**SAFE STRIP**

**SAFE and green Sensor Technologies for self-explaining and forgiving Road Interactive aPplications**

****

**Grant Agreement Number: 723211**

**Internal Report: ¨SAFE STRIP MQTT Broker¨**

**Legal Disclaimer**

The information in this document is provided “as is”, and no guarantee or warranty is given that the information is fit for any particular purpose. The above referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law. © 2017 by SAFE STRIP Consortium.

# SAFE STRIP MQTT server

SAFE STRIP communication between different entities is based on the MQTT protocol. MQTT is a very lightweight protocol that uses a publish/subscribe model. This makes it suitable for "machine to machine" messaging such as with low power sensors or mobile devices.

The SAFE STRIP MQTT server is Eclipse Mosquitto™. Eclipse Mosquitto™ is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 3.1 and 3.1.1.

Version currently installed is 1.4.15.

## Installation

The system is installed at Swarco Mizar premises as a docker image on a CentOS server. This allows to easily move the system in others environment if needed.

## Configuration

Eclipse Mosquitto™ for SAFE STRIP has been configured to accept connection only from client that presents themselves using a specific username/password pair, connection the system using secure connection (TLS 1.2).

The system is configured not to accept anonymous connection.

Furthermore, in the system is configured an access list: a user can connect only to a specific topic or list of topics. In details, the user *safestrip* is allowed only to access *SafeStrip/#* topics (where *#* is a wildcard that means any sub-topic).

The system has been left with the default configuration with regards to the persistence of the messages: the messages not delivered are stored every 30 minutes in the file system to maintain them.

## Connection to SAFE STRIP MQTT server

The server is configured to listen on address 93.62.253.212 on port 8883. The address is public and shall be used by any client outside Swarco Mizar network. For applications that will run on the internal Swarco Mizar network (as a docker), it is required that the address and port for the connection to the MQTT broker shall be configurable at system launching.

Eclipse Mosquitto™ for SAFE STRIP has been configured to accept connection only from clients that present themselves using a specific username/password pair:

* User: safestrip
* Pwd: S@f3str1p

The user *safestrip* is allowed only to access *SafeStrip/#* topics.

# MQTT Broker manual

## Publish/Subscribe

The MQTT protocol is based on the principle of publishing messages and subscribing to topics, or "pub/sub". Multiple clients connect to a broker and subscribe to topics that they are interested in. Clients also connect to the broker and publish messages to topics. Many clients may subscribe to the same topics and do with the information as they please. The broker and MQTT act as a simple, common interface for everything to connect to.

## Topics/Subscriptions

Messages in MQTT are published on topics. There is no need to configure a topic, publishing on it is enough. Topics are treated as a hierarchy, using a slash (/) as a separator. This allows sensible arrangement of common themes to be created, much in the same way as a filesystem. For example, multiple computers may all publish their hard drive temperature information on the following topic, with their own computer and hard drive name being replaced as appropriate:

* sensors/COMPUTER\_NAME/temperature/HARDDRIVE\_NAME

Clients can receive messages by creating subscriptions. A subscription may be to an explicit topic, in which case only messages to that topic will be received, or it may include wildcards. Two wildcards are available, + or #.

+ can be used as a wildcard for a single level of hierarchy. It could be used with the topic above to get information on all computers and hard drives as follows:

* sensors/+/temperature/+

As another example, for a topic of "a/b/c/d", the following example subscriptions will match:

* a/b/c/d
* +/b/c/d
* a/+/c/d
* a/+/+/d
* +/+/+/+

The following subscriptions will not match:

* a/b/c
* b/+/c/d
* +/+/+

# can be used as a wildcard for all remaining levels of hierarchy. This means that it must be the final character in a subscription. With a topic of "a/b/c/d", the following example subscriptions will match:

* a/b/c/d
* #
* a/#
* a/b/#
* a/b/c/#
* +/b/c/#

Zero length topic levels are valid, which can lead to some slightly non-obvious behaviour. For example, a topic of "a//topic" would correctly match against a subscription of "a/+/topic". Likewise, zero length topic levels can exist at both the beginning and the end of a topic string, so "/a/topic" would match against a subscription of "+/a/topic", "#" or "/#", and a topic "a/topic/" would match against a subscription of "a/topic/+" or "a/topic/#".

## Quality of Service

MQTT defines three levels of Quality of Service (QoS). The QoS defines how hard the broker/client will try to ensure that a message is received. Messages may be sent at any QoS level, and clients may attempt to subscribe to topics at any QoS level. This means that the client chooses the maximum QoS it will receive. For example, if a message is published at QoS 2 and a client is subscribed with QoS 0, the message will be delivered to that client with QoS 0. If a second client is also subscribed to the same topic, but with QoS 2, then it will receive the same message but with QoS 2. For a second example, if a client is subscribed with QoS 2 and a message is published on QoS 0, the client will receive it on QoS 0.

Higher levels of QoS are more reliable but involve higher latency and have higher bandwidth requirements.

* 0: The broker/client will deliver the message once, with no confirmation.
* 1: The broker/client will deliver the message at least once, with confirmation required.
* 2: The broker/client will deliver the message exactly once by using a four-step handshake.

## Retained Messages

All messages may be set to be retained. This means that the broker will keep the message even after sending it to all current subscribers. If a new subscription is made that matches the topic of the retained message, then the message will be sent to the client. This is useful as a "last known good" mechanism. If a topic is only updated infrequently, then without a retained message, a newly subscribed client may have to wait a long time to receive an update. With a retained message, the client will receive an instant update.

## Clean session / Durable connections

On connection, a client sets the "clean session" flag, which is sometimes also known as the "clean start" flag. If clean session is set to false, then the connection is treated as durable. This means that when the client disconnects, any subscriptions it has will remain and any subsequent QoS 1 or 2 messages will be stored until it connects again in the future. If clean session is true, then all subscriptions will be removed for the client when it disconnects.

## Wills

When a client connects to a broker, it may inform the broker that it has a will. This is a message that it wishes the broker to send when the client disconnects unexpectedly. The will message has a topic, QoS and retain status just the same as any other message.