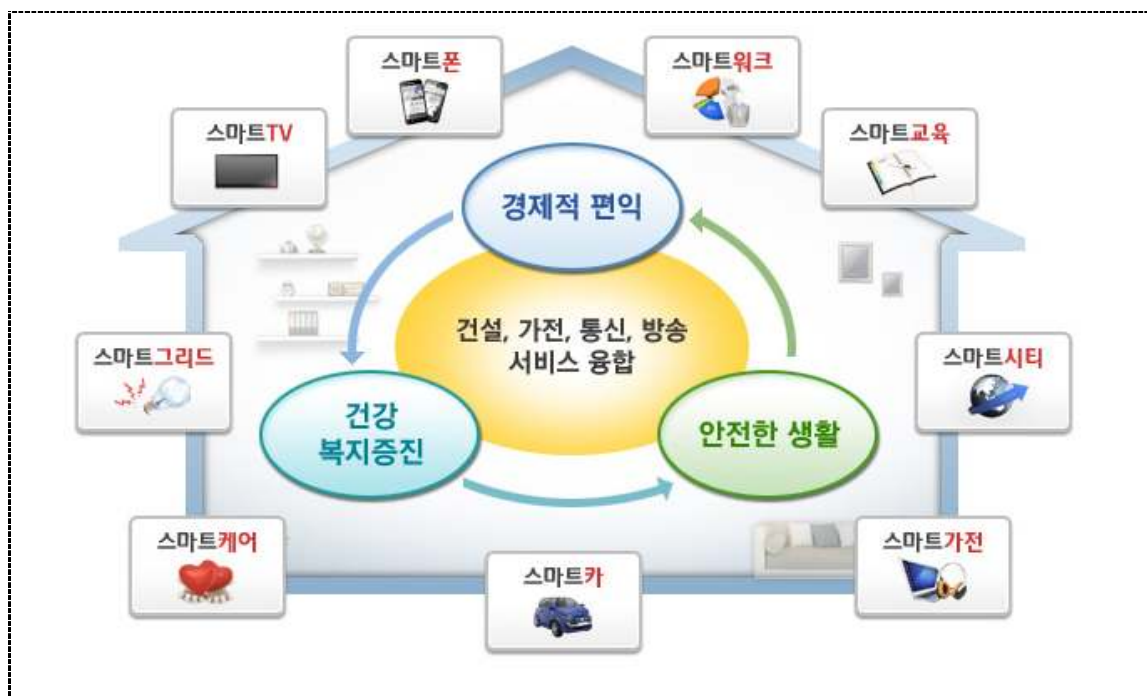


MQTT 프로젝트 응용편

1. 스마트 홈 (Smart Home) 제어 시스템

[1] 스마트홈의 정의



스마트 홈은 크게 홈오토메이션과 홈시큐리티 분야로 나눌 수 있다

홈오토메이션은 가정 내 정보 가전기기를 네트워크로 연결하여 시간, 장소에 구애 받지 않고 자유롭게 제어 관리 할 수 있는 서비스를 제공하고 홈 시큐리티는 주거 공간 내의 중요한 재산이나 시설을 외부 침입으로 보호 하고, 침입 감시, 출입 통제, CCTV등의 방법체계를 구성하여 불의의 상황에 신속하게 대처하여 범죄로부터의 손실을 사전에 예방하여 안전한 삶을 영위할 수 있도록 하는 것이다

그밖에도 스마트 홈 헬스케어와 에너지 관리를 위한 스마트 그린홈, 스마트 TV & 엔터테인먼트, 스마트 융합 가전 분야에 대한 관심도 증가하고 있다

[2] 스마트홈의 제어 시스템의 구현 기능

스마트홈의 다양한 기능 중에서 아래와 같이 홈 오토메이션과 홈 시큐리티 핵심 두 분야의 일부 기능을 프로젝트로 구현해보자

<1> 홈 오토메이션 기능

온도 습도 측정 센서를 사용하여 집안 온도 및 습도 조절한다

조도 측정 센서를 사용하여 집안 조명 조절한다

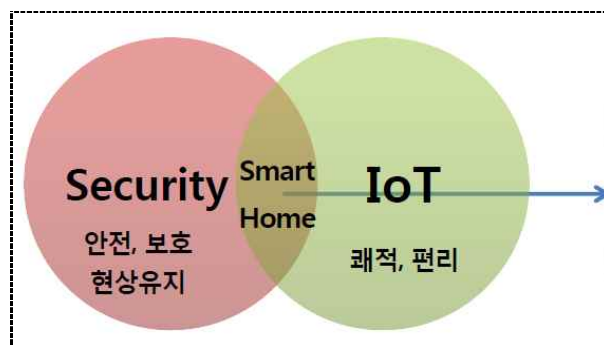
네트워크 접속을 이용 스마트폰으로 원격 모니터링 및 원격 제어를 할수 있다

<2> 홈 시큐리티 기능

인체감지 센서를 사용하여 외부 침입자 감지 및 사이렌 경보를 울린다

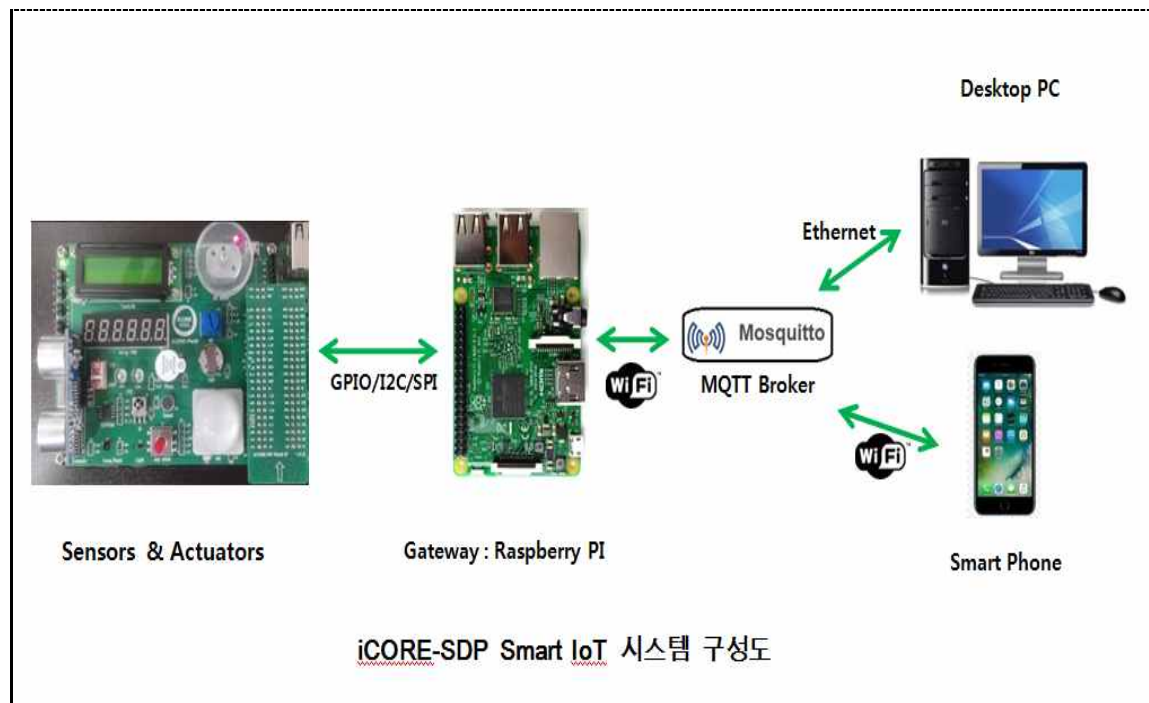
초음파 센서를 사용한 접근 물체 거리를 측정한다

원격으로 방문자를 모니터링 한다



[3] 스마트홈 제어 시스템 구성

Gateway 역할을 하는 라즈베리파이에 GPIO와 I2C,SPI로 연결되어 있는 각 센서에서 측정된 데이터들은 라즈베리파이 SD카드의 데이터베이스에 저장되고 이더넷이나 WiFi를 연결되어 있는 서버 PC와 안드로이드 스마트폰으로 전송된다



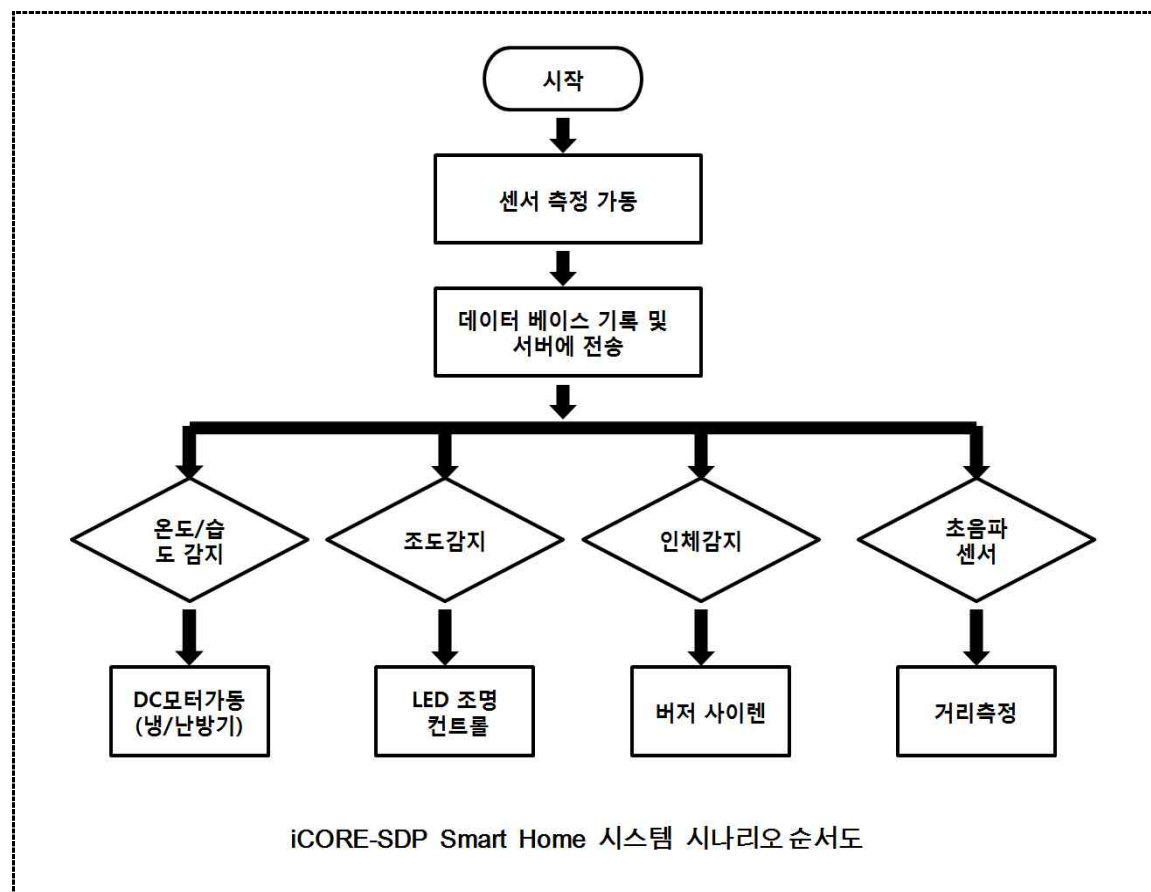
[4] 스마트홈 제어 시나리오

전체 시나리오를 순서도로 그리면 아래와 같다. 라즈베리파이를 부팅하여 연결되어 있는 각종 센서들을 측정을 가동시켜 데이터를 수집하고 데이터 베이스에 기록한다음 서버에 네트워크로 전송한다

온도 및 습도값에 따라 냉 난방기의 모터를 가동하여 실내 온도를 적절히 유지시킨다
조도 측정 값에 따라서 LED 조명(거실등, 안방등)의 밝기를 조절한다

홈 시큐리티 기능으로 인체 감지 센서에 의해 외부 침입이 감지되면 버저의 사이렌을 울려
알려준다 초음파 센서로 물체가 접근하는 거리를 측정하여 보고한다

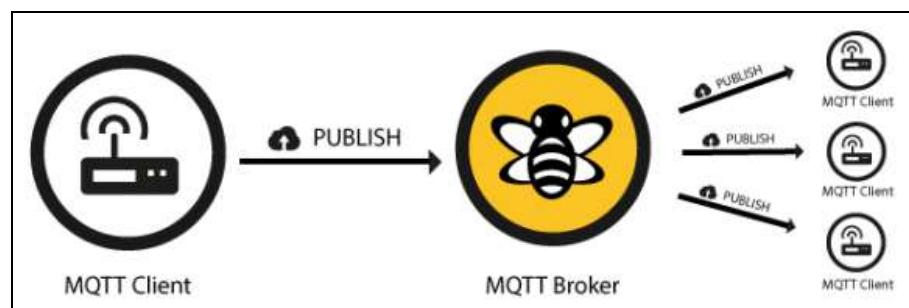
PC와 스마트폰을 사용하여 수집된 데이터를 분석하여 액추에이터들을 직접 제어 할 수
있도록 구현한다



[5] 스마트홈 제어를 위한 하드웨어

당사 iCORE-SDP 장비에는 게이트웨이 기능을 하는 라즈베리파이보드와 각종 센서 (초음파 센서, 인체 감지 센서, 사운드 센서, 온/습도 센서, Light 센서,)와 각종 액츄에이터(DC 모터, LED, 부저, Text LCD, FND) 가 탑재되어 있다

- 스마트홈 제어용 센서와 액츄에이터



[6] 스마트홈 제어 소프트웨어 구성

[1] MQTT 프로토콜은 무엇인가?

시중에 출시되어 있는 다수의 IoT 교육 장비들은 IoT 데이터를 수집하는 디바이스측에 소켓 서버나 웹서버를 설치하고 클라이언트 역할을 하는 스마트폰이나 PC에서 서버의 IP주소를 입력하여 서버에 접속하여 데이터를 수신해오는 방식을 사용하고 있다. 이러한 방식은 전통적인 네트워크 프로그래밍 기법을 사용하는 구형 방식으로 최근에 사용되는 IoT 시스템에서 자주 사용되는 방식과 다소 차이가 있다.

IoT 개념은 디바이스 쪽이 서버로 구현되기 보다는 다양한 노드의 장치에서 방대한 데이터를 수집하여 데이터를 가공 처리할 수 있는 서버로 전송하여 서버측이 데이터를 관리하고 처리하는 형태를 의미한다. 이를 위해서는 안정적이고 효율적인 네트워크 프로토콜이 필요로 하는데 그 중의 많이 사용되며 표준으로 자리를 잡아가려는 것 중 하나가 바로 MQTT 프로토콜이다.

OASIS(Organization for the Advancement of Structured Information Standards)가 사물 인터넷(Internet of Thing)을 위한 메시징 프로토콜로 MQTT(Message Queuing Telemetry Transport)를 선정했다고 발표했다.

HTTP가 사람들이 웹을 통해 정보를 공유할 수 있는 기반을 만들었던 것과 마찬가지로 MQTT 역시 수십억의 저렴한 내장형 데이터 수집 측정 디바이스를 온라인화할 수 있는 기반이 될 수 있다고 평가다.

MQTT는 제한된 컴퓨팅 성능과 빈약한 네트워크 연결 환경에 잘 맞는 메시징 프로토콜로, IBM과 시스템 공급업체인 유로테크(Eurotech)가 개발해 OASIS에 기증한 것이다.

MQTT는 이미 다양한 임베디드 시스템에서 폭넓게 사용되고 있다. 대표적인 사례가 병원으로, MQTT를 맥박조정기와 다른 의료 기구의 커뮤니케이션용 프로토콜로 사용하고 있다. 정유가스업계도 수천 마일에 이르는 송유관을 모니터링하는 데 MQTT를 사용하고 있다.

OASIS는 MQTT가 사물 인터넷의 표준 프로토콜로서 충분한 역할을 수행할 수 있도록 강화하기 위해 새로운 기술 위원회를 구성했다. 이 위원회는 MQTT가 더 다양한 네트워크와 디바이스, 소프트웨어 애플리케이션에서 이용할 수 있도록 할 계획이다..

[2] MQTT 패키지 설치 : 라즈베리파이 Gateway용 paho 설치 및 테스트

라즈베리파이에 paho-mqtt 패키지를 설치해야 한다

아래와 같이 라즈베리파이에 MQTT broker Mosquitto (Paho로 이름이 바뀌었음) 패키지를 설치한다 패키지 이름이 paho-mqtt 이며 Python3 용 설치를 위해 pip3 를 사용하여 설치하도록한다 (당사 iCORE-SDP 보드용 이미지에는 이미 설치되어 있다)

```
$ sudo pip3 install paho-mqtt
```

아래는 로컬 머신에서 실행되고 있는 브로커에 토픽을 구독(subscribe)하기 위한 파이썬 코드이다.

[sub.py] : Subscriber 구현 파이썬 코드

```
import paho.mqtt.client as mqtt

# 클라이언트가 서버에게서 CONNACK 응답을 받을 때 호출되는 콜백 함수
def on_connect(client, userdata, flags,rc):
    print ("Connected with result code " + str(rc))
    client.subscribe("hello/world")

# 서버에게서 PUBLISH 메시지를 받을 때 호출되는 콜백 함수
def on_message(client, userdata, msg):
    print ("Topic: ", msg.topic + '\nMessage: ' + str(msg.payload))

client = mqtt.Client()                # MQTT Client 오브젝트 생성
client.on_connect = on_connect        # on_connect callback 설정
client.on_message = on_message        # on_message callback 설정

client.connect("test.mosquitto.org", 1883, 60) # MQTT 서버에 연결

# 네트워크 트래픽을 처리, 콜백 디스패치, 재접속 등을 수행하는 블로킹 함수
client.loop_forever()
```

[pub.py] : Publisher 구현 파이썬 코드

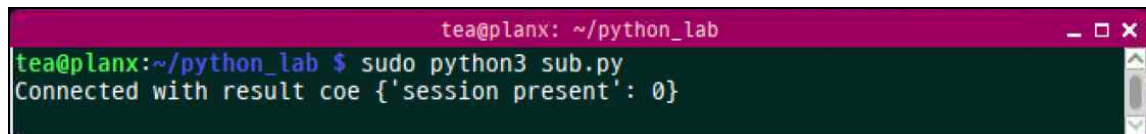
```
import paho.mqtt.client as mqtt

mqttc = mqtt.Client("python_pub")          # MQTT Client 오브젝트 생성
mqttc.connect("test.mosquitto.org", 1883)  # MQTT 서버에 연결

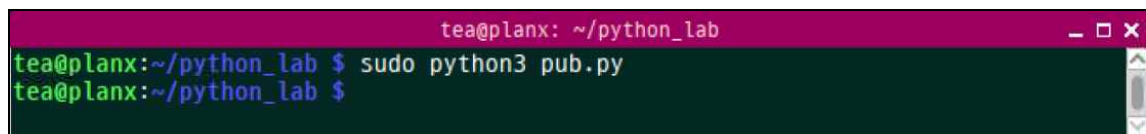
# 'hello/world' 토픽에 "Hello MQTT World!"라는 메시지를 발행한다
mqttc.publish("hello/world", "Hello MQTT World!")
mqttc.loop(2)                             # timeout = 2 초
```

라즈베리파이에서 두 개의 터미널 창을 열어서 위 두 개의 python 코드를 따로 실행시킨다
이 때 라즈베리파이는 인터넷 접속이 가능하도록 WiFi 를 미리 켜놓는다

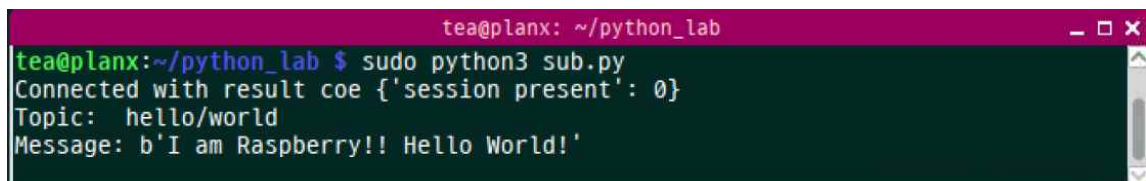
- Subscriber 실행 터미널

A terminal window with a pink title bar labeled 'tea@planx: ~/python_lab'. The command 'sudo python3 sub.py' has been executed, resulting in the output 'Connected with result code {'session present': 0}'.

- Publisher 실행 터미널

A terminal window with a pink title bar labeled 'tea@planx: ~/python_lab'. The command 'sudo python3 pub.py' has been executed, and the prompt has returned to 'tea@planx:~/python_lab \$'.

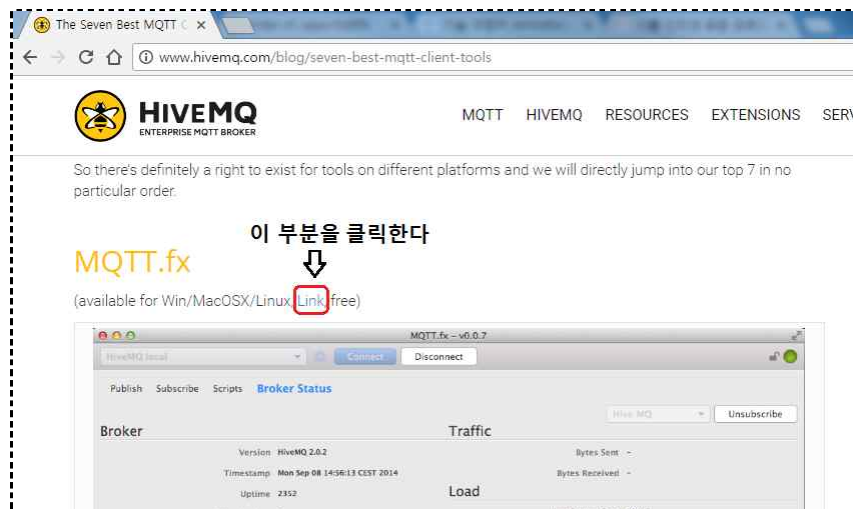
- Subscriber 의 수신 메시지 출력

A terminal window with a pink title bar labeled 'tea@planx: ~/python_lab'. The command 'sudo python3 sub.py' has been executed. The output shows 'Connected with result code {'session present': 0}', 'Topic: hello/world', and 'Message: b'I am Raspberry!! Hello World!'

[3] MQTT 패키지 설치 : Windows 용 MQTT.fx 툴 설치 및 테스트

실습시 배포된 mqttfx-1.4.2-windows-x64.exe 파일을 사용하여 직접 설치한다
(아래 사이트에 접속하여 MQTT.fx 툴을 다운로드 할 수 있다)

<http://www.hivemq.com/blog/seven-best-mqtt-client-tools>

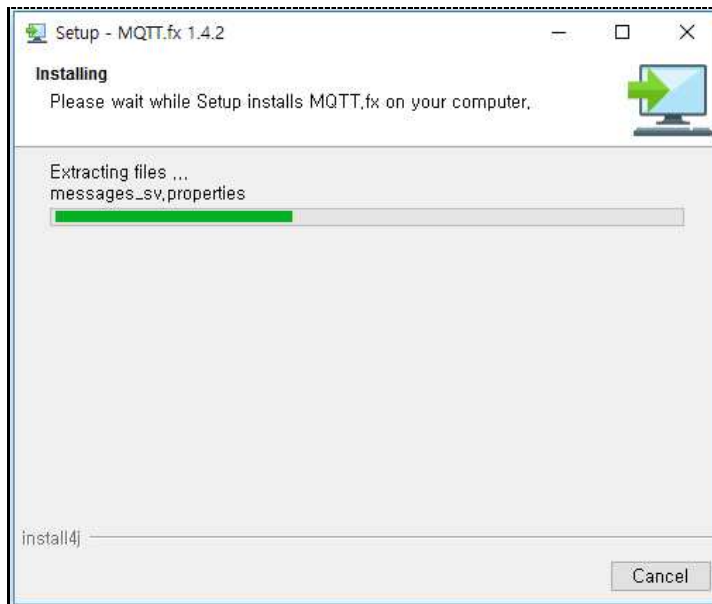


Link로 연결된 사이트에서 아래아 같이 접속하여 알맞은 버전을 다운로드하여 설치한다(생략)

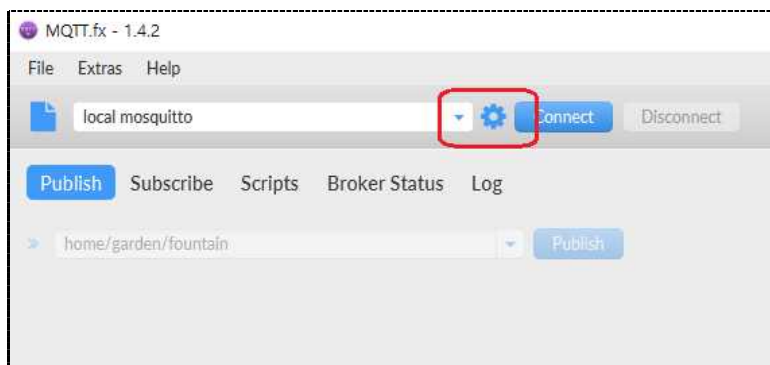


mqttfx-1.4.2-windows-x64.exe (Windows 64bit 용) 과
mqttfx-1.4.2-windows.exe (Windows 32bit 용) 중에서 하나를 설치하면 된다

- 설치 진행 화면



툴 설치 후 시작 메뉴에서  MQTTFX 툴을 찾아 실행한다
툴이 실행되면 아래 설정 부분을 클릭한다



Edit Connection Profile 창이 나타나면서 기본으로 설정되어 있는 값들이 보여 질 것이다
왼쪽 하단의 Profile 추가를 위해 + 버튼을 눌러 새로운 Project Name 과 Broker Address
값으로 설정해 준다

- 초기 기본 설정 값

Edit Connection Profiles

M2M Eclipse
local mosquitto

Connection Profile

Profile Name: local mosquitto

Broker Address: 127.0.0.1

Broker Port: 1883

Client ID: MQTT_FX_Client Generate

General | User Credentials | SSL/TLS | Proxy | Last Will and Testament

Connection Timeout: 30

Keep Alive Interval: 60

Clean Session: ☒

Auto Reconnect: ☒

Max Inflight: 10

MQTT Version: ☒ Use Default
3.1.1

Clear Publish History

Clear Subscription History

Revert Cancel OK Apply

Project Name을 "iCORE IoT Smart Home" 으로 변경하고
 Broker Address 를 "test.mosquitto.org" 로 변경해주고 Client ID는 MQTT_FX_Client를
 사용하지 않고 옆의 Generate 버튼을 눌러 새로 생성해준다
 나머지 항목들은 설정을 그대로 두고 [OK] 버튼을 누른다

M2M Eclipse
iCORE-SDP SMART HOME
local mosquitto

Connection Profile

Profile Name: iCORE-SDP SMART HOME

Broker Address: test.mosquitto.org

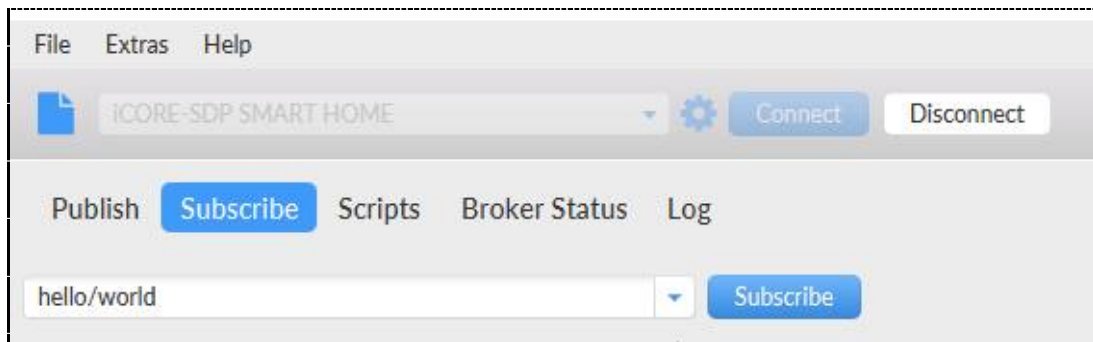
Broker Port: 1883

Client ID: 5f86ac1fc77b4559acf1b2b08de8ec60 Generate

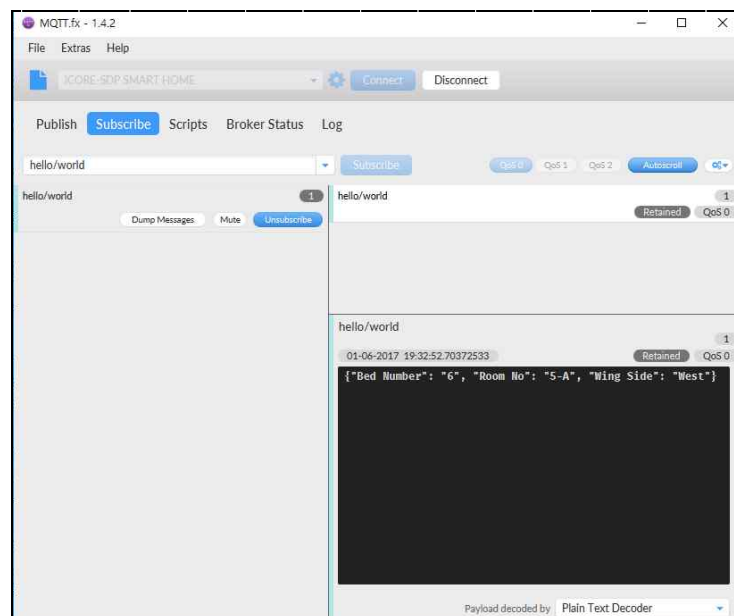
상단의 설정 버튼 오른쪽에 있는 Connect 버튼을 누른다



아래와 같이 Publish 오른쪽에 있는 Subscribe 버튼을 누르고 아래와 같이 "hello/world" 토픽을 입력한 다음 우측의 Subscribe 버튼을 누른다



아래와 같이 연결이 성공하면 하단에 hello/world 토픽의 메시지 창이 뜨게 된다



라즈베리 파이의 Publisher 터미널에서 pub.py를 다시 한번 실행해준다

cd python_lab/mqtt-mosquitto-broker 로 소스 디렉토리로 이동한 후 pub.py를 실행시킨다
- Publisher 실행 터미널

```
tea@planx: ~/python_lab
tea@planx:~/python_lab $ sudo python3 pub.py
tea@planx:~/python_lab $
```

MQTT.fx를 오른쪽 하단에 아래와 같이 라즈베리파이로부터 publishing 된 메시지가 출력 되는 것을 확인 할 수 있을 것이다

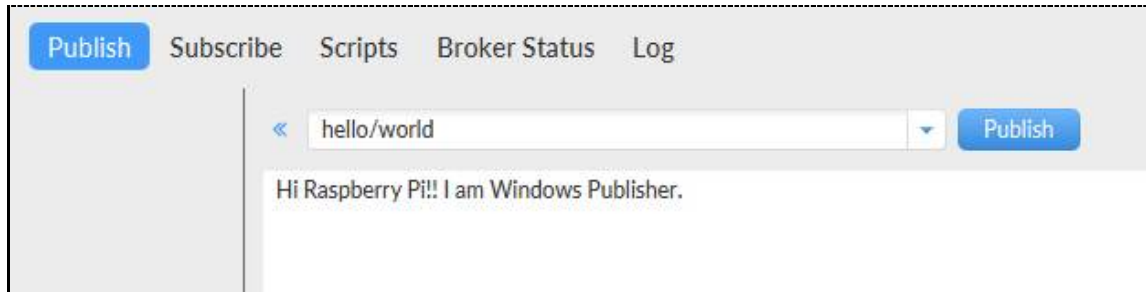


라즈베리파이 쪽에서 계속 publishing 하면 아래와 같이 메시지 버퍼 목록에서 추가로 보인다

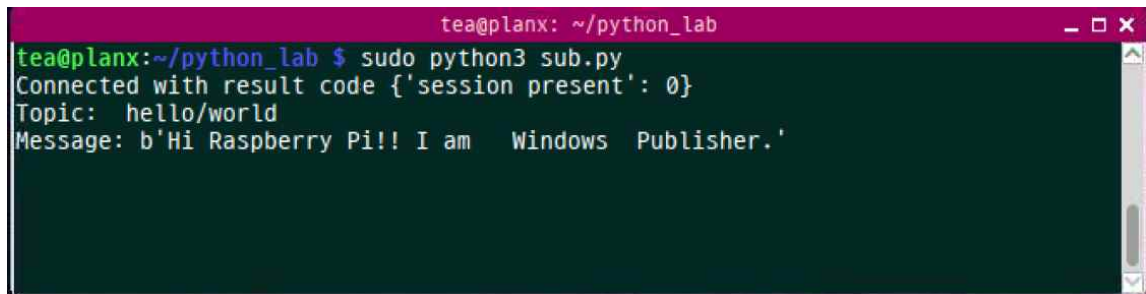


이번에는 Windows에서 라즈베리 파이로 publishing 해보자

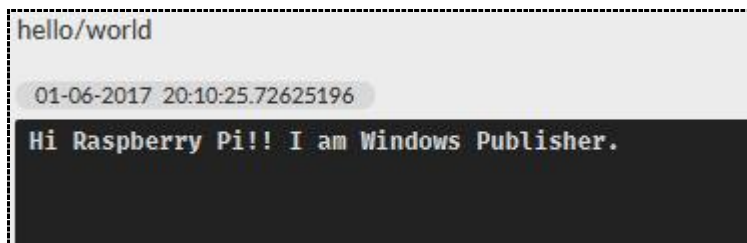
왼쪽 상단의 Publish 버튼을 선택하고 아래와 같이 토픽을 "hello/world" 로 입력해주고
아래와 같이 하단에 보낼 메시지를 입력한다음 오른쪽의 Publish 버튼을 눌러준다



-라즈베리파이 Subscriber 실행 터미널 출력 메시지



윈도우 MQTT.fx 톨의 Subscribe 메시지 화면도 동일한 메시지를 볼수 있을 것이다



[4] 측정 센서별 MQTT 구성을 위한 topic 설정

센서 종류	토픽 명	메시지 타입
온/습도 센서	icore-sdp/temp_humi	"T:27.06 H:29.72"
조도(light) 센서	icore-sdp/light	"214.17"
인체감지 센서	icore-sdp/pir	"10"
초음파 센서	icore-sdp/ultrasonic	"12.55 cm"
부저	icore-sdp/buzzer	"on"
DC 모터	icore-sdp/dc_motor	"on" or "off"
LED1	icore-sdp/led1	"on" or "off"
LED2	icore-sdp/led2	"on" or "off"

[5] 라즈베리파이 센서 측정 Subscriber 와 Publisher 용 python 코드를 구현한다

pi_pub.py : 온/습도 센서, 조도(light) 센서, 인체감지 센서, 초음파 센서 값을 broker로 발행(publish)한다

pi_sub.py : broker로 부터 데이터를 구독(subscribe)하여 보드에 연결되어 있는 부저, DC 모터 , LED1, LED2 와 같은 액추에이터를 가동시킨다

[pi_pub.py 소스 코드]

```
import paho.mqtt.client as mqtt
import RPi.GPIO as GPIO
import smbus2 as smbus
import time

# FOR I2C TEMP & HUMIDITY
bus = smbus.SMBus(1)
th_addr = 0x40
cmd_temp = 0xf3
```

```

cmd_humi = 0xf5
soft_reset = 0xfe
temp = 0.0
humi = 0.0
th_val = 0
th_data = [0, 0]

# FOR I2C LIGHT SENSOR
light_addr = 0x23
reset = 0x07
con_hr_mode = 0x10
one_hr_mode_1 = 0x20

#light_data1 = 0
#light_data2 = 0
#light_val = 0

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005

trig = 0
echo = 1
pir = 24

def get_ultrasonic_distance():
    GPIO.output(trig, False)
    time.sleep(0.1)

    GPIO.output(trig, True)
    time.sleep(0.00001)
    GPIO.output(trig, False)

    while GPIO.input(echo) == False : # Peri v 2.1
        #while GPIO.input(echo) == True : # Peri v 2.0
            pulse_start = time.time()

```



```

while GPIO.input(echo) == True :    # Peri v 2.1
#while GPIO.input(echo) == False : # Peri v 2.0
    pulse_end = time.time()

    pulse_duration = pulse_end - pulse_start
    distance = pulse_duration * 17000
    distance = round(distance, 2)

    #print ("Distance : ", distance, "cm")

    return distance

def get_pir_state():
    return GPIO.input(pir)

def get_temp_value():
    # temperature
    bus.write_byte(th_addr, cmd_temp)
    time.sleep(0.260)

    for i in range(0,2,1):
        th_data[i] = bus.read_byte(th_addr)

    th_val = th_data[0] << 8 | th_data[1]
    temp = -46.85+175.72/65536*th_val
    return temp

def get_humi_value():
    # humidity
    bus.write_byte(th_addr, cmd_humi)
    time.sleep(0.260)

    for i in range(0,2,1):
        th_data[i] = bus.read_byte(th_addr)

```

```

        th_val = th_data[0] << 8 | th_data[1]
        humi = -6.0+125.0/65536*th_val
        return humi

def get_light_value():
    #light
    time.sleep(0.2)
    data = bus.read_i2c_block_data(light_addr,one_hr_mode_1,2)
    return ((data[1] + (256 * data[0])) / 1.2)

def init_mqtt() :
    mqttc = mqtt.Client("raspi_pub")           # MQTT Client
    mqttc.connect("test.mosquitto.org", 1883)  # MQTT
    return mqttc

def main():

    mqttc = init_mqtt()

    GPIO.setwarnings(False)

    GPIO.setmode(GPIO.BCM)           # Use BCM GPIO numbers
    GPIO.setup(trig, GPIO.OUT)
    GPIO.setup(echo, GPIO.IN)         # Peri v 2.1
    #GPIO.setup(echo, GPIO.IN,GPIO.PUD_UP)  # Peri v 2.0

    GPIO.setup(pir, GPIO.IN)

    cnt = 0

    while True :

        val1 = get_temp_value()
        val2 = get_humi_value()

```

```

th_message = 'T:' + format(val1, '.2f') + ' H:' + format(val2, '.2f')
print(th_message)

mqttc.publish("icore-sdp/temp_humi", th_message) # 'temp & humidity' message
publishing

val3 = get_light_value()
light_message = format(val3, '.2f')
print('Light: ', light_message)

mqttc.publish("icore-sdp/light", light_message) # 'light' message publishing

distance = get_ultrasonic_distance()
dist_message = format(distance, '.2f') + ' cm'
print ('Distance:', dist_message)

mqttc.publish("icore-sdp/ultrasonic", dist_message) # 'ultra-sonic' message
publishing

state = get_pir_state()

if (state == True) :
    cnt += 1
    pir_message = format(cnt, 'd')
    print ('[PIR Count] =', pir_message)
    mqttc.publish("icore-sdp/pir", pir_message) # 'pir' message publishing

    #time.sleep(1)

if __name__ == '__main__':
    main()

```

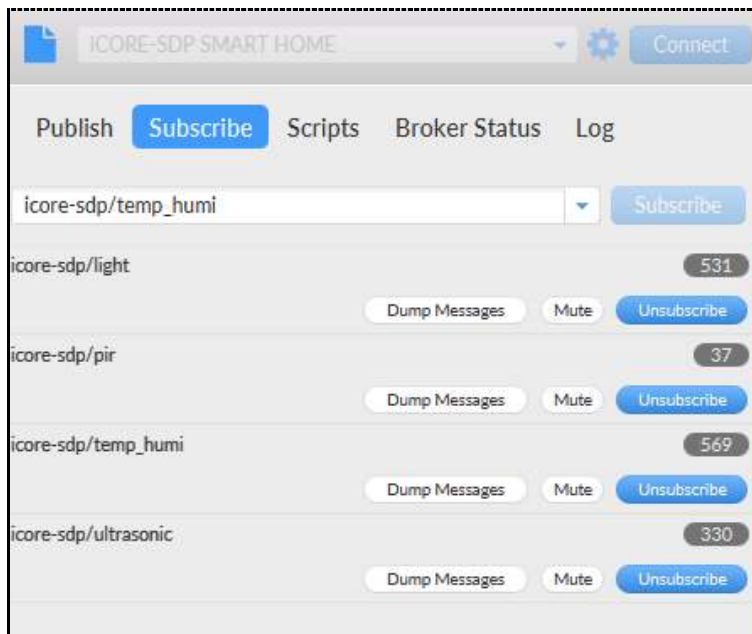
라즈베리파이의 터미널 상에서 위 파이썬 Publisher코드를 실행한다

```
tea@planx: ~/python_lab
tea@planx:~/python_lab $ sudo python3 pi_pub.py
T:29.16 H:49.06
Light: 428.33
Distance: 76.94 cm
T:29.16 H:49.06
Light: 428.33
Distance: 65.81 cm
T:29.16 H:49.06
Light: 428.33
Distance: 61.64 cm
```

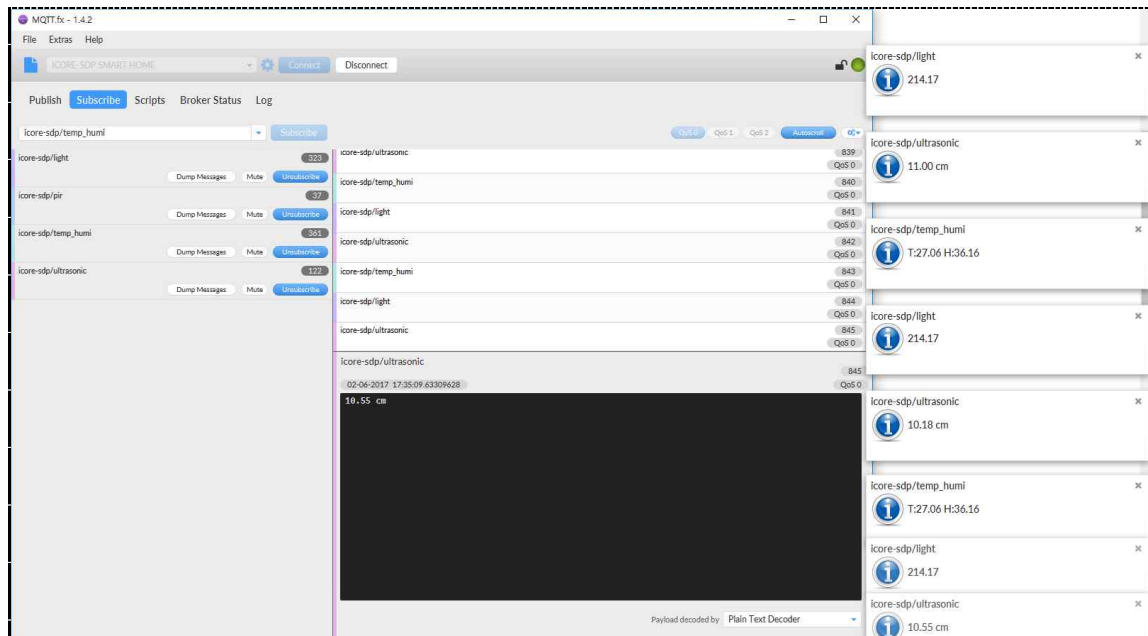
Windows의 MQTT.fx 툴을 실행하고 연결한 다음 아래와 같이 Subscribe 를 선택해서 아래 4 개의 토픽을 각각 입력하고 입력할 때마다 우측 Subscribe 버튼을 눌러준다

<입력할 topic 목록>

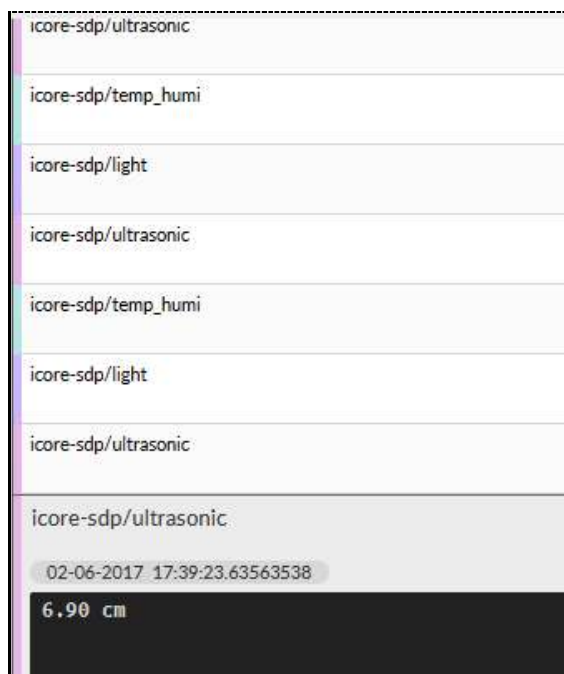
icore-sdp/temp_humi
icore-sdp/light
icore-sdp/pir
icore-sdp/ultrasonic



메시지 수신시 아래 우측과 같이 수신 알림 창이 나타났다가 사라지는 것을 볼수 있을 것이다



메시지 출력 창에는 라즈베리 파이에서 발행한 메시지들이 교대로 출력하는 것을 볼수 있을 것이다



[pi_sub.py 소스 코드]

```
# pi_sub.py
import paho.mqtt.client as mqtt

import RPi.GPIO as GPIO
import time

led1 = 14
led2 = 15

GPIO_RP = 4
GPIO_RN = 25
GPIO_EN = 12

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

buzzer_pin=13
GPIO.setup(buzzer_pin, GPIO.OUT)

scale = [ 261, 294, 329, 349, 392, 440, 493, 523 ]
scale2 = [ 1047, 1175, 1319, 1397, 1568, 1760, 1976, 2093 ]
p = GPIO.PWM(buzzer_pin, 100)
GPIO.output(buzzer_pin, True)

def buzzer_on():

    p.start(100)                # start the PWM on 100% duty cycle
    p.ChangeDutyCycle(90)      # change the duty cycle to 90%

    for i in range(8):
        p.ChangeFrequency(scale[i])
        time.sleep(0.1)
    p.stop()    # stop the PWM output

def led_init(led1,led2):
```

```

GPIO.setup(led1, GPIO.OUT)
GPIO.setup(led2, GPIO.OUT)

def led_on(led_pin):
    GPIO.output(led_pin, True)

def led_off(led_pin):
    GPIO.output(led_pin, False)

def motor_init():
    GPIO.setup(GPIO_RP, GPIO.OUT)
    GPIO.setup(GPIO_RN, GPIO.OUT)
    GPIO.setup(GPIO_EN, GPIO.OUT)

def motor_forward():
    print ('motor forward')
    GPIO.output(GPIO_RP, True)
    GPIO.output(GPIO_RN, False)
    GPIO.output(GPIO_EN, True)

def motor_backward():
    print ('motor backward')
    GPIO.output(GPIO_RP, False)
    GPIO.output(GPIO_RN, True)
    GPIO.output(GPIO_EN, True)

def motor_stop():
    print ('motor stop')
    GPIO.output(GPIO_EN, False)

def on_connect(client, userdata, flags, rc):
    print ("Connected with result code " + str(rc))
    client.subscribe("icore-sdp/buzzer")
    client.subscribe("icore-sdp/dc_motor")
    client.subscribe("icore-sdp/led1")
    client.subscribe("icore-sdp/led2")

```

```

def motor_message_control(buf_str):
    if buf_str == 'on':
        print('MOTOR: ',buf_str)
        motor_forward()

    elif buf_str == 'off':
        print('MOTOR: ',buf_str)
        motor_stop()

def led1_message_control(buf_str):
    if buf_str == 'on':
        print('LED1: ',buf_str)
        led_on(led1)

    elif buf_str == 'off':
        print('LED1: ',buf_str)
        led_off(led1)

def led2_message_control(buf_str):
    if buf_str == 'on':
        print('LED2: ',buf_str)
        led_on(led2)

    elif buf_str == 'off':
        print('LED2: ',buf_str)
        led_off(led2)

def buzzer_message_control(buf_str):
    if buf_str == 'on':
        print('buzzer: ',buf_str)
        buzzer_on()

def on_message(client, userdata, msg):
    buf_str = msg.payload.decode()    # decode() : convert byte to string
    print ('[Topic:', msg.topic + ' ] [Message:' + buf_str + ']')

```



```

    if msg.topic == 'icore-sdp/led1':
        led1_message_control(buf_str)

    elif msg.topic == 'icore-sdp/led2':
        led2_message_control(buf_str)

    elif msg.topic == 'icore-sdp/dc_motor':
        motor_message_control(buf_str)

    elif msg.topic == 'icore-sdp/buzzer':
        buzzer_message_control(buf_str)

if __name__ == "__main__":

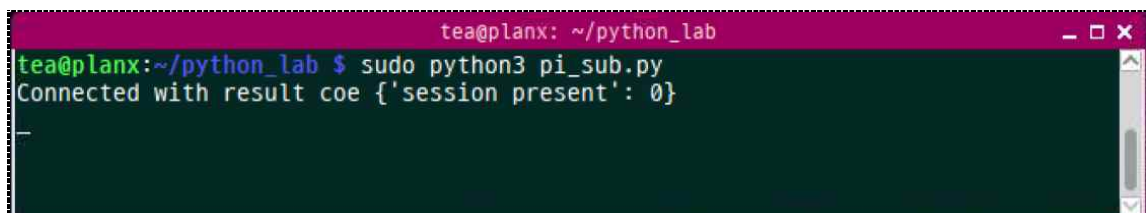
    led_init(led1,led2)

    motor_init()

    client = mqtt.Client()                # MQTT Client
    client.on_connect = on_connect        # on_connect callback
    client.on_message = on_message       # on_message callback
    client.connect("test.mosquitto.org", 1883, 60) # MQTT
    client.loop_forever()

```

라즈베리파이의 터미널 창에서 위의 파이썬 Subscriber 코드를 실행한다



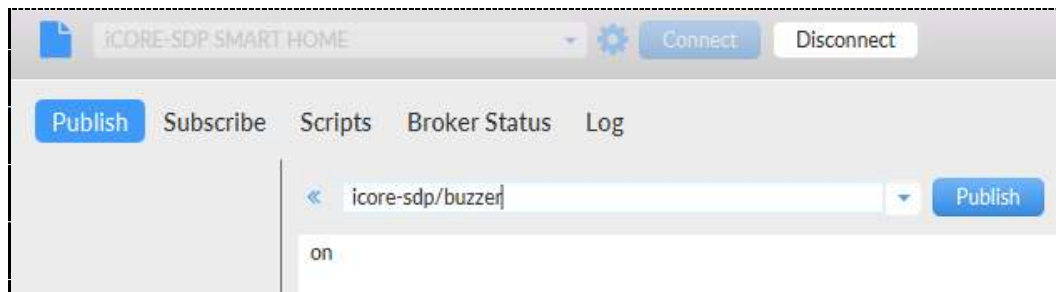
A terminal window with a pink title bar labeled 'tea@planx: ~/python_lab'. The terminal shows the command 'python3 pi_sub.py' being executed, followed by the output 'Connected with result code 0'.

```

tea@planx: ~/python_lab
tea@planx:~/python_lab $ python3 pi_sub.py
Connected with result code 0

```

Windows의 MQTT.fx 툴을 실행하고 연결한 다음 아래와 같이 좌측 Publish를 선택하고 토픽 입력란에 **"icore-sdp/buzzer"**를 입력해주고 메시지 입력창에 **"on"**을 입력한다음 우측 Publish 버튼을 눌러 메시지를 발행시킨다



라즈베리 파이에 부저 사이렌이 잠시 울리고 아래와 같이 Subscribe 창에 수신 메시지를 볼수 있을 것이다

```

tea@planx: ~/python_lab
tea@planx:~/python_lab $ sudo python3 pi_sub.py
Connected with result code {'session present': 0}
[Topic: icore-sdp/buzzer] [Message:on]
buzzer: on

```

아래 4 개의 토픽을 각각 입력하고 메시지 창에 **"on"** 을 입력하고 Publish 버튼을 누른다

부저는 **"on"** 메시지를 받으면 잠깐 동안 멜로디가 울리다 자동으로 멈추고
 모터는 **"on"** 메시지를 받으면 **"off"** 메시지를 수신할 때까지 계속 회전하게 된다
 2 개의 LED는 각각 **"on"** 메시지를 받으면 켜지고 **"off"** 메시지를 수신하면 다시 꺼진다

<입력할 topic 목록>

icore-sdp/buzzer
icore-sdp/dc_motor
icore-sdp/led1
icore-sdp/led2

[6] 안드로이드 스마트폰용 MQTT 앱 사용법

안드로이드 스마트폰에서 구글 Play에 접속하여 MQTT Dash 앱을 설치한다



MQTT Dash 앱은 안드로이드 폰에서 MQTT 프로토콜을 사용해 메시지 Subscribe 와 Publish를 처리할 수 있는 응용프로그램이다

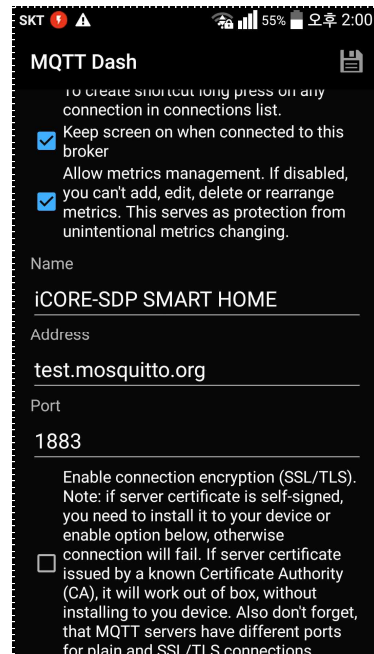
사용자가 topic명을 입력하여 원하는 화면을 구성할 수 있으므로 별도의 자바 응용프로그램을 작성하지 않고 MQTT 프로토콜 통신을 제어 할 수 있다



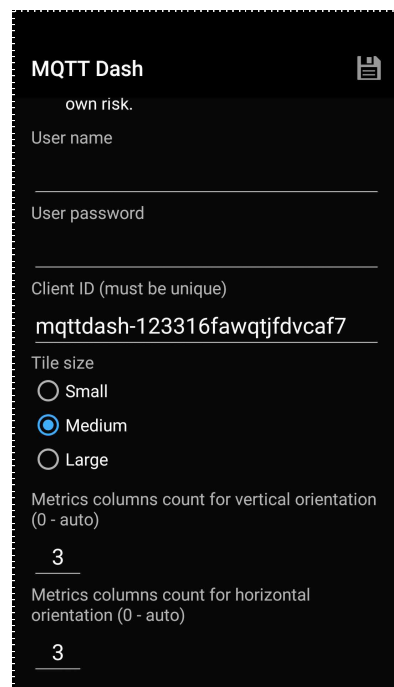
설치된 MQTT Dash를 실행하고 우측에 있는 (+) 버튼을 눌러 연결 구성 설정을 시작한다



아래와 같이 Name 설정란에 "iCORE-SDP SMART HOME"을 입력해주고
 Address 란에 MQTT Broker 주소인 "test.mosquitto.org" 를 입력한다(혹은 pi3 보드 ip주소)
 Port는 1883 으로 설정한다

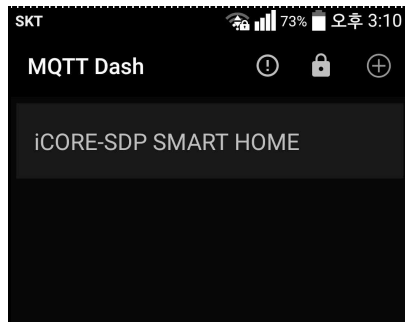


Client ID를 유일한 값으로 임의의 값을 입력해 주고 사용할 가로 세로 metric 값을
 3 으로 설정하고 우측 상단의 저장버튼을 누른다



라즈베리파이 쪽에 미리 Subscriber 와 Publisher 파이썬 코드를 실행시켜 놓는다
스마트폰과 라즈베리파이 모두 WiFi 인터넷에 연결 시켜 놓는다

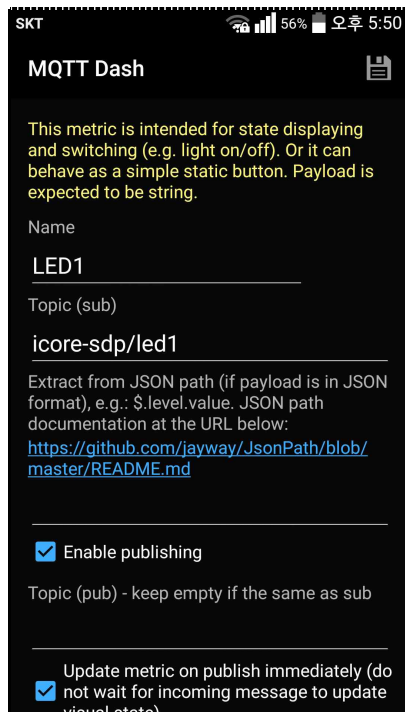
아래와 같이 메뉴가 생성 되어 있다 메뉴를 터치한다



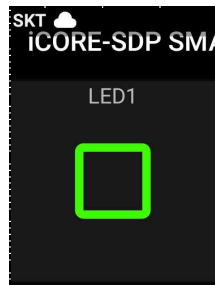
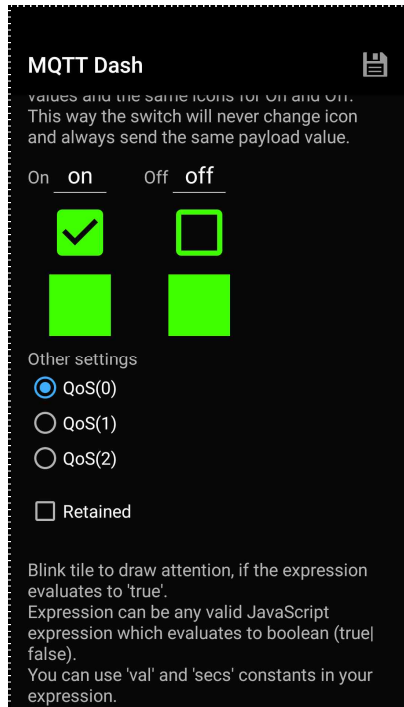
MQTT Broker 주소인 "test.mosquitto.org" 와의 접속이 성공하면 우측 상단의 (+) 버튼을 눌러 화면을 구성한다 여기서 8 개의 topic을 입력하여 구성해보자

(+) 버튼을 누르면 "Choose type" 메뉴가 뜨는데 "Switch/button"을 선택한다

Name 을 "LED1"으로 입력하고 Topic 란에는 "icore-sdp/led1" 을 입력한다

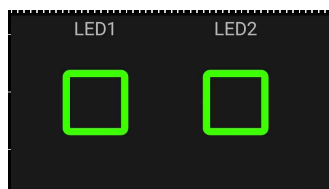


On 에는 "on"을 Off 에는 "off"를 입력하고 아래 색상 박스를 선택하여 아래와 같은 색상을 선택해 준 다음 우측 상단의 저장 버튼을 누르면 아래 우측과 같이 LED1 메뉴가 생성될 것이다



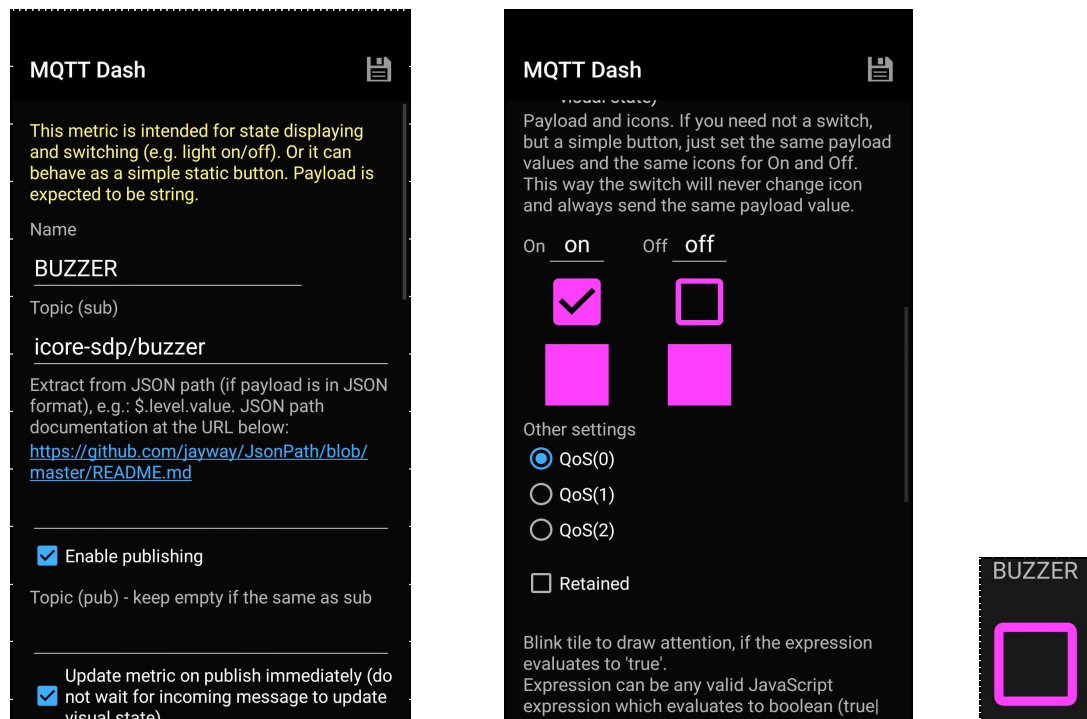
다시 (+) 버튼을 누르고 "Choose type" 메뉴가 뜨면 "Switch/button"을 선택한다
Name 을 "LED2"로 입력하고 Topic 란에는 "icore-sdp/led2" 을 입력한다

On 에는 "on" 을 Off 에는 "off"를 입력하고 아래 색상 박스를 선택하여 LED1 과 같은 색상을 선택해 준 다음 우측 상단의 저장 버튼을 누르면 아래 우측과 같이 LED2 메뉴가 생성될 것이다



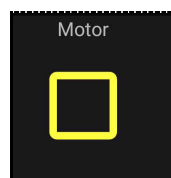
다시 (+) 버튼을 누르고 "Choose type" 메뉴가 뜨면 "Switch/button"을 선택한다
Name 을 "BUZZER"로 입력하고 Topic 란에는 "icore-sdp/buzzer" 을 입력한다

On 에는 "on" 을 Off 에는 "off"를 입력하고 아래 색상 박스를 선택하여 아래 우측과 같은 색상을 선택해 준 다음 우측 상단의 저장 버튼을 누르면 BUZZER 메뉴가 생성될 것이다



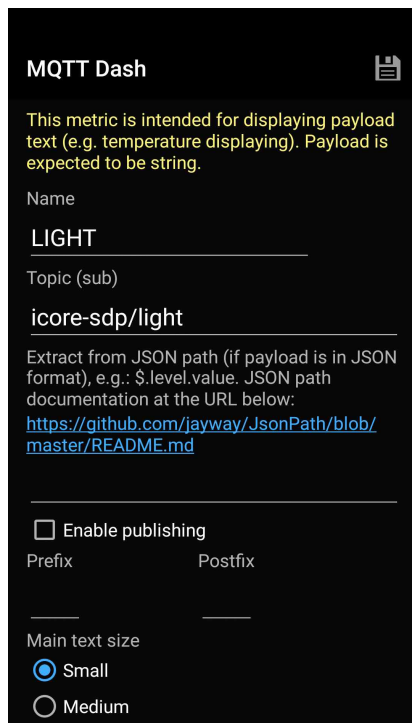
다시 (+) 버튼을 누르고 "Choose type" 메뉴가 뜨면 "Switch/button"을 선택한다
Name 을 "Motor"로 입력하고 Topic 란에는 "icore-sdp/dc_motor" 을 입력한다

On 에는 "on" 을 Off 에는 "off"를 입력하고 아래 색상 박스를 선택하여 아래와 같은 색상을 선택해 준 다음 우측 상단의 저장 버튼을 누르면 Motor 메뉴가 생성될 것이다

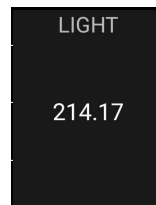


다시 (+) 버튼을 누르고 "Choose type" 메뉴가 뜨면 "Text"를 선택한다
Name 을 "LIGHT"로 입력하고 Topic 란에는 "icore-sdp/light" 를 입력한다

Enable publishing 을 선택 해제해주고 Main text size는 Small로 선택해준다
색상을 선택해 준 다음 우측 상단의 저장 버튼을 누르면 LIGHT 메뉴가 생성될 것이다



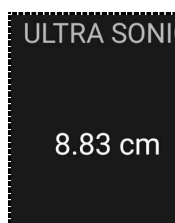
The image shows the MQTT Dash configuration interface. At the top, it says 'MQTT Dash' with a save icon. Below that, a note states: 'This metric is intended for displaying payload text (e.g. temperature displaying). Payload is expected to be string.' The 'Name' field is set to 'LIGHT'. The 'Topic (sub)' field is set to 'icore-sdp/light'. There is a section for 'Extract from JSON path' with a link to a GitHub page. Below that, the 'Enable publishing' checkbox is unchecked. There are fields for 'Prefix' and 'Postfix'. The 'Main text size' is set to 'Small' (indicated by a selected radio button). The 'Medium' radio button is also visible.



The image shows a small dashboard widget for the 'LIGHT' topic. It has a dark background with the text 'LIGHT' at the top and the value '214.17' below it.

다시 (+) 버튼을 누르고 "Choose type" 메뉴가 뜨면 "Text"를 선택한다
Name 을 "ULTRA SONIC"으로 입력하고 Topic 란에는 "icore-sdp/ultrasonic" 를 입력한다

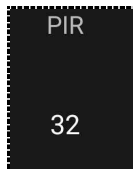
Enable publishing 을 선택 해제해주고 Main text size는 Small로 선택해준다
색상을 선택해 준 다음 우측 상단의 저장 버튼을 누르면 ULTRA SONIC 메뉴가 생성될 것이다



The image shows a small dashboard widget for the 'ULTRA SONIC' topic. It has a dark background with the text 'ULTRA SONIC' at the top and the value '8.83 cm' below it.

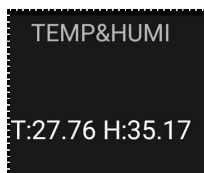
다시 (+) 버튼을 누르고 "Choose type" 메뉴가 뜨면 "Text"를 선택한다
Name 을 "PIR"으로 입력하고 Topic 란에는 "icore-sdp/pir" 을 입력한다

Enable publishing 을 선택 해제해주고 Main text size는 Small로 선택해준다
색상을 선택해 준 다음 우측 상단의 저장 버튼을 누르면 PIR 메뉴가 생성될 것이다



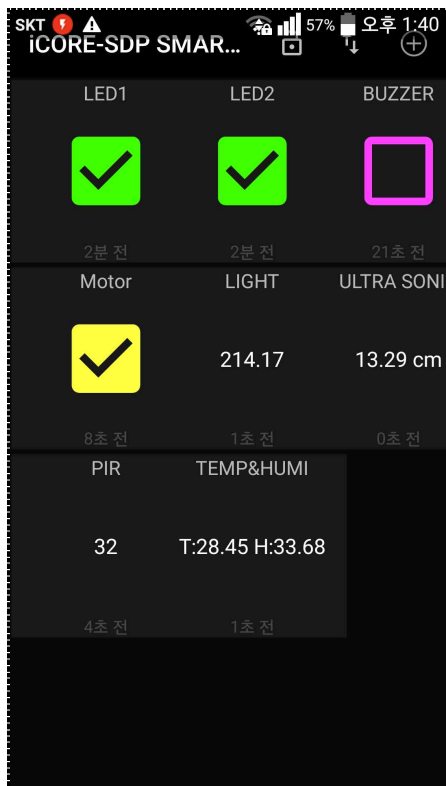
다시 (+) 버튼을 누르고 "Choose type" 메뉴가 뜨면 "Text"를 선택한다
Name 을 "TEMP&HUMI"으로 입력하고 Topic 란에는 "icore-sdp/temp_humi" 를 입력한다

Enable publishing 을 선택 해제해주고 Main text size는 Small로 선택해준다
색상을 선택해 준 다음 우측 상단의 저장 버튼을 누르면 TEMP&HUMI 메뉴가 생성될 것이다



Main text size 선택에서 아래로 내려가면 Main text color 설정부분이 있는데
출력될 텍스트 메시지의 색상을 선택할 수 있다
여기서는 모두 기본값인 흰색으로 되어 있다

이렇게 topic 을 모두 입력하면 아래와 같이 완성된다



LED1 과 LED2 버튼을 누르면 라즈베리파이 보드의 해당 LED가 켜지고 다시 누르면 꺼지는 것을 볼 수 있을 것이다

BUZZER는 버튼을 누르면 잠시 사이렌(멜로디)가 울리고 멈추게 된다 다시 버튼을 눌러 Off 인 상태에서 다시 버튼을 누르면 사이레이 울릴 것이다

Motor는 On 시 회전하고 Off시 회전을 멈춘다

나머지 항목들은 조도 및 오/습도, 거리등의 센서 값을 수신해서 Text로 출력해 준다

[MQTT broker를 라즈베리파이 보드로 설정하는 방법]

당사보드는 배포된 SD카드 이미지에 MQTT broker를 설치해놓아서 mosquitto broker를 사용하지 않고도 Topic를 보내고 받을 수 있다

Python 소스코드에서 MQTT broker 접속시 주소를 아래와 같이 변경해주면 된다

subscriber의 client.connect() 함수와 publisher 의 mqttc.connect() 함수의 인자로 broker의 주소를 설정해 주면 된다

broker : test.mosquitto.org	broker : iCORE-SDP 보드
<pre> <pi_sub.py> if __name__ == "__main__": led_init(led1,led2) motor_init() client = mqtt.Client() client.on_connect = on_connect client.on_message = on_message client.connect("test.mosquitto.org", 1883, 60) # MQTT client.loop_forever() </pre>	<pre> <pi_sub.py> if __name__ == "__main__": led_init(led1,led2) motor_init() client = mqtt.Client() client.on_connect = on_connect client.on_message = on_message client.connect("localhost", 1883, 60) # MQTT client.loop_forever() </pre>
<pre> <pi_pub.py> def init_mqtt() : mqttc = mqtt.Client("raspi_pub") # MQTT Client mqttc.connect("test.mosquitto.org", 1883) # MQTT return mqttc </pre>	<pre> <pi_pub.py> def init_mqtt() : mqttc = mqtt.Client("raspi_pub") # MQTT Client mqttc.connect("localhost", 1883) # MQTT return mqttc </pre>