G2 MQTT Protocol User's Guide

Gensym Corporation

Version Gensym.2018.1, March 2018

Table of Contents

Preface . . . 3
About this guide . . . 3
Audience . . . 3
Conventions . . . 3
Typographic . . . 3
Procedure Signatures . . . 4
Related Documentation . . . 4
Customer Support Services . . . 6
1. MQTT protocol . . . 9
1.1. Introduction . . . 9
1.2. Characteristics of MQTT . . . 9
1.3. MQTT protocol specification . . . 9
1.4. MQTT client . . . 10
1.5. Protocol options . . . 10
1.5.1. Explicit options . . . 10
1.5.2. Message/packet buffer . . . 10
1.5.3. When to deliver QoS=2 message . . . 11
1.5.4. Error handling . . . 11
1.5.5. MQTT under-specified . . . 13
1.6. Limits . . . 14
2. MQTT Knowledge Base . . . 17
2.1. Introduction . . . 17
2.2. Usage . . . 17
2.2.1. Startup/shutdown . . . 17
2.2.2. Sending messages . . . 17
2.2.3. Receiving messages . . . 17
2.3. Logging . . . 18
3. User API Reference . . . 19
3.1. Classes . . . 19
3.1.1. MQTT-SERVER . . . 19
3.2. Procedures . . . 21
3.2.1. G2-MQTT-DISCONNECT . . . 21
3.2.2. G2-MQTT-UNSUBSCRIBE . . . 21
3.2.3. G2-MQTT-SUBSCRIBE . . . 21
3.2.4. G2-MQTT-RECONNECT . . . 22

3.2.5. G2-MQTT-CONNECT 22
3.2.6. G2-MQTT-PUBLISH-EXACTLY-ONCE 23
3.2.7. G2-MQTT-PUBLISH-AT-LEAST-ONCE 23
3.2.8. G2-MQTT-PUBLISH-AT-MOST-ONCE 24
3.2.9. G2-MQTT-GET-UNCONFIRMED-MESSAGES 24
3.2.10. G2-MQTT-GET-UNCONFIRMED-SUBSCRIBES 25
3.2.11. G2-MQTT-GET-UNCONFIRMED-UNSUBSCRIBES 25

G2 MQTT Protocol User's Guide, Version Gensym.2018.1

Gensym Corporation
401 Congress Avenue
Suite 2650
Austin, Texas USA 78701
Telephone: (781) 265-7100

Preface

About this guide

This guide contains complete information about the G2 MQTT Protocol client implementation. This guide is designed to help users communicate with other nodes of a network using the MQTT protocol.

Audience

To understand and use this guide, you must be familiar with the G2 real-time expert system. In addition, you should be familiar with basics of the MQTT protocol and networking (TCP/IP) in general. If you encounter G2 terms or concepts that you do not understand, refer to the *G2 Reference Manual*.

Conventions

This guide uses the following typographic conventions and conventions for defining system procedures.

Typographic

Convention Examples	Description
g2-window, g2-window-1, ws-top-level, sys-mod	User-defined and system-defined G2 class names, instance names, workspace names, and module names
history-keeping-spec, temperature	User-defined and system-defined G2 attribute names
true, 1.234, ok, “Burlington, MA”	G2 attribute values and values specified or viewed through dialogs
Main Menu > Start KB Workspace > New Object create subworkspace Start Procedure	G2 menu choices and button labels
conclude that the x of y . .	Text of G2 procedures, methods, functions, formulas, and expressions
new-argument	User-specified values in syntax descriptions
text-string	Return values of G2 procedures and methods in syntax descriptions

Convention Examples	Description
File Name, OK, Apply, Cancel, General, Edit Scroll Area	GUIDE and native dialog fields, button labels, tabs, and titles
File > Save Properties	GMS and native menu choices
workspace	Glossary terms
`c:\Program Files\Gensym\`	Windows pathnames
`/usr/gensym/g2/kbs`	UNIX pathnames
`spreadsh.kb`	File names
`g2 -kb top.kb`	Operating system commands
`public void main()` `gsi_start`	Java, C and all other external c

Syntax conventions are fully described in the G2 Reference Manual.

Procedure Signatures

A procedure signature is a complete syntactic summary of a procedure or method. A procedure signature shows values supplied by the user in italics, and the value (if any) returned by the procedure underlined. Each value is followed by its type:

g2-clone-and-transfer-objects
(*list*: class item-list, *to-workspace*: class kb-workspace,*delta-x*: integer, *delta-y*: integer)
→ transferred-items: g2-list

Related Documentation

Integrity

- Integrity Relelease Notes
- Integrity Demo Guide
- Integrity User's Guide
- Integrity Reference Manual
- Integrity with AutoDiscovery User's Guide
- SymCure User's Guide
- Integrity Utililies Guide
- DXI3DB-Primer
- G2-SNMP Bridges Installation and User's Guide
- Integrity G2/Java Socket Manager User's Guide

- Integrity SNMP User's Guide

G2 Core Technology

- G2 Bundle Release Notes
- Getting Started with G2 Tutorials
- G2 Reference Manual
- G2 Language Reference Card
- G2 Developer's Guide
- G2 System Procedures Reference Manual
- G2 System Procedures Reference Card
- G2 Class Reference Manual
- Telewindows User's Guide
- G2 Gateway Bridge Developer's Guide

G2 Utilities

- G2 ProTools User's Guide
- G2 Foundation Resources User's Guide
- G2 Menu System User's Guide
- G2 XL Spreadsheet User's Guide
- G2 Dynamic Displays User's Guide
- G2 Developer's Interface User's Guide
- G2 OnLine Documentation Developer's Guide
- G2 OnLine Documentation User's Guide
- G2 GUIDE User's Guide
- G2 GUIDE/UIL Procedures Reference Manual

G2 Developers' Utilities

- Business Process Management System User's Guide
- Business Rules Management System User's Guide
- G2 Reporting Engine User's Guide
- G2 Web User's Guide
- G2 Event and Data Processing User's Guide
- G2 Run-Time Library User's Guide
- G2 Event Manager User's Guide

- G2 Dialog Utility User's Guide
- G2 Data Source Manager User's Guide
- G2 Data Point Manager User's Guide
- G2 Engineering Unit Conversion User's Guide
- G2 Error Handling Foundation User's Guide
- G2 Relation Browser User's Guide

Bridges and External Systems

- G2 ActiveXLink User's Guide
- G2 CORBALink User's Guide
- G2 Database Bridge User's Guide
- G2-ODBC Bridge Release Notes
- G2-Oracle Bridge Release Notes
- G2-Sybase Bridge Release Notes
- G2 JMail Bridge User's Guide
- G2 Java Socket Manager User's Guide
- G2 JMSLink User's Guide
- G2 OPCLink User's Guide
- G2-PI Bridge User's Guide
- G2-SNMP Bridge User's Guide
- G2 WebLink User's Guide

G2 JavaLink

- G2 JavaLink User's Guide
- G2 DownloadInterfaces User's Guide
- G2 Bean Builder User's Guide

G2 Diagnostic Assistant

- GDA User's Guide
- GDA Reference Manual
- GDA API Reference

Customer Support Services

Phone support is available from 9am to 6pm US Eastern time on Monday through Friday at the phone numbers listed below.

Gensym Global Support Phone Numbers for Standard and Gold Customers:

United States

Toll-Free - 1-855-453-8174; Toll - 1-512-861-2859

United Kingdom

0808-189-1290

France

0800-915549

If your country is not listed above, you can dial the US Toll Number 1-512-861-2859 to reach our support organization.

You can submit a ticket at anytime by going to our support portal at https://support.ignitetech.com or sending an email to support@ignitetech.com [mailto:support@ignitetech.com]. While you are able to submit tickets electronically by either method, our support portal is designed to collect key information to help resolve your issue faster, so the support portal is the preferred method for submitting tickets.

If you have any questions about Gensym Global Support, please check our support overview or review our FAQs on the support portal. You are also welcome to submit a ticket or call us and we would be happy to answer your questions.

Chapter 1. MQTT protocol

Provides an introduction to the MQTT protocol.

1.1. Introduction

MQTT (MQ Telemetry Transport or Message Queuing Telemetry Transport) is an ISO standard (ISO/IEC PRF 20922) publish-subscribe-based messaging protocol. It is designed for connections with remote locations where a "small code footprint" is required or the network bandwidth is limited such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts. The publish-subscribe messaging pattern requires a message broker.

1.2. Characteristics of MQTT

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

- Use of the publish/subscribe message pattern which provides one-to-many message distribution and decoupling of applications.
- A messaging transport that is agnostic to the content of the payload.
- Three qualities of service for message delivery:
  - "At most once", where messages are delivered according to the best efforts of the operating environment. Message loss can occur. This level could be used, for example, with ambient sensor data where it does not matter if an individual reading is lost as the next one will be published soon after.
  - "At least once", where messages are assured to arrive but duplicates can occur.
  - "Exactly once", where message are assured to arrive exactly once. This level could be used, for example, with billing systems where duplicate or lost messages could lead to incorrect charges being applied.

1.3. MQTT protocol specification

MQTT protocol, as implemented by G2, is defined by the ISO standard (ISO/IEC PRF 20922:2016), version 3.1.1. At the time of this writing, it was publicly available at: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

1.4. MQTT client

This document pertains only to the MQTT client that is implemented in G2 and one can't use G2 as a MQTT server.

1.5. Protocol options

MQTT protocol definition leaves some options for the implementer - i.e. one can implement some things or not or in a certain way. Here we list our choices on these matters. We did choose options taking into account the proposed (draft) MQTT 5.0.

1.5.1. Explicit options

Publish before CONNACK

It is an option on the client part whether to send data (publish) to the server before client receives CONNACK. G2 client will not send data (publish) before it receives a CONNACK from the server. In general, this option is to make very small client implementations on very small devices, but G2 is not of such ilk.

Not receiving PUBACK/PUBCOMP - Resending messages

MQTT 3.1.1 specifies that the "publisher" can (but doesn't have to) resend the message if it doesn't receive a PUBACK/PUBCOMP in a reasonable amount of time. Also, MQTT 5.0 prohibits resending messages except on re-connect, so we don't resend. We do, of course, have to resend on reconnect with CleanSession=0.

PINGRESP timer

MQTT specifies that we should close the connection if we don't receive PINGRESP in a reasonable amount of time. To simplify things, we conclude that the reasonable amount of time is the KEEP-ALIVE set for the MQTT-SERVER (i.e. there is no separate "PINGRESP timeout" parameter).

CONNACK timer

If we don't receive CONNACK in a reasonable amount of time, we should "break/close" the (TCP/IP) connection. There's no "universally reasonable amount of time" for G2, so this a parameter of the MQTT-SERVER object.

1.5.2. Message/packet buffer

MQTT leaves everything about message/packet buffers up to implementation. Obviously, we can't have 65535 packet buffers statically allocated per each MQTT connection - there may be a lot of them, and it would consume too much memory. OTOH, we can't dynamically allocate

them buffers “as we go along” (as we’re a real-time application).

So, to give the user some control on how much memory is used, we make this a configurable parameter, per MQTT-SERVER. This will only take effect at the start of a connection.

1.5.3. When to deliver QoS=2 message

MQTT explicitly allows the recipient of a message w/QoS=2 to either deliver the message “onward” on (the first) PUBLISH or on PUBREL.

We believe that “the sooner, the better” applies here, so, G2 delivers on PUBLISH. Also, MQTT 5.0 doesn’t have this option any more, but specifies “you need to start the delivery on PUBLISH, you don’t need to finish it before sending PUBREC or PUBCOMP”, but, in our case, there’s no “lengthy process” here, if we start, we’ll finish shortly.

1.5.4. Error handling

Unsolicited CONNACK

While MQTT does specify that “any protocol violation is a cause for closing the connection”, it doesn’t specify what is a protocol violation explicitly in most cases. The “unsolicited” CONNACK is not specified in this way.

Given that MQTT allows one to go on publishing even before receiving CONNACK, from our POV, it doesn’t make sense to break the connection in this case, so we ignore it (and log it).

Unsolicited PINGRESP

This is similar to Unsolicited CONNACK and we deal with it in a similar way - we ignore it (but log it for the user).

Unsolicited PUBREL

If we receive a PUBLISH, we send PUBREC back and await PUBREL. When we then receive PUBREL, we send PUBCOMP back.

But, if we receive a PUBREL “out of the blue” (without having received a PUBLISH for that packet ID before), we’ll ignore this and log it for the user.

Please observe that, while similar in an abstract way, we analyzed this independent of Invalid packet ID in SUBACK/PUBACK/PUBCOMP because there we’re analyzing the “sender” side, while here we’re analyzing the “receiver” side.

Unsolicited PUBCOMP

We send a PUBLISH/QoS=2, but receive a PUBCOMP “right away”, without a PUBREC. MQTT

specs are silent on how to handle this, though one can surmise that this is a protocol error.

The basic reasoning for not closing the connection is the same as for Unsolicited PUBREL - but, another question is: should we actually "go with it" and ignore that we didn't receive PUBREC? Or should we ignore this PUBCOMP?

It does seem to be right to expect a PUBREC and not treat "PUBCOMP w/out PUBREC as being OK". So, we ignore this and log it for the user.

Completely unsolicited PUBCOMP

Just to be thorough, let's mention "Completely unsolicited PUBCOMP", that is, receiving PUBCOMP when we haven't even sent a PUBLISH for its packet ID.

The only difference (from the "not-completely-unsolicited PUBCOMP") is that the only options here are "ignore" and "close connection", and, in line with other similar scenarios, we "ignore".

Receiving DISCONNECT

We (as a client) should never receive a DISCONNECT. While MQTT spec is silent on the matter, we take this to imply that if we do receive a DISCONNECT, it is a protocol violation (i.e. the other side thinks we're the server, and we're not). Thus, we close/break the connection (without sending DISCONNECT) in this case.

MQTT 5.0 allows for server sending a DISCONNECT, so, this will also ease our "upgrade" to it.

Receiving SUBSCRIBE or UNSUBSCRIBE

This is very similar to Receiving DISCONNECT and we treat it the same way (as a protocol violation and close the connection).

Receiving CONNECT

This is very similar to Receiving DISCONNECT and we treat it the same way (as a protocol violation and close the connection).

Not receiving SUBACK or UNSUBACK

MQTT doesn't deal with this, except not receiving (UN)SUBACK before the connection ends/breaks.

But, it does deal with not receiving PUBLISH confirmations (see Explicit options). So, we conclude that same stands for (UN)SUBACK - we can, but don't have to, resend the (UN)SUBSCRIBE.

Since we're using the reliable transport and subscriptions are kept by the server on

connection close, we deem this not worthy of adding complexity of resending the (UN)SUBSCRIBE on timeout.

But, user does have a need to see “are all my (un)subscriptions confirmed by the server”. So, we provide an API which gives back to the user a list of “unconfirmed subscriptions” and a separate list of “unconfirmed unsubscriptions”.

MQTT 5.0 forbids re-sending of (UN)SUBSCRIBE (although only implicitly), so this will also ease our upgrade to it.

Receiving Packet ID == 0

Message with a Packet ID == 0 is invalid. MQTT specs are not explicit what to do if we receive such a packet. It is a protocol error, so, one concludes that we should close the connection.

But, it does seem wrong to close a connection just because we received one Packet ID == 0. So, we drop/ignore the message, and report to the user, but do not close the connection.

Packet lengths

If the user tries to publish a message which is longer than the maximum allowed by MQTT spec, we indicate that error to the user (also giving the actual maximum allowed length). Similar, if our internal maximum is different than the MQTT spec maximum, we indicate that error (and give the actual maximum). If we receive (from the server) a packet whose length is longer than the maximum allowed by MQTT spec, that is an error and we close/break the connection.

1.5.5. MQTT under-specified

There are some things that are under-specified in MQTT specs. That is, if taken at “face value”, they don’t make much sense or are obviously lacking. Smaller items that have to do with error handling specifically we covered in Error handling, here we cover the “more significant ones”.

No SUBACK before end of connection

We sent a SUBSCRIBE, but have not received a SUBACK. Then, the connection drops (and not because of a disconnect request by the user but, say, network issues). Thus, we re-connect, with “CleanSession=0”. The question is: do we resend the unconfirmed SUBSCRIBE(s)?

If CleanSession is 0, that implies that the user expects the subscriptions to be “remembered” (as MQTT states that server keeps the subscriptions if CleanSession = 0), but, we don’t know if the connection dropped before the server received our SUBSCRIBE.

Another point is that MQTT specifies that client keeps the (published) messages if it uses CleanSession=0, which implies it keeps their packet IDs. But, SUBSCRIBE (and SUBACK) also

use (the same) packet IDs. Thus, it doesn't make sense to keep packet IDs (and content of said packets) for (published) messages, but not for subscriptions.

Thus, we do it like this:

- When we send SUBSCRIBE, we remember the subscriptions we sent (associated with the packet Id we sent in the SUBSCRIBE).
- When we receive SUBACK, we "forget" the subscriptions we sent for the packet ID that "came in" SUBACK.
- On successful CONNACK after our CONNECT with CleanSession=0, we resend all subscriptions (for all "pending" SUBSCRIBE packets), using their original packet IDs (even though it should not matter, as packet IDs lose their meaning in a new session, it will be easier to reason about protocol behavior if we use the original packet IDs).
- On "failed" CONNACK or on CONNECT with CleanSession=1, we "forget" all the subscriptions.

Invalid packet ID in SUBACK/PUBACK/PUBCOMP

These "acknowledge" packets "carry" the packet ID they acknowledge. But, what if that packet ID is not "awaiting acknowledgement" on the client side?

MQTT is silent on the matter. From a certain POV, you can conclude that this is a protocol error, thus, one should close/break the connection. But, actually, MQTT implies that this should be ignored, because it allows for re-sending of PUBLISH on timeout.

So, even though we don't re-send PUBLISH (or SUBSCRIBE), that is beside the point, MQTT actually wants us to ignore this. Of course, we log this as a warning to the user.

1.6. Limits

G2 MQTT implementation imposes certain limits regarding its configuration, as described below. These cannot be changed by the user. But, they can be changed, on request, for a future release.

Maximum amount of concurrent MQTT server connections supported

64

Maximum length of MQTT server address

44

Maximum length of Client ID

63

Maximum length of the user name

12

Maximum length of the password

12

Maximum size of MQTT message

4194304

Maximum length of a topic

255

Maximum number of topics in a SUBSCRIBE packet

256

Maximum size of Last Will message

1023

Maximum length of hex data trace

3072

Chapter 2. MQTT Knowledge Base

Provides information about the G2 MQTT KB.

2.1. Introduction

The MQTT KB is the module enabling use of MQTT protocol in a G2 KB application. It contains the User API. The reference for the API is documented at User API Reference. Here we give a description of the KB and how to use it.

2.2. Usage

To use the MQTT protocol in your G2 KB application, set the `mqtt-client.kb` as a required module. Please note that G2 can is only a MQTT client.

2.2.1. Startup/shutdown

Before you start using the MQTT protocol, you need to create one or more instances of MQTT-SERVER class.

Configure the connection by changing instance attributes and then call G2-MQTT-CONNECT. Upon successful connection, `MQTT-STATE` attribute of the instance changes to symbol `CONNECTED`. It is possible to create a rule to monitor that.

To close the connection, call G2-MQTT-DISCONNECT. Upon connection termination `MQTT-STATE` attribute of the instance changes to symbol `DISCONNECTED`.

2.2.2. Sending messages

To send a message, use G2-MQTT-PUBLISH-AT-MOST-ONCE, G2-MQTT-PUBLISH-AT-LEAST-ONCE or G2-MQTT-PUBLISH-EXACTLY-ONCE, for QoS 0, 1 or 2, respectively.

```
call g2-mqtt-publish-exactly-once (server-1, "topic1", "payload", false);
```

2.2.3. Receiving messages

First thing to do in order to receive messages is to define a "callback" procedure that will be

called upon a message arrival. This procedure should take three arguments: TOPIC: text, MESSAGE: text, UTF8-P: truth-value; the last argument will be TRUE if MESSAGE looks like a text string, not just byte vector. Then set MQTT-CALLBACK attribute of your MQTT-SERVER instance to the symbol naming the procedure.

Once the callback is set, you can use G2-MQTT-SUBSCRIBE to subscribe to some topics and start receiving messages from these topics. Once you don't want receive those messages any more, you should call G2-MQTT-UNSUBSCRIBE.

```
call g2-mqtt-subscribe (server-1,
                        sequence(
                          structure(topic: "some/topic0", qos: 0),
                          structure(topic: "some/topic1", qos: 1),
                          structure(topic: "some/topic2", qos: 2)));
```

```
call g2-mqtt-unsubscribe (server-1,
                          sequence("some/topic0", "some/topic1",
                                   "some/topic2"));
```

2.3. Logging

MQTT errors will be logged in general G2 log.

Additionally, MQTT-LOG-PACKET-DATA and MQTT-LOG-TOPIC-DATA attributes of MQTT-SERVER instance can be used to log packet and topic data, respectively.

Chapter 3. User API Reference

3.1. Classes

3.1.1. MQTT-SERVER

Direct Superior Classes

OBJECT

Attribute	Type	Initial Value	Initial Value Type	Description
MQTT-STATE	SYMBOL	DISCONNECTED	SYMBOL	Current state of MQTT connection
MQTT-HOST	WHOLE-STRING	localhost	WHOLE-STRING	MQTT server host
MQTT-PORT	POSITIVE-INTEGER	1883	POSITIVE-INTEGER	MQTT server port
MQTT-USE-TLS	TRUTH-VALUE		TRUTH-VALUE	Shall we use TLS?
MQTT-USERNAME	WHOLE-STRING		WHOLE-STRING	Username
MQTT-PASSWORD	WHOLE-STRING		WHOLE-STRING	Password
MQTT-WILL-RETAIN	TRUTH-VALUE		TRUTH-VALUE	Last Will retain
MQTT-WILL-QOS	NON-NEGATIVE-INTEGER	0	NON-NEGATIVE-INTEGER	Last Will QoS
MQTT-WILL-TOPIC	WHOLE-STRING		WHOLE-STRING	Last Will topic
MQTT-WILL-MESSAGE	WHOLE-STRING		WHOLE-STRING	Last Will message
MQTT-KEEP-ALIVE	POSITIVE-INTEGER	60	POSITIVE-INTEGER	Keep-Alive, in seconds; the maximum keep alive is 18h 12min 15 sec
MQTT-CLIENT-ID	WHOLE-STRING		WHOLE-STRING	MQTT client Id

Attribute	Type	Initial Value	Initial Value Type	Description
MQTT-CONNACK-TIMEOUT	POSITIVE-INTEGER	1000	POSITIVE-INTEGER	CONNACK timeout, in milliseconds; the minimum is 100 and the maximum is 10000
MQTT-MESSAGE-BUFFER-COUNT	POSITIVE-INTEGER	100	POSITIVE-INTEGER	Maximum number of outgoing packets awaiting confirmation
MQTT-IN-FLIGHT-MESSAGES-LIMIT	POSITIVE-INTEGER	10	POSITIVE-INTEGER	In-flight message limit
MQTT-LOG-PACKET-DATA	TRUTH-VALUE		TRUTH-VALUE	Shall we log packet data?
MQTT-LOG-TOPIC-DATA	TRUTH-VALUE		TRUTH-VALUE	Shall we log topic data?
MQTT-LOG-DIRECTORY	WHOLE-STRING		WHOLE-STRING	Log directory
MQTT-LOG-ROLLING-INTERVAL	POSITIVE-INTEGER	1	POSITIVE-INTEGER	Log rolling interval
MQTT-CALLBACK	SYMBOL		SYMBOL	Callback for incoming messages, a symbol; shall be a name of a procedure taking three arguments: TOPIC: text, MESSAGE: text, UTF8-P: truth-value; the last argument will be TRUE if MESSAGE looks like a text string, not just byte vector

Description

MQTT server

3.2. Procedures

3.2.1. G2-MQTT-DISCONNECT

Synopsis

`G2-MQTT-DISCONNECT`(`SERVER`: CLASS MQTT-SERVER)

Argument	Description
`SERVER`	MQTT server

It has no return values.

Description

Explicit disconnect by the user. It is ignored if we're already not connected to the server.

3.2.2. G2-MQTT-UNSUBSCRIBE

Synopsis

`G2-MQTT-UNSUBSCRIBE`(`SERVER`: CLASS MQTT-SERVER, `TOPICS`: SEQUENCE) → `RET0`: INTEGER

Argument	Description
`SERVER`	MQTT server
`TOPICS`	sequence of topics, like sequence("abc", "some/topic/0") etc.

Return Value	Description
`RET0`	packet id

Description

Unsubscribes from given topics. Please refer to MQTT protocol specification for details.The return value is the ID assigned to the outbound UNSUBSUBRIBE packet. If we're not connected, or in case of any other error which makes us not even try to subscribe, we raise an ERROR. Otherwise, the unsubscribe is started.

3.2.3. G2-MQTT-SUBSCRIBE

Synopsis

`G2-MQTT-SUBSCRIBE`(`SERVER`: CLASS MQTT-SERVER, `TOPICS`: SEQUENCE) → `RET0`: INTEGER

Argument	Description
SERVER	MQTT server
TOPICS	sequence of structure(topic: text, QoS: integer)

Return Value	Description
RET0	packet id

Description

Subscribes to given topics. Please refer to MQTT protocol specification for details.The return value is the ID assigned to the outbound SUBSUBRIBE packet. If we're not connected, or in case of any other error which makes us not even try to subscribe, we raise an ERROR. Otherwise, the subscribe is started.

3.2.4. G2-MQTT-RECONNECT

Synopsis

G2-MQTT-RECONNECT(SERVER: CLASS MQTT-SERVER)

Argument	Description
SERVER	MQTT server

It has no return values.

Description

This implies "CleanSession=0". Otherwise, it's same as G2-MQTT-CONNECT(server).

3.2.5. G2-MQTT-CONNECT

Synopsis

G2-MQTT-CONNECT(SERVER: CLASS MQTT-SERVER)

Argument	Description
SERVER	MQTT server

It has no return values.

Description

Establishes a connection to MQTT server. Please refer to MQTT protocol specification for

details. This implies “CleanSession=1”. The whole configuration (using SSL/TLS, username/password…) is kept in the `server` object. There is no return value. If we’re already connected, this is ignored. If something is wrong and we didn’t even try to connect, an ERROR is raised. Otherwise, the connection process is started and the `server` will be updated with the “state” of connection (similar to GSI).

3.2.6. G2-MQTT-PUBLISH-EXACTLY-ONCE

Synopsis

`G2-MQTT-PUBLISH-EXACTLY-ONCE`(`SERVER`: CLASS MQTT-SERVER, `TOPIC`: TEXT, `MESSAGE`: TEXT, `RETAIN`: TRUTH-VALUE) → `RET0`: INTEGER

Argument	Description
`SERVER`	MQTT server
`TOPIC`	topic to publish to
`MESSAGE`	a non-empty string or byte array
`RETAIN`	retain flag

Return Value	Description
`RET0`	packet id

Description

Publishes the message in given topic. Please refer to MQTT protocol specification for details.The return value is the ID assigned to the outbound PUBLISH packet. This is QoS=2. If we’re not connected, or in case of any other error which makes us not even try to publish, we raise an ERROR. Otherwise, the publish is started.

3.2.7. G2-MQTT-PUBLISH-AT-LEAST-ONCE

Synopsis

`G2-MQTT-PUBLISH-AT-LEAST-ONCE`(`SERVER`: CLASS MQTT-SERVER, `TOPIC`: TEXT, `MESSAGE`: TEXT, `RETAIN`: TRUTH-VALUE) → `RET0`: INTEGER

Argument	Description
`SERVER`	MQTT server
`TOPIC`	topic to publish to
`MESSAGE`	a non-empty string or byte array
`RETAIN`	retain flag

Return Value	Description
RET0	packet id

Description

Publishes the message in given topic. Please refer to MQTT protocol specification for details.The return value is the ID assigned to the outbound PUBLISH packet. This is QoS=1. If we're not connected, or in case of any other error which makes us not even try to publish, we raise an ERROR. Otherwise, the publish is started.

3.2.8. G2-MQTT-PUBLISH-AT-MOST-ONCE

Synopsis

G2-MQTT-PUBLISH-AT-MOST-ONCE(SERVER: CLASS MQTT-SERVER, TOPIC: TEXT, MESSAGE: TEXT, RETAIN: TRUTH-VALUE) → RET0: INTEGER

Argument	Description
SERVER	MQTT server
TOPIC	topic to publish to
MESSAGE	a non-empty string or byte array
RETAIN	retain flag

Return Value	Description
RET0	packet id

Description

Publishes the message in given topic. Please refer to MQTT protocol specification for details. The return value is a dummy packet ID=1, as it's QoS=0. If we're not connected, or in case of any other error which makes us not even try to publish, we raise an ERROR. Otherwise, the publish is started.

3.2.9. G2-MQTT-GET-UNCONFIRMED-MESSAGES

Synopsis

G2-MQTT-GET-UNCONFIRMED-MESSAGES(SERVER: CLASS MQTT-SERVER) → RET0: SEQUENCE

Argument	Description
SERVER	MQTT server

Return Value	Description
RET0	sequence of structure(packet-id: integer, text-p: truth-value, message: text)

Description

Returns a sequence of unconfirmed PUBLISH messages, if any. TEXT-P attribute of returned structures is TRUE when MESSAGE is probably a text string, not just a byte vector.

3.2.10. G2-MQTT-GET-UNCONFIRMED-SUBSCRIBES

Synopsis

G2-MQTT-GET-UNCONFIRMED-SUBSCRIBES(SERVER: CLASS MQTT-SERVER) → RET0: SEQUENCE

Argument	Description
SERVER	MQTT server

Return Value	Description
RET0	sequence-of-structs(packet-id: integer, topics: sequence-of-structs(topic: text, qos: integer)

Description

Returns a sequence of unconfirmed SUBSCRIBEs, each element being a structure with the following fields: PACKET-ID - an id of outbound SUBSCRIBE packet, TOPICS - sequence of topics we tried to subscribe to, each being a structure with TOPIC name and QOS value

3.2.11. G2-MQTT-GET-UNCONFIRMED-UNSUBSCRIBES

Synopsis

G2-MQTT-GET-UNCONFIRMED-UNSUBSCRIBES(SERVER: CLASS MQTT-SERVER) → RET0: SEQUENCE

Argument	Description
SERVER	MQTT server

Return Value	Description
RET0	sequence-of-structs(packet-id: integer, topics: sequence-of-texts

Description

Returns a sequence of unconfirmed UNSUBSCRIBEs, each element being a structure with the following fields: PACKET-ID - an id of outbound UNSUBSCRIBE packet, TOPICS - sequence of topics we tried to unsubscribe from