User Manual

DA16200 DA16600 MQTT Programmer Guide

UM-WI-010

Abstract

This MQTT Programmer Guide intends to assist software developers that implement applications with the DA16200 and DA16600 SDK. A certain degree of reader familiarity with programming environments, debugging tools, and software engineering process in general.

Contents

Figures

Tables

Terms and Definitions

MQTT	Message Queuing Telemetry Transport
DPM	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, Renesas Electronics

[2] DA16200 DA16600, SDK Programmer Guide, Renesas Electronics

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.

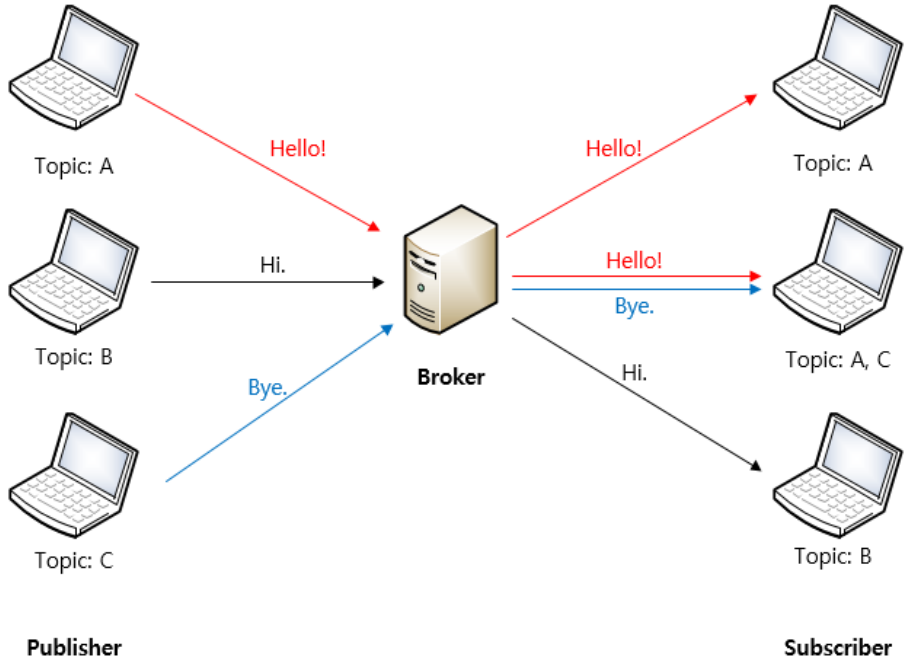

Figure 1: MQTT Messaging Concept

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

The DA16200 (DA16600) 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 (DA16600) SDK to use the MQTT function.

Enable the MQTT function as shown in the example below.

```
config_generic_sdk.h
...
#define         __SUPPORT_MQTT__              // Support MQTT
```

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 the next section.

Table 1: MQTT API List

`int mqtt_client_start(void)`	
Return	If succeeded, returns 0. If failed, returns an error code.
Description	Create the MQTT client thread.

`int mqtt_client_stop(void)`	
Return	If succeeded, returns 0. If there is no thread to terminate, returns -1.
Description	Terminate the MQTT client thread.

`int mqtt_client_check_conn(void)`	
Return	1 (true): Connected to a broker. 0 (false): Not connected.
Description	Check whether the MQTT session is connected.

`int mqtt_client_send_message(char *top, char *publish)`		
Return		0: Succeeded to publish. -1: Failed because mqtt is not connected. -2: Failed because the previous message send is in progress. -3: Failed because mqtt topic is missing. Other: Failed due to other cause. See enum “mqtt_client_error_code” to identify the cause.
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).

`int mqtt_client_send_message_with_qos(char *top, char *publish, timeout)`		
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 a timeout check.

int mqtt_client_unsub_topic(char *topic)		
Return		0: Succeeded to unsubscribe. 4: Failed because mqtt is not connected. 3: Failed because the topic is NULL 1: Failed because of memory alloc failure Other: Failed due to other cause. See enum “mqtt_client_error_code” to identify the cause.
Parameter	topic	Topic to unsubscribe
Description		Unsubscribe from the topic specified. Invoke this function only when mqtt client is in a connected state with Broker.

3.2 Configuration APIs

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

Table 2: Configuration API

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

void mqtt_sub_callback_set(void (*user_cb)(void))		
Return		None
Parameter	user_cb	user callback function to set
Description		Register a callback function that is invoked when a MQTT Subscriber is connected with a Broker.

void mqtt_pub_callback_set(void (*user_cb)(void))		
Return		None
Parameter	user_cb	user callback function to set
Description		Register a callback function that is invoked when publishing a message is done

void mqtt_msg_callback_set(void (*user_cb)(const char *buf, int len, const char *topic))		
Return		None
Parameter	user_cb	user callback function to set buf: PUBLISH message received len: PUBLISH message length topic: the topic of the PUBLISH message received
Description		Register a callback function that is invoked when a PUBLISH message arrives.

void mqtt_sub_disconn_cb_set(void (*user_cb)(void));		
Return		None
Parameter	user_cb	user callback function to set
Description		Register a callback function that is invoked when the MQTT client is disconnected

void mqtt_sub_disconn2_cb_set(void (*user_cb)(void));		
Return		None
Parameter	user_cb	user callback function to set
Description		- Register callback function called when MQTT Subscriber is disconnected by receiving a message with invalid unsupported length if the connection is clean_session=0 and qos > 0. - On receipt of this callback, application needs to clear the message in Broker by connecting with clean_sesion=1. - To use this API, __MQTT_CLEAN_SESSION_MODE_SUPPORT__ should be enabled. Check the section **오류! 참조 원본을 찾을 수 없습니다.**

void mqtt_subscribe_callback_set(void (*user_cb)(void));		
Return		None
Parameter	user_cb	user callback function to set
Description		Register a callback function that is invoked when a SUBSCRIBE request to a topic is finished

void mqtt_unsubscribe_callback_set(void (*user_cb)(void));		
Return		None
Parameter	user_cb	user callback function to set
Description		Register a callback function that is invoked when an UNSUBSCRIBE request is finished

Table 3: MQTT Messaging Configuration (String Type)

Name	Description	Example
DA16X_CONF_STR_MQTT_BROKER_IP	Broker IP address (or URI).	da16x_set_config_str(DA16X_CONF_STR_MQTT_BROKER_IP, "192.168.0.1");
DA16X_CONF_STR_MQTT_SUB_TOPIC	Subscriber topic (previous topics will be removed).	da16x_set_config_str(DA16X_CONF_STR_MQTT_SUB_TOPIC, topic);
DA16X_CONF_STR_MQTT_SUB_TOPIC_ADD	Subscriber topic to add (up to four).	da16x_set_config_str(DA16X_CONF_STR_MQTT_SUB_TOPIC_ADD, topic);

Name	Description	Example
DA16X_CONF_STR_MQTT_SUB_TOPIC_DEL	Subscriber topic to remove.	da16x_set_config_str(DA16X_CONF_STR_MQTT_SUB_TOPIC_DEL, topic);
DA16X_CONF_STR_MQTT_PUB_TOPIC	Topic to publish.	da16x_set_config_str(DA16X_CONF_STR_MQTT_PUB_PUB_TOPIC, "pub_topic");
DA16X_CONF_STR_MQTT_USERNAME	Username to log in to a broker.	da16x_set_config_str(DA16X_CONF_STR_MQTT_USERNAME, "mqtt_id");
DA16X_CONF_STR_MQTT_PASSWORD	Password to login to a broker.	da16x_set_config_str(DA16X_CONF_STR_MQTT_PASSWORD, "mqtt_password");
DA16X_CONF_STR_MQTT_WILL_TOPIC	Will Topic.	da16x_set_config_str(DA16X_CONF_STR_MQTT_WILL_TOPIC, "will_topic");
DA16X_CONF_STR_MQTT_WILL_MSG	Will Message.	da16x_set_config_str(DA16X_CONF_STR_MQTT_WILL_MSG, "will_msg");
DA16X_CONF_STR_MQTT_SUB_CLIENT_ID	MQTT client ID.	da16x_set_config_str(DA16X_CONF_STR_MQTT_SUB_CLIENT_ID, "sub_id");
DA16X_CONF_STR_MQTT_TLS_SNI	MQTT TLS SNI (Server Name Indication)	da16x_set_config_str(DA16X_CONF_STR_MQTT_TLS_SNI, "sni_str")

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

Table 4: MQTT Messaging Configuration (Integer Type)

Name	Description	Example
DA16X_CONF_INT_MQTT_SUB	MQTT operation (0: stop, 1: start).	da16x_set_config_int(DA16X_CONF_INT_MQTT_SUB, 1);
DA16X_CONF_INT_MQTT_AUTO	MQTT Auto-start at booting system (0: disable, 1: enable).	da16x_set_config_int(DA16X_CONF_INT_MQTT_AUTO, 1);
DA16X_CONF_INT_MQTT_PORT	Broker port number.	da16x_set_config_int(DA16X_CONF_INT_MQTT_PORT, 8883);
DA16X_CONF_INT_MQTT_QOS	QoS level (0~2).	da16x_set_config_int(DA16X_CONF_INT_MQTT_QOS, 2);
DA16X_CONF_INT_MQTT_TLS	TLS (0: disable, 1: enable).	da16x_set_config_int(DA16X_CONF_INT_MQTT_TLS, 1);
DA16X_CONF_INT_MQTT_WILL_QOS	QoS level of will messages (0~2).	da16x_set_config_int(DA16X_CONF_INT_MQTT_WILL_QOS, 1);
DA16X_CONF_INT_MQTT_PING_PERIOD	MQTT ping period (secs).	da16x_set_config_int(DA16X_CONF_INT_MQTT_PING_PERIOD, 86400);
DA16X_CONF_INT_MQTT_VER311	MQTT protocol : 1 (v3.1.1) / 0 (v3.1)	da16x_set_config_int(DA16X_CONF_INT_MQTT_VER311, 1)
DA16X_CONF_INT_MQTT_TLS_INCOMING	TLS incoming buffer size: default (1024*4), min (1024*2), max (1024*8)	da16x_set_config_int(DA16X_CONF_INT_MQTT_TLS_INCOMING, 1024*4)
DA16X_CONF_INT_MQTT_TLS_OUTGOING	TLS outgoing buffer size: default (1024*4), min (1024*2), max (1024*8)	da16x_set_config_int(DA16X_CONF_INT_MQTT_TLS_OUTGOING, 1024*4)

Name	Description	Example
DA16X_CONF_INT_MQTT_TLS_AUTHMODE	TLS peer certificate verification mode: 0 (not verify), 1 (optional), 2 (required), default is 1	da16x_set_config_int(DA16X_CONF_INT_MQTT_TLS_AUTHMODE, 1)
DA16X_CONF_INT_MQTT_CLEAN_SESSION	MQTT Clean Session mode (1: clean the previous session, 0: do not clean the previous session)	da16x_set_config_int(DA16X_CONF_INT_MQTT_CLEAN_SESSION, 1);

4 Example Code

DA16200 (DA16600) 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 (DA16600) 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.");

	da16x_set_nvcache_str(DA16X_CONF_STR_MQTT_BROKER_IP, "172.16.0.1");
	da16x_set_nvcache_int(DA16X_CONF_INT_MQTT_PORT, 1884);
	da16x_set_nvcache_str(DA16X_CONF_STR_MQTT_PUB_TOPIC, "da16k_pub");
	da16x_set_nvcache_str(DA16X_CONF_STR_MQTT_USERNAME, "username");
	da16x_set_nvcache_str(DA16X_CONF_STR_MQTT_PASSWORD, "password");
	da16x_set_nvcache_int(DA16X_CONF_INT_MQTT_QOS, 0);
	da16x_set_nvcache_int(DA16X_CONF_INT_MQTT_TLS, 0);

	da16x_nvcache2flash();

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

	tx_thread_sleep(300); // sleep 3 sec.(=300 ticks) /* for SDK V2.x.x.x */
	// vTaskDelay(300); /* for SDK V3.x.x.x */

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. /* for SDK V2.x.x.x */
		// vTaskDelay(100); /* for SDK V3.x.x.x */

		goto mqtt_pub_send;
	}

	return status;
}
```

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;

     da16x_set_nvcache_str(DA16X_CONF_STR_MQTT_BROKER_IP, "172.16.0.1");
     da16x_set_nvcache_int(DA16X_CONF_INT_MQTT_PORT, 1884);
     da16x_set_nvcache_str(DA16X_CONF_STR_MQTT_SUB_TOPIC, "da16k_sub");
     da16x_set_nvcache_str(DA16X_CONF_STR_MQTT_USERNAME, "username");
     da16x_set_nvcache_str(DA16X_CONF_STR_MQTT_PASSWORD, "password");
     da16x_set_nvcache_int(DA16X_CONF_INT_MQTT_QOS, 0);
     da16x_set_nvcache_int(DA16X_CONF_INT_MQTT_TLS, 0);

     da16x_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);
```

NOTE

The user thread that registers a message callback should have the same or higher priority than the “mqtt_client” thread (the priority of the mqtt_client thread is currently USER_PRI_APP(1)) for mqtt_client to be able to register the user callback before MQTT initialization.

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

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

              sprintf(msg, "DA16X 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_pub_callback_set(mqtt_pub_cb);
```

```
#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 */
        da16x_set_nvcache_int(DA16X_CONF_INT_MODE, 0);
        da16x_set_nvcache_str(DA16X_CONF_STR_SSID_0, "TEST_AP");
        da16x_set_nvcache_int(DA16X_CONF_INT_AUTH_MODE_0, CC_VAL_AUTH_WPA2);
        da16x_set_nvcache_int(DA16X_CONF_INT_ENCRYPTION_0, CC_VAL_ENC_CCMP);
        da16x_set_nvcache_str(DA16X_CONF_STR_PSK_0, "12345678");

        /* MQTT Setting */
        da16x_set_nvcache_int(DA16X_CONF_INT_MQTT_AUTO, 1);
        da16x_set_nvcache_str(DA16X_CONF_STR_MQTT_BROKER_IP, "172.16.0.1");
        da16x_set_nvcache_int(DA16X_CONF_INT_MQTT_PORT, 8883);
        da16x_set_nvcache_int(DA16X_CONF_INT_MQTT_QOS, 0);
        da16x_set_nvcache_int(DA16X_CONF_INT_MQTT_TLS, 1);
        da16x_set_nvcache_str(DA16X_CONF_STR_MQTT_SUB_TOPIC, "da16k");
        da16x_set_nvcache_str(DA16X_CONF_STR_MQTT_PUB_TOPIC, "da16k_sub");
        da16x_set_nvcache_int(DA16X_CONF_INT_MQTT_PING_PERIOD, 60);

        /* DPM after Rebooting */
        da16x_set_nvcache_int(DA16X_CONF_INT_DPM, 1);

        /* Enabled SNTP for TLS */
        da16x_set_nvcache_int(DA16X_CONF_INT_SNTP_CLIENT, 1);
```

```
        da16x_nvcache2flash();

        /* Input Certificate & Private Key */
        cert_flash_write(SFLASH_ROOT_CA_ADDR1, (char *)cert_buffer0,
                         strlen(cert_buffer0));
          cert_flash_write(SFLASH_CERTIFICATE_ADDR1, (char *)cert_buffer1,
                          strlen(cert_buffer1));
          cert_flash_write(SFLASH_PRIVATE_KEY_ADDR1, (char *)cert_buffer2,
                           strlen(cert_buffer2));
        reboot_func(SYS_REBOOT, DISCONNECT_SEND);
}
```

4.6 MQTT Client Sample

MQTT client sample is found in [SDK V2.x.x.x: ~/apps/common/examples/Network/MQTT_Client/src/mqtt_user_sample.c] or [SDK V3.x.x.x: ~/apps/common/examples/Network/MQTT_Client/src/mqtt_client_sample.c].

This simple application demonstrates receiving and sending an MQTT message. For building, running, and sample guide, see ~/doc/html/mqtt_client_sample.html (SDK V2.x.x.x).

5 Test

This section explains how to test the MQTT function on the DA16200 (DA16600) 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.

```
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

This section gives an example of publishing a non-QoS message.

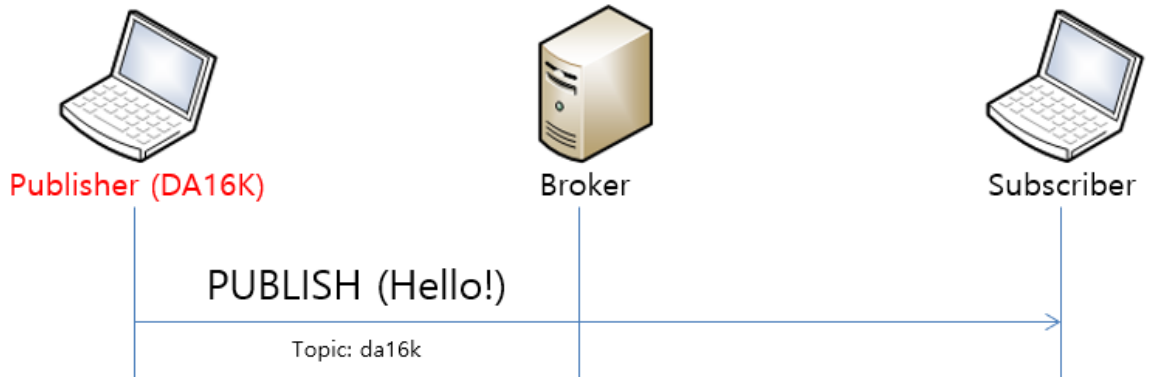

Figure 2: Publish Non-QoS Message

1. After the DA16200 (DA16600) 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 with the following command:

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

For example, `mqtt_config client_id abcd1111`

client_id should be unique per each device. By default, client_id is generated internally like “da16x_<the last 2 bytes of mac address>”. For example, 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
123456789012345678901234567890123456789012345678901234567890123456789012345678901
345678901234567890123456789012345678901234567890123456789012345678901234567890123
567890123456789012345678901234567890123456789012345678901234567890123456789012345
7890 …
```

Use the keyboard combinations Ctrl+C or Ctrl+Z to send the message.

5.3.2 QoS Message

This section gives an example of publishing a QoS message.

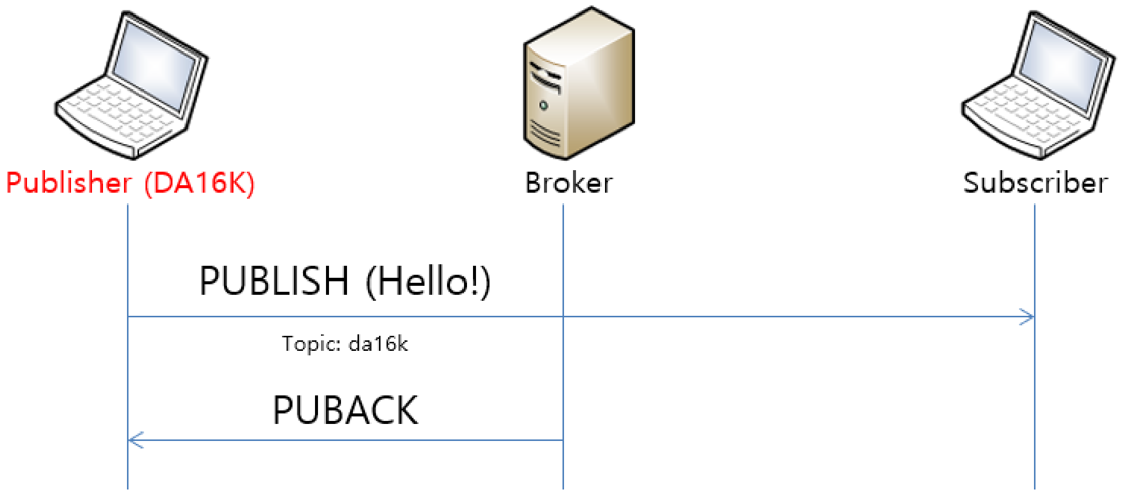

Figure 3: Publish QoS 1 Message

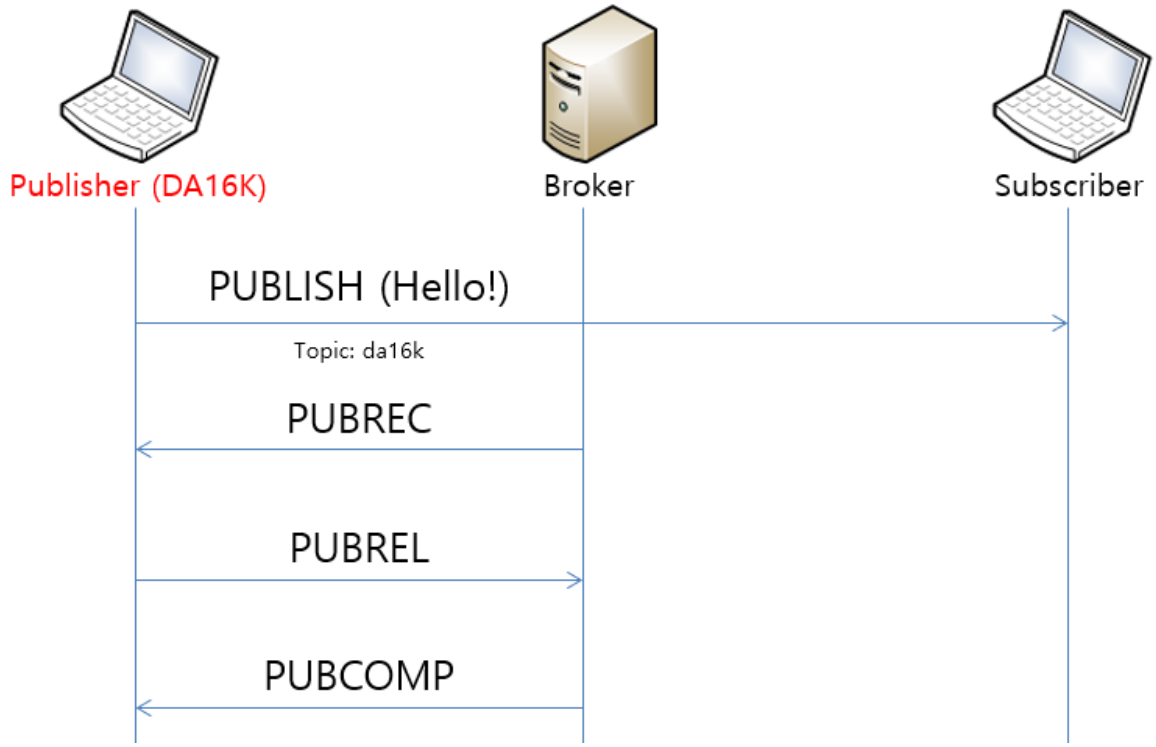

Figure 4: Publish QoS 2 Message

- 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 (DA16600) SDK provides a TLS encrypted session for secure MQTT messages.

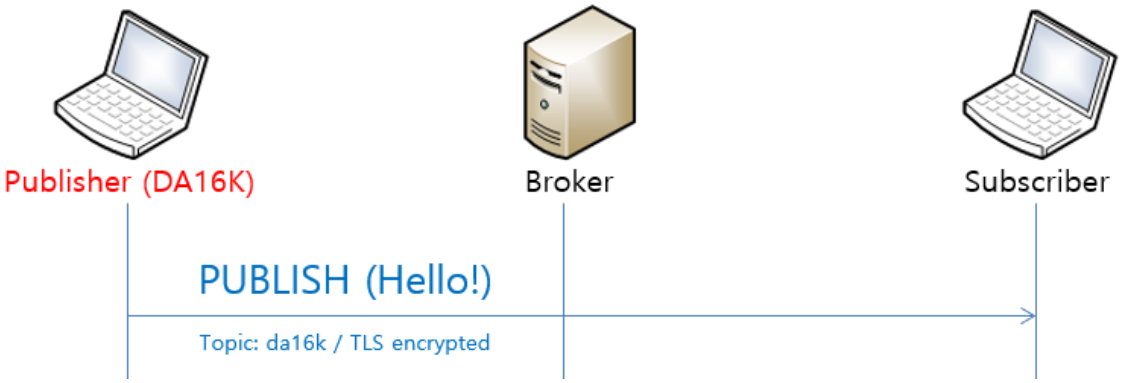

Figure 6: Publish Secure Message

NOTE
You need to store certificates in the DA16200 (DA16600) 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  0:03:18.53) 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>
```

```
C:₩mosquitto>mosquitto_sub -h 172.16.30.163 -p 8883 --cafile cas.pem --cert wifiuser.pem --key wifiuser.key --tls-ver
sion tlsv1 --insecure -t da16k
```

3. Set the current time in the DA16200 (DA16600) EVB to check if the certificate is valid.
   (If you want to use SNTP for time sync, input the command “net.sntp enable” to get the current time.)

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

4. Store three 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 username and password to authenticate users.

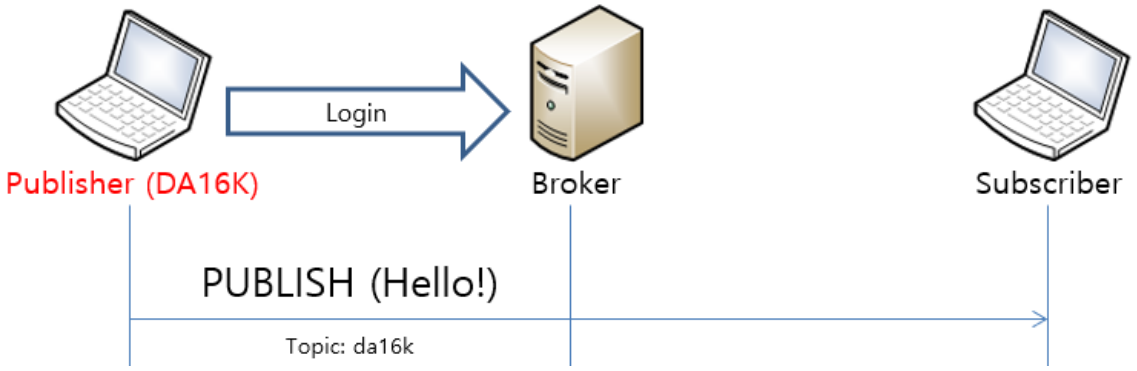

Figure 7: User Login

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 `mosquito_sub` command to log in successfully to the broker.

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

5. On `mqtt_client` (DUT), set the username and password, and start mqtt_client.

```
[/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>
```

NOTE

- The max length of the console command is 158 so to type in a password exceeding the limit of the console, use the command “mqtt_config long_password”
- The max length of the buffer is currently 160 for a password, 64 for a username. If you want to change max length, modify MQTT_USERNAME_MAX_LEN or MQTT_PASSWORD_MAX_LEN, if required

5.4 Subscriber

5.4.1 Setup

1. 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
```

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

```
[/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 (DA16600) 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>
```

1. Run the broker as below.

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

2. Add three 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)
```

3. 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>

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
```

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): 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 the DPM mode enabled. In addition, an MQTT sample implementation (Section 4.3 and Section 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
```

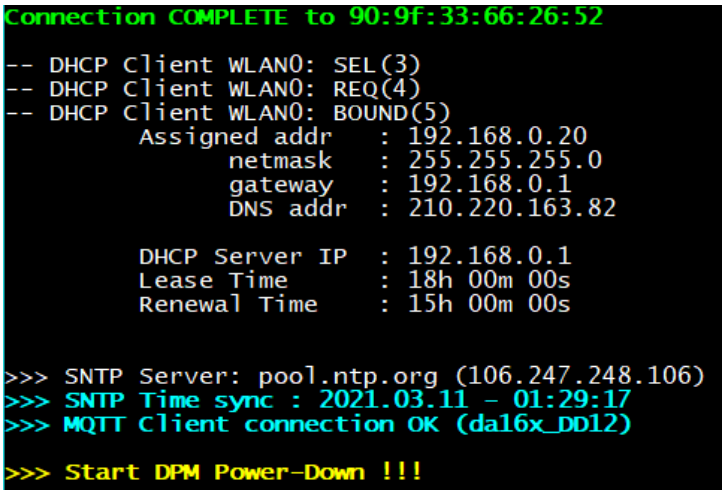

Figure 8: DPM Sleep after MQTT Connection

#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 (DA16600) wakes up from DPM Sleep and prints the message.

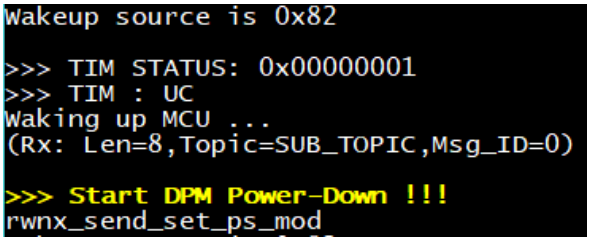

Figure 9: MQTT UC Wakeup

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

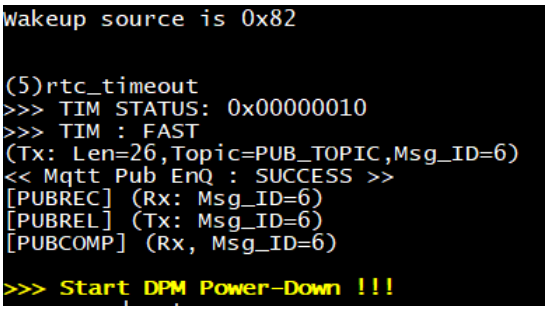

Figure 10: MQTT Wakeup for Sending Message

5.5.1 MQTT Reconnection Scheme

When the broker is disconnected, MQTT Client tries to reconnect to the broker based on the following scheme.

Non-DPM mode

1. MQTT Client tries reconnection six times (MQTT_CONN_MAX_RETRY) and is terminated after the max number of trials is reached.

DPM mode

1. After disconnection from the broker is recognized, the system wakes up from the DPM Sleep, and MQTT Client tries reconnection three times (MQTT_RESTART_MAX_RETRY) and the system enters DPM Sleep when the trials fail.
2. In five seconds, the system wakes up and MQTT Client tries reconnection with the broker. If it fails in connecting to the broker, the system enters the DPM Sleep.
3. “Step 2” is repeated six times (MQTT_CONN_MAX_RETRY) and MQTT Client is terminated after the max number of trials (MQTT_CONN_MAX_RETRY) is reached. The system then enters DPM Sleep.
4. In case other DPM wakeup (User Wakeup, user RTC Wakeup, UC, and so forth) happens after “Step 3”, “Step 2” is repeated six times.

5.5.2 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 the 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 (DA16600) was in the Sleep mode until it woke up to post a periodic status message to the broker. Once DA16200 (DA16600) received the response, it enters and stays in Sleep mode until the next Status Message Tx time (the interval depends on application).

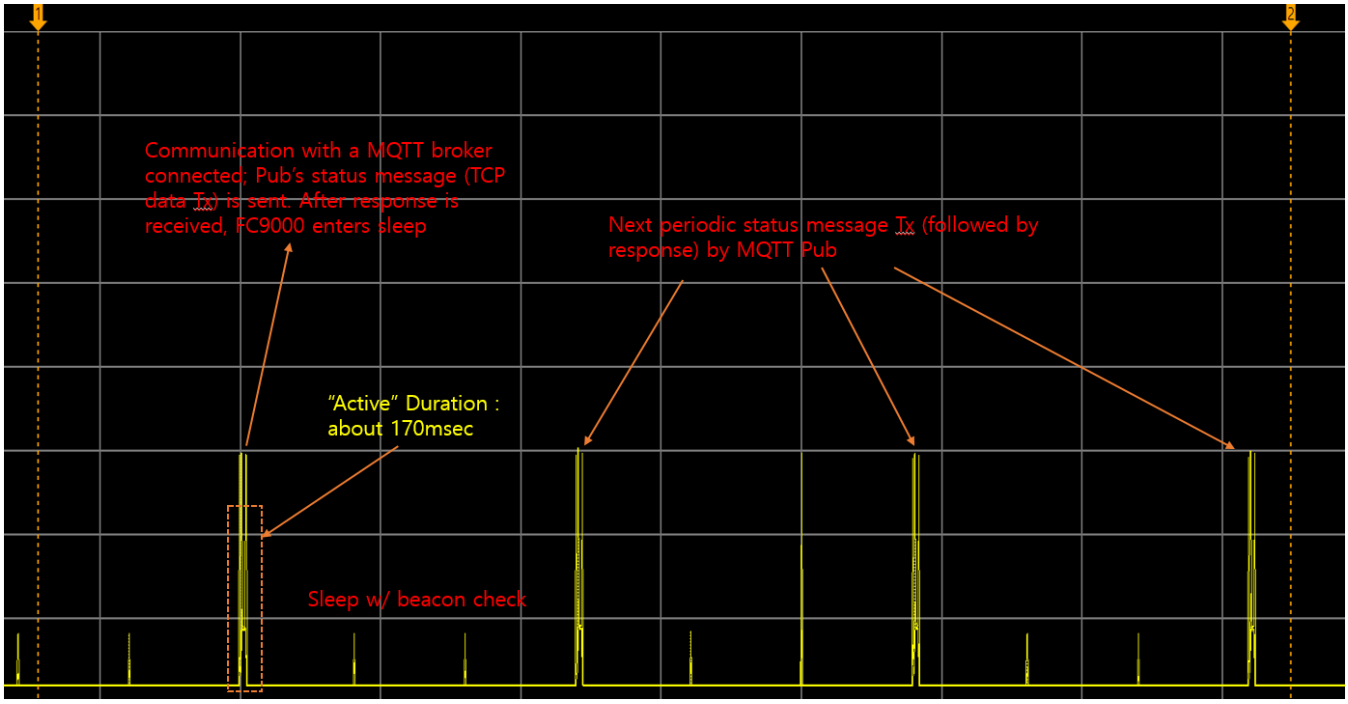

Figure 11: MQTT Communication

5.6 MQTT CleanSession=0 Test Guide

5.6.1 CleanSession=0 Mode

When an MQTT Client (Mqttc onward) establishes a connection with an MQTT Broker (Broker onward), there are two types of session; CleanSession=1 and CleanSession=0.

CleanSession=1: default session type. When the Broker receives a connect request from an Mqttc that tries to connect with an option “CleanSession=1” (which is default config on DA16x), Broker treats the connection as a “new” session. If an existing session associated with the same client_id is found, the Broker clears that previous session and creates a new one with the client_id.

CleanSession=0: When the Broker receives a connect request from an Mqttc that tries to connect with an option “CleanSession=0”, the Broker first tries to find a session (session data) with the same client_id. If it finds one, it keeps using that session for the new Mqttc.

While Mqttc is connected to the Broker, there may be times when the TCP connection becomes unstable and disconnected (e.g. mqtt ping failed). This may cause some messages that had been published to the Broker during that time to not be delivered to a subscriber. If new messages (with QoS > 0) are published to the Broker and for sessions that have been configured in “CleanSession=0”, the Broker retains and re-sends them when the Mqttc is re-connected. Mqttc (if

CleanSession=0 is enabled) also should retain the state of the unfinished / unacked messages until reconnection.

```
1647307743: New connection from 192.168.0.2 on port 8883.
1647307743: New client connected from 192.168.0.2 as da16x_D9CC (c1, k60).
1647307743: Sending CONNACK to da16x_D9CC (0, 0)
1647307743: Received SUBSCRIBE from da16x_D9CC
1647307743:     SUB_TOPIC (QoS 2)
1647307743: da16x_D9CC 2 SUB_TOPIC
1647307743: Sending SUBACK to da16x_D9CC
```

Figure 12: Broker console - CleanSession=1 connection

```
1647307894: Client da16x_D9CC disconnected.
1647307898: New connection from 192.168.0.2 on port 8883.
1647307898: New client connected from 192.168.0.2 as da16x_D9CC (c0, k60).
1647307898: Sending CONNACK to da16x_D9CC (0, 0)
1647307898: Received SUBSCRIBE from da16x_D9CC
1647307898:     SUB_TOPIC (QoS 2)
1647307898: da16x_D9CC 2 SUB_TOPIC
1647307898: Sending SUBACK to da16x_D9CC
```

Figure 13: Broker console - CleanSession=0 connection

Even with CleanSession=0 connection, the Broker does not maintain session data if Mqttc is disconnected in the following cases.

- If a new message is published with QoS 0 after Mqttc is disconnected
- If Mqttc's connection QoS is 0.

DA16x supports CleanSession=0 mode in the following way

- "CleanSession=0 feature" is enabled by default in SDK v3.2.3.0 or higher, disabled by default in SDK v2.4.x.x. For verfiication of this feature in SDK v2.4.x.x, "CleanSession=0 feature" should be enabled in config_generic_sdk.h (__MQTT_CLEAN_SESSION_MODE_SUPPORT__)
- If an application uses **QoS 1 or Qos 2 and CleanSession=0**, the message (payload) size (both Tx and Rx) should be pre-decided (because there's limitation in the dpm user pool size). By default, 100 bytes are defined.

  #define MQTT_MSG_TBL_PRESVD_MAX_PLAYLOAD_LEN 100

  Depending on the application's expected maximum payload size, a different value can be defined.

  The DPM User Pool has a limited size (approximatly 8K in total) in the system.

  First check the available "free" DPM User Pool size using the console command "dpm user_pool", and then calculate the max payload length and message number for the application if needed.

The default configuraiton (payload_len: 100, max_count: 10)
allocates approximatly 1.9KB of dpm user pool (Check mq_msg_tbl_presvd_t for detail).
Search the following compiler options in config_generic_sdk.h.

```
//max payload length of a preserved message
#define MQTT_MSG_TBL_PRESVD_MAX_PLAYLOAD_LEN        100
// max number of preserved messages
#define MQTT_MSG_TBL_PRESVD_MAX_MSG_CNT             10
```

- The following console command is provided to configure CleanSession mode:
  mqtt_client clean_session <1|0>

5.6.1.1 CleanSession and QoS Matrix Table for PUBLISH Rx

Subscriber			Unacked Message Delivery (After MQTT Reconnection)	QoS (Effective Actual)	Publisher Message's QoS
Case	Clean Session	QoS			
1	1	0	X	**0**	0
2	1	1	X	**0**	0
3	1	2	X	**0**	0
4	1	0	X	**0**	1
5	1	1	X	**1**	1
6	1	2	X	**1**	1
7	1	0	X	**0**	2
8	1	1	X	**1**	2
9	1	2	X	**2**	2
10	0	0	X	**0**	0
11	0	1	X	**0**	0
12	0	2	X	**0**	0
13	0	0	X	**0**	1
14	0	1	**O**	**1**	1

Subscriber			Unacked Message Delivery (After MQTT Reconnection)	QoS (Effective Actual)	Publisher Message's QoS
Case	Clean Session	QoS			
15	0	2	O	1	1
16	0	0	X	0	2
17	0	1	O	1	2
18	0	2	O	2	2

Table 5 CleanSession and QoS matrix in message Rx

Basically, with CleanSession=1, no unacked message delivery happens when a mqtt reconnect happens (marked as x).

With CleanSession=0, only case 14, 15, 17, and 18 makes message redelivery happen for messages that had been delivered to the Broker while the Mqttc was offline (marked as O).

5.6.1.2 CleanSession and QoS Matrix Table for PUBLISH Tx

Expectation 1	Application assumes "sending message" will fail, and waits until mqtt gets re-connected before retrying.
Behavior 1	Application does message send retry.
Expectation 2	Application assumes "sending message" will resume when mqtt gets re-connected.
Behavior 2	Application waits until message send retry by mqtt is complete.

Publisher			Expectation if MQTT gets disconnected (while QoS 1/2 message is not fully acked or QoS 0 Send is being sent)	Behaviour expected when MQTT Client re-connected
Case	Clean Session	QoS		
1	1	0	Expectation 1	Behaviour 1
2	1	1	Expectation 1	Behaviour 1
3	1	2	Expectation 1	Behaviour 1
4	0	0	Expectation 1	Behaviour 1
5	0	1	Expectation 2	Behaviour 2
6	0	2	Expectation 2	Behaviour 2

Table 6 CleanSession and QoS matrix in message Tx

When publishing a message from DA16x, applcation's expectation and action/behaviour may be different if CleanSession=0 and QoS 1 or 2 are used in some specific cases.

In normal network condition, there's no difference in message send behaviour between CleanSession=0 and CleanSession=1.

In some abnormal cases where QoS 1/2's ACK message (PUBACK, PUBREC, PUBREL, or PUBCOMP) gets lost due to bad network conditions (which can cause a Mqttc re-connection), CleanSession=0 can recover the previous message state and resume communication with theBroker.

However, if CleanSession=1 is used, when Mqttc is disconnected, it can safely re-transmit the message when Mqttc is re-connected. Depending on the use case, either approach (CleanSession=0 or CleanSession=1) can be utilized.

5.6.2 Test steps

5.6.2.1 How to connect with CleanSession=0

```
[/DA16200/NET] # mqtt_client stop

[/DA16200/NET] # mqtt_config clean_session 0

[/DA16200/NET] # mqtt_config qos 2

[/DA16200/NET] # mqtt_config status

MQTT Client Information:
  - MQTT Status        : Not Running
  - Broker IP          : 192.168.0.230
  - Port               : 8883
  - Pub. Topic         : PUB_TOPIC
  - Sub. Topic         : SUB_TOPIC
  - QoS Level          : 2
  - TLS                : Enable
  - Clean Session      : No
  - TLS ALPN           : (None)
  - TLS SNI            : (None)
  - TLS CIPHER SUIT    : (None)
  - Ping Period        : 60
  - TLS Incoming buf   : 4096(bytes)
  - TLS Outgoing buf   : 4096(bytes)
  - TLS Auth mode      : 1
  - User name          : (None)
  - Password           : (None)
  - Client ID          : (default: da16x_D9CC)
  - MQTT VER           : 3.1
[/DA16200/NET] # mqtt_client start

MQTT CleanSession=0 Support Mode enabled.
[/DA16200/NET] # >>> MQTT Client connection OK (da16x_D9CC)
```


To activate "**CleanSession=0 support mode**" in DA16x, QoS should be 1 or 2 and CleanSession option should be set to 0.

If either option (CleanSession and QoS) is not set as above, CleanSession=0 support mode is disabled.

5.6.2.2 How to restart CleanSession=0 test

If you want to do re-test (fresh new test) with CleanSession=0 mode, depending on the previous session type, you may need the Broker to "clear the previous session".

The reason is that since an Mqttc connects with CleanSession=0, the Broker does not delete the session data until the Mqttc re-connects with CleanSession=1.

Case 1: Previous session is CleanSession=1 and you want to restart a new CleanSession=0 test

```
[/DA16200/NET] # mqtt_client stop

 [mqtt_subscriber_main] mosquitto_loop_forever exited (rc=17, sock=0, errno=0,
runContinueSub=0)
[mqtt_client] terminated
[/DA16200/NET] # mqtt_config clean_session 0

[/DA16200/NET] # mqtt_client start

MQTT CleanSession=0 Support Mode enabled.
[/DA16200/NET] # >>> MQTT Client connection OK (da16x_D9CC)
```

Case 2: Previoius session is CleanSession=0 and you want to do re-test of CleanSession=0.

```
[/DA16200/NET] # mqtt_client stop

 [mqtt_subscriber_main] mosquitto_loop_forever exited (rc=17, sock=0, errno=0,
runContinueSub=0)
[mqtt_client] terminated
[/DA16200/NET] #
[/DA16200/NET] #
[/DA16200/NET] #
[/DA16200/NET] # mqtt_config clean_session 1

[/DA16200/NET] # mqtt_client start

[/DA16200/NET] # >>> MQTT Client connection OK (da16x_D9CC)

[/DA16200/NET] #
[/DA16200/NET] # mqtt_client stop

 [mqtt_subscriber_main] mosquitto_loop_forever exited (rc=17, sock=0, errno=0,
runContinueSub=0)
[mqtt_client] terminated
[/DA16200/NET] #
[/DA16200/NET] #
[/DA16200/NET] # mqtt_config clean_session 0

[/DA16200/NET] # mqtt_client start

MQTT CleanSession=0 Support Mode enabled.
```

```
[/DA16200/NET] # >>> MQTT Client connection OK (da16x_D9CC)
```

5.6.2.3 PUBLISH Rx Test

Test Steps

Test steps are as follows under non-DPM and DPM mode.

In non-DPM mode:

- DA16x: connect to Broker
- Publisher: send one or two messages
- DA16x: check if the messages are received.
- DA16x: disconnect from Broker
- Publisher: send one or two messages (let say msg_A)
- DA16x: reconnect to Broker
- DA16x: check if msg_A (sent while DA16x is offline) is received

DPM mode:

- DA16x: connect to Broker. Enter DPM Sleep
- Publisher: send one or two messages
- DA16x: check if the messages are received.
- DA16x: turn off AP. Wait for the mqtt keep alive period to finish (to make sure Broker recognizes the Mqttc disconnection)
- Publisher: send one or two messages (let say msg_A)
- DA16x: turn on AP. Wait until DA16x is connected to AP
- DA16x: reconnected to AP and check if msg_A (sent while DA16x is offline) is received.

Note:

- mosquitto broker (Broker), mosquitto publisher (Publisher), and DA16x (Subscriber) are used for the test.
- message length from publisher should be less than or equal to 100. If longer messages are sent, they may not be restored properly when mqtt is reconnected.

Test Steps - Example 1 (non-DPM)

Below are the test steps for case 15 (non-DPM mode)

[DA16x] Connect Mqttc with CleanSession=0 and QoS 2

```
[/DA16200/NET] # mqtt_client stop

 [mqtt_subscriber_main] mosquitto_loop_forever exited (rc=17, sock=0, errno=0,
runContinueSub=0)
[mqtt_client] terminated
```



```
[/DA16200/NET] #
[/DA16200/NET] #
[/DA16200/NET] #
[/DA16200/NET] #
[/DA16200/NET] # mqtt_config qos 2

[/DA16200/NET] # mqtt_config clean_session 0

[/DA16200/NET] # mqtt_client start

MQTT CleanSession=0 Support Mode enabled.
[/DA16200/NET] # >>> MQTT Client connection OK (da16x_D9CC)
```

[Other Publisher] Publish messages

```
C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert
wifiuser.pem --key wifiuser.key --tls-version tlsv1 --insecure -q 1 -t SUB_TOPIC -
m "hello_qos_0"

C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert
wifiuser.pem --key wifiuser.key --tls-version tlsv1 --insecure -q 1 -t SUB_TOPIC -
m "hello_qos_1"
```

[DA16x] check the messages are successfully received

```
[/DA16200/NET] #
[/DA16200/NET] # (Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=1)
[PUBACK] (Tx: Msg_ID=1)
(Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=2)
[PUBACK] (Tx: Msg_ID=2)
```

[DA16x] disconnect from Broker

```
[/DA16200/NET] #
[/DA16200/NET] # mqtt_client stop

 [mqtt_subscriber_main] mosquitto_loop_forever exited (rc=17, sock=0, errno=0,
runContinueSub=0)
[mqtt_client] terminated
```

[Other Publisher] publish two messages (while DA16x is in disconnected state)

```
C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert
wifiuser.pem --key wifiuser.key --tls-version tlsv1 --insecure -q 1 -t SUB_TOPIC -
m "hello_qos_2"

C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert
wifiuser.pem --key wifiuser.key --tls-version tlsv1 --insecure -q 1 -t SUB_TOPIC -
m "hello_qos_3"
```

[DA16x] re-connects to the Broker and checks if the two messages that had been published while DA16x was in a disconnected state are received successfully.

```
[/DA16200/NET] #
[/DA16200/NET] # mqtt_client start
```

```
MQTT CleanSession=0 Support Mode enabled.
[/DA16200/NET] # (Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=3)
[PUBACK] (Tx: Msg_ID=3)
(Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=4)
[PUBACK] (Tx: Msg_ID=4)
>>> MQTT Client connection OK (da16x_D9CC)
```

Test Steps - Example 2 (DPM)

Below are the test steps for case 18 (DPM mode)

(mosquitto broker and mosquitto publisher are used for the test)

[DA16x] Connect with CleanSession=0 and QoS 2

```
[/DA16200/NET] # mqtt_config qos 2

[/DA16200/NET] # mqtt_config clean_session 0

[/DA16200/NET] # mqtt_config status

MQTT Client Information:
  - MQTT Status  : Not Running
  - Broker IP          : 192.168.0.230
  - Port               : 8883
  - Pub. Topic         : PUB_TOPIC
  - Sub. Topic         : SUB_TOPIC
  - QoS Level          : 2
  - TLS                : Enable
  - Clean Session      : No
  - TLS ALPN           : (None)
  - TLS SNI            : (None)
  - TLS CIPHER SUIT    : (None)
  - Ping Period        : 60
  - TLS Incoming buf   : 4096(bytes)
  - TLS Outgoing buf   : 4096(bytes)
  - TLS Auth mode      : 1
  - User name          : (None)
  - Password           : (None)
  - Client ID          : (default: da16x_D9CC)
  - MQTT VER           : 3.1
[/DA16200/NET] #
[/DA16200/NET] #
[/DA16200/NET] # dpm on
[DP
Wakeup source is 0x0
[dpm_init_retmemory] DPM INIT CONFIGURATION(1)

...

System Mode : Station Only (0)
>>> Start DA16X Supplicant ...
>>> DA16x Supp Ver2.7 - 2020_07
>>> Wi-Fi mode : b/g/n -> b/g (for DPM)
>>> MAC address (sta0) : d4:3d:39:10:d9:cc
```

```
...
Connection COMPLETE to 00:11:32:ce:8e:6f

-- DHCP Client WLAN0: SEL(6)
-- DHCP Client WLAN0: REQ(1)
-- DHCP Client WLAN0: CHK(8)
-- DHCP Client WLAN0: BOUND(10)
         Assigned addr   : 192.168.1.195
               netmask   : 255.255.255.0
               gateway   : 192.168.1.1
               DNS addr  : 192.168.1.1

         DHCP Server IP  : 192.168.1.1
         Lease Time      : 24h 00m 00s
         Renewal Time    : 20h 00m 00s

MQTT CleanSession=0 Support Mode enabled.
>>> Hello World #2 ( network dependent application ) !!!
>>> MQTT Client connection OK (da16x_D9CC)
>>> Start DPM Power-Down !!!
```

[Other Publisher] Publish messages

```
C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert
wifiuser.pem --key wifiuser.key --tls-version tlsv1 --insecure -q 2 -t SUB_TOPIC -
m "hello_qos_1"

C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert
wifiuser.pem --key wifiuser.key --tls-version tlsv1 --insecure -q 2 -t SUB_TOPIC -
m "hello_qos_2"
```

[DA16x] check the messages are successfully received

```
Wakeup source is 0x82

>>> Start DA16X Supplicant ...
>>> TIM STATUS: 0x00000001
>>> TIM : UC
>>> Hello World #1 ( Non network dependent application ) !!!
  MQTT CleanSession=0 Support Mode enabled.
>>> Hello World #2 ( network dependent application ) !!!
(Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=1)
[PUBREC] (Tx: Msg_ID=1)
[PUBREL] (Rx: Msg_ID=1)
[PUBCOMP] (Tx: Msg_ID=1)
>>> Start DPM Power-Down !!!
[i3ed11_dpm_tcp_ack_proc] TCP Update SEQ Ný
Wakeup source is 0x82

>>> Start DA16X Supplicant ...
>>> TIM STATUS: 0x00000001
>>> TIM : UC
>>> Hello World #1 ( Non network dependent application ) !!!
  MQTT CleanSession=0 Support Mode enabled.
>>> Hello World #2 ( network dependent application ) !!!
(Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=2)
```

```
[PUBREC] (Tx: Msg_ID=2)
[PUBREL] (Rx: Msg_ID=2)
[PUBCOMP] (Tx: Msg_ID=2)
>>> Start DPM Power-Down !!!
[i3ed11_dpm_tcp_ack_proc] TCP Update SEQ Num(20d7)
PS TIME 130369 us
```

[DA16x] Turn off AP

```
Wakeup source is 0x82

>>> Start DA16X Supplicant ...
>>> TIM STATUS: 0x00000008
>>> TIM : No BCN

>>> Network Interface (wlan0) : DOWN
[wpa_supplicant_event_disassoc] CTRL-EVENT-DISCONNECTED bssid=00:11:32:ce:8e:6f
reason=4 locally_generated=1
Fast scan, freq=2432, num_ssids=1
!!! No selected network !!!
Fast scan, freq=2432, num_ssids=1
>>> Hello World #1 ( Non network dependent application ) !!!
!!! No selected network !!!

### User Call-back : Wi-Fi disconnected ( reason_code = 4 ) ...
Fast scan, freq=2432, num_ssids=1
!!! No selected network !!!
!!! No selected network !!!
!!! No selected network !!!
!!! No selected network !!!
!!! No selected network !!!

rtc_timeout (tid:14)
!!! No selected network !!!
[dpm_timer_process] 'mqtt_sub' is not ready. Callback can't be called. (/14)
>> Abnormal DPM(1) operation after 1 second
...
```

[Broker] make sure Mqttc is disconnected

```
...
1647318510: Socket error on client da16x_D9CC, disconnecting.
...
```

[Other Publisher] publish two messages (while DA16x is in a disconnected state)

```
C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert
wifiuser.pem --key wifiuser.key --tls-version tlsv1 --insecure -q 2 -t SUB_TOPIC -
m "hello_qos_3"

C:\mosquitto>mosquitto_pub -h 192.168.0.230 -p 8883 --cafile cas.pem --cert
wifiuser.pem --key wifiuser.key --tls-version tlsv1 --insecure -q 2 -t SUB_TOPIC -
m "hello_qos_4"
```

[DA16x] Turn ON AP

[DA16x] Wait until AP is connected and see whether “hello_qos_3” and “hello_qos_4” are received

```
...
Wakeup source is 0x82


System Mode : Station Only (0)
>>> Start DA16X Supplicant ...
>>> DA16x Supp Ver2.7 - 2020_07
>>> Wi-Fi mode : b/g/n -> b/g (for DPM)
>>> MAC address (sta0) : d4:3d:39:10:d9:cc
>>> sta0 interface add OK
>>> Start STA mode...
>>> Hello World #1 ( Non network dependent application ) !!!

>>> Network Interface (wlan0) : UP
>>> Associated with 00:11:32:ce:8e:6f

Connection COMPLETE to 00:11:32:ce:8e:6f

-- DHCP Client WLAN0: SEL(6)
-- DHCP Client WLAN0: REQ(1)
-- DHCP Client WLAN0: CHK(8)
-- DHCP Client WLAN0: BOUND(10)
         Assigned addr   : 192.168.1.195
               netmask   : 255.255.255.0
               gateway   : 192.168.1.1
               DNS addr  : 192.168.1.1

         DHCP Server IP  : 192.168.1.1
         Lease Time      : 24h 00m 00s
         Renewal Time    : 20h 00m 00s

  MQTT CleanSession=0 Support Mode enabled.
>>> Hello World #2 ( network dependent application ) !!!
(Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=3)
[PUBREC] (Tx: Msg_ID=3)
(Rx: Len=11,Topic=SUB_TOPIC,Msg_ID=4)
[PUBREC] (Tx: Msg_ID=4)
>>> MQTT Client connection OK (da16x_D9CC)
[PUBREL] (Rx: Msg_ID=3)
[PUBCOMP] (Tx: Msg_ID=3)
[PUBREL] (Rx: Msg_ID=4)
[PUBCOMP] (Tx: Msg_ID=4)
>>> Start DPM Power-Down !!!
```

5.6.2.4 PUBLISH Tx Test

Test Steps

Test steps are as follows.

- DA16x: connect to Broker
- DA16x: send a messages
- DA16x: check if the message send is successful

Note:

- Message length from DA16x should be less than or equal to 100 for case 5 and 6 configuration. Sending longer messages returns failure. For cases other than case 5 or 6, message length limit is 3K.

Test Steps – Example

Below is the test steps for case 5 (non-DPM mode)

```
[/DA16200/NET] # mqtt_config qos 1

[/DA16200/NET] # mqtt_config clean_session 0

[/DA16200/NET] # mqtt_client start

  MQTT CleanSession=0 Support Mode enabled.
[/DA16200/NET] # user cb: on_connect
user cb: on_subscribe
>>> MQTT Client connection OK (da16x_D9CC)

[/DA16200/NET] #
[/DA16200/NET] # mqtt_client -m hello_q1

[/DA16200/NET] # (Tx: Len=8,Topic=PUB_TOPIC,Msg_ID=2)
<< Mqtt Pub EnQ : SUCCESS >>
[PUBACK] (Rx, Msg_ID=2)
user cb: on_publish(mid=2)
```

5.7 Reset

The following command clears all MQTT configurations:

```
[/DA16200/NET]# mqtt_config reset
```

6 Certificate

DA16200 (DA16600) 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 // for SDK v3.2.3.0 or higher, use "cert write ca1"
```

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 // for SDK v3.2.3.0 or higher, use "cert write cert1"
```

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 // for SDK v3.2.3.0 or higher, use "cert write key1"
```

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 // for SDK v3.2.3.0 or higher, use "cert status"#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 SDK 3.x, cert status is shown as below
[/DA16200/NET] # cert status

#1:
  For MQTT, CoAPs Client
  - Root CA     : Found
  - Certificate : Found
  - Private Key : Found
  - DH Parameter: Empty

#2:
  For HTTPs, OTA
  - Root CA     : Empty
  - Certificate : Empty
```

```
  - Private Key : Empty
  - DH Parameter: Empty
```

In case you want to remove all the credentials stored:

```
[/DA16200/NET]# cert 3 // in SDK v3.x, use "cert del all"
```

6.2 CA, Client Cert, and Client Key

- Cert 0: CA

```
-----BEGIN CERTIFICATE-----
MIID+TCCAuGgAwIBAgIJANqqHCazDkkOMA0GCSqGSIb3DQEBCwUAMIGSMQswCQYD
VQQGEwJVUzETMBEGA1UECAwKQ2FsaWZvcm5pYTEUMBIGA1UEBwwLU2FudGEgQ2xh
cmExFzAVBgNVBAoMDldpLUZpIEFsbGlhbmNlMR0wGwYDVQQDDBRXRkEgUm9vdCBD
ZXJ0aWZpY2F0ZTEgMB4GCSqGSIb3DQEJARYRc3VwcG9ydEB3aS1maS5vcmcwHhcN
MTMwMzExMTkwMjI2WhcNMjMwMzA5MTkwMjI2WjCBkjELMAkGA1UEBhMCVVMxEzAR
BgNVBAgMCkNhbGlmb3JuaWExFDASBgNVBAcMC1NhbnRhIENsYXJhMRcwFQYDVQQK
DA5XaS1GaSBBbGxpYW5jZTEdMBsGA1UEAwwUV0ZBIFJvb3QgQ2VydGlmaWNhdGUx
IDAeBgkqhkiG9w0BCQEWEXN1cHBvcnRAd2ktZmkub3JnMIIBIjANBgkqhkiG9w0B
AQEFAAOCAQ8AMIIBCgKCAQEA6TOCu20m+9zLZITYAhGmtxwyJQ/1xytXSQJYX8LN
YUS/N3HG2QAQ4GKDh7DPDI13zhdc0yOUE1CIOXa1ETKbHIU9xABrL7KfX2HCQ1nC
PqRPiW9/wgQch8Aw7g/0rXmg1zewPJ36zKnq5/5Q1uyd8YfaXBzhxm1IYlwTKMlC
ixDFcAeVqHb74mAcdel1lxdagHvaL56fpUExm7GyMGXYd+Q2vYa/o1UwCMGfMOj6
FLHwKpy62KCoK3016HlWUlbpg8YGpLDt2BB4LzxmPfyH2x+Xj75mAcllOxx7GK0r
cGPpINRsr4vgoltm4Bh1eIW57h+gXoFfHCJLMG66uhU/2QIDAQABo1AwTjAdBgNV
HQ4EFgQUCwPCPlSiKL0+Sd5y8V+Oqw6XZ4IwHwYDVR0jBBgwFoAUCwPCPlSiKL0+
Sd5y8V+Oqw6XZ4IwDAYDVR0TBAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAQEAsNxO
z9DXb7TkNFKtPOY/7lZig4Ztdu6Lgf6qEUOvJGW/Bw1WxlPMjpPk9oI+JdR8ZZ4B
9QhE+LZhg6SJbjK+VJqUcTvnXWdg0e8CgeUw718GNZithIElWYK3Kh1cSo3sJt0P
z9CiJfjwtBDwsdAqC9zV9tgp09QkEkav84X20VxaITa3H1QuK/LWSn/ORrzcX0Il
10YoF6 Hz3ZWa65mUoMzd8DYtCyGtcbYrSt+NMCqRB186PDQn5XBCytgF8VuiCyyk
Z04hqHLzAFc21P9yhwKGi3BHD/Sep8fvr9y4VpMIqHQm2jaFPxY1VxhPSV+UHoE1
fCPitIJTp/iXi7uXTQ==
-----END CERTIFICATE-----
```

- Cert 1: Client Cert

```
-----BEGIN CERTIFICATE-----
MIIEBTCCAu2gAwIBAgICEEYwDQYJKoZIhvcNAQELBQAwgZIxCzAJBgNVBAYTAlVT
MRMwEQYDVQQIDApDYWxpZm9ybmlhMRQwEgYDVQQHDAtTYW50YSBDbGFyYTEXMBUG
A1UECgwOV2ktRmkgQWxsaWFuY2UxHTAbBgNVBAMMFFdGQSBSb290IENlcnRpZmlj
YXRlMSAwHgYJKoZIhvcNAQkBFhFzdXBwb3J0QHdpLWZpLm9yZzAeFw0xMzA1MTAy
MzQ0NTFaFw0yMzA1MDgyMzQ0NTFaMH4xCzAJBgNVBAYTAlVTMRMwEQYDVQQIDApD
YWxpZm9ybmlhMRcwFQYDVQQKDA5XaS1GaSBBbGxpYW5jZTEfMB0GA1UEAwwWQ2xp
ZW50IENlcnRpZmljYXRlIElETDEgMB4GCSqGSIb3DQEJARYRc3VwcG9ydEB3aS1m
aS5vcmcwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDco7kQ4W2b/fBw
2ZgSAXVWBCmJW0yax8K682kRiHlB1aJ5Im8rTEktZlPDQVhoO3Ur+Ij9Y8ukD3Hq
CMa95T1a3Ly9lhDIME/VRqRgZRGa/FC/jkiK9u9SgIXPZqJk1s/JVG7a7deC8BvK
iqIFhXoHl0N4nJxwVA8kag48dXbrTxrOH26C9qwU85iS/c1ozHJMmq052WPSQZII
Sq8+VmxlCbbXxQ7kU2oZkxDqW3hMI3OPS80s8q4tMzuitvzFO0JvAHgh4IFyE7yg
nIE7+lM9f3EwzECw9nEBdL7AvfnYLhlIEI8S8wZTUpnd8XA5Qs7Qa4rLNuqI273Z
IWJLh9v/AgMBAAGjeDB2MA8GA1UdEwEB/wQFMAMCAQAwCwYDVR0PBAQDAgXgMBYG
A1UdJQEB/wQMMAoGCCsGAQUFBwMCMB0GA1UdDgQWBBQtC2mx8POhZRfB+sj4wX1Z
OzdrCzAfBgNVHSMEGDAWgBQLA8I+VKIovT5J3nLxX46rDpdngjANBgkqhkiG9w0B
AQsFAAOCAQEAsvHJ+J2YTCsEA69vjSmsGoJ2iEXDuHI+57jIo+8qRVK/m1is1eiU
AefExDtxxKTEPPtourlYO8A4i7oep9T43Be8nwVZdmxzfu14cdLKQrE+viCuHQTc
BLSoAv6/SZa3MeIRkkDSPtCPTJ/Pp+VYPK8gPlc9pwEs/KLgFxK/Sq6RDNjTPs6J
MChxi1iSdUES8mz3JDhQ2RQWVuibPorhgVqNTyXBjFUbYxTVl3hBCtK/Bd4IyFTX
Li90XXHseNbj2sHu3PFBU7PG5mhKQMUOYqvQzEDIXT6SDA+PrepqrwXn/BL861K6
GV4LcfKBg0HHdW9gJByZCN02igFTw56Kzg==
-----END CERTIFICATE-----
```


- Cert 2: Client Key

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEA3KO5EOFtm/3wcNmYEgF1VgQpiVtMmsfCuvNpEYh5QdWieSJv
K0xJLWZTw0FYaDt1K/iI/WPLpA9x6gjGveU9Wty8vZYQyDBP1UakYGURmvxQv45I
ivbvUoCFz2aiZNbPyVRu2u3XgvAbyoqiBYV6B5dDeJyccFQPJGoOPHV2608azh9u
gvasFPOYkv3NaMxyTJqtOdlj0kGSCEqvPlZsZQm218UO5FNqGZMQ6lt4TCNzj0vN
LPKuLTM7orb8xTtCbwB4IeCBchO8oJyBO/pTPX9xMMxAsPZxAXS+wL352C4ZSBCP
EvMGU1KZ3fFwOULO0GuKyzbqiNu92SFiS4fb/wIDAQABAoIBAQDcnbCc2mt5AM98
Z3aQ+nhSy9Kkj2/njDqAKIc0ituEIpNUwEOcbaj2Bk1W/W3iuyEMGHURuMmUgAUN
WD0w/5j705+9ieG56eTJgts1r5mM+SHch+6tVQAz5GLn4N4cKlaWHyDBM/S77k47
lacwEijUkkFaxm3+O27woEMf3OxNl24KmRenMYBhqcsoT4BYBw3Bh8xe+XN95rXj
2BdIbr5+RWGc9Zsz4o5Wmd4mL/JvbKeohrsecien4TZRzWFku93XV5kie1c1aJy1
nJ85bGJk4focmP/2ToxQysTbPYCxHVTIHuADK/qf9SGHJ9F7EBHE7+0isuwBbqOD
OzS8rHdRAoGBAPCXlaHumEkLIRv3enhpHPBYxnDndNCtT1T6+Cuit/vfo6K6oA7p
iUaej/GPZsDKXhayeTiEaq7QMinUtGkiCgGlVtXghXuCZz6KrH19W6wzC6Pbokmq
BZak4LQcvGavt3VzjliAKLcdn6nQt/+bp/jKDJOKVbvb30sjS035Ah4zAoGBAOrF
BgE9UTEnfQHIh7pyiM1DAomBbdrlRos8maQl26cHqUHN3+wy1bGHLzOjYFFoAasx
eizw7Gudgbae28WIP1yLGrpt15cqVAvlCYmBtZ3C98FuT3FYgEEZpWNmE8Om+5UM
td+mtMjonWAPkCYC+alqUZzeIs+CZs5CHKYCDqcFAoGBAOfkQv38GV2102jARJPQ
RGtINaRXApmrod43s4Fjac/kAzVyiZk18PFXHUhnvlMt+jgIN5yIzMbHtsHo2SbH
/zsM4MBuklm0G80FHjIp5HT6EksSA77amF5VdptDYzfaP4p+IYIdrKCqddzYZrCA
mArMvAhs+iuCRhuG3is+SZNPAoGAHs6r8w2w0dp0tP8zkGvnN8hLVO//EnJzx2G0
Z63wHQMMWu5BLCWflSRANW6C/SvAzE450hvralPI6cX+4PT4G5TFdSFk4RlU3hq4
Has/wewLxv5Kvnz2l5Rd96U1gr8u1GhOlYKyxop/3FMuf050pJ6nBwa/WquqAfb6
+23ZrmECgYEA6l0GFHwMFBNnpPuxHgYgS5+4g3+8DhZZIDc7IflBCBWF/ZwbM+nH
+JSxiYYjvD7zIBhndqERcZ+fvbZTQ8oymr3j5AESM0ZfAHbft6IFQWjDUC3IDUF/
4F0cUidFC8smu6Wa2tjvSIz7DfvmDsn1l+7s9qQvDxdyPas0IkL/v8w=
-----END RSA PRIVATE KEY-----
```

// Mosquitto 1.4.14 License

Eclipse Distribution License 1.0

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

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
1.11	14-Jun-2022	Added functions: mqtt_client_unsub_topic(), mqtt_sub_callback_set(), mqtt_pub_callback_set(), mqtt_msg_callback_set(), mqtt_sub_disconn_cb_set(), mqtt_subscribe_callback_set(), mqtt_unsubscribe_callback_set() mqtt_sub_disconn2_cb_set()
1.10	11-Apr-2022	Updated mqtt_client_send_message() : Return value diversified Updated Configuration parameters CleanSession=0 mode test guide added New Certificate commands for SDK v3.2.3.0
1.9	08-Apr-2022	Update logo, disclaimer, copyright.
1.8	29-Nov-2021	Title was changed.
1.7	08-Aug-2021	Section 4.6 : path to sample file updated
1.6	01-Apr-2021	SDK V3.x.x.x support added.
1.5	11-Mar-2021	Section 4.6 and Section 5.5.1 added. Section 4.5 cmd_mqtt_sample( ) updated. Symbol name change: from FC9K/fc9k to DA16X/da16x. mqtt_config long_password added. Terms updated; typo fixed. Figure labels added.
1.4	27-Nov-2020	Command "mqtt_config client_id" 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.

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.

RoHS Compliance

Dialog Semiconductor's suppliers certify that its products are in compliance with the requirements of Directive 2011/65/EU of the European Parliament on the restriction of the use of certain hazardous substances in electrical and electronic equipment. RoHS certificates from our suppliers are available on request.

Important Notice and Disclaimer

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu

Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:

https://www.renesas.com/contact/