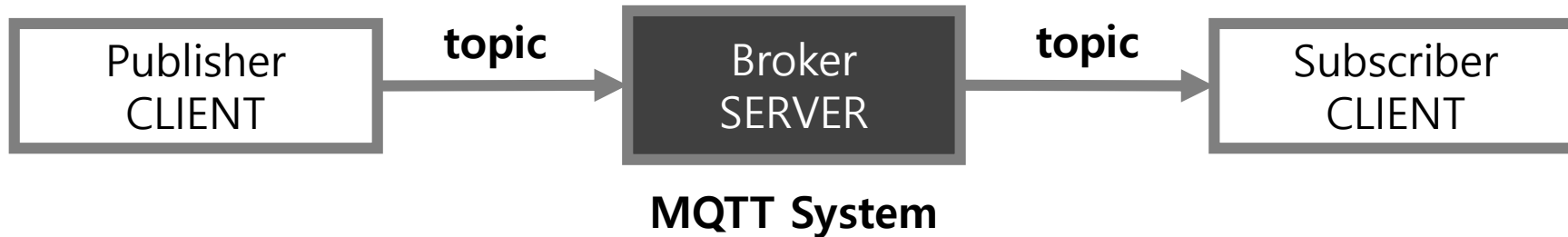


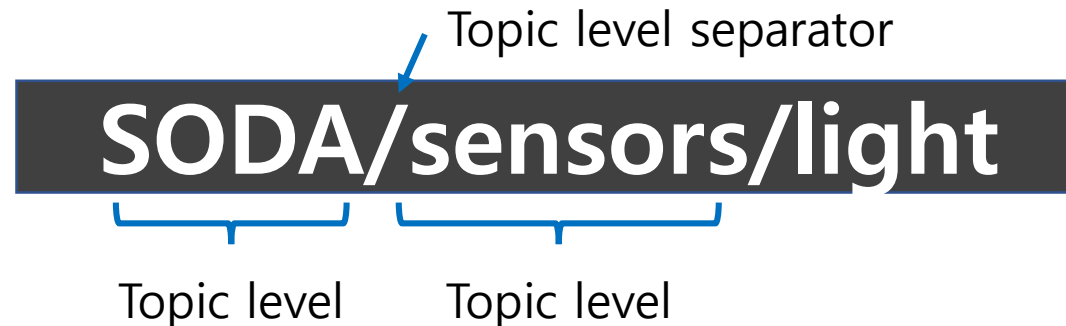
MQTT

- MQTT는 IoT 장치를 위해 낮은 대역폭을 사용하는 메시징 프로토콜
 - 브로커
 - 메시지 큐 서버의 한 종류인 MQTT 중계 서버
 - SODA에는 오픈 소스인 mosquitto 가 설치되어 있음
 - 토픽
 - 메시지 발행, 구독 패턴의 기준
 - 클라이언트
 - 브로커에 토픽을 발행하거나 구독



MQTT 토픽

- 메시지에 대한 발행, 구독 패턴의 기준
 - 클라이언트 사이 미리 정의한 의미대로 정보를 교환할 수 있게 함
 - 대소 문자를 구분하는 UTF-8 문자열로 파일 시스템과 같이 슬래시로 계층 구분
 - \$SYS 토픽을 제외한 나머지는 사용자 정의
 - \$SYS 토픽은 브로커에 대한 정보 공개용
 - 토픽 필터인 '+'와 '#'로 와일드카드 적용
 - +: 해당 레벨만 와일드카드 적용
 - #: 자신을 포함해 모든 하위 레벨에 와일드카드 적용

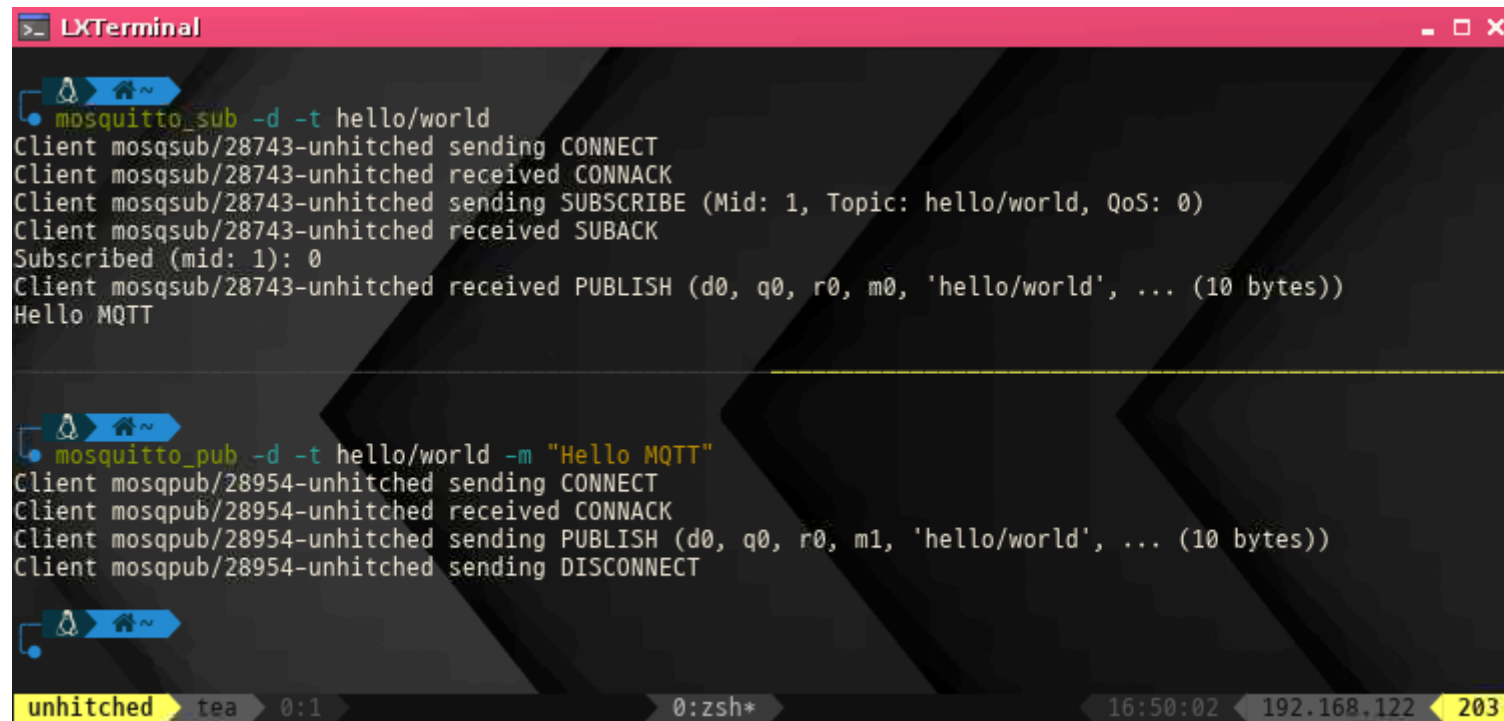


MQTT 테스트

- 타깃에는 MQTT 브로커가 항상 실행 중임
 - 구독자
 - `mosquitto_sub -h 192.168.101.101 -d -t hello/world`
 - -h: 브로커 주소로 로컬이면 생략 가능
 - -d: 디버깅을 위해 내부 패킷 출력
 - -t: 브로커에서 구독할 토픽
 - 게시자
 - `mosquitto_pub -h 192.168.101.101 -d -t hello/world -m "hello MQTT"`
 - -m: 브로커에 게시할 메시지

MQTT 테스트

- 타깃에는 MQTT 브로커가 항상 실행 중임
 - 터미널의 tmux를 수평으로 분할한 후 게시자와 구독자 테스트
 - <ctrl>b "



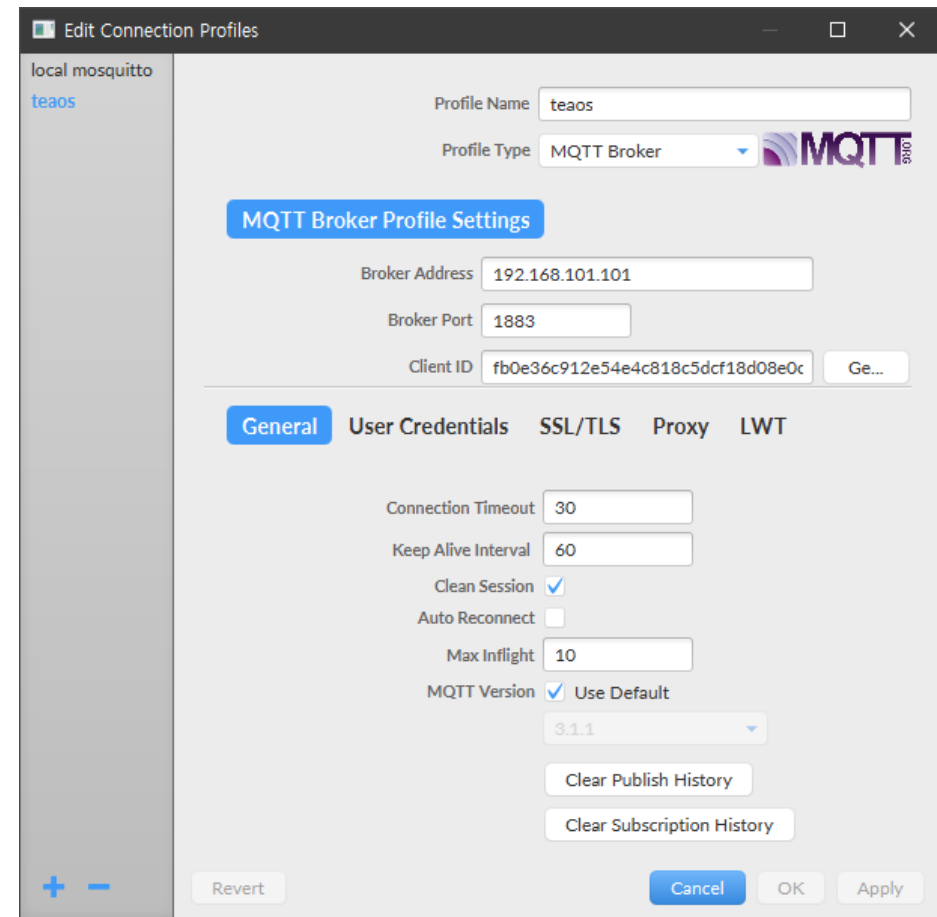
```
LXTerminal
mosquitto_sub -d -t hello/world
Client mosqsub/28743-unhitched sending CONNECT
Client mosqsub/28743-unhitched received CONNACK
Client mosqsub/28743-unhitched sending SUBSCRIBE (Mid: 1, Topic: hello/world, QoS: 0)
Client mosqsub/28743-unhitched received SUBACK
Subscribed (mid: 1): 0
Client mosqsub/28743-unhitched received PUBLISH (d0, q0, r0, m0, 'hello/world', ... (10 bytes))
Hello MQTT

mosquitto_pub -d -t hello/world -m "Hello MQTT"
Client mosqpub/28954-unhitched sending CONNECT
Client mosqpub/28954-unhitched received CONNACK
Client mosqpub/28954-unhitched sending PUBLISH (d0, q0, r0, m1, 'hello/world', ... (10 bytes))
Client mosqpub/28954-unhitched sending DISCONNECT

unhitched tea 0:1 0:zsh* 16:50:02 192.168.122 203
```

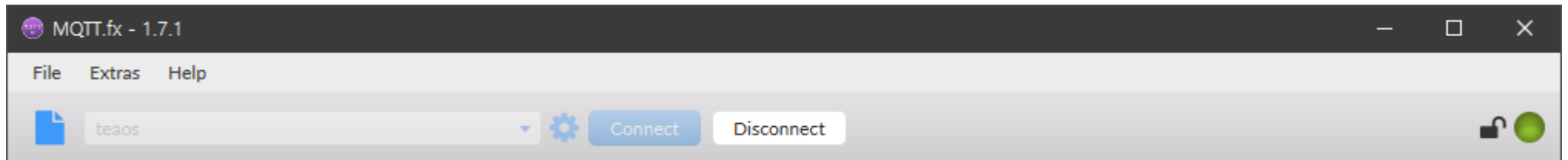
MQTT 테스트

- 호스트에서 테스트
 - 호스트에 MQTT.fx 설치
 - <http://www.jensd.de/apps/mqttfx/1.7.1/>
 - MQTT.fx 실행
 - '메뉴 > Extras > Edit Connection Profiles' 선택
 - Profile name: SODA
 - Broker Address: 192.168.101.101
 - Client ID: 'Generate' 버튼을 눌러 생성
 - 'Apply' 버튼을 눌러 저장한 후 설정 종료

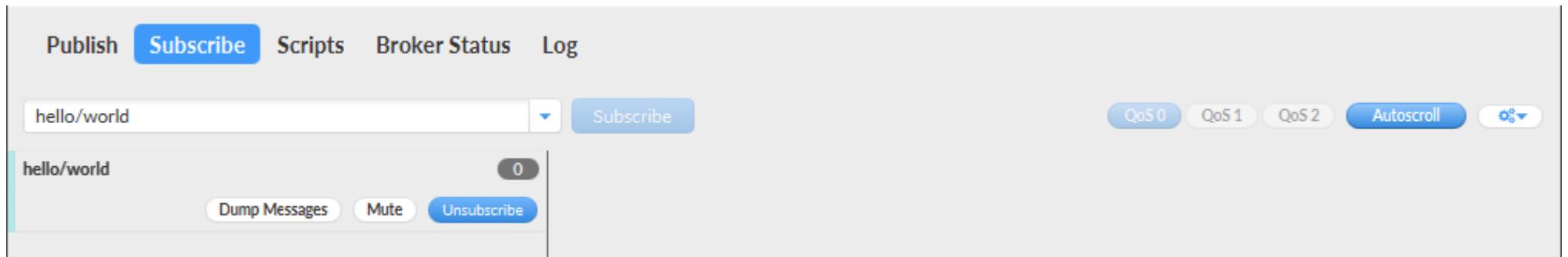


MQTT 테스트

- 호스트에서 테스트 (계속)
 - 'SODA' 프로필에서 'Connect' 버튼을 눌러 브로커에 연결



- Subscribe 탭 선택
 - 토픽에 'hello/world'를 입력한 후 'Subscribe' 버튼 선택

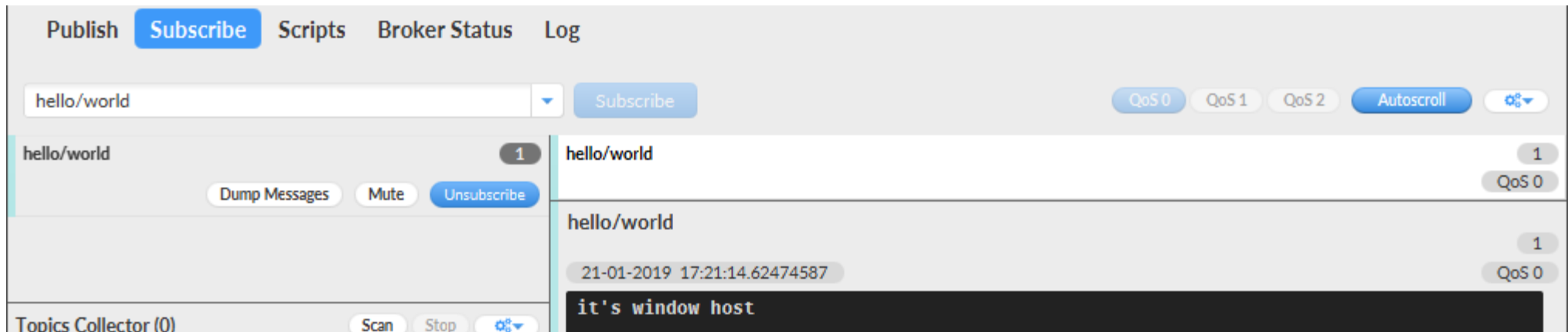


MQTT 테스트

- 호스트에서 테스트 (계속)
 - Publish 탭 선택
 - 토픽에 'hello/world', 메시지 창에 메시지를 입력한 후 'Publish' 버튼 선택



- Subscribe 탭 또는 타깃의 mosquitto_sub ...에서 결과 확인



paho-mqtt

- 멀티 플랫폼을 지원하는 MQTT 클라이언트 라이브러리
 - 발행자 및 구독자 구현에 사용
 - 응용프로그램은 발행자 또는 구독자가 될 수 있음
 - 하나의 응용프로그램에 발행자와 구독자를 모두 구현할 수 있음
 - 하나의 응용프로그램에 여러 개의 발행자 또는 구독자를 구현할 수 있음
 - 비동기 메시지 처리 메커니즘 지원
 - 내부 메시지 루프를 통해 상황 별로 사용자 등록한 함수 콜백
 - 브로커에 연결 알림
 - 구독한 토픽의 메시지 수신 알림
 - 발행한 토픽의 메시지 게시 알림
 - 파이썬 인터프리터가 설치된 호스트는 패키지 추가 설치 필요
 - `pip install paho-mqtt`

paho-mqtt

- paho.mqtt.client 모듈의 Client 클래스
 - `__init__(client_id="", clean_session=True, userdata=None, protocol=MQTTv311, transport="tcp")`
 - MQTT 클라이언트 객체 반환
 - `client_id`: 브로커에 연결할 때 사용할 클라이언트 ID. 기본값은 자동 생성
 - `clean_session`: 세션 모드 설정. 기본값인 `True`는 클린 세션 사용
 - `False`로 설정하면 명시적으로 `client_id` 부여한 상태에서 영구 세션 사용
 - `userdata`: 콜백 함수가 호출될 때 함께 전달할 사용자 데이터
 - `protocol`: 사용할 프로토콜 버전
 - `transport`: 전송 계층 선택. 기본값은 TCP

paho-mqtt

- on_connect: connect()의 결과로 호출할 콜백 설정
 - on_connect(client, userdata, flags, rc)
 - client: 생성된 Client 객체
 - userdata: Client 객체를 생성할 때 전달한 인자
 - flags: 브로커가 보낸 응답 플래그
 - rc: 결과 코드로 0이면 브로커에 성공적으로 연결. 나머지는 실패
- on_message: subscribe()로 가입한 메시지가 수신될 때 호출할 콜백 설정
 - on_message(client, userdata, message)
 - message: MQTTMessage 객체 타입의 수신 메시지
 - topic: 토픽, payload: 메시지, qos: 메시지 품질, retain: 지속 데이터 여부
- on_publish: publish()로 브로커에 메시지를 전달한 후 호출할 콜백 설정
 - on_publish(client, userdata, mid)
 - mid: 메시지 ID (또는 패킷 ID)

paho-mqtt

- `connect(host, port=1883, keepalive=60, bind_address="")`
 - 브로커에 연결
 - `host`: 문자열 형식의 브로커 주소
 - `port`: 브로커의 포트 번호. 기본값은 1883
 - `keepalive`: 킵얼라이브 만료 시간. 기본값은 60초
 - `bind_address`: 네트워크 인터페이스가 여러 개일 때 바인드 대상
 - 결과는 `on_connect`에 설정한 사용자 콜백 함수를 통해 전달됨
- `subscribe(topic, qos=0)`
 - 하나 이상의 토픽 구독
 - `topic`: 구독할 토픽으로 튜플로 묶으면 여러 개의 토픽 전달 가능
 - `on_subscribe`에 설정한 사용자 콜백 함수가 있으면 호출됨
 - 메시지를 받으면 `on_message`에 설정한 사용자 콜백 함수를 통해 반환

paho-mqtt

- `publish(topic, payload=None, qos=0, retain=False)`
 - 브로커에 메시지 발생
 - `topic`: 메시지에 부여할 토픽
 - `payload`: 게시할 메시지. 문자열, `int`, `float`는 내부에서 바이트 배열로 변환
 - 최대 크기는 263,435,455byte
 - `retain`: `True`이면 브로커에 마지막 메시지 보존. 기본값은 `False`로 보존하지 않음
 - 반환값은 `MQTTMessageInfo`로 `rc` 값이 0이면 성공적으로 전송
 - `on_publish`에 설정한 사용자 콜백 함수가 있으면 호출됨

메시지 발행

- 브로커에 메시지 발행

```
import paho.mqtt.client as mqtt
```

```
def publish_message():
```

```
    data = input("Enter message: ")
```

```
    client.publish("SODA/hello", data) #SODA/hello 토픽에 input()으로 받은 사용자 데이터 게시
```

```
def do_connect(client, userdata, flags, rc):
```

```
    print("ok connect")
```

```
    publish_message()
```

```
def do_publish(client, userdata, mid):
```

```
    print("ok publish")
```

```
    publish_message()
```

```
client = mqtt.Client()
```

```
client.on_connect = do_connect
```

```
client.on_publish = do_publish
```

```
client.connect("192.168.101.101") #실제 브로커 주소 사용
```

```
client.loop_forever()
```

토픽 구독

- 브로커에서 토픽 구독

```
import paho.mqtt.client as mqtt

def do_connect(client, userdata, flags, rc):
    print("ok connect")
    client.subscribe("hello/world")

def do_message(client, userdata, message):
    print("[%s] %s"%(message.topic, message.payload.decode()))

client = mqtt.Client()

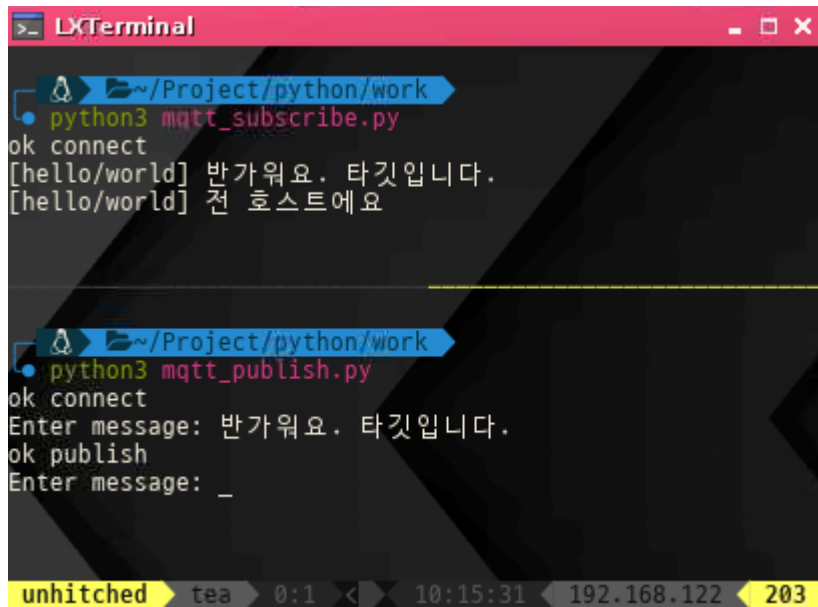
client.on_connect = do_connect
client.on_message = do_message

client.connect("192.168.101.101") #실제 브로커 주소 사용

client.loop_forever()
```

메시지 발행, 토픽 구독 실행

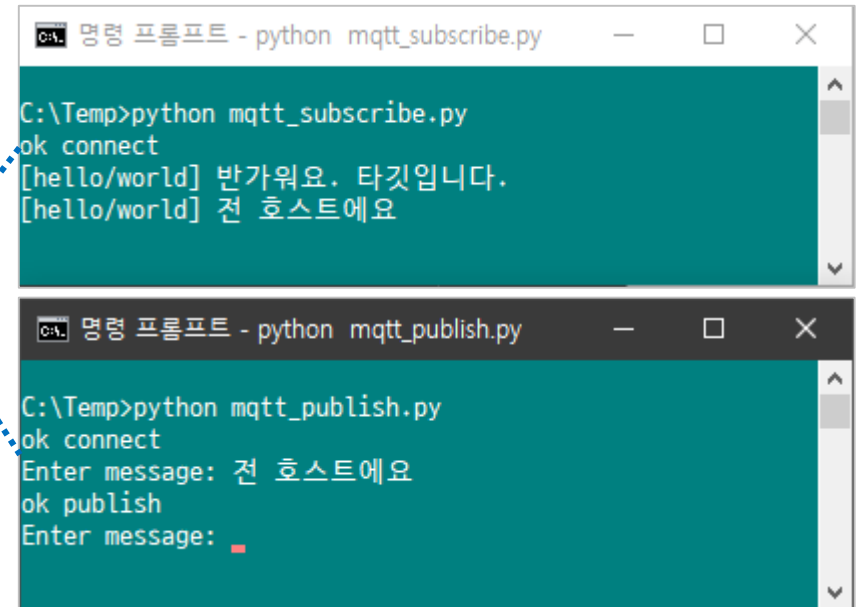
- 발행과 구독 프로그램 실행
 - 브로커가 실행 중인 타깃에서 양쪽 모두 실행 가능
 - 다른 타깃을 사용한다면 코드에서 브로커 주소 변경
 - 호스트에 paho-mqtt를 설치했다면 호스트에서도 실행 가능
 - 발행 프로그램에서 메시지를 발생하면 구독 프로그램에 표시됨



```
> LXTerminal
~/Project/python/work
python3 mqtt_subscribe.py
ok connect
[hello/world] 반가워요. 타깃입니다.
[hello/world] 전 호스트예요

~/Project/python/work
python3 mqtt_publish.py
ok connect
Enter message: 반가워요. 타깃입니다.
ok publish
Enter message: _
```

Broker



```
C:\Temp>python mqtt_subscribe.py
ok connect
[hello/world] 반가워요. 타깃입니다.
[hello/world] 전 호스트예요

C:\Temp>python mqtt_publish.py
ok connect
Enter message: 전 호스트예요
ok publish
Enter message: _
```

채팅

- 채팅을 위해 메시지 발행과 토픽 구독을 함께 구현
 - 프로그램을 실행하면 자신의 `nic_name` 입력
 - 토픽
 - 발행 토픽: `"hello" + nic_name + "/chat"`
 - 구독 필터: `"hello/+chat"`
 - 수신한 메시지 중 자신이 게시한 메시지는 출력 안 함
- 발행과 구독을 함께 처리할 때는 별도의 스레드 필요
 - 대기가 있는 쪽을 작업 스레드에서 처리
 - 발행은 사용자 데이터를 얻기 위해 `input()` 사용
 - `input()`은 입력 대기가 있음

채팅

- 채팅을 위해 메시지 발행과 토픽 구독을 함께 구현 (계속)

```
import paho.mqtt.client as mqtt
import threading

nic_name = None

def __task_publish(client):
    topic = "hello/" + nic_name + "/chat"
    data = input()

    client.publish(topic, data)

def do_connect(client, userdata, flags, rc):
    if rc == 0:
        print("hi~ %s"%(nic_name))
        client.subscribe("hello+/chat")
        threading.Thread(target=__task_publish, args=(client,)).start()
    else:
        print("not connect")

def do_publish(client, userdata, mid):
    threading.Thread(target=__task_publish, args=(client,)).start()
```

채팅

- 채팅을 위해 메시지 발행과 토픽 구독을 함께 구현 (계속)

```
def do_message(client, usrdata, message):
    t_nic_name = message.topic.split('/')[1]

    if nic_name != t_nic_name:
        print(' ' * 40, "[%s] %s"%(t_nic_name, message.payload.decode()))

def main():
    global nic_name

    nic_name = input("your nic name: ")

    client = mqtt.Client()
    client.on_connect = do_connect
    client.on_publish = do_publish
    client.on_message = do_message
    client.connect("192.168.101.101") #실제 브로커 주소 사용
    client.loop_forever()

if __name__ == "__main__":
    main()
```

채팅

- 채팅을 위해 메시지 발행과 토픽 구독을 함께 구현 (계속)



```
LXTerminal
~/Project/python/work
python3 mqtt_chat.py
your nic name: 하하맨
hi~ 하하맨

[우주천사] 모두 안녕!!!
[얼짱맨] 미투....

반가워~
[우주천사] 그럼
[얼짱맨] 난 심심해
[얼짱맨] 뭐 재밌는 일 없을까?

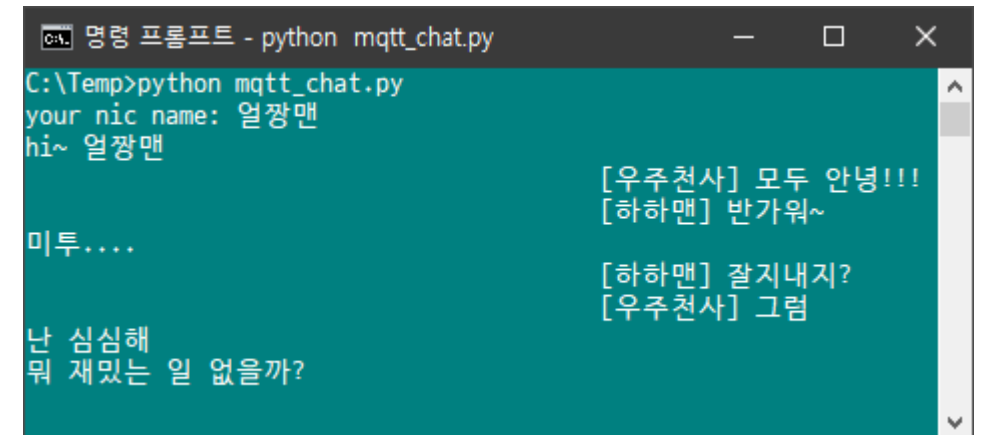
잘지내지?

~/Project/python/work
python3 mqtt_chat.py
your nic name: 우주천사
hi~ 우주천사
모두 안녕!!!

[하하맨] 반가워~
[얼짱맨] 미투....
[하하맨] 잘지내지?

그럼
[얼짱맨] 난 심심해
[얼짱맨] 뭐 재밌는 일 없을까?

unhitched tea 0:1 0:python3* 13:13:29 192.168.122 203
```



```
명령 프롬프트 - python mqtt_chat.py
C:\Temp>python mqtt_chat.py
your nic name: 얼짱맨
hi~ 얼짱맨

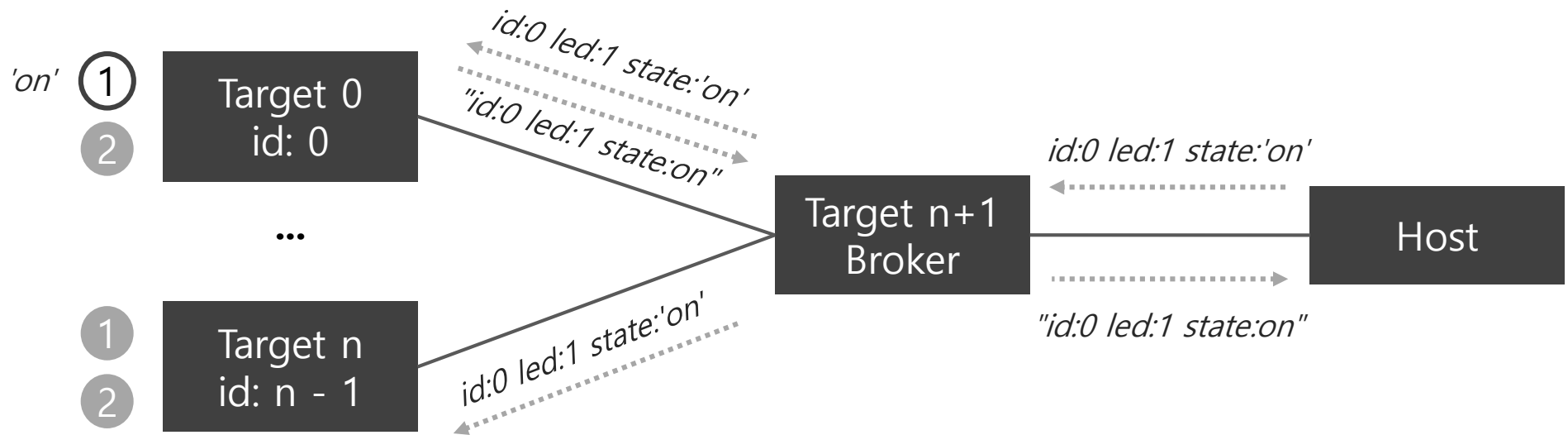
[우주천사] 모두 안녕!!!
[하하맨] 반가워~

미투....
[하하맨] 잘지내지?
[우주천사] 그럼

난 심심해
뭐 재밌는 일 없을까?
```

여러 개의 타겟 LED 제어

- 호스트에서 여러 개의 타겟 LED 켜고 끄기
 - 타겟 구분
 - 프로그램이 실행될 때 입력하는 일련 번호로 식별(id)
 - 호스트는 타겟 id를 알고 있다고 가정 함



여러 개의 타깃 LED 제어

- 호스트에서 여러 개의 타깃 LED 켜고 끄기 (계속)
 - 토픽
 - SODA/led/action:
 - 호스트에서 브로커에 JSON 문자열로 해당 타깃의 LED 제어 메시지 전달
 - json: "{₩"id₩": 0, ₩"led₩": 1, ₩"action₩": ₩"on₩"}"
 - 메시지를 수신한 타깃은 JSON 문자열을 딕셔너리로 변경
 - dict: { 'id':0, 'led':1, 'action': 'on'}
 - 'id'가 자신이면 'led'와 'action'을 참조해 LED 상태 변경
 - SODA/led/state
 - LED의 상태를 변경한 타깃은 문자열로 브로커에 이를 알림
 - "id:%d, led:%d, state:%s"%(led_info['id'], led_info['led'], led_info['action'])
 - 호스트는 변경된 타깃의 LED 상태를 수신 메시지로 파악

여러 개의 타깃 LED 제어

- 호스트에서 여러 개의 타깃 LED 켜고 끄기 (계속)
 - 메시지
 - json 패키지는 파이썬 표준 라이브러리
 - dumps(): 딕셔너리를 JSON 문자열로 변환
 - loads(): JSON 문자열을 딕셔너리로 변환
 - 호스트에서 타깃으로 전달되는 LED 상태 변경 메시지는 JSON 사용
 - led_info = {
 - 'id': <0 ~ n>,
 - 'led':<1 or 2>,
 - 'state': <'on' or 'off'>
 - 타깃에서 호스트로 전달되는 LED 상태 알림 메시지는 문자열 사용
 - "id:0, led:1, state:on"

여러 개의 타겟 LED 제어

- 호스트에서 타겟의 LED 켜고 끄기 (계속)
 - mqtt_led_target.py

```
import paho.mqtt.client as mqtt
import subprocess
import json

id = None

def do_connect(client, userdata, flags, rc):
    if rc == 0:
        print("ok connect")
        client.subscribe("SODA/led/action")
    else:
        print("not connect")

def do_message(client, userdata, message):
    led_info = json.loads(message.payload)

    if id != led_info['id']:
        return
```

여러 개의 타겟 LED 제어

- 호스트에서 타겟의 LED 켜고 끄기 (계속)
 - mqtt_led_target.py

```
if led_info['action'] == 'on':
    subprocess.Popen("echo 1 > /sys/class/gpio/gpio%d/value"%(led_info['led'] + 13), shell=True)
elif led_info['action'] == 'off':
    subprocess.Popen("echo 0 > /sys/class/gpio/gpio%d/value"%(led_info['led'] + 13), shell=True)

client.publish("SODA/led/state", "id:%d, led:%d, state:%s"%(id, led_info['led'], led_info['action']))

def main():
    global id

    id = int(input("input your id: "))

    client = mqtt.Client()
    client.on_connect = do_connect
    client.on_message = do_message
    client.connect("192.168.101.101")    #실제 브로커 주소 사용
    client.loop_forever()

if __name__ == "__main__":
    main()
```


여러 개의 타겟 LED 제어

- 호스트에서 타겟의 LED 켜고 끄기 (계속)
 - mqtt_led_host.py

```
import paho.mqtt.client as mqtt
import threading
import json

def __task_publish(client):
    led_info = {'id':None, 'led':None, 'action':None}

    led_info['id'] = int(input("Target ID: "))
    led_info['led'] = int(input("LED: "))
    led_info['action'] = input("Action: ")
    client.publish("SODA/led/action", json.dumps(led_info))

def do_connect(client, userdata, flags, rc):
    if rc == 0:
        print("ok connect")
        client.subscribe("SODA/led/state")
        threading.Thread(target=__task_publish, args=(client,)).start()
    else:
        print("not connect")
```

여러 개의 타겟 LED 제어

- 호스트에서 타겟의 LED 켜고 끄기 (계속)
 - mqtt_led_host.py

```
def do_publish(client, userdata, mid):
    threading.Thread(target=__task_publish, args=(client,)).start()

def do_message(client, userdata, message):
    print(' ' * 20, "<<<", message.payload.decode())

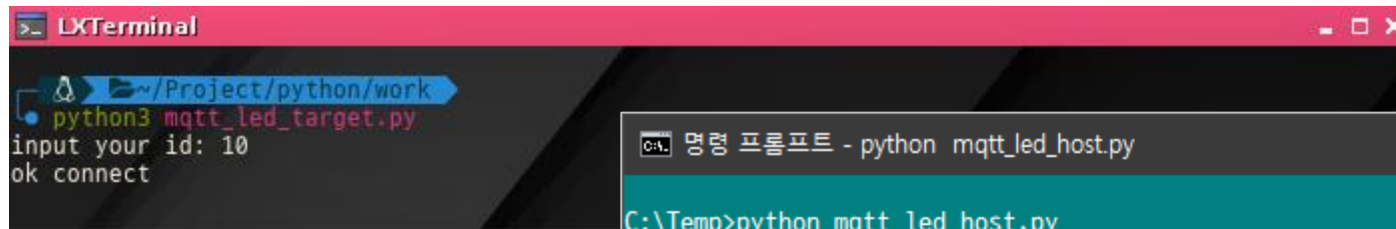
def main():
    client = mqtt.Client()

    client.on_connect = do_connect
    client.on_publish = do_publish
    client.on_message = do_message
    client.connect("192.168.101.101") #실제 브로커 주소 사용
    client.loop_forever()

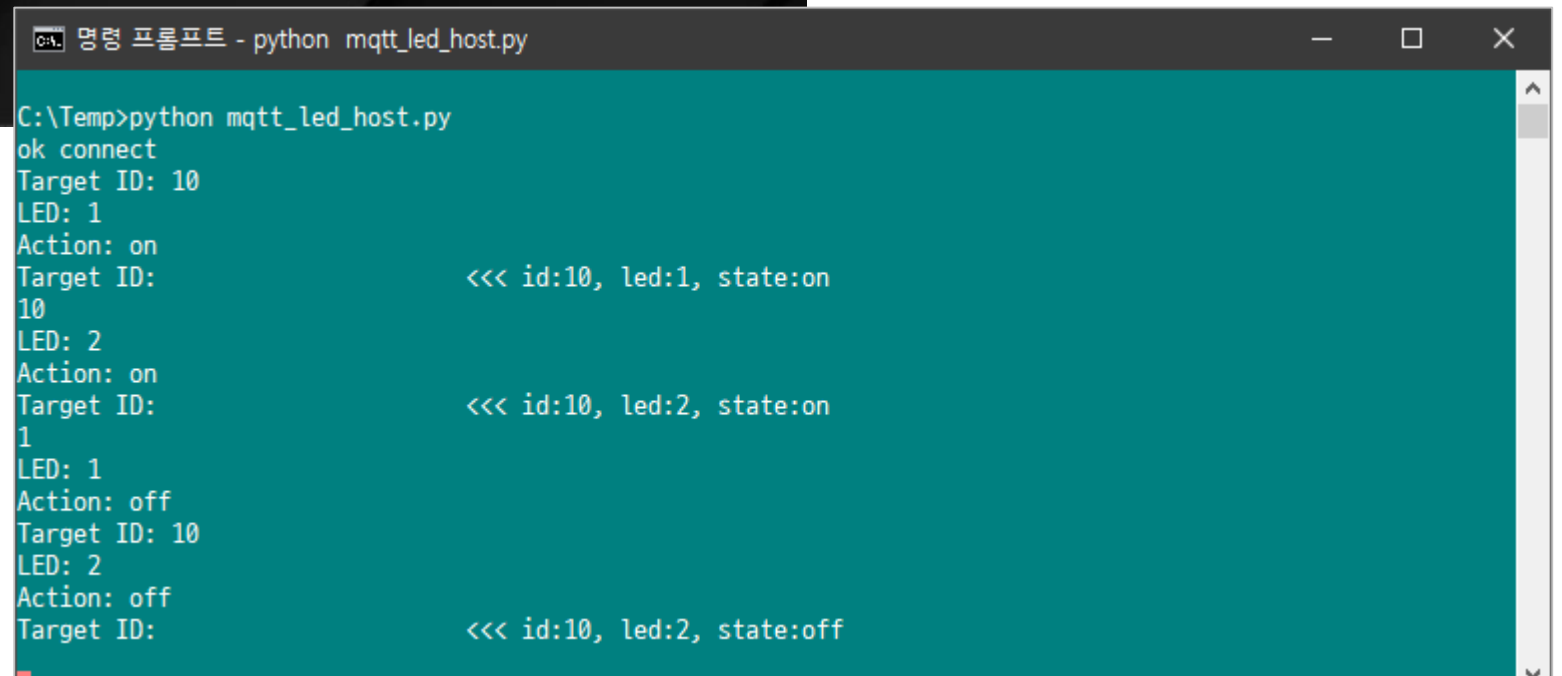
if __name__ == "__main__":
    main()
```

여러 개의 타깃 LED 제어

- 호스트에서 타깃의 LED 켜고 끄기 (계속)
 - 호스트용 프로그램은 다른 타깃에서 실행해도 됨



```
LXTerminal
~/Project/python/work
python3 mqtt_led_target.py
input your id: 10
ok connect
```



```
명령 프롬프트 - python mqtt_led_host.py
C:\Temp>python mqtt_led_host.py
ok connect
Target ID: 10
LED: 1
Action: on
Target ID: <<< id:10, led:1, state:on
10
LED: 2
Action: on
Target ID: <<< id:10, led:2, state:on
1
LED: 1
Action: off
Target ID: 10
LED: 2
Action: off
Target ID: <<< id:10, led:2, state:off
```