Backwards Compatibility

- MQTT is an asymmetric protocol, with distinct client and server implementations. Clients can be quite simple, servers tend to be more complex. There is a wide range of client implementations available, compared with a smaller number of server implementations, and there are many more client instances deployed than servers
- By “Backwards Compatibility” we chiefly mean the ability of a client coded to speak an older version of the protocol being able to connect to, and successfully use, a server that implements a newer version of the protocol
  - This is automatic if the new version of the protocol is a true superset of the older one, otherwise it only happens if a server supports both versions.
- “Backwards compatibility” might also mean the ability of a newer client to work with an older server provided it doesn't use any features that aren't available in the older protocol version.
- “Full interoperability” means that old clients will work with new servers, and that new clients will work with old servers. This implies that the protocol doesn't change at all.
- Up to now, there have only been two widely-used versions of MQTT, namely MQTT v3 and MQTT v3.1. They differ only in that 3.1 has an optional username/password in the Connect command, so MQTT v3.1 is backwards compatible (in both senses) with v3.

Assertion in the Charter: The standardised MQTT 1.0 doesn’t have to be fully interoperable with today’s MQTT v3.1 but it must be straightforward for server implementors to provide the first form of backwards compatibility.

Compatibility statements from the TC's Charter

“Changes to the input document, other than editorial changes and other points of clarification, will be limited to the Connect command, and should be backward compatible with implementations of previous versions of the specification such that a client coded to speak an older version of the protocol will be able to connect to, and successfully use, a server that implements a newer version of the protocol....A goal of the TC is to minimize disruption to existing implementations, making it straightforward to support both the Version 3.1 of the MQTT specification and the OASIS standard.”

This permits the following:

1) Tightening up the words in the spec, clarifying issues of interpretation, identifying compliance requirements and the like
    - This would give full interoperability, except in corner cases where implementations have interpreted the v3.1 input spec in different ways
2) Changes to the Protocol Name and/or Version that is supplied on the CONNECT command
    - This would mean that a new server wanting to support 3.1 clients would have to CONNECT commands with both Name/Version combinations. This would be useful if there are cases where the server implementation has had to be changed to conform to a clarification in the spec
    - A new client would only connect to an old server in cases where that server didn't validate the Protocol Name or Version
3) Other minor changes to the CONNECT command would appear to be permitted, should we wish to make them

MQTT 3.1 command format - Connect

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

Protocol Name and Version number									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (6)	0	0	0	0	0	1	1	0
byte 3	'M'	0	1	0	0	1	1	0	1
byte 4	'Q'	0	1	0	1	0	0	0	1
byte 5	'I'	0	1	0	0	1	0	0	1
byte 6	's'	0	1	1	1	0	0	1	1
byte 7	'd'	0	1	1	0	0	1	0	0
byte 8	'p'	0	1	1	1	0	0	0	0
byte 9	Version (3)	0	0	0	0	0	0	1	1
Connect Flags									
byte 10	User name flag (1) Password flag (1) Will RETAIN (0) Will QoS (01) Will flag (1) Clean Session (1)	1	1	0	0	1	1	1	x
Keep Alive timer									
byte 11	KeepAlive MSB (0)	0	0	0	0	0	0	0	0
byte 12	KeepAlive LSB (10)	0	0	0	0	1	0	1	0
Payload fields									

Fixed Header
Same format for all message types

Variable header
Format differs by message type – but has a fixed structure for each type

In the CONNECT message, these flags are used to specify the fields that are present in the payload (if present they must appear in a given order).

In CONNECT the Payload is used to carry additional parameters, in the form of length-prefixed strings. Could be extended to include self-defining parameters.