**v1.0.0**

**임무 소프트웨어 Guide**

**시간동기화**

**msw\_timesync**

**목 차**

[1 개요 3](#_Toc76657592)

[2 시간동기화 4](#_Toc76657593)

[2.1 시간동기화 SW 4](#_Toc76657594)

[2.2 시간동기화 과정 5](#_Toc76657595)

[3 임무Lib 및 임무SW 개발 6](#_Toc76657596)

[3.1 임무Lib, 임무SW와 개발도구 6](#_Toc76657597)

[3.2 임무Lib 개발 6](#_Toc76657598)

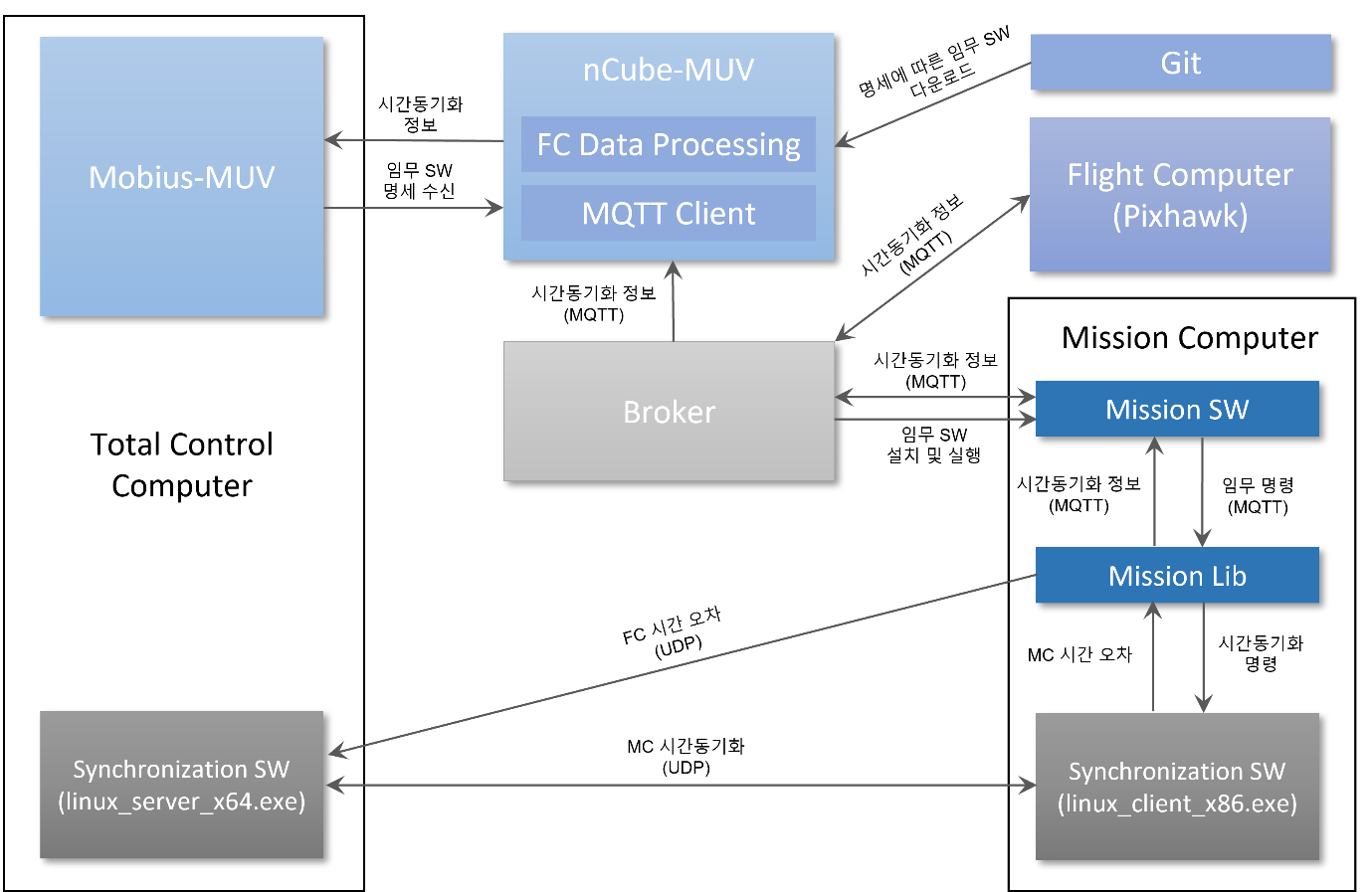
[3.3 임무SW 개발 13](#_Toc76657599)

# 개요

이 문서는 임무를 수행하는 다수의 드론에 탑재된 임무 컴퓨터(Mission Computer, MC)와 비행제어 컴퓨터(Flight Controller, FC) 간 시간 동기를 제어하기 위한 방법을 설명하는 문서이다.

시간동기화 프로그램은 드론이 임무를 수행하는 동안 상시 동작하며 MC는 동일한 기체에 탑재된 FC와 다른 기체의 MC 간 시간 오차를 나타내는 두 가지 오프셋을 계산하여 중앙에 전송한다. MC에서 동작하는 시간동기화 프로그램은 다수의 MC 간 오프셋을 계산하기 위한 동기화 서버가 중앙에서 동작하며 MC에서 FC와 다수의 MC 간 시간 오차를 보정한다.

이를 위해 시간동기화 SW와 동기화 과정에 대해 설명하고 임무라이브러리(Library, Lib), 임무 Lib를 포함한 형태로 nCube-MUV와 연동되어 외부와 통신하며 임무 Lib를 제어하는 임무 소프트웨어(Software, SW)에 대해 설명하고 동작하기 위한 방법을 설명한다.

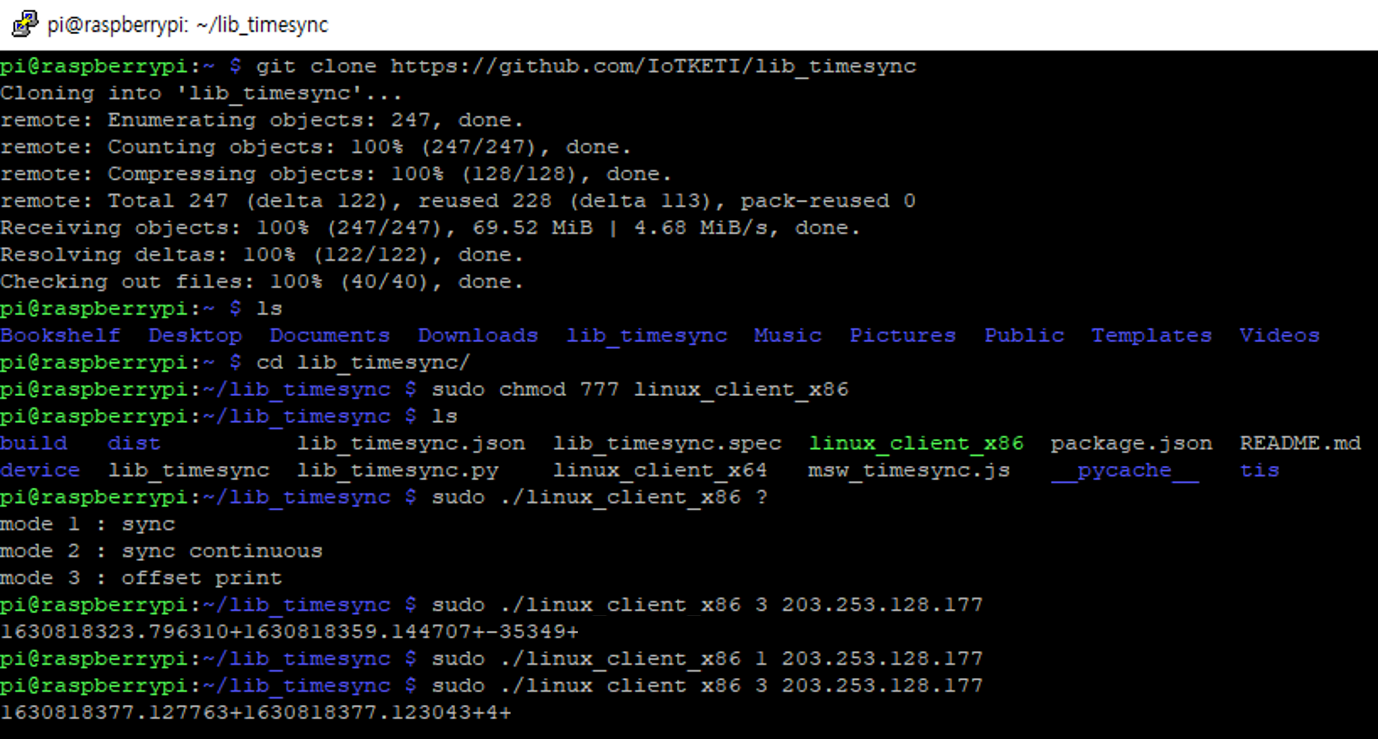


<시간동기화 임무 소프트웨어(msw\_timesync) 아키텍처>

# 시간동기화

## 시간동기화 SW

다수의 드론이 임무를 수행하는 환경을 고려하여 MC 간 5ms 이하의 동기화를 수행하기 위한 시간동기화 알고리즘은 서버와 클라이언트 단의 SW로 구분되어 개발된다. 동기화 SW는 Two-way 알고리즘에 기반하여 socket 통신을 이용한 C언어로 개발되며 실행파일로 만들어진다. 서버 동기화 SW는 중앙에서 상시 동작하며, 클라이언트의 동기화 SW는 MC에서 실행되어 시간동기화 수행 및 오프셋을 계산하여 출력할 수 있다.

 <Raspberry pi를 이용한 동기화 SW 실행 및 검증>

5ms 동기화 확인

동기화 SW 실행

git clone lib\_timesync

KETI IP (중앙동기화 서버)

## 시간동기화 과정

시간동기화 과정은 Raspberry pi의 MC를 가정하여 설명한다. 다수의 MC의 시간 동기는 임무가 수행되는 모든 상황에서 중앙의 동기화 서버를 기준으로 제어된다. MC에서 실행되는 시간동기화 임무Lib는 FC와 serial로 연결되어 nCube-MUV를 통해 중계되는 timesync 메시지를 수신하며 FC의 GPS 기반 system time과 MC 간 시간 오차를 계산한다. 계산된 FC 오프셋을 중앙의 동기화 서버에 전송하여 동기화 서버는 GPS와의 시간 오차가 보정된 가상 시간을 설정한다. 이후 동기화 SW를 통해 GPS가 반영된 시간으로 MC와 FC 간 시간 오차를 보정한다.

FC

MC

Serial connection

Telemetry1

GPS1

<Raspberry pi의 MC와 FC의 Serial 연결 및 GPS 연결>

# 임무Lib 및 임무SW 개발

## 임무Lib, 임무SW와 개발도구

임무Lib는 임무 장비와 직접적으로 연결되어 임무 장비를 제어하는 소프트웨어이다. 필수 구현 요소로 임무 장비와 연결하기 위한 인터페이스(Serial, I2C, PWM 등) 연동, 내부적으로 임무SW와 통신하기 위한 MQTT 통신에 대한 내용이 있다. 또한 임무 장비, 사용자 또는 환경에 따라 데이터 모델 정의, 통신 모델 정의 등의 내용을 추가할 수 있다. 임무Lib를 임무SW에서 사용을 위해 개발자가 사전에 개발하여 응용프로그램 형태로 배포해야 한다.

임무SW는 임무Lib를 포함한 형태로 개발되며 외부와 통신하며 임무Lib를 제어하는 소프트웨어이다. 임무SW는 응용프로그램 형태의 임무Lib를 실행하기 위한 스크립트를 정의하는 내용이 필수이며, 또한 외부와 통신하기 위해 통신 모델을 정의하고 데이터 모델을 정의하는 내용을 필수로 한다.

개발자는 개발도구를 통해 임무SW를 좀 더 쉽게 개발하고 배포하여 드론에 탑재까지 가능하다. 개발도구는 블록코딩 형식으로 개발할 수 있으며 Github와 연동하여 이미 배포된 임무Lib와 임무SW를 재사용이 가능하고 개발된 임무SW를 배포할 수 있으며 드론과 임무에 대한 명세를 작성함으로써 자동으로 드론에 탑재와 동작이 가능하다.

## 임무Lib 개발

* 임무를 수행하며 중앙에서 제어되는 모든 드론의 MC와 FC 간 시간 동기를 제어하기 위한 임무Lib를 사전에 개발한다. 임무Lib는 MC에 탑재되어 시간 동기를 실질적으로 제어하며 시간동기화를 자동화하는 역할을 수행한다. 임무Lib에는 아래의 목록이 필수적으로 구현되어야 한다. 개발된 임무Lib는 최종적으로 응용프로그램 형태로 뒤에 설명하는 임무SW에 포함된 형태로 동작한다.
* 임무Lib는 크게 4가지 부분으로 나뉜다.

1. MC와 내부적으로 통신하는 것과 동시에 nCube-MUV의 중계를 통해 FC와 데이터를 송수신하여 FC의 시간 오차를 계산하기 위한 MQTT 통신 예시코드

def on\_connect(client,userdata,flags, rc):

    global monitor

    print('[msw\_mqtt\_connect] connect to ', broker\_ip)

    sub\_container\_name = lib['control'][0]

    control\_topic = '/MUV/control/' + lib['name'] + '/' + sub\_container\_name

    lib\_mqtt\_client.subscribe(control\_topic, 0)

    print ('[lib]control\_topic\n', control\_topic)

    f = fifo()

    mav = ardupilotmega.MAVLink(f)

    # Send timesync

    monitor.tx\_time = dt.timestamp(dt.now())

    m = mav.timesync\_encode(0, int( monitor.tx\_time ))

    m.pack(mav)

    tx\_msg = m.get\_msgbuf()

    client.publish(monitor.topic\_req, tx\_msg)

    print('Time synch is published')

def on\_disconnect(client, userdata, flags, rc=0):

    print(str(rc))

def on\_subscribe(client, userdata, mid, granted\_qos):

    print("subscribed: " + str(mid) + " " + str(granted\_qos))

def on\_message(client, userdata, msg):

    global missionPort

    global monitor

    message = str(msg.payload.decode("utf-8"))

    print('[ ' + msg.topic + ' ] ' + message)

    f = fifo()

    mav = ardupilotmega.MAVLink(f)

    # Time sync message reception

    mavMsg = bytearray.fromhex(" ".join(message[i:i + 2] for i in range(0, len(message), 2)))

    if msg.topic == monitor.topic\_timesync:

        rx\_msg = mav.parse\_char(mavMsg)

        if rx\_msg.tc1 == 0:

            # Send timesync

            monitor.tx\_time = dt.timestamp(dt.now())

            m = mav.timesync\_encode(0, int( monitor.tx\_time ))

            m.pack(mav)

            tx\_msg = m.get\_msgbuf()

            client.publish(monitor.topic\_req, tx\_msg)

        else:

            rx\_time = dt.timestamp(dt.now())

            if monitor.fc\_lt != 0: monitor.fc\_lt = (monitor.fc\_lt + (rx\_time - monitor.tx\_time)/2)/2

            else: monitor.fc\_lt = (rx\_time - monitor.tx\_time) / 2

    else:

        # System time message reception

        print('System time is received')

        rx\_msg = mav.parse\_char(mavMsg)

        now = float( dt.timestamp( dt.now() ) )

        if rx\_msg.time\_unix\_usec is not 0:

            monitor.fc\_time = float( rx\_msg.time\_unix\_usec / 1e6 )

            monitor.fc\_offset = int( ( (monitor.fc\_time + monitor.fc\_lt) - now ) \* 1000 )

            print('Time calculation')

        else:

            monitor.fc\_time = 0

            monitor.fc\_offset = 0

def msw\_mqtt\_connect(broker\_ip, port):

    global lib\_topic

    global lib\_mqtt\_client

    lib\_topic = ''

    lib\_mqtt\_client = mqtt.Client()

    lib\_mqtt\_client.on\_connect = on\_connect

    lib\_mqtt\_client.on\_disconnect = on\_disconnect

    lib\_mqtt\_client.on\_subscribe = on\_subscribe

    lib\_mqtt\_client.on\_message = on\_message

    lib\_mqtt\_client.connect(broker\_ip, port)

    lib\_mqtt\_client.loop\_start()

    return lib\_mqtt\_client

def send\_data\_to\_msw (data\_topic, obj\_data):

    global lib\_mqtt\_client

    lib\_mqtt\_client.publish(data\_topic, obj\_data)

1. MC의 시간 오차를 주기적으로 측정하여 시간동기화를 위한 동기화 SW를 실행하고 계산된 FC의 시간 오차를 중앙 동기화 서버에 전송하는 예시코드

class Monitor(Thing):

    # Initialize

    def \_\_init\_\_(self):

        Thing.\_\_init\_\_(self)

        self.protocol = 'up'

        self.interval = 5

        self.topic = []

        self.topic\_systime = ''

        self.topic\_timesync = ''

        self.topic\_req = ''

        self.name = 'Monitor'

        self.server\_addr = ''

        self.server\_port = ''

        self.trans\_protocol = 'udp'

        self.threshold = 5

        self.ct\_path = ''

        self.tx\_time = []

        self.fc\_lt = 0

        self.fc\_time = 0

        self.fc\_offset = 0

        # client path check

        if os.path.exists('./linux\_client\_x86'):

            self.ct\_path = os.path.abspath('linux\_client\_x86')

        elif os.path.exists('./device/linux\_client\_x86'):

            self.ct\_path = os.path.abspath('./device/linux\_client\_x86')

        else:

            for name in os.listdir('./'):

                if name.find('\_timesync') != -1:

                    if os.path.exists('./' + name + '/linux\_client\_x86'):

                        self.ct\_path = os.path.abspath('./' + name + '/linux\_client\_x86')

                        break

                    elif os.path.exists('./' + name + '/device/linux\_client\_x86'):

                        self.ct\_path = os.path.abspath('./' + name + '/device/linux\_client\_x86')

                        break

        # OS address bit check

        (os\_bit, \_) = platform.architecture()

        if os\_bit == '32bit':

            self.client\_sw = self.ct\_path[:-2] + '86'

        elif os\_bit == '64bit':

            self.client\_sw = self.ct\_path[:-2] + '64'

        print(self.client\_sw)

        # Change of ownership

        subprocess.call(['sudo', 'chmod', '777', self.client\_sw])

    # Thing dependent get function

    def get(self, key):

        if key in self.topic:

            # protocol check

            if self.trans\_protocol == 'tcp':

                self.\_protocol = 1

            elif self.trans\_protocol == 'udp':

                self.\_protocol = 0

            payload = dict()

            Index = True

            while Index:

                # Time offset calculation

                mc\_offset = subprocess.getoutput( self.client\_sw + ' 3 ' + self.server\_addr + ' ' + self.server\_port + ' ' + str(self.\_protocol) )

                data\_temp = mc\_offset.split('+')

                del data\_temp[-1]

                try:

                    payload['server'] = dt.fromtimestamp( float( data\_temp[0] ) ).astimezone(timezone('Asia/Seoul')).strftime('%Y%m%dT%H%M%S%f')[:-3]

                    payload['mc\_time'] = dt.fromtimestamp( float( data\_temp[1] ) ).astimezone(timezone('Asia/Seoul')).strftime('%Y%m%dT%H%M%S%f')[:-3]

                    payload['mc\_offset'] = int( data\_temp[2] )

                    Index = False

                except IndexError:

                    time.sleep(3)

                    continue

            # Check the FC connection

            if self.fc\_offset == 0:

                payload['fc\_time'] = payload['mc\_time']

            else:

                payload['fc\_time'] = dt.fromtimestamp( self.fc\_time ).astimezone(timezone('Asia/Seoul')).strftime('%Y%m%dT%H%M%S%f')[:-3]

payload['fc\_offset'] = int( data\_temp[2] - self.fc\_offset )

            payload = json.dumps(payload, indent=4)

            # Time offset check

            if abs(float(data\_temp[2])) > float(self.threshold):

                # Excute synchronizer

                subprocess.call(['sudo', self.client\_sw, '1', self.server\_addr, self.server\_port, str(self.\_protocol), str(self.threshold)], stdout = subprocess.PIPE, stderr = subprocess.PIPE)

                print('Synchronizer is executed')

            # Return the calculated time offset

            return payload

        else :

            pass

    # Function to measure RTT of the FC link

    def rtt\_measure(self, muv\_tis, sc):

        settings = {

            'SendTerm'       : 4,

            'InitialPacket'  : 15,

            'TransmitPacket' : 5,

            'Hz'             : 0.6,

        }

        f = fifo()

        mav = ardupilotmega.MAVLink(f)

        ADDR = (self.server\_addr, int(self.server\_port))

        count = tmp = 0

        sock = socket(AF\_INET, SOCK\_DGRAM)

        start = time.time()

        initial = 0

        while True:

            try:

                # 2hz

                time.sleep(settings['Hz'])

                # Send timesync

                self.tx\_time = dt.timestamp(dt.now())

                m = mav.timesync\_encode(0, int( self.tx\_time ))

                m.pack(mav)

                tx\_msg = m.get\_msgbuf()

                sc.publish(self.topic\_req, tx\_msg)

                # send ms measure

                count = count + 1

                tmp = tmp + (self.fc\_offset / settings['TransmitPacket'])

                if count is settings['TransmitPacket']:

                    enteredTime = time.time() - start

                    if settings['SendTerm'] - enteredTime >= 0:

                        time.sleep(settings['SendTerm'] - enteredTime)

                    if tmp is not 0:

                        if initial < settings['InitialPacket']:

                            sock.sendto(str(tmp).encode(), ADDR)

                            initial = initial + 1

                        elif abs(tmp) <= 120000:

                            sock.sendto(str(tmp).encode(), ADDR)

                    count = 0

                    tmp = 0

                    # startTime initialization

                    start = time.time()

            except KeyboardInterrupt:

                muv\_tis.join()

                sc.close()

                return 0

1. 시간동기화의 상황을 관측하기 위해 계산된 시간 오차를 데이터 프로토콜에 맞게 변환하여 Mobius로 전송하는 예시코드

class MUV\_TIS(Thread):

    # Initialize

    def \_\_init\_\_(self, thing, sc, rf\_sc = None):

        Thread.\_\_init\_\_(self)

        self.thing = thing

        self.sc = sc

        self.rf\_sc = rf\_sc

    def run(self):

        # ================== communication and routine ======================

        try:

            if self.rf\_sc == None: conn = self.sc

            else: conn = self.rf\_sc.ser

            # Creation of threads

            thr = MUV\_up(conn, self.thing)

            thr.start()

            # FC thread

            FC\_thread = threading.Thread(target = self.thing.rtt\_measure(thr, self.sc))

            FC\_thread.start()

        except KeyboardInterrupt:

            self.sc.close()

class MUV\_up(Thread):

    def \_\_init\_\_(self, rf\_sc, thing):

        Thread.\_\_init\_\_(self)

        self.rf\_sc = rf\_sc

        self.thing = thing

        self.topic = thing.topic

        self.interval = thing.interval

        print(thing.name + ' interface is started')

    def run(self):

        # Data transmission

        while True:

            try:

                if len(self.topic) == 0:

                    break

                # If uplink array is vacant

                if not len(self.topic):

                    break

                # Update delay (second)

                time.sleep(self.interval)

                # Sending sensor values

                payload = self.thing.read(self.topic)

                print(payload)

                self.rf\_sc.publish(self.thing.topic, payload)

            except KeyboardInterrupt:

                self.rf\_sc.close()

class Thing:

    # Initialize value object

    def \_\_init\_\_(self, interval = 0.5):

        self.interval = interval

        self.tag = []

        self.name = 'Thing'

    # Formation function for sensor values

    def encode(self, con, data):

        payload = '{"ctname":"%s","con":"%s"}<EOF>' % (con, data)

        return payload

    # Function to read payload data

    def read(self, key):

        data = self.get(key)

        payload = self.encode(key, data)

        return data

1. 임무 Lib가 시작되는 메인 코드 예시

def main():

    os.system('sudo timedatectl set-ntp off')

    my\_lib\_name = 'lib\_timesync'

    msw\_dir\_name = 'msw\_' + my\_lib\_name.split('\_')[1] + '\_' + 'msw\_' + my\_lib\_name.split('\_')[1]

    try:

        lib = dict()

        print('./' + msw\_dir\_name + '/' + my\_lib\_name + '.json')

        with open('./' + msw\_dir\_name + '/' + my\_lib\_name + '.json', 'r') as f:

            lib = json.load(f)

    except:

        lib = dict()

        lib["name"] = my\_lib\_name

        lib["target"] = 'armv6'

        lib["description"] = "[name] [server ip] [interval] [protocol] [threshold] [server port]"

        lib["scripts"] = "./lib\_timesync 203.253.128.177 5 udp 5 5005"

        lib["data"] = ["TimeSync", "Req"]

        lib["control"] = ["system\_time", "timesync"]

        lib = json.dumps(lib, indent=4)

        lib = json.loads(lib)

        with open('./' + msw\_dir\_name + '/' + my\_lib\_name + '.json', 'w', encoding='utf-8') as json\_file:

            json.dump(lib, json\_file, indent=4)

    broker\_ip = 'localhost'

    port = 1883

    # Inforamtion for time server

    monitor = Monitor()

    '''

    예시: argv[n] = [파라미터 = default value]

    argv[1] = [서버주소 = keti 서버]

    argv[2] = [업로드 주기 = 3초]

    argv[3] = [소켓 프로토콜 = udp]

    argv[4] = [동기화 문턱 값 = 5ms]

    argv[5] = [동기화 port = 5005]

    '''

    # parsing input parameters

    argv = sys.argv

    if len(argv) > 2:

        input\_par = argv

        del input\_par[0]

    else:

        input\_par = argv[1].split(' ')

    print(input\_par)

    if len(input\_par) < 2: monitor.server\_addr = '203.253.128.177'

    else : monitor.server\_addr = input\_par[0]

    if len(input\_par) < 3: monitor.interval = 3

    else : monitor.interval = int( input\_par[1] )

    if len(input\_par) < 4: monitor.trans\_protocol = 'udp'

    else : monitor.trans\_protocol = input\_par[2]

    if len(input\_par) < 5: monitor.threshold = 5

    else : monitor.threshold = int( input\_par[3] )

    if len(input\_par) < 6: monitor.server\_port = '5005'

    else : monitor.server\_port = input\_par[4]

    # Define resource

    container\_name = lib["data"][0]

    monitor.topic = '/MUV/data/' + lib["name"] + '/' + container\_name

    monitor.topic\_req = '/MUV/data/' + lib["name"] + '/' + lib["data"][1]

    monitor.topic\_systime = '/MUV/control/' + lib["name"] + '/' + lib["control"][0]

    monitor.topic\_timesync = '/MUV/control/' + lib["name"] + '/' + lib["control"][1]

    # TAS thread

    msw\_mqtt\_connect(broker\_ip, port)

    lib\_mqtt\_client.subscribe(monitor.topic\_timesync)

    lib\_mqtt\_client.subscribe(monitor.topic\_systime)

    monitor\_tis = MUV\_TIS(monitor, lib\_mqtt\_client).start()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

* 임무Lib는 임무SW와 연동을 위해 실행파일 형태로 배포한다. Python으로 개발된 시간동기화 임무Lib는 아래의 명령어를 통해 응용프로그램 형태로 만들 수 있다.

$ python3 –m PyInstaller –F lib\_timesync.py

* Python으로 개발되어 응용프로그램 형태로 배포된 시간동기화 임무Lib는 아래 Github 주소에서 확인할 수 있다.

<https://github.com/IoTKETI/lib_timesync.git>

## 임무SW 개발

* + 1. MUV 개발도구
* 개발자가 임무SW를 손쉽게 개발하고 배포하여 MC에 자동으로 탑재까지 가능하도록 지원하는 블록코딩 형태의 웹 기반 지원도구이다.
* <http://203.253.128.177:7505> 주소로 접속하여 로그인하여 사용 가능하다.
* 코딩을 소스코드를 작성하는 형태가 아닌 블록코딩을 통해 쉽게 개발할 수 있으며 Github와 연동하여 업로드가 가능하고 approval이라는 명세를 입력함으로써 드론에 탑재가 가능하다.
  + 1. 임무 S/W 개발과정

1. 아래 사진과 같이 임무SW의 기본이 되는 형태까지 블록으로 코딩한다. 이 구조는 임무SW도 동일한 구조를 가진다.

텍스트이(가) 표시된 사진

자동 생성된 설명

1. 사전에 개발해서 배포한 임무Lib를 호출한다.

* 임무Lib 이름: 사전에 응용 프로그램 형태로 배포한 임무Lib의 이름
* 임무Lib 저장소: Github에 배포된 임무Lib를 가져오기 위한 주소

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명

* 임무Lib: lib\_timesync
* 배포된 저장소 주소:   
  https://github.com/IoTKETI/lib\_timesync

1. 생성된 임무Lib 블록 추가 및 임무SW의 이름 입력

테이블이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

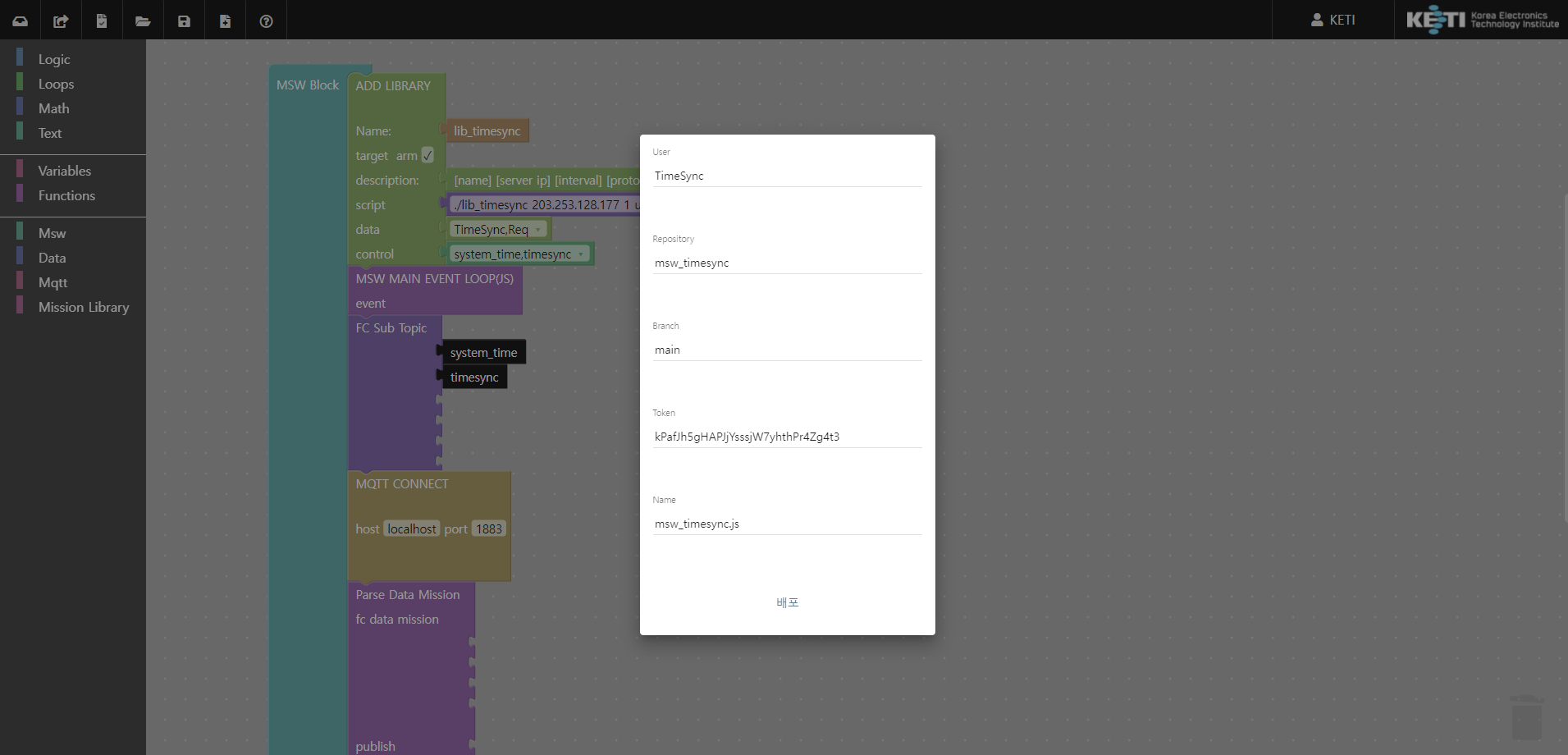
자동 생성된 설명

1. 블록으로 코딩한 임무SW를 배포하기 전 소스코드 형태로 확인

텍스트이(가) 표시된 사진

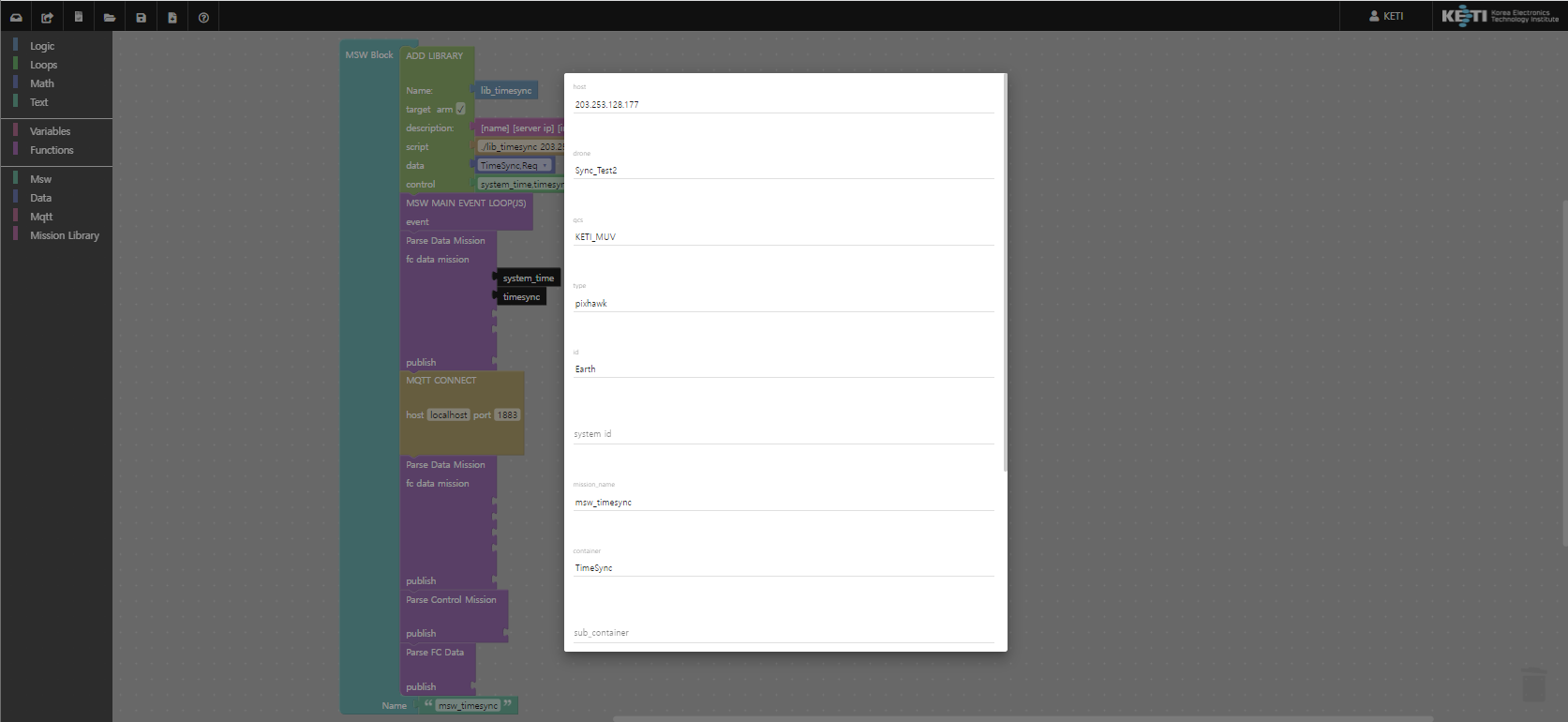
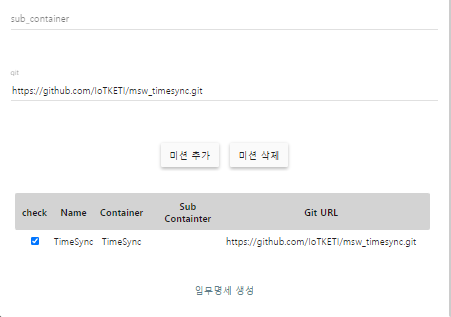
자동 생성된 설명

1. 임무SW 배포에 필요한 정보 입력하여 배포

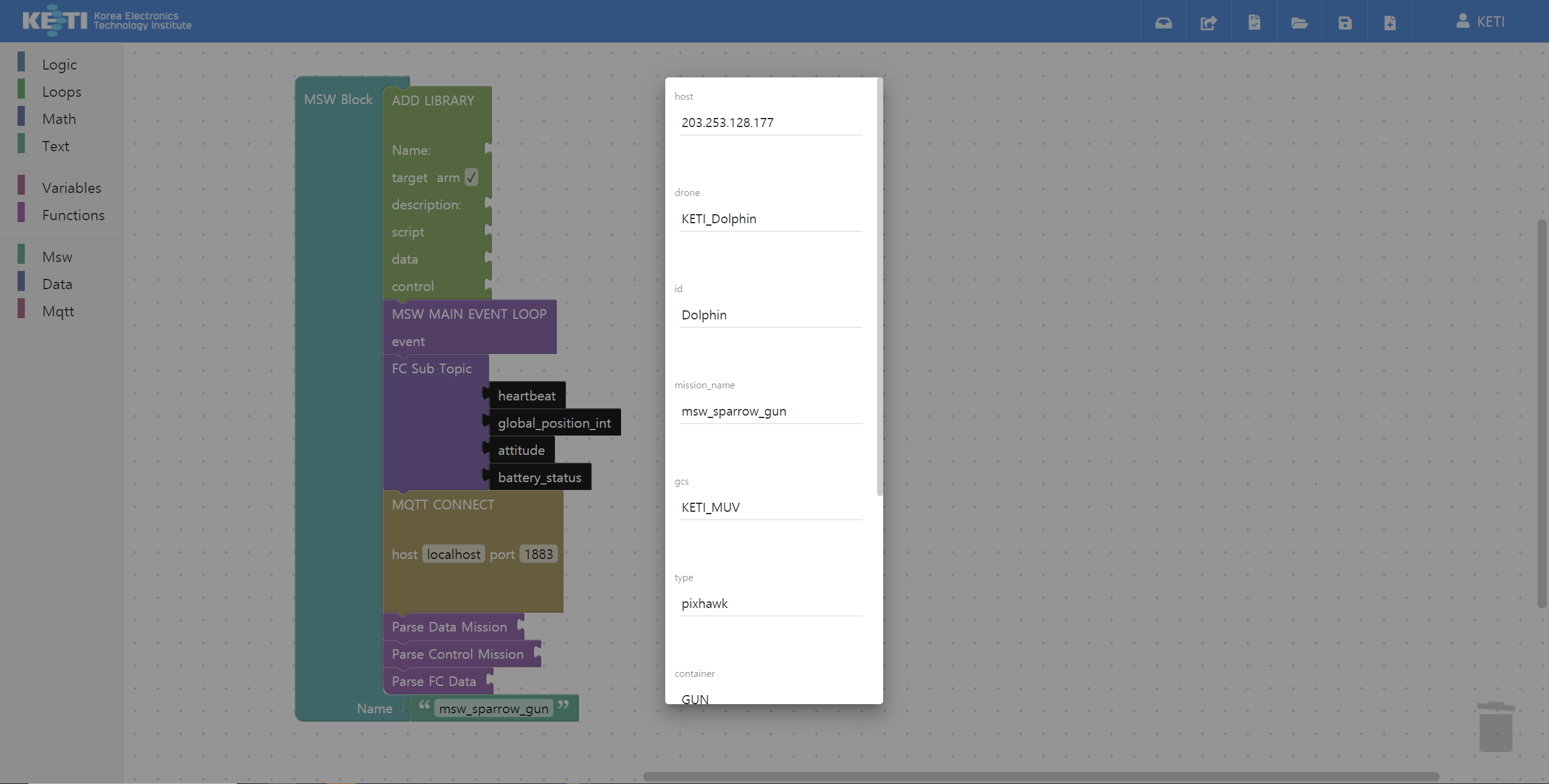
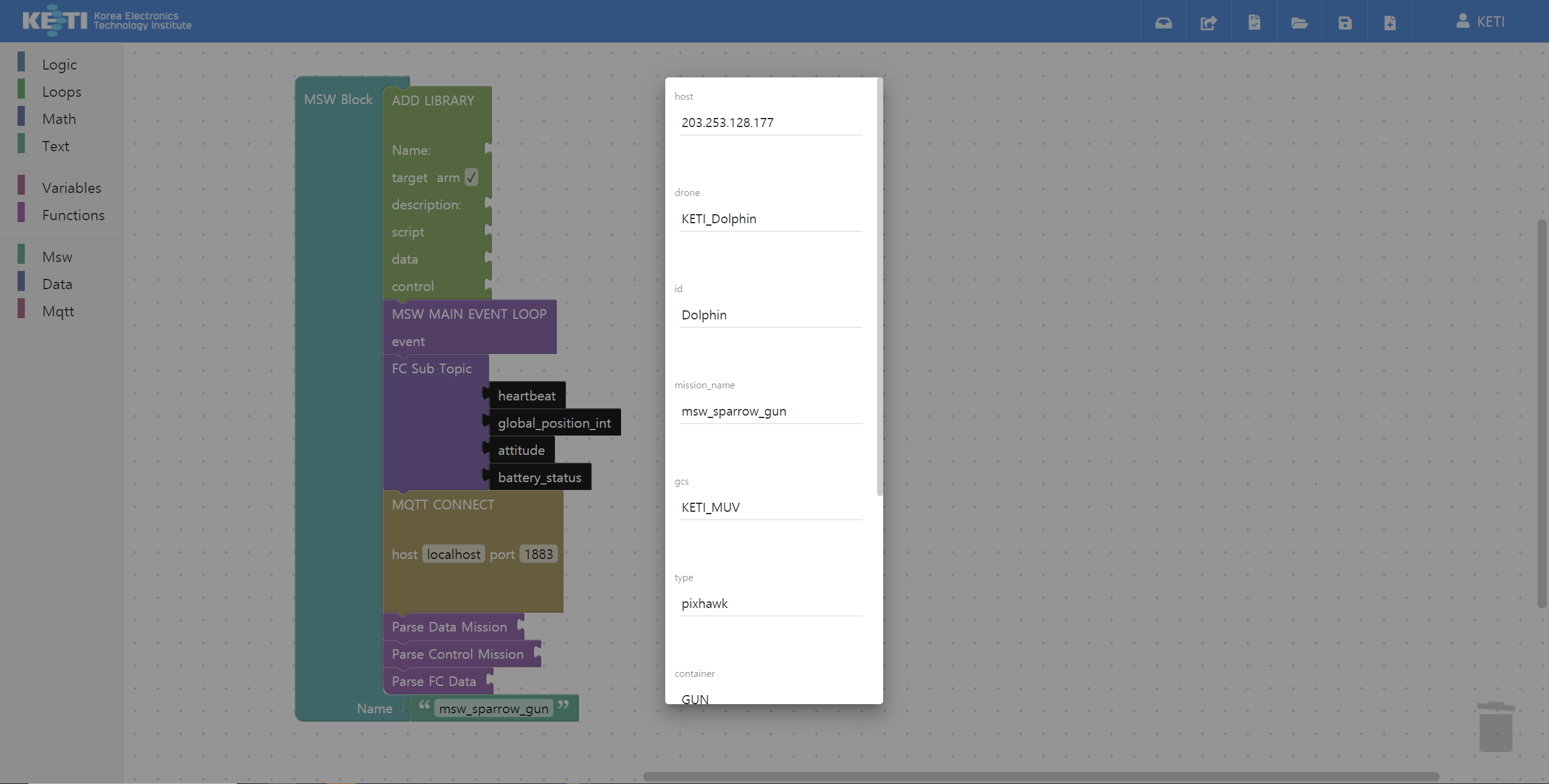
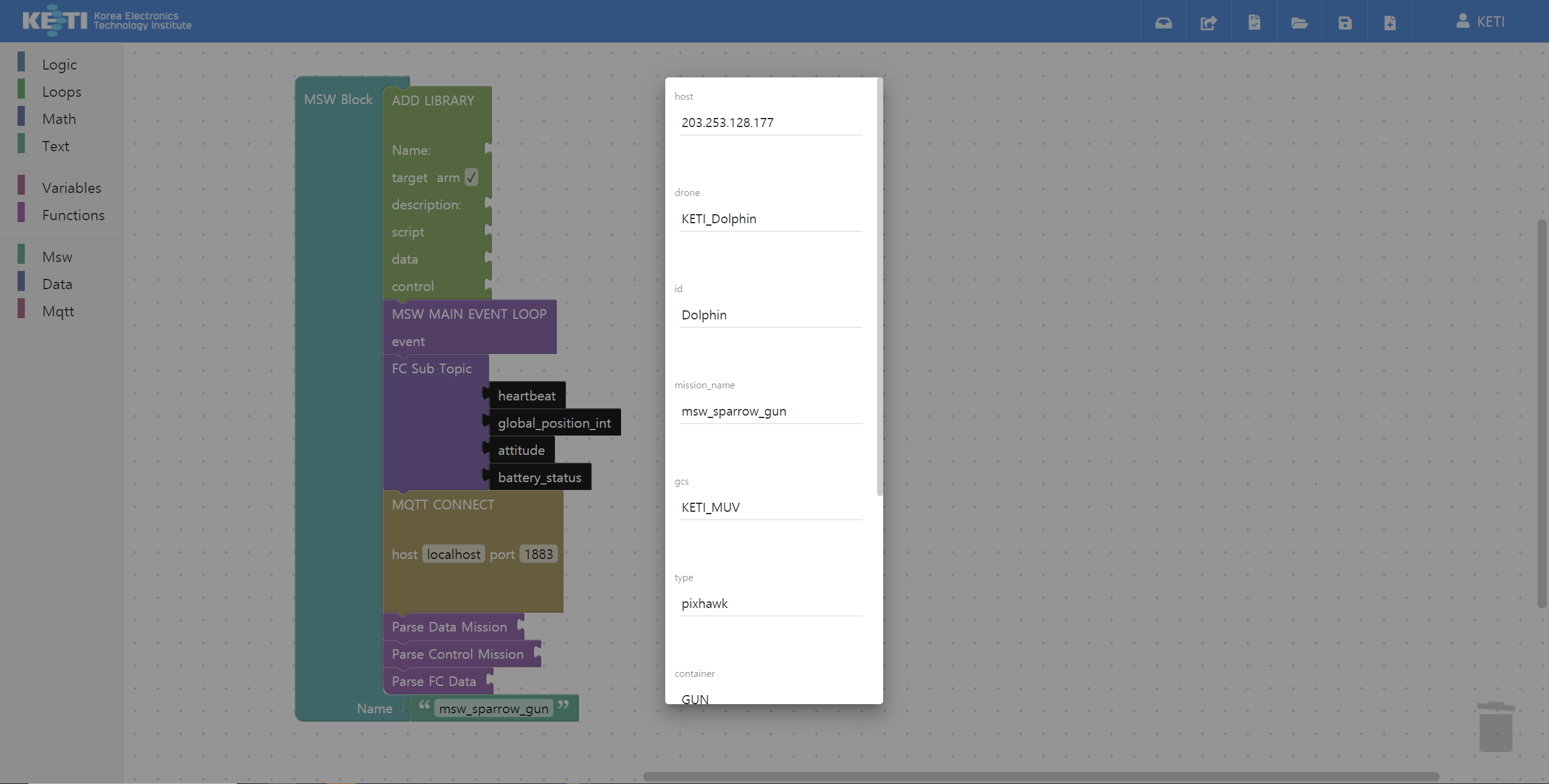
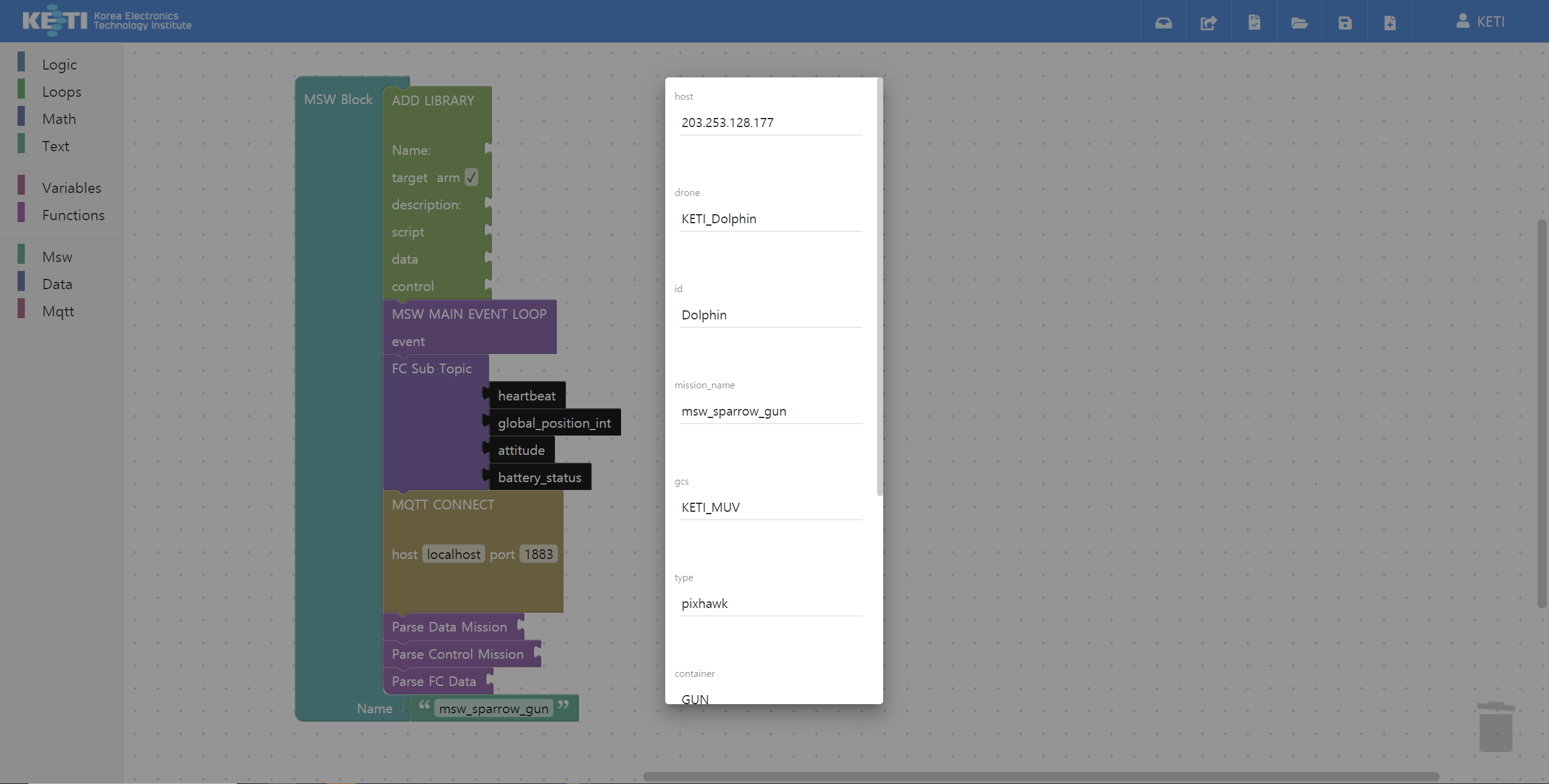
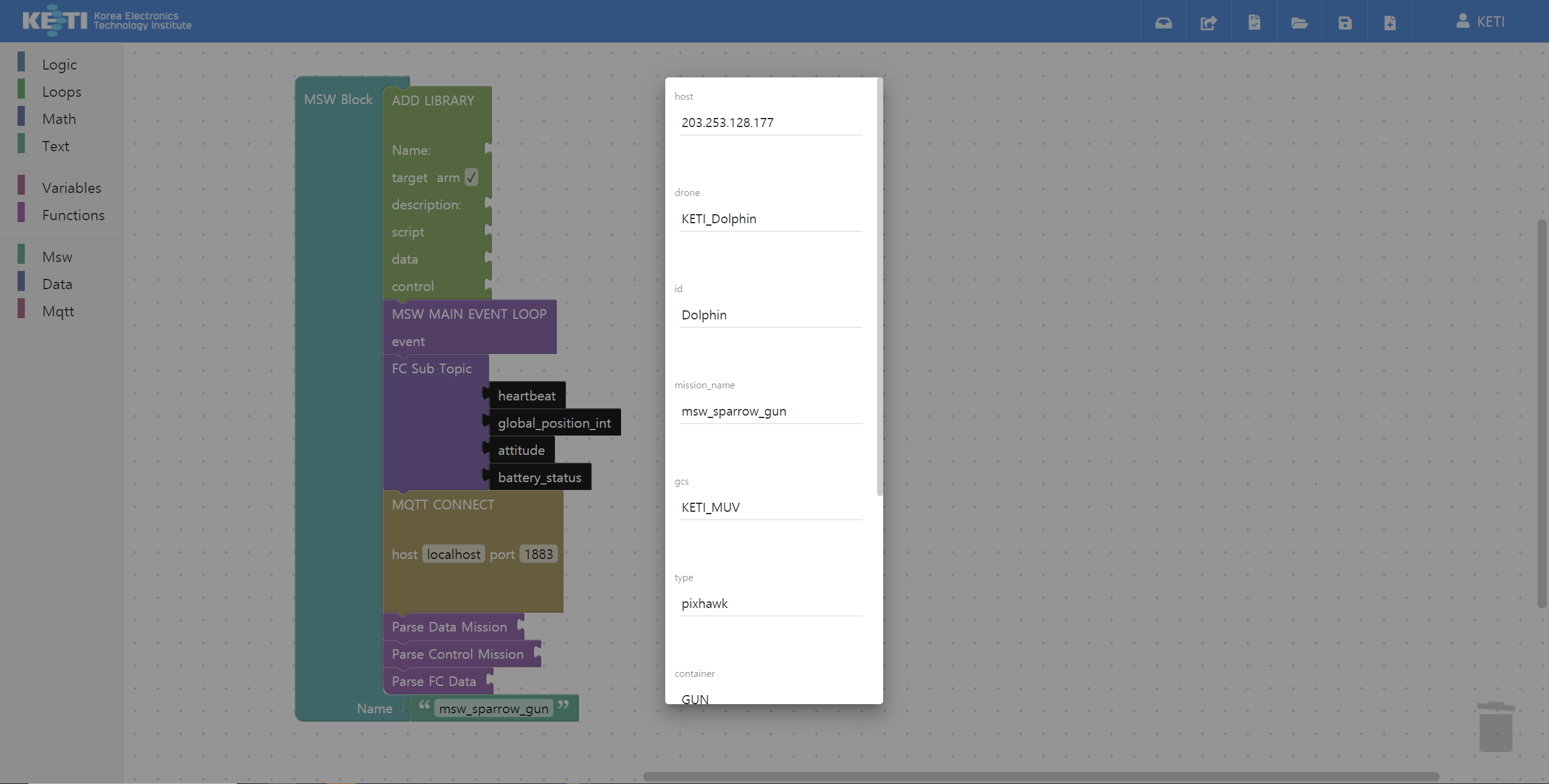


* Github에 배포할 사용자 이름
* 배포할 저장소 이름
* Github에 생성할 Branch
* 계정 접근을 위한 Github 토큰
* 배포할 코드의 파일 이름

1. 드론 탑재를 위한 드론 명세 작성

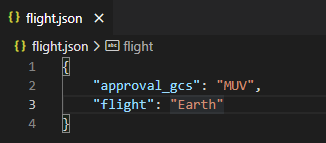


* 중앙서버 주소
* 드론 이름
* 드론 데이터 저장되는 AE 이름
* 드론 타입
* 드론 타입
* System ID
* 임무SW 이름
* 임무 데이터가 업로드 되는 컨테이너 이름
* 사용자가 명령을 전달할 컨테이너 이름
* Github에 배포된 임무SW 저장소 주소
* 미션 추가 후 체크, 임무명세 생성

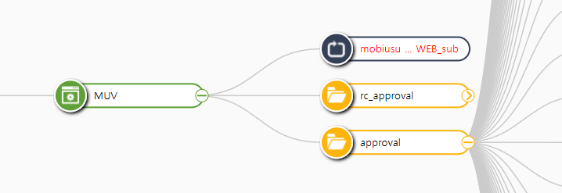
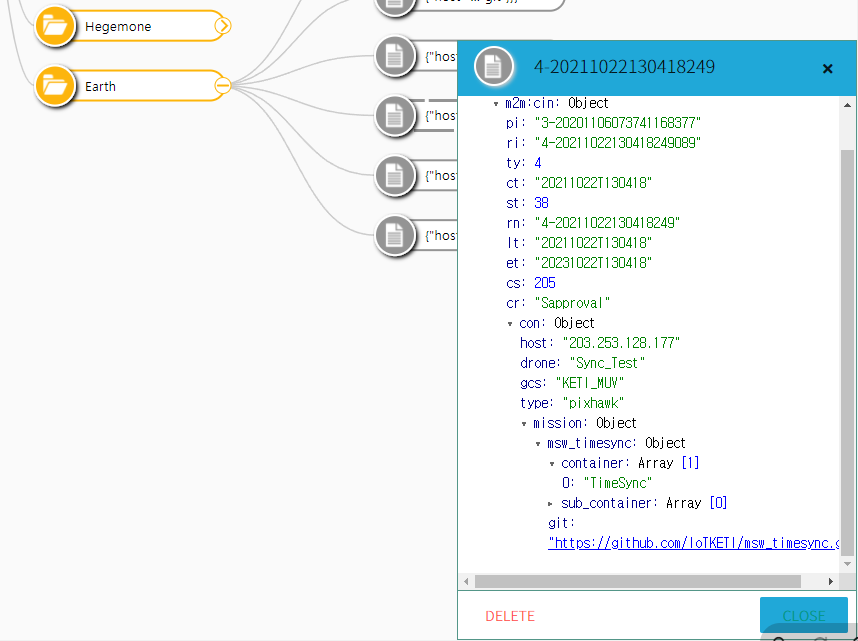


1. MC의 nCube-MUV 폴더에서 filght.json 정보 수정 (nCube-MUV 가이드 참고)

* flight에 ⑥의 과정에서 업로드한 명세의 드론 ID를 추가한다



1. 명세 업로드 확인



1. 시간동기화 오프셋 확인

* /Mobius/{GCS 이름}/Mission\_Data/{드론 이름}/msw\_timesync/**TimeSync** 컨테이너 아래에 새로 갱신되는 Content Instance의 con값에 서버, MC 및 FC의 시간과 MC 및 FC의 오프셋 값을 확인할 수 있다.
* 시간동기화 알고리즘이 정상적으로 작동한다면 MC 및 FC의 오프셋이 요구되는 5ms 안의 시간 오차 안으로 동기화가 진행된다.

