

# User Manual

## DA16200 MQTT Programmer Guide

UM-WI-010

### Abstract

*This MQTT Programmer Guide intends to assist software developers that implement applications with the DA16200 SDK. A certain degree of reader familiarity with programming environments, debugging tools and software engineering process in general is assumed.*

---

---

## Contents

<b>Abstract</b> .....	<b>1</b>
<b>Contents</b> .....	<b>2</b>
<b>Figures</b> .....	<b>3</b>
<b>Tables</b> .....	<b>3</b>
<b>Terms and Definitions</b> .....	<b>4</b>
<b>References</b> .....	<b>4</b>
<b>1 Overview</b> .....	<b>5</b>
<b>2 SDK Build</b> .....	<b>5</b>
<b>3 Application Programming Interface</b> .....	<b>6</b>
3.1 Operation APIs .....	6
3.2 Configuration APIs .....	6
<b>4 Example Code</b> .....	<b>8</b>
4.1 MQTT Publisher .....	8
4.2 MQTT Subscriber .....	9
4.3 Receiving / Processing Message (From MQTT Publisher).....	9
4.4 Periodic Message Publishing.....	9
4.5 Running Sub and Pub at the Same Time (with DPM) .....	10
<b>5 Test</b> .....	<b>11</b>
5.1 Test Environment .....	11
5.2 Setup .....	11
5.3 Publisher .....	13
5.3.1 Non-QoS Message .....	13
5.3.2 QoS Message .....	15
5.3.3 MQTT over TLS .....	16
5.3.4 Username and Password .....	17
5.4 Subscriber .....	18
5.4.1 Setup.....	18
5.4.2 MQTT over TLS .....	18
5.4.3 Username and Password .....	19
5.4.4 WILL.....	20
5.5 MQTT Pub/Sub Test with DPM and TLS .....	20
5.5.1 DPM Power Profile .....	23
5.6 Reset.....	23
<b>6 Certificate</b> .....	<b>24</b>
6.1 Certificate Commands.....	24
6.2 CA, Client Cert, and Client Key.....	25
<b>Revision History</b> .....	<b>29</b>

**DA16200 MQTT Programmer Guide**

**Figures**

Figure 1: MQTT Messaging Concept ..... 5  
 Figure 2: Publish Non-QoS Message ..... 13  
 Figure 3: Publish QoS 1 Message..... 15  
 Figure 4: Publish QoS 2 Message..... 15  
 Figure 5: Configure Parameters and Publish a Message ..... 16  
 Figure 6: Publish Secure Message ..... 16  
 Figure 7: User Login ..... 17  
 Figure 8: MQTT Communication ..... 23

**Tables**

Table 1: MQTT API List..... 6  
 Table 2: Configuration API ..... 7  
 Table 3: MQTT Messaging Configuration (String Type) ..... 7  
 Table 4: MQTT Messaging Configuration (Integer Type)..... 7

---

## DA16200 MQTT Programmer Guide

### Terms and Definitions

MQTT	Dynamic Power Management
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
API	Application Programming Interface
AP	Access Point
QoS	Quality of Service
TLS	Transport Layer Security

### References

- [1] <DA16200>, Datasheet, Dialog Semiconductor
- [2] <DA16200>, SDK Programmer Guide, Dialog Semiconductor

DA16200 MQTT Programmer Guide

### 1 Overview

MQTT (Message Queue Telemetry Transport) is an ISO standard (ISO/IEC PRF 20922) publish-subscribe-based messaging protocol. It works on top of the TCP/IP protocol. The publisher sends (PUBLISH) messages to the subscriber through the broker. The subscriber needs to keep the connection with the broker by TCP session while the publisher can disconnect the session with the broker after sending a message.

As shown in Figure 1, once the broker receives a message with a specific topic the message is sent to subscribers that already registered with the topic. A subscriber can register with more than one topic. There can be many or no subscribers which registered with a specific topic.

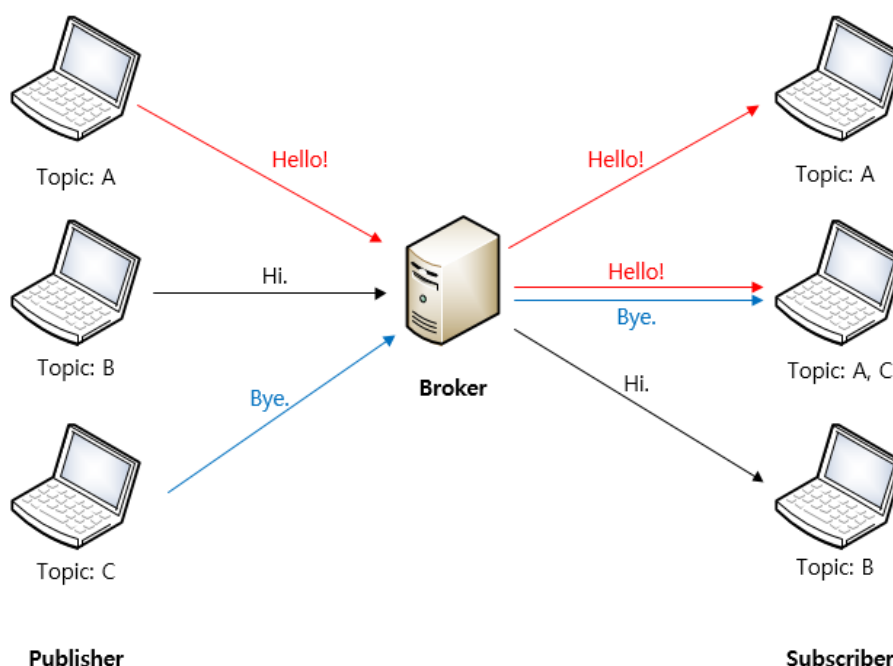


Figure 1: MQTT Messaging Concept

The exchange of MQTT messages supports QoS (Quality of Service). QoS has 3 levels (0, 1 and 2) and the process of each QoS level is as below.

The DA16200 supports both publisher and subscriber functions and allows simultaneous use. The subscriber function supports DPM mode. TLS is available for message encryption.

### 2 SDK Build

Some source files should be modified in the DA16200 SDK to use the MQTT function.

Enable the MQTT function as shown in the example below.

```
[\src\customer\config_generic_sdk.h
#define      SUPPORT MQTT          // Support MQTT
```

## DA16200 MQTT Programmer Guide

### 3 Application Programming Interface

#### 3.1 Operation APIs

The APIs listed in [Table 1](#) are used to create or terminate the MQTT thread, to check the status, and to publish a message. The configuration to execute MQTT protocols is explained in [section 0](#).

**Table 1: MQTT API List**

<code>int mqtt_client_start(void)</code>		
Return	If succeeded, returns 0. If failed, returns an error code.	
Description	Create the MQTT client thread.	
<code>int mqtt_client_stop(void)</code>		
Return	If succeeded, returns 0. If there is no thread to terminate, returns -1.	
Description	Terminate the MQTT client thread.	
<code>int mqtt_client_check_conn(void)</code>		
Return	1 (true): Connected to a broker. 0 (false): Not connected.	
Description	Check whether the MQTT session is connected.	
<code>int mqtt_client_send_message(char *top, char *publish)</code>		
Return	0: Succeeded to publish. -1: Failed to publish because Publisher is not ready to send.	
Parameter	top	Topic (if NULL, the MQTT publisher sends a PUBLISH message with the topic stored in NVRAM.)
	publish	Message to be published.
Description	Publisher sends an MQTT message (PUBLISH).	
<code>int mqtt_client_send_message_with_qos(char *top, char *publish, timeout)</code>		
Return	0: Succeeded to publish. -1: Failed to publish because Publisher is not ready to send. -2: Failed to publish because the timeout expired.	
Parameter	top	Topic (if NULL, the MQTT module sends a PUBLISH message with the topic stored in NVRAM.)
	publish	Message to be published.
	timeout	Timeout to wait for a previous QoS message to process completely (unit: 10 ms).
Description	Publisher sends an MQTT message (PUBLISH) with timeout check.	

#### 3.2 Configuration APIs

With `NVRAM` items, the user can configure MQTT messaging. This allows to configure the publisher and the subscriber.

## DA16200 MQTT Programmer Guide

**Table 2: Configuration API**

<code>int mqtt_client_config_initialize(void)</code>	
Return	If succeeded, returns 0 (MOSQ_ERR_SUCCESS. If failed, returns an error code.
Description	Reset all MQTT Configurations.

**Table 3: MQTT Messaging Configuration (String Type)**

Name	Description	Example
FC9K_CONF_STR_MQTT_BROKER_IP	Broker IP address (or URI).	<code>fc9k_set_config_str(FC9K_CONF_STR_MQTT_BROKER_IP, "192.168.0.1");</code>
FC9K_CONF_STR_MQTT_SUB_TOPIC	Subscriber topic (previous topics will be removed).	<code>fc9k_set_config_str(FC9K_CONF_STR_MQTT_SUB_TOPIC, topic);</code>
FC9K_CONF_STR_MQTT_SUB_TOPIC_ADD	Subscriber topic to add (up to 4).	<code>fc9k_set_config_str(FC9K_CONF_STR_MQTT_SUB_TOPIC_ADD, topic);</code>
FC9K_CONF_STR_MQTT_SUB_TOPIC_DEL	Subscriber topic to remove.	<code>fc9k_set_config_str(FC9K_CONF_STR_MQTT_SUB_TOPIC_DEL, topic);</code>
FC9K_CONF_STR_MQTT_PUB_TOPIC	Topic to publish.	<code>fc9k_set_config_str(FC9K_CONF_STR_MQTT_PUB_PUB_TOPIC, "pub topic");</code>
FC9K_CONF_STR_MQTT_USERNAME	Username to login to a broker.	<code>fc9k_set_config_str(FC9K_CONF_STR_MQTT_USERNAME, "mqtt id");</code>
FC9K_CONF_STR_MQTT_PASSWORD	Password to login to a broker.	<code>fc9k_set_config_str(FC9K_CONF_STR_MQTT_PASSWORD, "mqtt password");</code>
FC9K_CONF_STR_MQTT_WILL_TOPIC	Will Topic.	<code>fc9k_set_config_str(FC9K_CONF_STR_MQTT_WILL_TOPIC, "will topic");</code>
FC9K_CONF_STR_MQTT_WILL_MSG	Will Message.	<code>fc9k_set_config_str(FC9K_CONF_STR_MQTT_WILL_MSG, "will msg");</code>
FC9K_CONF_STR_MQTT_SUB_CLIENT_ID	MQTT client ID.	<code>fc9k_set_config_str(FC9K_CONF_STR_MQTT_SUB_CLIENT_ID, "sub id");</code>

**Note 1** Up to eight subscriber topics can be registered, and only one publisher topic can be registered.

**Table 4: MQTT Messaging Configuration (Integer Type)**

Name	Description	Example
FC9K_CONF_INT_MQTT_SUB	MQTT operation (0: stop, 1: start).	<code>fc9k_set_config_int(FC9K_CONF_INT_MQTT_SUB, 1);</code>
FC9K_CONF_INT_MQTT_AUTO	MQTT Auto-start at booting system (0: disable, 1: enable).	<code>fc9k_set_config_int(FC9K_CONF_INT_MQTT_AUTO, 1);</code>
FC9K_CONF_INT_MQTT_PORT	Broker port number.	<code>fc9k_set_config_int(FC9K_CONF_INT_MQTT_PORT, 8883);</code>
FC9K_CONF_INT_MQTT_QOS	QoS level (0~2).	<code>fc9k_set_config_int(FC9K_CONF_INT_MQTT_QOS, 2);</code>
FC9K_CONF_INT_MQTT_TLS	TLS (0: disable, 1: enable).	<code>fc9k_set_config_int(FC9K_CONF_INT_MQTT_TLS, 1);</code>
FC9K_CONF_INT_MQTT_WILL_QOS	QoS level of will messages (0~2).	<code>fc9k_set_config_int(FC9K_CONF_INT_MQTT_WILL_QOS, 1);</code>
FC9K_CONF_INT_MQTT_PING_PERIOD	MQTT ping period (secs).	<code>fc9k_set_config_int(FC9K_CONF_INT_MQTT_PING_PERIOD, 86400);</code>

## DA16200 MQTT Programmer Guide

### 4 Example Code

DA16200 MQTT publisher and subscriber are configured by NVRAM items. Once all configurations are done, you just need to run the subscriber thread or publish an MQTT message.

#### 4.1 MQTT Publisher

Set the configurations for the MQTT broker, publisher topic and so on (once after boot, or when you want to change). Call `mqtt_client_start()` and `mqtt_client_send_message()` with a message, then DA16200 will temporarily connect to the broker and publish the message.

```
#include "mqtt_client.h"
#include "common_config.h"

int mqtt_pub_example(void)
{
    int status;
    char send_msg[16] = {0, };

    strcpy(send_msg, "Hello broker.");

    fc9k_set_nvcache_str(FC9K_CONF_STR_MQTT_BROKER_IP, "172.16.0.1");
    fc9k_set_nvcache_int(FC9K_CONF_INT_MQTT_PORT, 1884);
    fc9k_set_nvcache_str(FC9K_CONF_STR_MQTT_PUB_TOPIC, "da16k_pub");
    fc9k_set_nvcache_str(FC9K_CONF_STR_MQTT_USERNAME, "username");
    fc9k_set_nvcache_str(FC9K_CONF_STR_MQTT_PASSWORD, "password");
    fc9k_set_nvcache_int(FC9K_CONF_INT_MQTT_QOS, 0);
    fc9k_set_nvcache_int(FC9K_CONF_INT_MQTT_TLS, 0);

    fc9k_nvcache2flash();

    status = mqtt_client_start();
    if (status)
    {
        PRINTF("Failed to initialize MQTT Client.");
        return status;
    }

    tx_thread_sleep(300); // sleep 3 sec. (=300 ticks)

mqtt_pub_send:
    status = mqtt_client_check_conn();
    if (!status)
    {
        mqtt_client_send_message(NULL, send_msg);
    }
    else
    {
        tx_thread_sleep(100); // sleep 1 sec.
        goto mqtt_pub_send;
    }

    return status;
}
```



## DA16200 MQTT Programmer Guide

### 4.2 MQTT Subscriber

Set the configurations for the MQTT broker, subscriber topic and so on (once before running the subscriber thread). Call `mqtt_client_start()` and then the subscriber thread will start.

```
#include "mqtt_client.h"
#include "common_config.h"

int mqtt_sub_example(void)
{
    int status;

    fc9k_set_nvcache_str(FC9K_CONF_STR_MQTT_BROKER_IP, "172.16.0.1");
    fc9k_set_nvcache_int(FC9K_CONF_INT_MQTT_PORT, 1884);
    fc9k_set_nvcache_str(FC9K_CONF_STR_MQTT_SUB_TOPIC, "da16k_sub");
    fc9k_set_nvcache_str(FC9K_CONF_STR_MQTT_USERNAME, "username");
    fc9k_set_nvcache_str(FC9K_CONF_STR_MQTT_PASSWORD, "password");
    fc9k_set_nvcache_int(FC9K_CONF_INT_MQTT_QOS, 0);
    fc9k_set_nvcache_int(FC9K_CONF_INT_MQTT_TLS, 0);

    fc9k_nvcache2flash();

    status = mqtt_client_start();

    return status;
}
```

### 4.3 Receiving / Processing Message (From MQTT Publisher)

When an MQTT message is received, it is received via a callback, which a user programmer needs to register. To be able to add a callback, section 4.2 should be done first (file `mqtt_client.h` should be included).

```
mqtt_client_set_msg_cb(mqtt_msg_cb);
```

The following example code shows a callback sample implementation. In this implementation, when a message is received, and if payload is "1", a certain message is printed.

```
static void mqtt_msg_cb (const char *buf, int len, const char *topic)
{
    if (strcmp(message->payload, "1", 1) == 0)
    {
        char msg[64] = {0, };

        sprintf(msg, "FC9K status: Not bad (%d)", ++mqtt_sample_msg_id);
        mqtt_client_send_message(NULL, msg);
    }
}
```

### 4.4 Periodic Message Publishing

When an MQTT Publisher session is connected, you can register a periodic message function. To be able to send it, you should call the callback API as below.

With MQTT Publisher, you can post a periodic message. Section 4.1 should be done first.

```
mqtt_client_set_pub_cb(mqtt_pub_cb);
```

## DA16200 MQTT Programmer Guide

```
#define MQTT_PUB_MSG_PERIODIC 30 // secs
static void mqtt_pub_cb(void)
{
    if (!dpm_mode_is_wakeup())
    {
        dpm_timer_create(SAMPLE_MQTT_CLIENT, "timer1",
                        mqtt_pub_send_periodic,
                        MQTT_PUB_MSG_PERIODIC,
                        MQTT_PUB_MSG_PERIODIC);
    }
}
```

For a duplicate RTC timer registration, it is only registered on Power-On-Boot. That means, when the RTC timer is expired, a message is printed on the console.

```
static void mqtt_pub_send_periodic(char *timer_name)
{
    char msg[64] = {0, };

    strcpy(msg, "DA16K Periodic Message");
    mqtt_client_send_message(NULL, msg);
}
```

### 4.5 Running Sub and Pub at the Same Time (with DPM)

```
void cmd_mqtt_sample(int argc, char *argv[])
{
    /* Wi-Fi Connection Setting */
    fc9k_set_nvcache_int(FC9K_CONF_INT_MODE, 0);
    fc9k_set_nvcache_str(FC9K_CONF_STR_SSID_0, "TEST_AP");
    fc9k_set_nvcache_int(FC9K_CONF_INT_AUTH_MODE_0, CC_VAL_AUTH_WPA2);
    fc9k_set_nvcache_int(FC9K_CONF_INT_ENCRYPTION_0, CC_VAL_ENC_CCMP);
    fc9k_set_nvcache_str(FC9K_CONF_STR_PSK_0, "12345678");

    /* MQTT Setting */
    fc9k_set_nvcache_int(FC9K_CONF_INT_MQTT_AUTO, 1);
    fc9k_set_nvcache_str(FC9K_CONF_STR_MQTT_BROKER_IP, "172.16.0.1");
    fc9k_set_nvcache_int(FC9K_CONF_INT_MQTT_PORT, 8883);
    fc9k_set_nvcache_int(FC9K_CONF_INT_MQTT_QOS, 0);
    fc9k_set_nvcache_int(FC9K_CONF_INT_MQTT_TLS, 1);
    fc9k_set_nvcache_str(FC9K_CONF_STR_MQTT_SUB_TOPIC, "da16k");
    fc9k_set_nvcache_str(FC9K_CONF_STR_MQTT_PUB_TOPIC, "da16k_sub");
    fc9k_set_nvcache_int(FC9K_CONF_INT_MQTT_PING_PERIOD, 60);

    /* DPM after Rebooting */
    fc9k_set_nvcache_int(FC9K_CONF_INT_DPM, 1);

    /* Enabled Sntp for TLS */
    fc9k_set_nvcache_int(FC9K_CONF_INT_SNTP_CLIENT, 1);

    fc9k_nvcache2flash();

    /* Input Certificate & Private Key */
    cert_flash_write(FLASH_ROOT_CA_ADDR1, (char *)cert_buffer0);
    cert_flash_write(FLASH_CERTIFICATE_ADDR1, (char *)cert_buffer1);
    cert_flash_write(FLASH_PRIVATE_KEY_ADDR1, (char *)cert_buffer2);
}
```

## DA16200 MQTT Programmer Guide

```
reboot_func(SYS_REBOOT, DISCONNECT_SEND);
}
```

## 5 Test

This section explains how to test the MQTT function on the DA16200 debug console window.

### 5.1 Test Environment

For this test the mosquitto MQTT broker is used, which you can download from the following url:

<https://mosquitto.org/download/>

If you feel that the broker installation is difficult, Dialog Semiconductor can provide it so that you can extract and run it on your Windows PC.

### 5.2 Setup

Open a command window and go to the mosquitto folder.

1. Run a broker.

```
mosquitto -v -p <Port Number>
```

```
C:\#mosquitto>mosquitto -v -p 1884
1582173416: mosquitto version 1.4.14 (build date 11/07/2017 0:03:18.53) starting
1582173416: Using default config.
1582173416: Opening ipv6 listen socket on port 1884.
1582173416: Opening ipv4 listen socket on port 1884.
```

2. Open a new command window and run a subscriber.

```
mosquitto_sub -h <Broker IP> -p <Port Number> -t <Topic>
```

```
C:\#mosquitto>mosquitto_sub -h 172.16.30.163 -p 1884 -t da16k
```

The following message is shown in the broker window.

```
C:\#mosquitto>mosquitto -v -p 1884
1582173276: mosquitto version 1.4.14 (build date 11/07/2017 0:03:18.53) starting
1582173276: Using default config.
1582173276: Opening ipv6 listen socket on port 1884.
1582173276: Opening ipv4 listen socket on port 1884.
1582173309: New connection from 172.16.30.163 on port 1884.
1582173309: New client connected from 172.16.30.163 as mosqsub|13800-KR-ENG-LT (c1, k60)
1582173309: Sending CONNACK to mosqsub|13800-KR-ENG-LT (0, 0)
1582173309: Received SUBSCRIBE from mosqsub|13800-KR-ENG-LT
1582173309: da16k (QoS 0)
1582173309: mosqsub|13800-KR-ENG-LT 0 da16k
1582173309: Sending SUBACK to mosqsub|13800-KR-ENG-LT
```

3. Open a new command window and publish a message.

```
mosquitto_pub -h <Broker IP> -p <Port Number> -t <Topic> -m "<Message>"
```

```
C:\#mosquitto>mosquitto_pub -h 172.16.30.163 -p 1884 -t da16k -m "Hello World!"
```

The following message is shown in the broker window.

---

**DA16200 MQTT Programmer Guide**

---

```
582173567: New connection from 172.16.30.163 on port 1884.  
582173567: New client connected from 172.16.30.163 as mosqpub|3508-KR-ENG-LT- (c1, k60).  
582173567: Sending CONNACK to mosqpub|3508-KR-ENG-LT- (0, 0)  
582173567: Received PUBLISH from mosqpub|3508-KR-ENG-LT- (d0, q0, r0, m0, 'da16k', ... (12 bytes))  
582173567: Sending PUBLISH to mosqsub|17076-KR-ENG-LT (d0, q0, r0, m0, 'da16k', ... (12 bytes))  
582173567: Received DISCONNECT from mosqpub|3508-KR-ENG-LT-  
582173567: Client mosqpub|3508-KR-ENG-LT- disconnected.
```

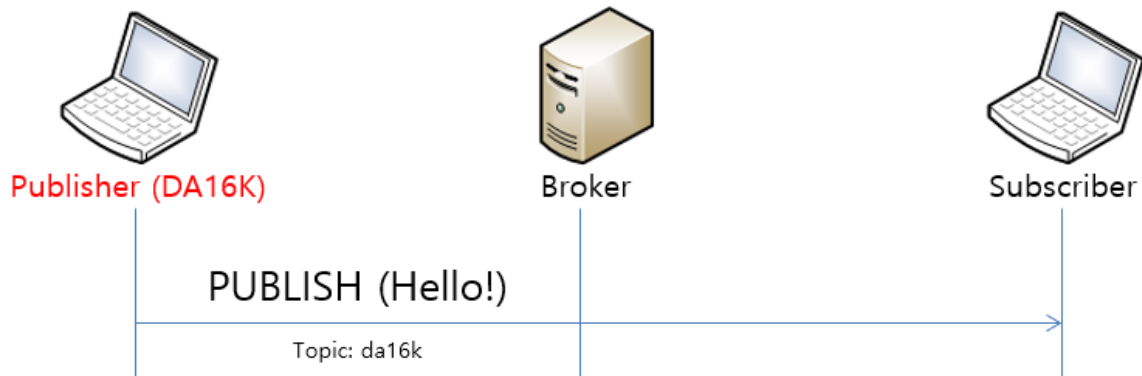
The subscriber receives the message.

```
C:\#mosquitto>mosquitto_sub -h 172.16.30.163 -p 1884 -t da16k  
Hello World!
```

### 5.3 Publisher

#### 5.3.1 Non-QoS Message

Here is an example of publishing a non-QoS message.



**Figure 2: Publish Non-QoS Message**

1. After the DA16200 EVB is connected to an AP, configure the parameters and publish a message.

```

[/DA16200]# net
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config pub_topic <Topic>
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)

[/DA16200/NET]# mqtt_client -m <Message>
...
[/DA16200/NET]# mqtt_client stop
    
```

Optionally, "client\_id" can also be set using the following command.

```
[/DA16200/NET]# mqtt_config client_id <client_id_string>
```

e.g.) `mqtt_config client_id abcd1111`

client\_id should be unique per each device. By default client\_id is generated internally such as "da16x\_<the last 2 bytes of mac address>" : e.g.) da16x\_FCFA

2. When message transmission `-m "Hello!"` is successful, you can see the following messages:

Hello! (Send, Len: 6, Topic: da16k, Message ID: 1) The following syntax allows to send a message with a new topic:

```
[/DA16200/NET] mqtt_client -m <Message> <NewTopic>
```

If the previous parameters for broker, port and topics are not changed, then you do not need to set the parameters for the publication of every message.

\* The max length of the console command is 158. To send a longer PUBLISH, write the following command:

```

[/DA16200/NET] mqtt_client -l
Typing data: (MQTT Publisher message)
Cancel - CTRL+D, End of Input - CTRL+C or CTRL+Z
    
```



DA16200 MQTT Programmer Guide

5.3.2 QoS Message

This section gives an example of publishing a QoS message.

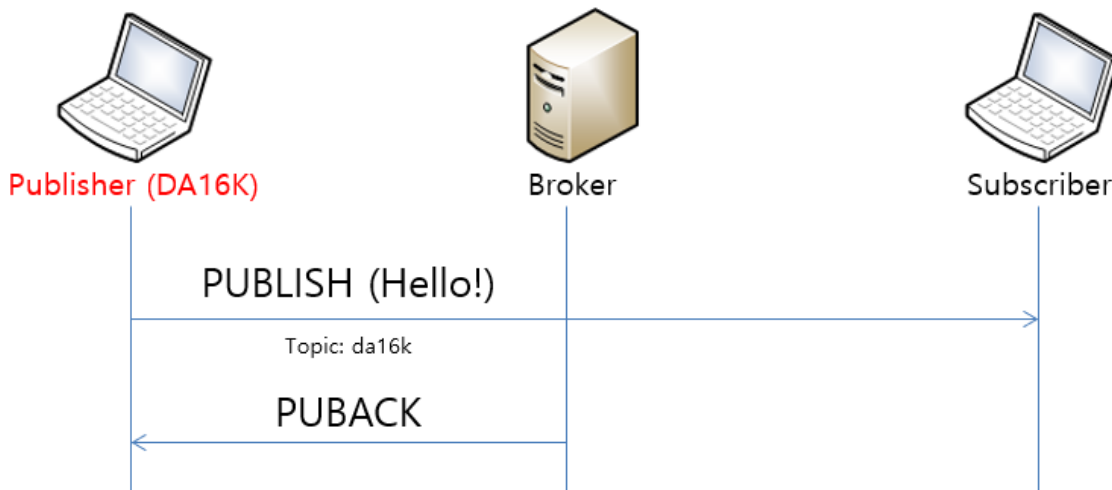


Figure 3: Publish QoS 1 Message

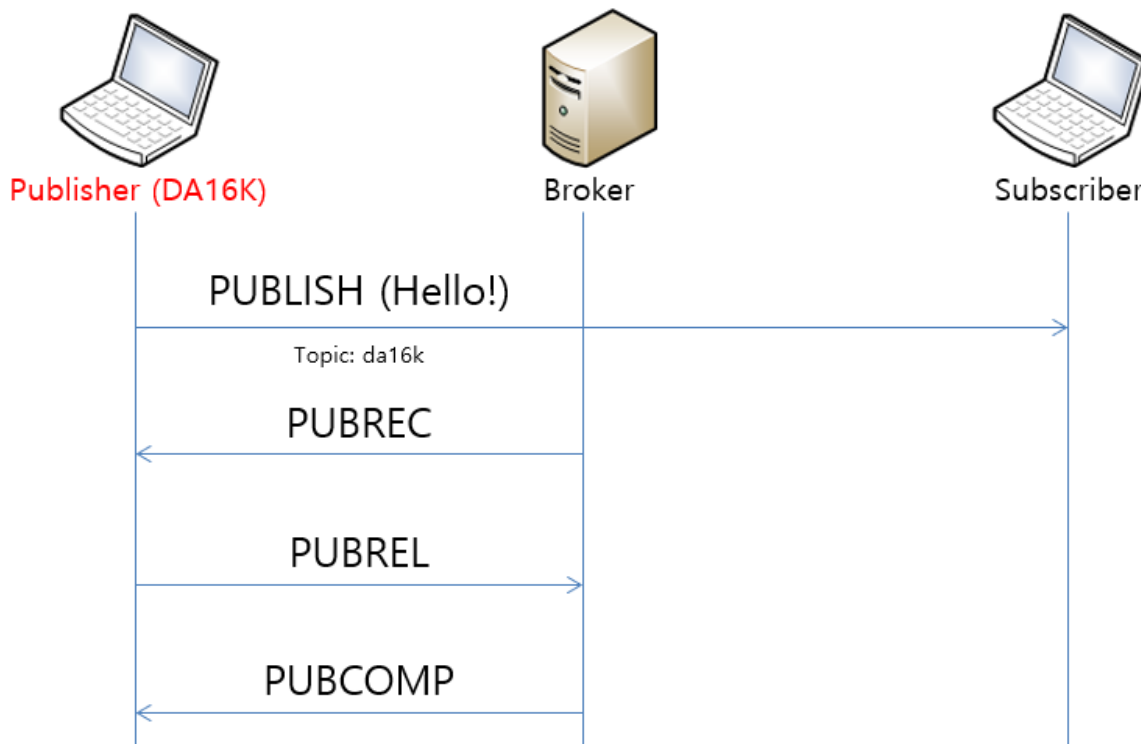


Figure 4: Publish QoS 2 Message

## DA16200 MQTT Programmer Guide

1. Configure the parameters and publish a message.

```
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config pub_topic <Topic>
[/DA16200/NET]# mqtt_config qos <QoS Level>
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)

[/DA16200/NET]# mqtt_client -m <Message>
```

```
1582175804: Received PUBLISH from PUB-da16x_FD26 (d0, q1, r0, m1, 'da16k', ... (6 bytes))
1582175804: Sending PUBACK to PUB-da16x_FD26 (Mid: 1)
```

```
1582175859: Received PUBLISH from PUB-da16x_FD26 (d0, q2, r0, m1, 'da16k', ... (6 bytes))
1582175859: Sending PUBREC to PUB-da16x_FD26 (Mid: 1)
1582175859: Received PUBREL from PUB-da16x_FD26 (Mid: 1)
1582175859: Sending PUBCOMP to PUB-da16x_FD26 (Mid: 1)
```

Figure 5: Configure Parameters and Publish a Message

### 5.3.3 MQTT over TLS

The DA16200 SDK provides a TLS encrypted session for secure MQTT messages.

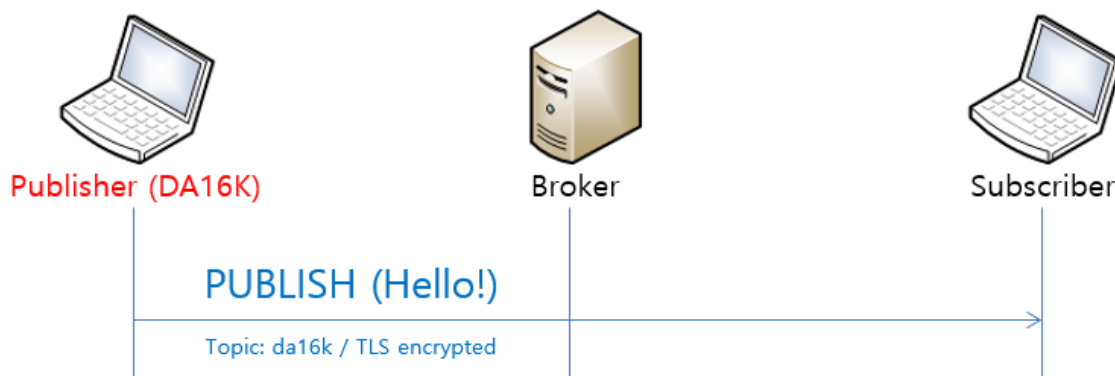


Figure 6: Publish Secure Message

**Note 1** You need to store certificates in the DA16200 EVK to use TLS encryption. This procedure is explained in section 6.

1. Run a broker with a secure port.

```
mosquitto -c mosquitto.conf -p <Port Number> -v
```

```
C:\#mosquitto>mosquitto -c mosquitto.conf -p 8883 -v
1582174980: mosquitto version 1.4.14 (build date 11/07/2017 0:09:18.53) starting
1582174980: Config loaded from mosquitto.conf.
1582174980: Opening ipv6 listen socket on port 8883.
1582174980: Opening ipv4 listen socket on port 8883.
```

2. Run a subscriber.

```
mosquitto_sub -h <Broker IP> -p <Port> --cafile <CA Certificate> --cert <Client Certificate> --key <Client Private Key> --tls-version <TLS Protocol Version> --insecure -t <Topic>
```



## DA16200 MQTT Programmer Guide

```
C:\#mosquitto>mosquitto_sub -h 172.16.30.163 -p 8883 --cafile cas.pem --cert wifiuser.pem --key wifiuser.key --tls-version tlsv1 --insecure -t da16k
```

3. Set the current time in the DA16200 EVB in order to check if the certificate is valid.  
(If you want to use SNTP for time sync, the input the command "net.sntp enable" to get the current time.)

```
[/DA16200]# time set <yyyy-mm-dd> <hh:mm:ss>
```

4. First store 3 Certificates (see section 6.1) in the DUT, and then follow the steps below.

```
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config pub_topic <Topic>
[/DA16200/NET]# mqtt_config tls 1
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)

[/DA16200/NET]# mqtt_client -m <Message>
```

### 5.3.4 Username and Password

1. Set up a user name and password to authenticate users.

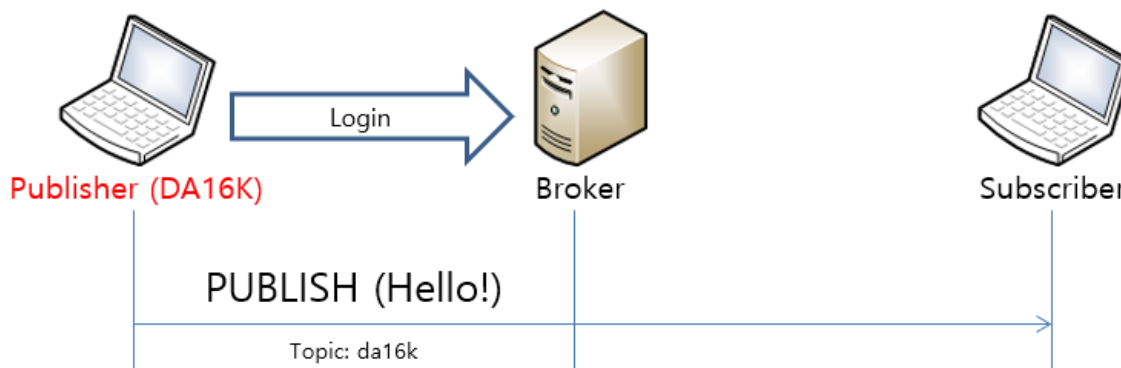


Figure 7: User Login

2. Run a broker with a secure port. You need to prepare the configuration file.

```
mosquitto -c <Config File> -p <Port> -v
```

```
C:\#mosquitto>mosquitto -c mosq_idpw.conf -p 1900 -v
1582176530: mosquitto version 1.4.14 (build date 11/07/2017 0:03:18.53) starting
1582176530: Config loaded from mosq_idpw.conf.
1582176530: Opening ipv6 listen socket on port 1900.
1582176530: Opening ipv4 listen socket on port 1900.
```

In the mosquitto package provided by Dialog Semiconductor, file `mosq_idpw.conf` is used for the <Config File> parameter, and user accounts are registered in file `p1.txt`.

3. You can add a new account in this file with the following command:

```
mosquitto_passwd.exe -b p1.txt <username> <password>
```

4. At the mosquitto command prompt, please run the `mosquitto_sub` command to login successfully to the broker.

```
mosquitto_sub -h <broker_ip> -p <port> -t <topic> -u <id> -P <pass>
```

## DA16200 MQTT Programmer Guide

- On `mqtt_client` (DUT), set the username and password, and try.

```
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config pub_topic <Topic>
[/DA16200/NET]# mqtt_config tls 0
[/DA16200/NET]# mqtt_config username <Username>
[/DA16200/NET]# mqtt_config password <Password>
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)

[/DA16200/NET]# mqtt_client -m <Message>
```

## 5.4 Subscriber

### 5.4.1 Setup

- Configure the parameters and start the subscriber.

```
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config sub_topic 1 <Topic>
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)
...
[/DA16200/NET]# mqtt_client stop
```

- You can register multiple topics. You should add the parameter for the number of topics in the command (up to 4).

```
[/DA16200/NET]# mqtt_client stop
[/DA16200/NET]# mqtt_config sub_topic <Topic count> <Topic#1> <Topic#2> ...
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)
[/DA16200/NET]# mqtt_config sub_topic add <New topic>
[/DA16200/NET]# mqtt_config sub_topic del <Topic to remove>
```

### 5.4.2 MQTT over TLS

You need to set the current time in the DA16200 EVB to check if the certificate is valid.  
(if SNTP is auto-started during boot, you do not need to do this step.)

```
[/DA16200]# time set <yyyy-mm-dd> <hh:mm:ss>
```

- Run the broker as below.

```
mosquitto -c mosquitto.conf -p <Port Number> -v
```

- First add 3 Certificates (see section 6.1) for the DUT, and then do the steps below.

```
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config sub_topic 1 <Topic>
[/DA16200/NET]# mqtt_config tls 1
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)
```

- Run a publisher on your PC.

```
mosquitto_pub -h <Broker IP> -p <Port> --cafile <CA Certificate> --cert <Client Certificate> --key <Client Private Key> --tls-version <TLS Protocol Version> -t <Topic> --insecure -m <message>
```

---

## DA16200 MQTT Programmer Guide

---

```
Example: mosquitto_pub -h 192.168.0.101 -p 1884 --cafile cas.pem --cert
wifiuser.pem --key wifiuser.key --tls-version tlsv1 -t da16k --insecure -m "hello"
```

### 5.4.3 Username and Password

1. DUT: Set username and password.

```
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config sub_topic 1 <Topic>
[/DA16200/NET]# mqtt_config tls 0
[/DA16200/NET]# mqtt_config username <Username>
[/DA16200/NET]# mqtt_config password <Password>
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)
```

2. In the mosquitto package provided by Dialog Semiconductor, file `mosq_idpw.conf` is used for the <Config File> parameter and user accounts are registered in file `p1.txt`. You can add a new account in this file with the following command.

```
mosquitto_pub -h [Broker IP] -p [port] -t [topic] -m <message> -u [id] -P
[password]
```

Example:

```
mosquitto_pub -h 192.168.0.101 -p 1884 -t da16k -u mike -P 1234 -m hello
```

## DA16200 MQTT Programmer Guide

### 5.4.4 WILL

1. Sub#1 (DUT): Set the will message.

```
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config sub_topic 1 <Topic>
[/DA16200/NET]# mqtt_config will_topic <Topic>
[/DA16200/NET]# mqtt_config will_message <Message>
[/DA16200/NET]# mqtt_config will_qos <QoS Level>
[/DA16200/NET]# mqtt_client start
>>> MQTT Client connection OK (da16x_FFFE)
```

2. Broker: write the following command.

```
>mosquitto -v -p 1884
```

3. Sub#2 (PC): write the following command.

```
>mosquitto_sub -h 192.168.0.101 -t da16k -p 1884 -q 0
```

4. Sub#1 (DUT): now please try an unexpected disconnection.

```
[/DA16200/NET]# reset

>>> Network Interface (wlan0): DOWN
[mqtt_subscriber_main] Request mqtt_restart
[wpa_supplicant_event_disassoc] CTRL-EVENT-DISCONNECTED bssid=ec:08:6b:d6:53:62
reason=3 locally_generated=1

DA16200 ROM-Boot [ffffc000]
[MROM]
```

5. Sub#2 (PC): wait until the following will message is printed.

```
>mosquitto_sub -h 192.168.0.101 -t da16k -p 1884 -q 2
imwill
```

### 5.5 MQTT Pub/Sub Test with DPM and TLS

In this test, the Pub and Sub are run with DPM mode enabled. In addition, an MQTT sample implementation (4.3 and 4.4) is also enabled where the message callback and the Periodic Pub message posting are implemented.

1. Broker: run with TLS enabled.

```
>mosquitto -c mosquitto.conf -p 8883 -v
```

2. Sub#2 (PC): write the following command.

```
>mosquitto_sub -h 192.168.0.101 -p 8883 --cafile cas.pem --cert wifiuser.pem --key
wifiuser.key --tls-version tlsv1 -t da16k --insecure
```

3. Sub-Pub#1 (DUT): write the following command.

```
[/DA16200/NET]# mqtt_config auto 1
[/DA16200/NET]# mqtt_config broker <Broker IP>
[/DA16200/NET]# mqtt_config port <Port Number>
[/DA16200/NET]# mqtt_config sub_topic 1 <Topic>
[/DA16200/NET]# mqtt_config pub_topic <Topic>
[/DA16200/NET]# mqtt_config tls 1
[/DA16200/NET]# sntp_enable
[/DA16200/NET]# nvram.setenv dpm_mode 1
[/DA16200/NET]# reboot
```

## DA16200 MQTT Programmer Guide

```

Connection COMPLETE to 88:36:6c:d9:b0:68

-- DHCP Client WLAN0: SEL
-- DHCP Client WLAN0: REQ
-- DHCP Client WLAN0: BOUND
    Assigned addr   : 192.168.0.35
      netmask      : 255.255.255.0
      gateway     : 192.168.0.1
      DNS addr    : 168.126.63.1

    DHCP Server IP : 192.168.0.1
    Lease Time     : 24h 00m 00s
    Renewal Time  : 20h 00m 00s

>>> SNTP Server: pool.ntp.org (211.233.84.186)
>>> SNTP Time sync : 2020.04.09 - 02:14:51
>>> MQTT client connection OK (da16x_FFFE)

>>> Start DPM Power-Down !!!
  
```

#Pub (PC): you can try to send the Pub message as below.

```
>mosquitto_pub -h 192.168.0.101 -p 1884 --cafile cas.pem --cert wifiuser.pem --key
wifiuser.key --tls-version tlsv1 -t da16k --insecure -m "Hello World!!"
```

When the message is received, DA16200 wakes up from DPM Sleep and prints the message.

```

wakeup source is 0x82

>>> TIM STATUS: 0x00000001
>>> TIM : UC
Hello world! (Recv, Len: 12, Topic: da16k, Message ID: 0)

>>> Start DPM Power-Down !!!
rwnx_send_set_ps_mode PS TIME (us) 440326
  
```

If the code examples in sections 4.3 to 4.5 are applied, MQTT publisher starts to post a periodic message every 30 seconds and the MQTT subscriber processes the received PUBLISH messages.

```

(5)rtc_timeout
>>> TIM STATUS: 0x00000010
>>> TIM : FAST
DA16K Periodic Message (14) (Send, Len: 27, Topic: _da16k, Message ID: 15)

>>> Start DPM Power-Down !!!
rwnx_send_set_ps_mode PS TIME (us) 470123
  
```

---

**DA16200 MQTT Programmer Guide**

---

```
wakeup source is 0x82
>>> TIM STATUS: 0x00000001
>>> TIM : UC
1 (Recv, Len: 1, Topic: da16k, Message ID: 0)
DA16K status : Not bad (9) (send, Len: 26, Topic: _da16k, Message ID: 10)
>>> Start DPM Power-Down !!!
```

## DA16200 MQTT Programmer Guide

### 5.5.1 DPM Power Profile

With Keysight, a current consumption measuring tester, you can check how DPM works in MQTT communication. DPM allows the system to stay in sleep mode most of the time and only wake up (and stay active for only a small amount of time to get the job done) when needed.

In the Keysight snapshot below, DA16200 was in sleep mode until it woke up to post a periodic status message to the broker. Once DA16200 received the response, it enters and stays in sleep mode until the next Status Message Tx time (the interval depends on application).

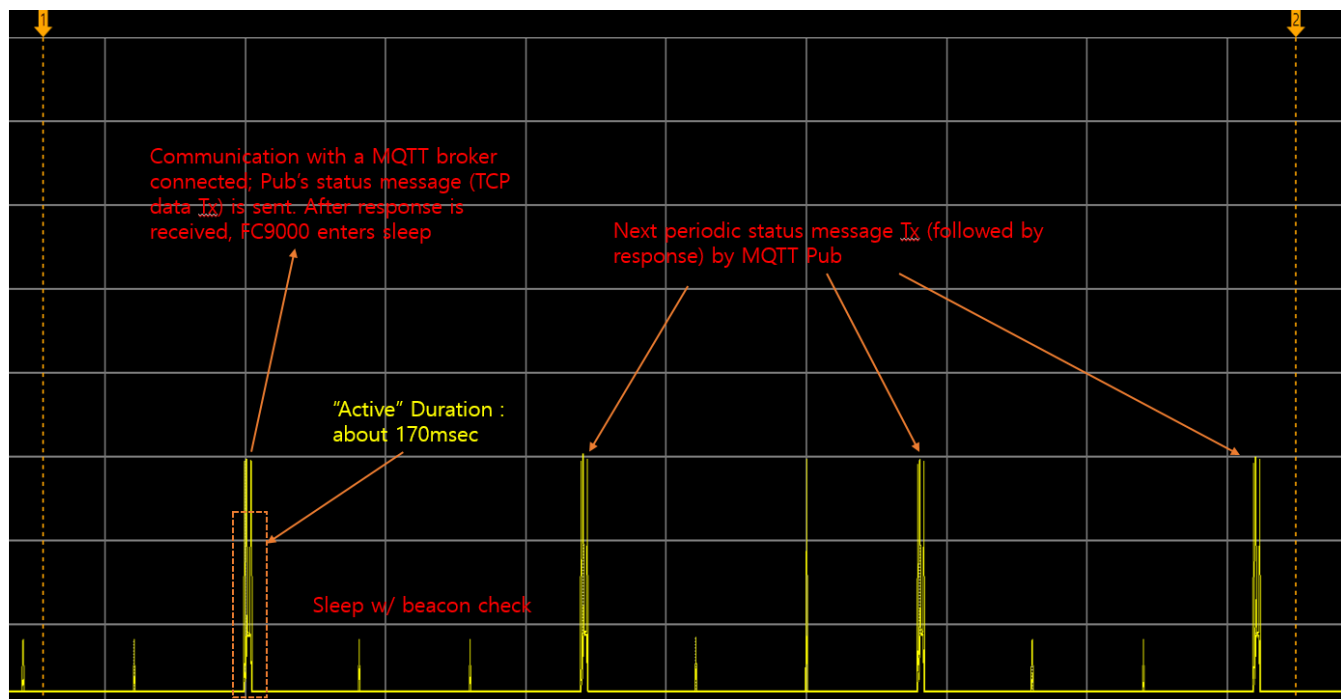


Figure 8: MQTT Communication

### 5.6 Reset

The following command clears all MQTT configurations:

```
[/DA16200/NET]# mqtt config reset
```

## DA16200 MQTT Programmer Guide

### 6 Certificate

DA16200 provides methods to store certificates in the serial flash with the use of console commands.

#### 6.1 Certificate Commands

1. Store a CA certificate.

```
[/DA16200/NET]# net
[/DA16200/NET]# cert 0
Typing data: (certificate value)
Cancel - CTRL+D, End of Input - CTRL+C or CTRL+Z
// Copy & paste certificate data in the terminal window and press "CTRL+C" or "CTRL+Z"
(see section 6.2)
```

2. Store a client certificate.

```
[/DA16200/NET]# cert 1
Typing data: (certificate value)
Cancel - CTRL+D, End of Input - CTRL+C or CTRL+Z
// Copy & paste certificate data in the terminal window and press "CTRL+C" or "CTRL+Z"
(see section 6.2)
```

3. Store a client key.

```
[/DA16200/NET]# cert 2
Typing data: (certificate value)
Cancel - CTRL+D, End of Input - CTRL+C or CTRL+Z
// Copy & paste certificate data in the terminal window and press "CTRL+C" or "CTRL+Z"
(see section 6.2)
```

4. After adding cert/keys, please check if they are successfully stored.

```
[/DA16200/NET]# cert
#1 (MQTT, Enterprise)
Root CA: O
Certificate: O
Private Key: O
DH Parameter: X
#2 (HTTPs, CoAPs Client)
Root CA: X
Certificate: X
Private Key: X
DH Parameter: X
```

In case you want to remove all the credentials stored:

```
[/DA16200/NET]# cert 3
```



DA16200 MQTT Programmer Guide

6.2 CA, Client Cert, and Client Key

- Cert 0: CA

```

-----BEGIN CERTIFICATE-----
MIID+TCCAUgGawIBAgIJANqgHCazDkkOMA0GCSqGSIb3DQEBCwUAMIGSMQswCQYD
VQQGEwJVUzETMBEGA1UECAwKQ2FsaWZvcml5YUUEUMBIGA1UEBwwLU2FudGEgQ2xh
cmExFzAVBgNVBAoMD1dpLUZpIEFsbG1hbmNlMR0wGwYDVQDDBRXRkEgUm9vdCBD
ZXJ0aWZpY2F0ZTEGMB4GCSqGSIb3DQEJARYRc3VwcG9ydEB3aS1maS5vcmcwHhcN
MTMwMzExMTkwMjI2WhcnMjMwMzA5MTkwMjI2WjCBkjELMAKGA1UEBhMCVVMxEzAR
BgNVBAGMCKNhG1mb3JuaWEeFDASBgNVBACMC1NhbnRhIENsYXJhMRcwFQYDVQK
DA5XaS1GaSBbbGxpYW5jZTEdMBSGA1UEAwwUV0ZBIFJvb3QgQ2VydG1maWNhdGUx
IDAeBgkqhkiG9w0BCQEWEXN1cHBvcnRAD2ktZmkub3JnMIIBIjANBgkqhkiG9w0B
AQEFAAOCAQ8AMIIBCgKCAQEAE6TOCu20m+9zLZITYAhGmtxwyJQ/1xytXSQJYX8LN
YUS/N3HG2QAQ4GKDh7DPDI13zhdc0yOUE1CIOXa1ETKbHIU9xABrL7KfX2HCQ1nC
PqRPiW9/wgQch8Aw7g/0rXmg1zewPJ36zKnq5/5Q1uyd8YfaXBzhxmiIY1wTKM1C
ixDFcAeVqHb74mAcde111xdagHvaL56fpUExm7GyMGXYd+Q2vYa/olUwCMGfMOj6
FLHwKpy62KCoK3016H1WU1bpg8YgPLDt2BB4LzxmPfyH2x+Xj75mAc1lOxx7GK0r
cGPPINrSr4vgo1tm4Bh1eIW57h+gXoFfHCJLMG66uhU/2QIDAQABolAwTjAdBgNV
HQ4EFgQUcWPCPLSiKL0+Sd5y8V+Oqw6XZ4IwHwYDVR0jBBGwFoAUCwPCPLSiKL0+
Sd5y8V+Oqw6XZ4IwDAYDVR0TBAAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAsNxO
z9DXb7TkNFKtPOY/7lZig4Ztdu6Lgf6qEUOvJGW/Bw1Wx1PMjPpk9oI+JdR8ZZ4B
9QhE+LZhg6SjBjK+VJqUCtVnXWdg0e8CgeUw718GNZithIE1WYK3Kh1cSo3sJt0P
z9CiJfjwtdWsdAqC9zV9tgp09QkEkav84X20VxaITa3H1QuK/LWSn/ORrzcx0I1
10YoF6 Hz3Zwa65mUoMzd8DyTcyGtcbYrSt+NMCGRB186PDQn5XBCytgF8VuiCyyk
Z04hghLZAFc21P9yhwKGI3BHD/Sep8fvr9y4VpMIqHQM2jaFPxY1VxhPSV+UHoE1
fCPitIJTp/iXi7uXTQ==
-----END CERTIFICATE-----
    
```

- Cert 1: Client Cert

```

-----BEGIN CERTIFICATE-----
MIIEBTCCAu2gAwIBAgICEEYwDQYJKoZIhvcNAQELBQAwZlIxZCZAJBgNVBAYTA1VT
MRMwEQYDVQQIDApDYWxpZm9ybmlhMRQwEgYDVQHDAtTYW50YSBDbGFyYUUEXMBUG
A1UECgwOV2ktRmkqQWwSaWFuY2UxHTAbBgNVBAMFFdGQsSBSb290IENlcnRpZmlj
YXR1MSAwwH4YJKoZIhvcNAQkBFhFzdXBw3J0Q0HdpLWZpLm9yZzAeFw0xMzA1MTAy
MzQ0NTFhFw0yMzA1MDgyMzQ0NTFhFmH4xCZAJBgNVBAYTA1VTMRMwEQYDVQQIDApD
YWxpZm9ybmlhMRcwFQYDVQKDA5XaS1GaSBbbGxpYW5jZTEfMBOGA1UEAwwWQ2xp
ZW50IENlcnRpZmljYXR1IE1ETDEgMB4GCSqGSIb3DQEJARYRc3VwcG9ydEB3aS1m
aS5vcmcwgwEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCdco7kQ4W2b/fBw
2ZgSAXVWBCmJW0yax8K682kRiH1B1aJ5Im8rTEktZ1PDQVho03Ur+Ij9Y8ukD3Hq
Cma95T1a3Ly9lhDIME/VRqRgZRGa/FC/jkiK9u9SgIXPZqJk1s/JVG7a7deC8BvK
iqIFhXoH10N4nJxwVA8kag48dXbrTxxOH26C9qwU85iS/clozHJMmq052WPSQZII
Sq8+Vmx1CbbXxQ7kU2ozkxDqW3hMI3OPS80s8q4tMzuitvzF00JvAHgh4IFyE7yg
nIE7+1M9f3EwzECw9nEBdL7AvfnYLh1IEI8S8wZTUpond8XA5Qs7Qa4rLNuqI273Z
IWJLh9v/AgMBAAGjEDB2MA8GA1UdEwEB/wQFMAMCAQAwCwYDVR0PBAQDAgXgMByG
A1UdJQEB/wQMAoGCCsGAQUFBwMCMBOGA1UdDgQWBBQtC2mx8POhZRfB+sJ4wX1Z
OzdrCzAfBgNVHSMEGDAWgBQLA8I+VKIovT5J3nLxX46rDpdngjANBgkqhkiG9w0B
AQsFAAOCAQEAsvHJ+J2YTCS EA69vjSmsGoJ2iEXDuHI+57jIo+8qrVK/mlisleiU
AefExDtxxKTEPPtourlY08A4i7oep9T43Be8nwVZdmxzfU14cdLKQRE+viCuHQTC
BLSoAv6/SZa3MeIRkkDSPTCPTJ/Pp+VYPK8gPlc9pwEs/KLgFxFK/Sq6RDNjTPs6J
MChxiliSdUES8mz3JDhQ2RQWuibPorhgVqNTyXBjFubYxTV13hBCtK/Bd4IyFTX
Li90XXHseNb2sHu3PFBU7PG5mhKQMUOYqvQzEDIXT6SDA+PrepqrwXn/BL861K6
GV4LcfKBg0HHdW9gJBYZCN02igFTw56Kzg==
-----END CERTIFICATE-----
    
```

DA16200 MQTT Programmer Guide

- Cert 2: Client Key

```

-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAA3KO5EOfTm/3wcNmYEgF1VgQpiVtMmsfCuvNpEYh5QdWieSjv
K0xJLWzTw0FYaDt1K/iI/WPLpA9x6gjGveU9Wty8vZYQyDBF1UakYGURmvxQv45I
ivbvUoCFz2aiZNBpyVRu2u3XgvAbyoqiBYV6B5dDeJyccFQPJGGoOPHV2608azh9u
gvasFPQYkv3NaMxyTJqtOd1j0kGSCeqvPlZsZQm218UO5FNqGZMQ61t4TCNzj0vN
LPKuLTM7orb8xTtCbwb4IeCBch08oJyBO/pTPX9xMMxAsPZxAXS+wL352C4ZSBCP
EvMGU1KZ3fFwOULO0GuKyzbqiNu92SFis4fb/wIDAQABAoIBAQDcnbCc2mt5AM98
Z3aQ+nhSy9Kkj2/njDqAKIc0ituEIPNUwEOcbaj2Bk1W/W3iuyEMGHURuMmUgAUN
WD0w/5j705+9ieG56eTJgts1r5mM+SHch+6tVQAz5GLn4N4cKlaWHyDBM/S77k47
lacwEijUkkFaxm3+O27woEMf30xNl24KmRenMYBhqcsot4BYBw3Bh8xe+XN95rXj
2BdIbr5+RWGc9Zsz4o5Wmd4mL/JvbKeohrsecien4TZRzWFku93XV5kielc1aJy1
nJ85bGJk4focmP/2ToxQysTbPYCxHVTIHuADK/qf9SGHJ9F7EBHE7+0isuwBbqOD
OzS8rHdRAoGBAPCXlaHumEkLIRv3enhpHPBYxnDndNctT1T6+Cuit/vfo6K6oA7p
iUaej/GPZsDKXhayeTiEaq7QMinUtGkiCgGlVtXghXuCZz6KrH19W6wzC6Pbokmq
BZak4LQcvGavt3VzjliAKLcdn6nQt/+bp/jKDJOKVbvb30sjS035Ah4zAoGBAOrF
BgE9UTEEnfQHTh7pyiM1DAomBbdrlRos8maQl26cHqUHN3+wy1bGHLzOjYFFoAasx
eizw7Gudgbae28WIPlyLGrpt15cqVAvlCYmBtZ3C98FuT3FYgEEZpWNmE8Om+5UM
td+mtMjonWAPkCYC+alqUZzeIs+Czs5CHKYCDqcFAoGBAOfkQv38GV2102jARJPQ
RGtINaRXApmrod43s4Fjac/kAzVyiZk18PFXHUhnlMt+jgIN5yIzMoHtsHo2SbH
/zsM4MBuklm0G80FHjIp5HT6EksSA77amF5VdptDYzfaP4p+IYIdrKCqddzYZrCA
mArMvAhs+iuCRhuG3is+SZNPaoGAHs6r8w2w0dp0tP8zkGvnN8hLVO//EnJzx2G0
Z63wHQMMWu5BLCWfLsRANW6C/SvAzE450hvralPI6cX+4PT4G5TFdSFk4RlU3hq4
Has/wewLxv5Kvz215Rd96U1gr8ulGh01YKyxop/3FMuf050pJ6nBwa/WquqAf6b
+23ZrmECgYEA6l0GFHwMFBnnpPuxHgYgS5+4g3+8DhZZIDc7If1BCBWF/ZwbM+nH
+JSxiYYjvD7zIBhndqERCz+fvbZTQ8oymr3j5AESM0ZfAHbft6IFQWjDUC3IDUF/
4F0cUIdFC8smu6Wa2tjvSIz7DfvmDsn1l+7s9qQvDxdyPas0IkL/v8w=
-----END RSA PRIVATE KEY-----
    
```

---

**DA16200 MQTT Programmer Guide**

---

// Mosquitto 1.4.14 License

Eclipse Distribution License 1.0

Copyright (c) 2007, Eclipse Foundation, Inc. and its licensors.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification,  
are permitted provided that the following conditions are met:

- \* Redistributions of source code must retain the above copyright notice,  
this list of conditions and the following disclaimer.
- \* Redistributions in binary form must reproduce the above copyright notice,  
this list of conditions and the following disclaimer  
in the documentation and/or other materials provided with the distribution.
- \* Neither the name of the Eclipse Foundation, Inc.  
nor the names of its contributors may be used to endorse or  
promote products derived from this software without  
specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"  
AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,  
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  
PURPOSE  
ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE  
LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR  
CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF  
SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS  
INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN  
CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)  
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE  
POSSIBILITY OF SUCH DAMAGE.

-----

// MiniUPnPc License

Copyright (c) 2005-2016, Thomas BERNARD

---

## DA16200 MQTT Programmer Guide

---

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- \* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- \* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- \* The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

---

UMAC GPL License

Linux kernel 3.9.0 rc3 version(backport 4.2.6-1)

---

## Revision History

Revision	Date	Description
1.4	20-Aug-2020	Command "mqtt_config client" added
1.3	30-Mar-2020	Several small updates.
1.2	26-Nov-2019	Finalized for publication
1.1	15-Nov-2019	Editorial review
1.0	30-Aug-2019	Preliminary DRAFT Release

## DA16200 MQTT Programmer Guide

### Status Definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

### Disclaimer

Unless otherwise agreed in writing, the Dialog Semiconductor products (and any associated software) referred to in this document are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of a Dialog Semiconductor product (or associated software) can reasonably be expected to result in personal injury, death or severe property or environmental damage. Dialog Semiconductor and its suppliers accept no liability for inclusion and/or use of Dialog Semiconductor products (and any associated software) in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Information in this document is believed to be accurate and reliable. However, Dialog Semiconductor does not give any representations or warranties, express or implied, as to the accuracy or completeness of such information. Dialog Semiconductor furthermore takes no responsibility whatsoever for the content in this document if provided by any information source outside of Dialog Semiconductor.

Dialog Semiconductor reserves the right to change without notice the information published in this document, including, without limitation, the specification and the design of the related semiconductor products, software and applications. Notwithstanding the foregoing, for any automotive grade version of the device, Dialog Semiconductor reserves the right to change the information published in this document, including, without limitation, the specification and the design of the related semiconductor products, software and applications, in accordance with its standard automotive change notification process.

Applications, software, and semiconductor products described in this document are for illustrative purposes only. Dialog Semiconductor makes no representation or warranty that such applications, software and semiconductor products will be suitable for the specified use without further testing or modification. Unless otherwise agreed in writing, such testing or modification is the sole responsibility of the customer and Dialog Semiconductor excludes all liability in this respect.

Nothing in this document may be construed as a license for customer to use the Dialog Semiconductor products, software and applications referred to in this document. Such license must be separately sought by customer with Dialog Semiconductor.

All use of Dialog Semiconductor products, software and applications referred to in this document is subject to Dialog Semiconductor's [Standard Terms and Conditions of Sale](#), available on the company website ([www.dialog-semiconductor.com](http://www.dialog-semiconductor.com)) unless otherwise stated.

Dialog, Dialog Semiconductor and the Dialog logo are trademarks of Dialog Semiconductor Plc or its subsidiaries. All other product or service names and marks are the property of their respective owners.

© 2020 Dialog Semiconductor. All rights reserved

## Contacting Dialog Semiconductor

#### United Kingdom (Headquarters)

*Dialog Semiconductor (UK) LTD*  
Phone: +44 1793 757700

#### Germany

*Dialog Semiconductor GmbH*  
Phone: +49 7021 805-0

#### The Netherlands

*Dialog Semiconductor B.V.*  
Phone: +31 73 640 8822

#### Email:

[enquiry@diasemi.com](mailto:enquiry@diasemi.com)

#### North America

*Dialog Semiconductor Inc.*  
Phone: +1 408 845 8500

#### Japan

*Dialog Semiconductor K. K.*  
Phone: +81 3 5769 5100

#### Taiwan

*Dialog Semiconductor Taiwan*  
Phone: +886 281 786 222

#### Web site:

[www.dialog-semiconductor.com](http://www.dialog-semiconductor.com)

#### Hong Kong

*Dialog Semiconductor Hong Kong*  
Phone: +852 2607 4271

#### Korea

*Dialog Semiconductor Korea*  
Phone: +82 2 3469 8200

#### China (Shenzhen)

*Dialog Semiconductor China*  
Phone: +86 755 2981 3669

#### China (Shanghai)

*Dialog Semiconductor China*  
Phone: +86 21 5424 9058