MQTT Client Application

User guide

Version 0.2

May 2016

Redpine Signals, Inc.
2107 N. First Street, #540
San Jose, CA 95131.
Tel: (408) 748-3385
Fax: (408) 705-2019
Email: info@redpinesignals.com
Website: www.redpinesignals.com

About this Document

This document describes the process of bringing up the RS9113 based module as a MQTT client.

Disclaimer:

The information in this document pertains to information related to Redpine Signals, Inc. products. This information is provided as a service to our customers, and may be used for information purposes only. Redpine assumes no liabilities or responsibilities for errors or omissions in this document. This document may be changed at any time at Redpine's sole discretion without any prior notice to anyone. Redpine is not committed to updating this document in the future.

Table of Contents

Table of Figures

Table of Tables

No table of figures entries found.

1 Introduction

This project is applicable to all the WiSeConnect variants like WiSeConnect Plus, WiSeMCU and WyzBee. The term WiSeConnect refers to its appropriate variant.

1.1 Protocol Overview

MQTT is a publish-subscribe based "light weight" messaging protocol for using on top of the TCP/IP protocol.

The MQTT connection itself is always between one client and the broker, no client is connected to another client directly.

MQTT client

A MQTT client is any device from a micro controller to a full fledged server, that has a MQTT library running and is connecting to an MQTT broker over any kind of network.

MQTT Clients can share the information on a particular topic using MQTT protocol.

MQTT clients connect to the MQTT broker using TCP connection and can subscribe and publish on any desired topic.

The other clients which are subscribed for that topic will receive the published messages.

MQTT Broker

The publish-subscribe messaging pattern requires a message broker.

The broker is primarily responsible for receiving all messages, filtering them,

deciding like who is interested in it and then sending the message to all subscribed clients.

It also holds the session of all persisted clients including subscriptions and missed messages.

Another responsibility of the broker is the authentication and authorization of clients.

A simple demonstration of subscribing and publishing of temperature is shown below

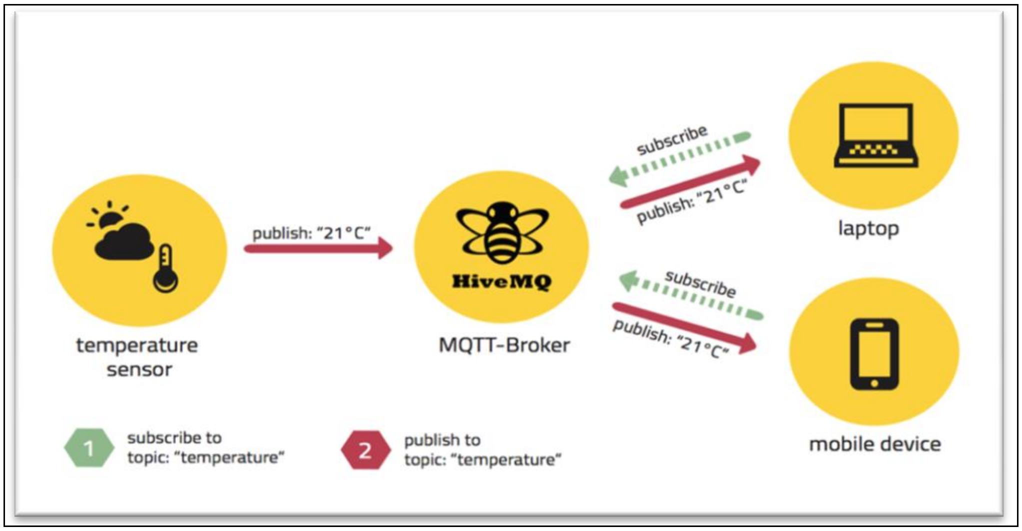


Figure 1: Demonstration of MQTT protocol

1.2 Application Overview

1.2.1 Overview

This application demonstrates how to configure WiSeConnect device as MQTT client and how to establish connection with MQTT broker and how to subscribe, publish and receive the MQTT messages from MQTT broker.

In this application, WiSeConnect device configured as WiFi station and connects to Access Point. After successful WiFi connection, application connects to MQTT broker and subscribes to the topic **"REDPINE"** and publishes a message **"THIS IS MQTT CLIENT DEMO FROM REDPINE"** on that subscribed topic. And application waits to receive the data published on subscribed topic by other clients.

1.2.2 Sequence of Events

This Application explains user how to:

- Connect to Access Point
- Establish MQTT client connection with MQTT broker
- Subscribe the topic **"REDPINE"**
- Publish message **"THIS IS MQTT CLIENT DEMO FROM REDPINE"** on the subscribed topic **"REDPINE"**
- Receive data published by other clients on the same subscribed topic **"REDPINE"**.

1.3 Application Setup

The WiSeConnect in its many variants supports SPI and UART interfaces. Depending on the interface used, the required set up is as below:

1.3.1 SPI based Setup Requirements

- Windows PC1 with CooCox IDE
- Spansion (MB9BF568NBGL) micro controller

Note: If user does not have Spansion (MB9BF568NBGL) host platform, please go through the SPI-Porting guide **\sapis\docs\RS9113-WiSeConnect-SAPI-Porting-Guide-vx.x.pdf** for SAPIs porting to that particular platform.

- WiSeConnect device
- Windows PC2 with with MQTT broker installed in it
- Windows PC3 with with MQTT client utility installed in it

Note: MQTT broker for different OS platforms can be downloaded from the link ***http://mosquitto.org/download/***

Ex: Install "mosquitto-1.4.8-install-win32.exe"

MQTT Utility which has to be installed in Windows PC 3 can be downloaded from the below given link

https://www.eclipse.org/downloads/download.php?file=/paho/1.0/org.eclipse.paho.mqtt.utility-1.0.0.jar

1.3.2 UART/USB-CDC based Setup Requirements

- Windows PC1 with Dev-C++ IDE
- WiSeConnect device
- WiFi Access point
- Windows PC2 with with MQTT broker installed in it
- Windows PC3 with with MQTT client utility installed in it

Note: MQTT broker for different OS platforms can be downloaded from the link ***http://mosquitto.org/download/***

Ex: Install "mosquitto-1.4.8-install-win32.exe"

MQTT Utility which has to be installed in Windows PC 3 can be downloaded from the below given link

https://www.eclipse.org/downloads/download.php?file=/paho/1.0/org.eclipse.paho.mqtt.utility-1.0.0.jar

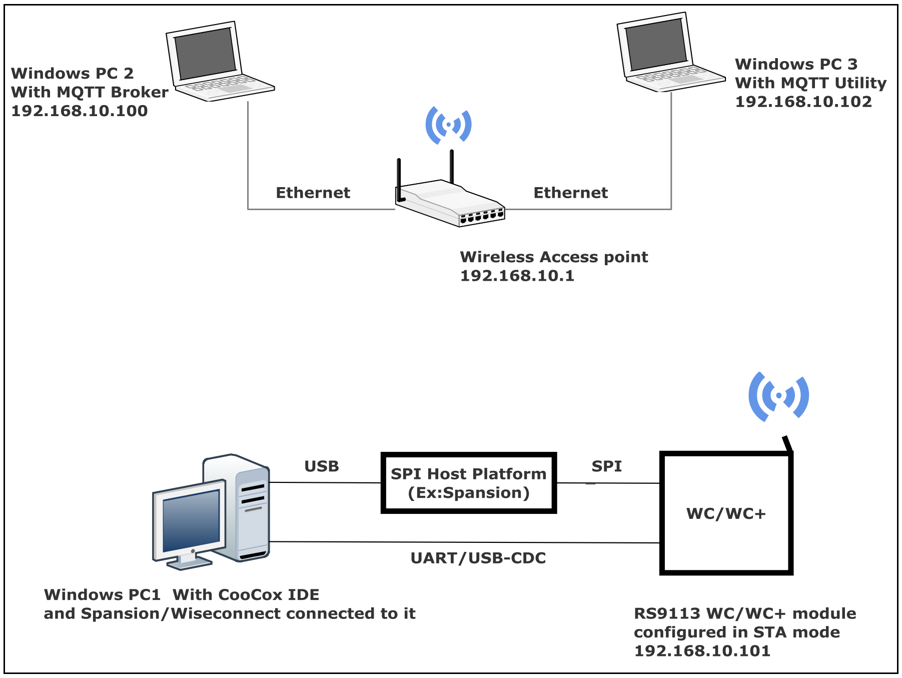


Figure 2: MQTT Client demo set up

2 Configuration and Execution of the Application

The example application is available in the Release at **{Release $}/host/sapis/examples.** These examples will have to be initialized, configured and executed to test the application. The initialization varies based on the interface but configuration and execution are the common.

2.1 Initializing the Application

2.1.1 SPI Interface

If User using SPI interface, Please refer the document ***sapis/platforms/spansion_MB9BF568NBGL/RS9113-WiSeConnect_SAPIS_Spansion_Project_User_guide.pdf*** for opening the ***mqtt_client*** example in CooCox IDE.

2.1.2 UART/USB-CDC Interface

If User using UART interface, Please refer the document ***sapis/platforms/ windows_uart/RS9113-WiSeConnect_SAPIS_Windows_Project_UserGuide.pdf*** for opening the ***mqtt_client*** example in Dev-C++ IDE

2.2 Configuring the Application

1. Open sapis/examples/wlan/mqtt_client/rsi_mqtt_client.c file and update/modify following macros :

SSID refers to the name of the Access point.

```
      #define SSID                    "<ap name>"
```

SECURITY_TYPE refers to the type of security. In this application STA supports Open, WPA-PSK, WPA2-PSK securities.

Valid configuration is:

RSI_OPEN - For OPEN security mode

RSI_WPA - For WPA security mode

RSI_WPA2 - For WPA2 security mode

```
      #define SECURITY_TYPE             RSI_OPEN
```

PSK refers to the secret key if the Access point configured in WPA-PSK/WPA2-PSK security modes.

```
      #define PSK                       "<psk>"
```

CLIENT_PORT port refers device MQTT client port number

```
      #define CLIENT_PORT               5001
```

SERVER_PORT port refers remote MQTT broker/server port number

```
      #define SERVER_PORT               1883
```

SERVER_IP_ADDRESS refers remote peer IP address (Windows PC2) to connect with MQTT broker/server socket.

IP address should be in long format and in little endian byte order.

Example: To configure “192.168.0.100” as remote IP address, update the macro **SERVER_IP_ADDRESS** as **0x6400A8C0**.

```
#define SERVER_IP_ADDRESS      0x6400A8C0
```

MQTT client keep alive period

```
#define RSI_KEEP_ALIVE_PERIOD     100
```

Memory to initialize MQTT client Info structure

```
#define MQTT_CLIENT_INIT_BUFF_LEN 3500
```

Global buffer or memory which is used for MQTT client initialization. This buffer is used for the MQTT client information storage.

```
uint8_t mqqt_client_buffer[MQTT_CLIENT_INIT_BUFF_LEN];
```

QOS indicates the level of assurance for delivery of an Application Message.

QoS levels are:

0 - At most once delivery

1 - At least once delivery

2 - Exactly once delivery

```
#define QOS                        0
```

RSI_MQTT_TOPIC refers to which topic WiSeConnect MQTT client is supposed to subscribe.

```
#define RSI_MQTT_TOPIC       "REDPINE"
```

MQTT Message to publish on the topic subscribed

```
uint8_t publish_message[] ="THIS IS MQTT CLIENT DEMO
FROM REDPINE"
```

MQTT Client ID with which MQTT client connects to MQTT broker/server

```
uint8_t clientID[] = "MQTTCLIENT"
```

User name for login credentials

```
int8_t  username[] = "username"
```

Password for login credentials

```
int8_t  password[] = "password"
```

NUMEBR_OF_PACKETS refers how many packets to receive from TCP client

```
#define NUMBER_OF_PACKETS     <no of packets>
```

Application memory length which is required by the driver

```
#define  GLOBAL_BUFF_LEN       8000
```

To configure IP address

DHCP_MODE refers whether IP address configured through DHCP or STATIC

```
#define DHCP_MODE   1
```

Note: If user wants to configure STA IP address through DHCP then set **DHCP_MODE** to 1 and skip configuring the following **DEVICE_IP, GATEWAY** and **NETMASK** macros.

(Or)

If user wants to configure STA IP address through **STATIC** then set **DHCP_MODE** macro to “0” and configure following **DEVICE_IP**, **GATEWAY** and **NETMASK** macros.

IP address to be configured to the device in STA mode should be in long format and in little endian byte order.

Example: To configure “192.168.10.10” as IP address, update the macro **DEVICE_IP** as 0x0A0AA8C0.

```
        #define  DEVICE_IP            0X0A0AA8C0
```

IP address of the gateway should also be in long format and in little endian byte order

Example: To configure “192.168.10.1” as Gateway, update the macro GATEWAY as 0x010AA8C0

```
        #define  GATEWAY              0x010AA8C0
```

IP address of the network mask should also be in long format and in little endian byte order

Example: To configure “255.255.255.0” as network mask, update the macro **NETMASK** as 0x00FFFFFF

```
        #define   NETMASK             0x00FFFFFF
```

The following parameters are configured if OS is used.

WLAN task priority is given and this should be of low priority

```
        #define RSI_WLAN_TASK_PRIORITY       1
```

Driver task priority is given and this should be of highest priority

```
        #define RSI_DRIVER_TASK_PRIORITY     1
```

WLAN Task stack size is configured by this macro

```
        #define RSI_WLAN_TASK_STACK_SIZE     500
```

Driver Task stack size is configured by this macro

```
        #define RSI_DRIVER_TASK_STACK_SIZE   500
```

2. Open ***sapis/include/rsi_wlan_config.h*** file and update/modify following macros,

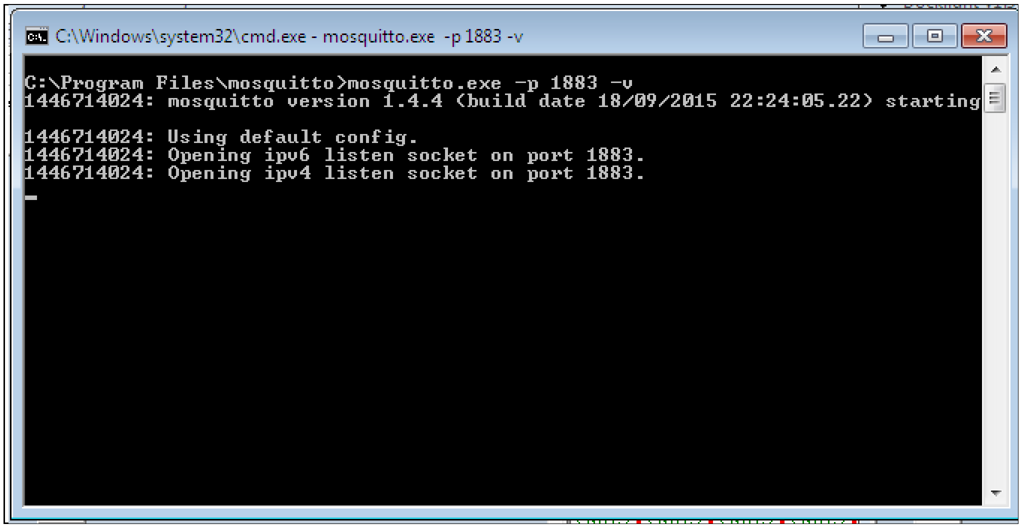
```
#define  CONCURRENT_MODE              RSI_DISABLE
#define  RSI_FEATURE_BIT_MAP          FEAT_SECURITY_OPEN
#define  RSI_TCP_IP_BYPASS            RSI_DISABLE
#define  RSI_TCP_IP_FEATURE_BIT_MAP   TCP_IP_FEAT_DHCPV4_CLIENT
#define  RSI_CUSTOM_FEATURE_BIT_MAP    0
#define  RSI_BAND                     RSI_BAND_2P4GHZ
```

2.3 Executing the Application

1. Configure the Access point in OPEN/WPA-PSK/WPA2-PSK mode to connect WiSeConnect device in STA mode.
2. Install MQTT broker in Windows PC2 which is connected to Access Point through LAN
3. Run MQTT broker in Windows PC2 using following command :

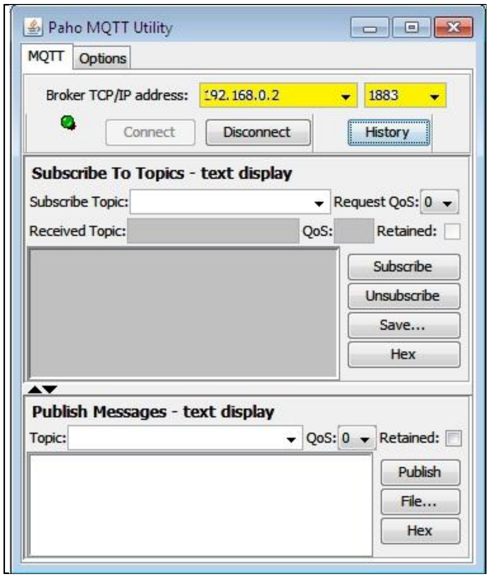
   Open Command prompt and go to MQTT installed folder (Ex: C:\Program Files\mosquitto) and run following command,

   ```
   mosquito.exe -p 1883 -v
   ```



```
C:\Program Files\mosquitto>mosquitto.exe -p 1883 -v
1446714024: mosquitto version 1.4.4 (build date 18/09/2015 22:24:05.22) starting

1446714024: Using default config.
1446714024: Opening ipv6 listen socket on port 1883.
1446714024: Opening ipv4 listen socket on port 1883.
```

4. Open MQTT client utility in Windows PC3 and connect to MQTT broker by giving Windows PC2 IP address and MQTT broker port number in Broker TCP/IP address field.

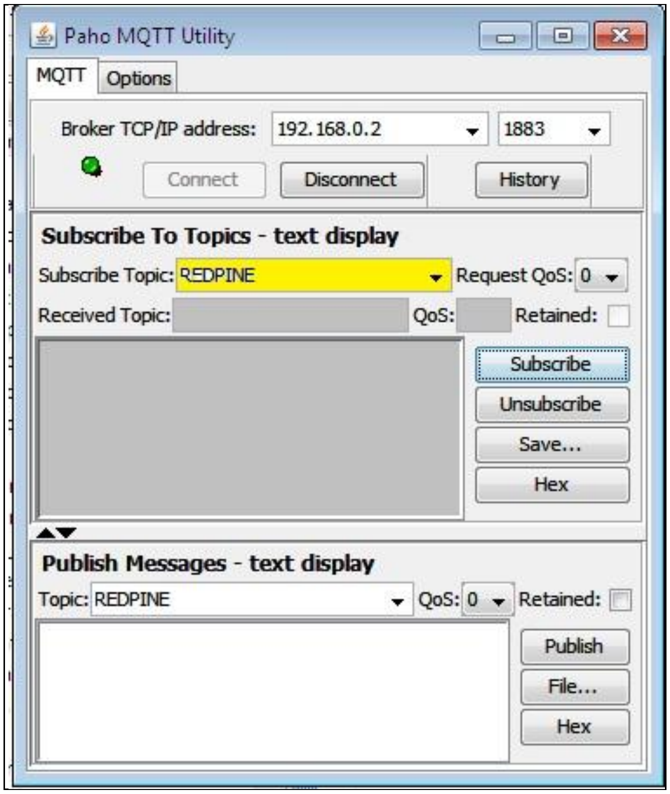


Broker TCP/IP address: 192.168.0.2 1883

Request QoS: 0

QoS: 0

5. After successful connection, subscribe to the topic from MQTT client utility.



Paho MQTT Utility

Broker TCP/IP address: 192.168.0.2 1883

Subscribe To Topics - text display

Subscribe Topic: REDPINE Request QoS: 0

Received Topic: QoS: Retained: □

Publish Messages - text display

Topic: REDPINE QoS: 0 Retained: □

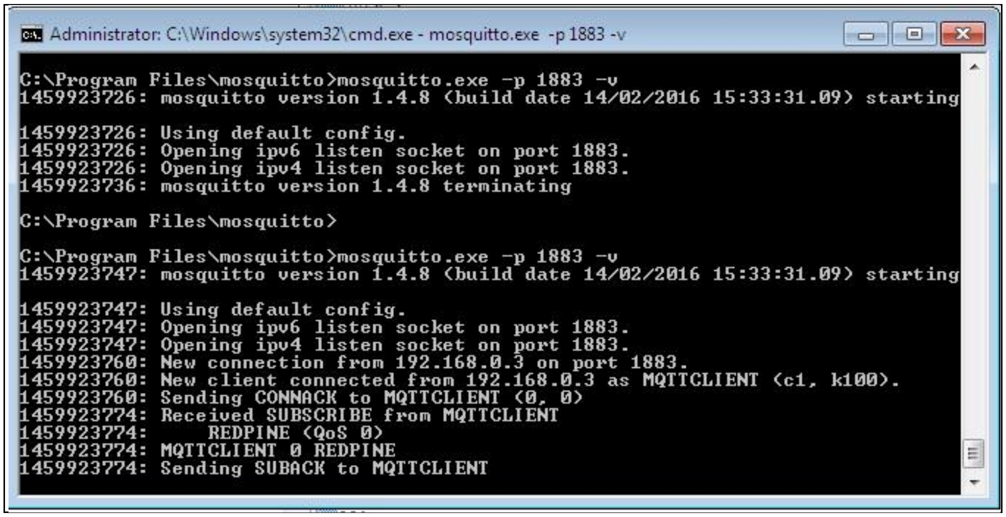
6. **SPI Interface**

If User using SPI interface, Please refer the document ***sapis/platforms/spansion_MB9BF568NBGL/RS9113-WiSeConnect_SAPIS_Spansion_Project_User_guide.pdf*** for executing the ***mqtt_client*** example in CooCox IDE.

7. **UART/USB-CDC Interface**

If User using UART interface, Please refer the document ***sapis/platforms/ windows_uart/RS9113-WiSeConnect_SAPIS_Windows_Project_UserGuide.pdf*** for executing the ***mqtt_client*** example in Dev-C++ IDE

8. After the program gets executed, WiSeConnect Device will be connected to same Access point having the configuration same as that of in the application and get IP.
9. Once the WiSeConnect Device is connected to the MQTT broker, Device subscribe to the topic `RSI_MQTT_TOPIC (Ex: "REDPINE").` User can see the client connected and subscribe information in the MQTT broker,

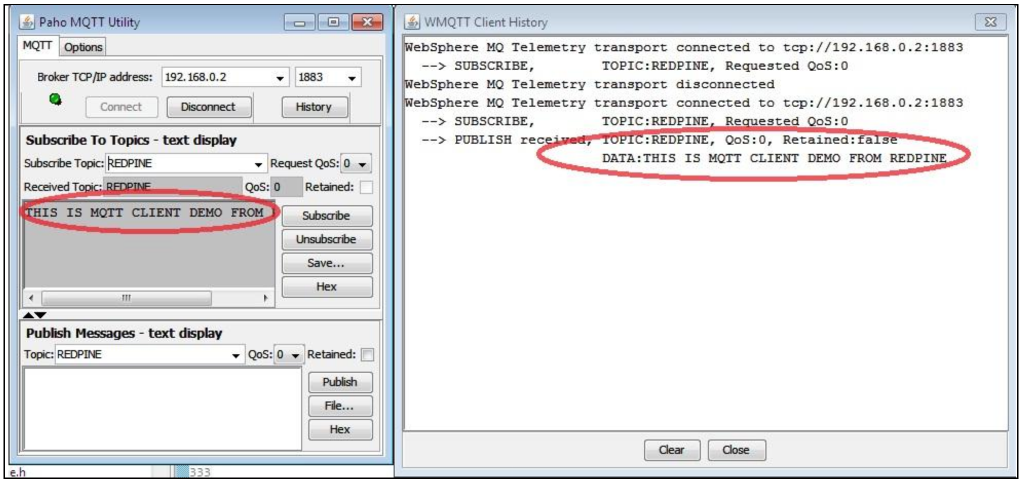


Administrator: C:\Windows\system32\cmd.exe - mosquitto.exe -p 1883 -v

```
C:\Program Files\mosquitto>mosquitto.exe -p 1883 -v
1459923726: mosquitto version 1.4.8 (build date 14/02/2016 15:33:31.09) starting

1459923726: Using default config.
1459923726: Opening ipv6 listen socket on port 1883.
1459923726: Opening ipv4 listen socket on port 1883.
1459923736: mosquitto version 1.4.8 terminating

C:\Program Files\mosquitto>

C:\Program Files\mosquitto>mosquitto.exe -p 1883 -v
1459923747: mosquitto version 1.4.8 (build date 14/02/2016 15:33:31.09) starting

1459923747: Using default config.
1459923747: Opening ipv6 listen socket on port 1883.
1459923747: Opening ipv4 listen socket on port 1883.
1459923760: New connection from 192.168.0.3 on port 1883.
1459923760: New client connected from 192.168.0.3 as MQTTCLIENT (c1, k100).
1459923760: Sending CONNACK to MQTTCLIENT (0, 0)
1459923774: Received SUBSCRIBE from MQTTCLIENT
1459923774:     REDPINE (QoS 0)
1459923774: MQTTCLIENT 0 REDPINE
1459923774: Sending SUBACK to MQTTCLIENT
```

10. After successful subscription to the topic **`RSI_MQTT_TOPIC (Ex: “REDPINE”),`** Device publish a message which is given in **`publish_message`** array (Ex: “THIS IS MQTT CLIENT DEMO FROM REDPINE”) on the subscribed topic.
11. MQTT client utility which is running on Windows PC3 will receive the message published by WiSeConnect device as it subscribed to the same topic.

Please refer the given below images for MQTT client utility image and message history.



Paho MQTT Utility

Broker TCP/IP address: 192.168.0.2 1883

Subscribe To Topics - text display

Subscribe Topic: REDPINE Request QoS: 0

Received Topic: REDPINE QoS: 0 Retained: □

THIS IS MQTT CLIENT DEMO FROM

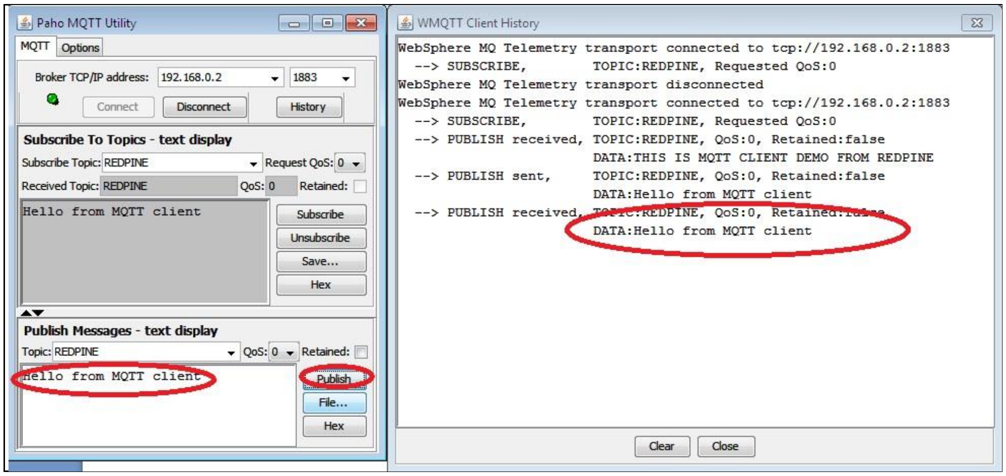
Publish Messages - text display

Topic: REDPINE QoS: 0 Retained: □

WMQTT Client History

```
WebSphere MQ Telemetry transport connected to tcp://192.168.0.2:1883
  --> SUBSCRIBE,        TOPIC:REDPINE, Requested QoS:0
WebSphere MQ Telemetry transport disconnected
WebSphere MQ Telemetry transport connected to tcp://192.168.0.2:1883
  --> SUBSCRIBE,        TOPIC:REDPINE, Requested QoS:0
  --> PUBLISH received, TOPIC:REDPINE, QoS:0, Retained:false
                        DATA:THIS IS MQTT CLIENT DEMO FROM REDPINE
```

12. Now publish a message using MQTT Utility on the same topic. Now this message is the message received by WiSeConnect Device.



Broker TCP/IP address: 192.168.0.2 1883

Subscribe Topic: REDPINE Request QoS: 0

Received Topic: REDPINE QoS: 0 Retained: □

```
Hello from MQTT client
```

Topic: REDPINE QoS: 0 Retained: □

```
Hello from MQTT client
```

```
WebSphere MQ Telemetry transport connected to tcp://192.168.0.2:1883
  --> SUBSCRIBE,        TOPIC:REDPINE, Requested QoS:0
WebSphere MQ Telemetry transport disconnected
WebSphere MQ Telemetry transport connected to tcp://192.168.0.2:1883
  --> SUBSCRIBE,        TOPIC:REDPINE, Requested QoS:0
  --> PUBLISH received, TOPIC:REDPINE, QoS:0, Retained:false
                        DATA:THIS IS MQTT CLIENT DEMO FROM REDPINE
  --> PUBLISH sent,     TOPIC:REDPINE, QoS:0, Retained:false
                        DATA:Hello from MQTT client
  --> PUBLISH received, TOPIC:REDPINE, QoS:0, Retained:false
                        DATA:Hello from MQTT client
```

NOTE: Multiple MQTT client instances can be created

3 Limitations

MQTT client application keeps on polling for the data to receive on the subscribed topic irrespective of receive timeout mentioned in the rsi`_mqtt_poll_for_recv_data` API.