximum value that it sent in the CONNECT packet. |
| [MQTT-3.3.2-11] | A Server MUST NOT send a PUBLISH packet with a Topic Alias greater than the Topic Alias Maximum value sent by the Client in the CONNECT packet. |
| [MQTT-3.3.2-12] | A Server MUST accept all Topic Alias values greater than 0 and less than or equal to the Topic Alias Maximum value that it returned in the CONNACK packet. |
| [MQTT-3.3.2-13] | The Response Topic MUST be a UTF-8 Encoded String. |
| [MQTT-3.3.2-14] | The Response Topic MUST NOT contain wildcard characters. |
| [MQTT-3.3.2-15] | The Server MUST send the Response Topic unaltered to all subscribers receiving the Application Message. |
| [MQTT-3.3.2-16] | The Server MUST send the Correlation Data unaltered to all subscribers receiving the Application Message. |
| [MQTT-3.3.2-17] | The Server MUST send all User Properties unaltered in a PUBLISH packet when forwarding the Application Message to a Client. |
| [MQTT-3.3.2-18] | The Server MUST maintain the order of User Properties when forwarding the Application Message. |
| [MQTT-3.3.2-19] | The Content Type MUST be a UTF-8 Encoded String. |
| [MQTT-3.3.2-20] | A Server MUST send the Content Type unaltered to all subscribers receiving the Application Message. |
| [MQTT-3.3.4-1] | The receiver of a PUBLISH Packet MUST respond with the packet as determined by the QoS in the PUBLISH Packet. |
| [MQTT-3.3.4-2] | In this case the Server MUST deliver the message to the Client respecting the maximum QoS of all the matching subscriptions. |
| [MQTT-3.3.4-3] | If the Client specified a Subscription Identifier for any of the overlapping subscriptions the Server MUST send those Subscription Identifiers in the message which is published as the result of the subscriptions. |
| [MQTT-3.3.4-4] | If the Server sends a single copy of the message it MUST include in the PUBLISH packet the Subscription Identifiers for all matching subscriptions which have a Subscription Identifiers, their order is not significant. |
| [MQTT-3.3.4-5] | If the Server sends multiple PUBLISH packets it MUST send, in each of them, the Subscription Identifier of the matching subscription if it has a Subscription Identifier. |
| [MQTT-3.3.4-6] | A PUBLISH packet sent from a Client to a Server MUST NOT contain a Subscription Identifier. |
| [MQTT-3.3.4-7] | The Client MUST NOT send more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets for which it has not received PUBACK, PUBCOMP, or PUBREC with a Reason Code of 128 or greater from the Server. |
| [MQTT-3.3.4-8] | The Client MUST NOT delay the sending of any packets other than PUBLISH packets due to having sent Receive Maximum PUBLISH packets without receiving acknowledgements for them. |
| [MQTT-3.3.4-9] | The Server MUST NOT send more than Receive Maximum QoS 1 and QoS 2 PUBLISH packets for which it has not received PUBACK, PUBCOMP, or PUBREC with a Reason Code of 128 or greater from the Client. |
| [MQTT-3.3.4-10] | The Server MUST NOT delay the sending of any packets other than PUBLISH packets due to having sent Receive Maximum PUBLISH packets without receiving acknowledgements for them. |
| [MQTT-3.4.2-1] | The Client or Server sending the PUBACK packet MUST use one of the PUBACK Reason Codes. |
| [MQTT-3.4.2-2] | The sender MUST NOT send this property if it would increase the size of the PUBACK packet beyond the Maximum Packet Size specified by the receiver. |
| [MQTT-3.4.2-3] | The sender MUST NOT send this property if it would increase the size of the PUBACK packet beyond the Maximum Packet Size specified by the receiver. |
| [MQTT-3.5.2-1] | The Client or Server sending the PUBREC packet MUST use one of the PUBREC Reason Codes. |
| [MQTT-3.5.2-2] | The sender MUST NOT send this property if it would increase the size of the PUBREC packet beyond the Maximum Packet Size specified by the receiver. |
| [MQTT-3.5.2-3] | The sender MUST NOT send this property if it would increase the size of the PUBREC packet beyond the Maximum Packet Size specified by the receiver. |
| [MQTT-3.6.1-1] | Bits 3,2,1 and 0 of the Fixed Header in the PUBREL packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network Connection. |
| [MQTT-3.6.2-1] | The Client or Server sending the PUBREL packet MUST use one of the PUBREL Reason Codes. |
| [MQTT-3.6.2-2] | The sender MUST NOT send this Property if it would increase the size of the PUBREL packet beyond the Maximum Packet Size specified by the receiver. |
| [MQTT-3.6.2-3] | The sender MUST NOT send this property if it would increase the size of the PUBREL packet beyond the Maximum Packet Size specified by the receiver. |
| [MQTT-3.7.2-1] | The Client or Server sending the PUBCOMP packets MUST use one of the PUBCOMP Reason Codes. |
| [MQTT-3.7.2-2] | The sender MUST NOT use this Property if it would increase the size of the PUBCOMP packet beyond the Maximum Packet Size specified by the receiver. |
| [MQTT-3.7.2-3] | The sender MUST NOT send this property if it would increase the size of the PUBCOMP packet beyond the Maximum Packet Size specified by receiver. |
| [MQTT-3.8.1-1] | Bits 3,2,1 and 0 of the Fixed Header of the SUBSCRIBE packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network Connection |
| [MQTT-3.8.3-1] | The Topic Filters MUST be a UTF-8 Encoded String. |
| [MQTT-3.8.3-2] | The Payload MUST contain at least one Topic Filter and Subscription Options pair. |
| [MQTT-3.8.3-3] | Bit 2 of the Subscription Options represents the No Local option. If the value is 1, Application Messages MUST NOT be forwarded to a connection with a ClientID equal to the ClientID of the publishing connection. |
| [MQTT-3.8.3-4] | It is a Protocol Error to set the No Local bit to 1 on a Shared Subscription. |
| [MQTT-3.8.3-5] | The Server MUST treat a SUBSCRIBE packet as malformed if any of Reserved bits in the Payload are non-zero. |
| [MQTT-3.8.4-1] | When the Server receives a SUBSCRIBE packet from a Client, the Server MUST respond with a SUBACK packet. |
| [MQTT-3.8.4-2] | The SUBACK packet MUST have the same Packet Identifier as the SUBSCRIBE packet that it is acknowledging. |
| [MQTT-3.8.4-3] | If a Server receives a SUBSCRIBE packet containing a Topic Filter that is identical to a Non‑shared Subscription’s Topic Filter for the current Session then it MUST replace that existing Subscription with a new Subscription. |
| [MQTT-3.8.4-4] | If the Retain Handling option is 0, any existing retained messages matching the Topic Filter MUST be re-sent, but Application Messages MUST NOT be lost due to replacing the Subscription. |
| [MQTT-3.8.4-5] | If a Server receives a SUBSCRIBE packet that contains multiple Topic Filters it MUST handle that packet as if it had received a sequence of multiple SUBSCRIBE packets, except that it combines their responses into a single SUBACK response. |
| [MQTT-3.8.4-6] | The SUBACK packet sent by the Server to the Client MUST contain a Reason Code for each Topic Filter/Subscription Option pair. |
| [MQTT-3.8.4-7] | This Reason Code MUST either show the maximum QoS that was granted for that Subscription or indicate that the subscription failed. |
| [MQTT-3.8.4-8] | The QoS of Payload Messages sent in response to a Subscription MUST be the minimum of the QoS of the originally published message and the Maximum QoS granted by the Server. |
| [MQTT-3.9.2-1] | The Server MUST NOT send this Property if it would increase the size of the SUBACK packet beyond the Maximum Packet Size specified by the Client. |
| [MQTT-3.9.2-2] | The Server MUST NOT send this property if it would increase the size of the SUBACK packet beyond the Maximum Packet Size specified by the Client. |
| [MQTT-3.9.3-1] | The order of Reason Codes in the SUBACK packet MUST match the order of Topic Filters in the SUBSCRIBE packet. |
| [MQTT-3.9.3-2] | The Server sending the SUBACK packet MUST send one of the Subscribe Reason Code values for each Topic Filter received. |
| [MQTT-3.10.1-1] | Bits 3,2,1 and 0 of the Fixed Header of the UNSUBSCRIBE packet are reserved and MUST be set to 0,0,1 and 0 respectively. The Server MUST treat any other value as malformed and close the Network Connection |
| [MQTT-3.10.3-1] | The Topic Filters in an UNSUBSCRIBE packet MUST be UTF-8 Encoded Strings. |
| [MQTT-3.10.3-2] | The Payload of an UNSUBSCRIBE packet MUST contain at least one Topic Filter. |
| [MQTT-3.10.4-1] | The Topic Filters (whether they contain wildcards or not) supplied in an UNSUBSCRIBE packet MUST be compared character-by-character with the current set of Topic Filters held by the Server for the Client. If any filter matches exactly then its owning Subscription MUST be deleted. |
| [MQTT-3.10.4-2] | When a Server receives UNSUBSCRIBE It MUST stop adding any new messages which match the Topic Filters, for delivery to the Client. |
| [MQTT-3.10.4-3] | When a Server receives UNSUBSCRIBE It MUST complete the delivery of any QoS 1 or QoS 2 messages which match the Topic Filters and it has started to send to the Client. |
| [MQTT-3.10.4-4] | The Server MUST respond to an UNSUBSCRIBE request by sending an UNSUBACK packet. |
| [MQTT-3.10.4-5] | The UNSUBACK packet MUST have the same Packet Identifier as the UNSUBSCRIBE packet. Even where no Topic Subscriptions are deleted, the Server MUST respond with an UNSUBACK. |
| [MQTT-3.10.4-6] | If a Server receives an UNSUBSCRIBE packet that contains multiple Topic Filters, it MUST process that packet as if it had received a sequence of multiple UNSUBSCRIBE packets, except that it sends just one UNSUBACK response. |
| [MQTT-3.11.2-1] | The Server MUST NOT send this Property if it would increase the size of the UNSUBACK packet beyond the Maximum Packet Size specified by the Client. |
| [MQTT-3.11.2-2] | The Server MUST NOT send this property if it would increase the size of the UNSUBACK packet beyond the Maximum Packet Size specified by the receiver. |
| [MQTT-3.11.3-1] | The order of Reason Codes in the UNSUBACK packet MUST match the order of Topic Filters in the UNSUBSCRIBE packet. |
| [MQTT-3.11.3-2] | The Server sending the UNSUBACK packet MUST use one of the UNSUBSCRIBE Reason Code values for each Topic Filter received. |
| [MQTT-3.12.4-1] | The Server MUST send a PINGRESP packet in response to a PINGREQ packet. |
| [MQTT-3.14.0-1] | A Server MUST NOT send a DISCONNECT until after it has sent a CONNACK with Reason Code of less than 0x80. |
| [MQTT-3.14.1-1] | The Client or Server MUST validate that reserved bits are set to 0. If they are not zero it sends a DISCONNECT packet with a Reason code of 0x81 (Malformed Packet). |
| [MQTT-3.14.2-1] | The Client or Server sending the DISCONNECT packet MUST use one of the DISCONNECT Reason Codes. |
| [MQTT-3.14.2-2] | The Session Expiry Interval MUST NOT be sent on a DISCONNECT by the Server. |
| [MQTT-3.14.2-3] | The sender MUST NOT use this Property if it would increase the size of the DISCONNECT packet beyond the Maximum Packet Size specified by the receiver. |
| [MQTT-3.14.2-4] | The sender MUST NOT send this property if it would increase the size of the DISCONNECT packet beyond the Maximum Packet Size specified by the receiver. |
| [MQTT-3.14.4-1] | After sending a DISCONNECT packet the sender MUST NOT send any more MQTT Control Packets on that Network Connection. |
| [MQTT-3.14.4-2] | After sending a DISCONNECT packet the sender MUST close the Network Connection. |
| [MQTT-3.14.4-3] | On receipt of DISCONNECT with a Reason Code of 0x00 (Success) the Server MUST discard any Will Message associated with the current Connection without publishing it. |
| [MQTT-3.15.1-1] | Bits 3,2,1 and 0 of the Fixed Header of the AUTH packet are reserved and MUST all be set to 0. The Client or Server MUST treat any other value as malformed and close the Network Connection. |
| [MQTT-3.15.2-1] | The sender of the AUTH Packet MUST use one of the Authenticate Reason Codes. |
| [MQTT-3.15.2-2] | The sender MUST NOT send this property if it would increase the size of the AUTH packet beyond the Maximum Packet Size specified by the receiver |
| [MQTT-3.15.2-3] | The sender MUST NOT send this property if it would increase the size of the AUTH packet beyond the Maximum Packet Size specified by the receiver. |
| [MQTT-4.1.0-1] | The Client and Server MUST NOT discard the Session State while the Network Connection is open. |
| [MQTT-4.2.0-1] | A Client or Server MUST support the use of one or more underlying transport protocols that provide an ordered, lossless, stream of bytes from the Client to Server and Server to Client. |
| [MQTT-4.1.0-2] | The Server MUST discard the Session State when the Network Connection is closed and the Session Expiry Interval has passed. |
| [MQTT-4.3.1-1] | In the QoS 0 delivery protocol, the sender MUST send a PUBLISH packet with QoS 0 and DUP flag set to 0. |
| [MQTT-4.3.2-1] | In the QoS 1 delivery protocol, the sender MUST assign an unused Packet Identifier each time it has a new Application Message to publish. |
| [MQTT-4.3.2-2] | In the QoS 1 delivery protocol, the sender MUST send a PUBLISH packet containing this Packet Identifier with QoS 1 and DUP flag set to 0. |
| [MQTT-4.3.2-3] | In the QoS 1 delivery protocol, the sender MUST treat the PUBLISH packet as “unacknowledged” until it has received the corresponding PUBACK packet from the receiver. |
| [MQTT-4.3.2-4] | In the QoS 1 delivery protocol, the receiver MUST respond with a PUBACK packet containing the Packet Identifier from the incoming PUBLISH packet, having accepted ownership of the Application Message. |
| [MQTT-4.3.2-5] | In the QoS 1 delivery protocol, the receiver after it has sent a PUBACK packet the receiver MUST treat any incoming PUBLISH packet that contains the same Packet Identifier as being a new Application Message, irrespective of the setting of its DUP flag. |
| [MQTT-4.3.3-1] | In the QoS 2 delivery protocol, the sender MUST assign an unused Packet Identifier when it has a new Application Message to publish. |
| [MQTT-4.3.3-2] | In the QoS 2 delivery protocol, the sender MUST send a PUBLISH packet containing this Packet Identifier with QoS 2 and DUP flag set to 0. |
| [MQTT-4.3.3-3] | In the QoS 2 delivery protocol, the sender MUST treat the PUBLISH packet as “unacknowledged” until it has received the corresponding PUBREC packet from the receiver. |
| [MQTT-4.3.3-4] | In the QoS 2 delivery protocol, the sender MUST send a PUBREL packet when it receives a PUBREC packet from the receiver with a Reason Code value less than 0x80. This PUBREL packet MUST contain the same Packet Identifier as the original PUBLISH packet. |
| [MQTT-4.3.3-5] | In the QoS 2 delivery protocol, the sender MUST treat the PUBREL packet as “unacknowledged” until it has received the corresponding PUBCOMP packet from the receiver. |
| [MQTT-4.3.3-6] | In the QoS 2 delivery protocol, the sender MUST NOT re-send the PUBLISH once it has sent the corresponding PUBREL packet. |
| [MQTT-4.3.3-7] | In the QoS 2 delivery protocol, the sender MUST NOT apply Application Message expiry if a PUBLISH packet has been sent. |
| [MQTT-4.3.3-8] | In the QoS 2 delivery protocol, the receiver MUST respond with a PUBREC containing the Packet Identifier from the incoming PUBLISH packet, having accepted ownership of the Application Message. |
| [MQTT-4.3.3-9] | In the QoS 2 delivery protocol, the receiver if it has sent a PUBREC with a Reason Code of 0x80 or greater, the receiver MUST treat any subsequent PUBLISH packet that contains that Packet Identifier as being a new Application Message. |
| [MQTT-4.3.3-10] | In the QoS 2 delivery protocol, the receiver until it has received the corresponding PUBREL packet, the receiver MUST acknowledge any subsequent PUBLISH packet with the same Packet Identifier by sending a PUBREC. It MUST NOT cause duplicate messages to be delivered to any onward recipients in this case. |
| [MQTT-4.3.3-11] | In the QoS 2 delivery protocol, the receiver MUST respond to a PUBREL packet by sending a PUBCOMP packet containing the same Packet Identifier as the PUBREL. |
| [MQTT-4.3.3-12] | In the QoS 2 delivery protocol, the receiver After it has sent a PUBCOMP, the receiver MUST treat any subsequent PUBLISH packet that contains that Packet Identifier as being a new Application Message. |
| [MQTT-4.3.3-13] | In the QoS 2 delivery protocol, the receiver MUST continue the QoS 2 acknowledgement sequence even if it has applied Application Message expiry. |
| [MQTT-4.4.0-1] | When a Client reconnects with Clean Start set to 0 and a session is present, both the Client and Server MUST resend any unacknowledged PUBLISH packets (where QoS > 0) and PUBREL packets using their original Packet Identifiers. This is the only circumstance where a Client or Server is REQUIRED to resend messages. Clients and Servers MUST NOT resend messages at any other time. |
| [MQTT-4.4.0-2] | If PUBACK or PUBREC is received containing a Reason Code of 0x80 or greater the corresponding PUBLISH packet is treated as acknowledged, and MUST NOT be retransmitted. |
| [MQTT-4.5.0-1] | When a Server takes ownership of an incoming Application Message it MUST add it to the Session State for those Clients that have matching Subscriptions. |
| [MQTT-4.5.0-2] | The Client MUST acknowledge any Publish packet it receives according to the applicable QoS rules regardless of whether it elects to process the Application Message that it contains. |
| [MQTT-4.6.0-1] | When the Client re-sends any PUBLISH packets, it MUST re-send them in the order in which the original PUBLISH packets were sent (this applies to QoS 1 and QoS 2 messages). |
| [MQTT-4.6.0-2] | The Client MUST send PUBACK packets in the order in which the corresponding PUBLISH packets were received (QoS 1 messages). |
| [MQTT-4.6.0-3] | The Client MUST send PUBREC packets in the order in which the corresponding PUBLISH packets were received (QoS 2 messages). |
| [MQTT-4.6.0-4] | The Client MUST send PUBREL packets in the order in which the corresponding PUBREC packets were received (QoS 2 messages). |
| [MQTT-4.6.0-5] | When a Server processes a message that has been published to an Ordered Topic, it MUST send PUBLISH packets to consumers (for the same Topic and QoS) in the order that they were received from any given Client. |
| [MQTT-4.6.0-6] | A Server MUST treat every, Topic as an Ordered Topic when it is forwarding messages on Non‑shared Subscriptions. |
| [MQTT-4.7.0-1] | The wildcard characters can be used in Topic Filters, but MUST NOT be used within a Topic Name. |
| [MQTT-4.7.1-1] | The multi-level wildcard character MUST be specified either on its own or following a topic level separator. In either case it MUST be the last character specified in the Topic Filter. |
| [MQTT-4.7.1-2] | The single-level wildcard can be used at any level in the Topic Filter, including first and last levels. Where it is used, it MUST occupy an entire level of the filter. |
| [MQTT-4.7.2-1] | The Server MUST NOT match Topic Filters starting with a wildcard character (# or +) with Topic Names beginning with a $ character. |
| [MQTT-4.7.3-1] | All Topic Names and Topic Filters MUST be at least one character long. |
| [MQTT-4.7.3-2] | Topic Names and Topic Filters MUST NOT include the null character (Unicode U+0000). |
| [MQTT-4.7.3-3] | Topic Names and Topic Filters are UTF-8 Encoded Strings; they MUST NOT encode to more than 65,535 bytes. |
| [MQTT-4.7.3-4] | When it performs subscription matching the Server MUST NOT perform any normalization of Topic Names or Topic Filters, or any modification or substitution of unrecognized characters. |
| [MQTT-4.8.2-1] | A Shared Subscription's Topic Filter MUST start with $share/ and MUST contain a ShareName that is at least one character long. |
| [MQTT-4.8.2-2] | The ShareName MUST NOT contain the characters "/", "+" or "#", but MUST be followed by a "/" character. This "/" character MUST be followed by a Topic Filter. |
| [MQTT-4.8.2-3] | The Server MUST respect the granted QoS for the Clients subscription. |
| [MQTT-4.8.2-4] | The Server MUST complete the delivery of the message to that Client when it reconnects. |
| [MQTT-4.8.2-5] | If the Clients Session terminates before the Client reconnects, the Server MUST NOT send the Application Message to any other subscribed Client. |
| [MQTT-4.8.2-6] | If a Client responds with a PUBACK or PUBREC containing a Reason Code of 0x80 or greater to a PUBLISH packet from the Server, the Server MUST discard the Application Message and not attempt to send it to any other Subscriber. |
| [MQTT-4.9.0-1] | The Client or Server MUST set its initial send quota to a non-zero value not exceeding the Receive Maximum. |
| [MQTT-4.9.0-2] | Each time the Client or Server sends a PUBLISH packet at QoS > 0, it decrements the send quota. If the send quota reaches zero, the Client or Server MUST NOT send any more PUBLISH packets with QoS > 0. |
| [MQTT-4.9.0-3] | The Client and Server MUST continue to process and respond to all other MQTT Control Packets even if the quota is zero. |
| [MQTT-4.12.0-1] | If the Server does not support the Authentication Method supplied by the Client, it MAY send a CONNACK with a Reason Code of 0x8C (Bad authentication method) or 0x87 (Not Authorized) as described in section 4.13 and MUST close the Network Connection. |
| [MQTT-4.12.0-2] | If the Server requires additional information to complete the authorization, it can send an AUTH packet to the Client. This packet MUST contain a Reason Code of 0x18 (Continue authentication). |
| [MQTT-4.12.0-3] | The Client responds to an AUTH packet from the Server by sending a further AUTH packet. This packet MUST contain a Reason Code of 0x18 (Continue authentication). |
| [MQTT-4.12.0-4] | The Server can reject the authentication at any point in this process. It MAY send a CONNACK with a Reason Code of 0x80 or above as described in section 4.13, and MUST close the Network Connection. |
| [MQTT-4.12.0-5] | If the initial CONNECT packet included an Authentication Method property then all AUTH packets, and any successful CONNACK packet MUST include an Authentication Method Property with the same value as in the CONNECT packet. |
| [MQTT-4.12.0-6] | If the Client does not include an Authentication Method in the CONNECT, the Server MUST NOT send an AUTH packet, and it MUST NOT send an Authentication Method in the CONNACK packet. |
| [MQTT-4.12.0-7] | If the Client does not include an Authentication Method in the CONNECT, the Client MUST NOT send an AUTH packet to the Server. |
| [MQTT-4.12.1-1] | If the Client supplied an Authentication Method in the CONNECT packet it can initiate a re-authentication at any time after receiving a CONNACK. It does this by sending an AUTH packet with a Reason Code of 0x19 (Re-authentication). The Client MUST set the Authentication Method to the same value as the Authentication Method originally used to authenticate the Network Connection. |
| [MQTT-4.12.1-2] | If the re-authentication fails, the Client or Server SHOULD send DISCONNECT with an appropriate Reason Code and MUST close the Network Connection. |
| [MQTT-4.13.1-1] | When a Server detects a Malformed Packet or Protocol Error, and a Reason Code is given in the specification, it MUST close the Network Connection. |
| [MQTT-4.13.2-1] | The CONNACK and DISCONNECT packets allow a Reason Code of 0x80 or greater to indicate that the Network Connection will be closed. If a Reason Code of 0x80 or greater is specified, then the Network Connection MUST be closed whether or not the CONNACK or DISCONNECT is sent. |
| [MQTT-6.0.0-1] | MQTT Control Packets MUST be sent in WebSocket binary data frames. If any other type of data frame is received the recipient MUST close the Network Connection. |
| [MQTT-6.0.0-2] | A single WebSocket data frame can contain multiple or partial MQTT Control Packets. The receiver MUST NOT assume that MQTT Control Packets are aligned on WebSocket frame boundaries. |
| [MQTT-6.0.0-3] | The Client MUST include “mqtt” in the list of WebSocket Sub Protocols it offers. |
| [MQTT-6.0.0-4] | The WebSocket Subprotocol name selected and returned by the Server MUST be “mqtt”. |

1. Summary of new features in MQTT v5.0 (non-normative)

The following new features are added to MQTT v5.0

* Session expiry

Split the Clean Session flag into a Clean Start flag which indicates that the session should start without using an existing session, and a Session Expiry interval which says how long to retain the session after a disconnect. The session expiry interval can be modified at disconnect. Setting of Clean Start to 1 and Session Expiry Interval to 0 is equivalent in MQTT v3.1.1 of setting Clean Session to 1.

* Message expiry

Allow an expiry interval to be set when a message is published.

* Reason code on all ACKs

Change all response packets to contain a reason code. This include CONNACK, PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK, DISCONNECT, and AUTH. This allows the invoker to determine whether the requested function succeeded.

* Reason string on all ACKs

Change most packets with a reason code to also allow an optional reason string. This is designed for problem determination and is not intended to be parsed by the receiver.

* Server disconnect

Allow DISCONNECT to be sent by the Server to indicate the reason the connection is closed.

* Payload format and content type

Allow the payload format (binary, text) and a MIME style content type to be specified when a message is published. These are forwarded on to the receiver of the message.

* Request / Response

Formalize the request/response pattern within MQTT and provide the Response Topic and Correlation Data properties to allow response messages to be routed back to the publisher of a request. Also, add the ability for the Client to get configuration information from the Server about how to construct the response topics.

* Shared Subscriptions

Add shared subscription support allowing for load balanced consumers of a subscription

* Subscription ID

Allow a numeric subscription identifier to be specified on a SUBSCRIBE, and returned on the message when it is delivered. This allows the Client to determine which subscription or subscriptions caused the message to be delivered.

* Topic Alias

Decrease the size of the MQTT packet overhead by allowing the topic name to be abbreviated to a small integer. The Client and Server independently specify how many topic aliases they allow.

* Flow control

Allow the Client and Server to independently specify the number of outstanding reliable messages (QoS>0) they allow. The sender pauses sending such messages to stay below this quota. This is used to limit the rate of reliable messages, and to limit how many are in flight at one time.

* User properties

Add User Properties to most packets. User properties on PUBLISH are included with the message and are defined by the Client applications. The user properties on PUBLISH and Will Properties are forwarded by the Server to the receiver of the message. User properties on the CONNECT, SUBSCRIBE, and UNSUBSCRIBE packets are defined by the Server implementation. The user properties on CONNACK PUBACK, PUBREC, PUBREL, PUBCOMP, SUBACK, UNSUBACK and AUTH packets are defined by the sender, and are unique to the sender implementation. The meaning of user properties is not defined by MQTT.

* Maximum Packet Size

Allow the Client and Server to independently specify the maximum packet size they support. It is an error for the session partner to send a larger packet.

* Optional Server feature availability

Define a set of features which the Server does not allow and provide a mechanism for the Server to specify this to the Client. The features which can be specified in this way are: Maximum QoS, Retain Available, Wildcard Subscription Available, Subscription Identifier Available, and Shared Subscription Available. It is an error for the Client to use features that the Server has declared are not available.

It is possible in earlier versions of MQTT for a Server to not implement a feature by declaring that the Client is not authorized for that function. This feature allows such optional behavior to be declared and adds specific Reason Codes when the Client uses one of these features anyway.

* Enhanced authentication

Provide a mechanism to enable challenge/response style authentication including mutual authentication. This allows SASL style authentication to be used if supported by both Client and Server, and includes the ability for a Client to re-authenticate within a connection.

* Subscription options

Provide subscription options primarily defined to allow for message bridge applications. These include an option to not send messages originating on this Client (noLocal), and options for handling retained messages on subscribe.

* Will delay

Add the ability to specify a delay between the end of the connection and sending the will message. This is designed so that if a connection to the session is re-established then the will message is not sent. This allows for brief interruptions of the connection without notification to others.

* Server Keep Alive

Allow the Server to specify the value it wishes the Client to use as a keep alive. This allows the Server to set a maximum allowed keepalive and still have the Client honor it.

* Assigned ClientID

In cases where the ClientID is assigned by the Server, return the assigned ClientID. This also lifts the restriction that Server assigned ClientIDs can only be used with Clean Session=1 connections.

* Server reference

Allow the Server to specify an alternate Server to use on CONNACK or DISCONNECT. This can be used as a redirect or to do provisioning.