

# 임무 소프트웨어 Guide

LGU LTE 품질측정

msw\_lgu\_lte

# 목 차

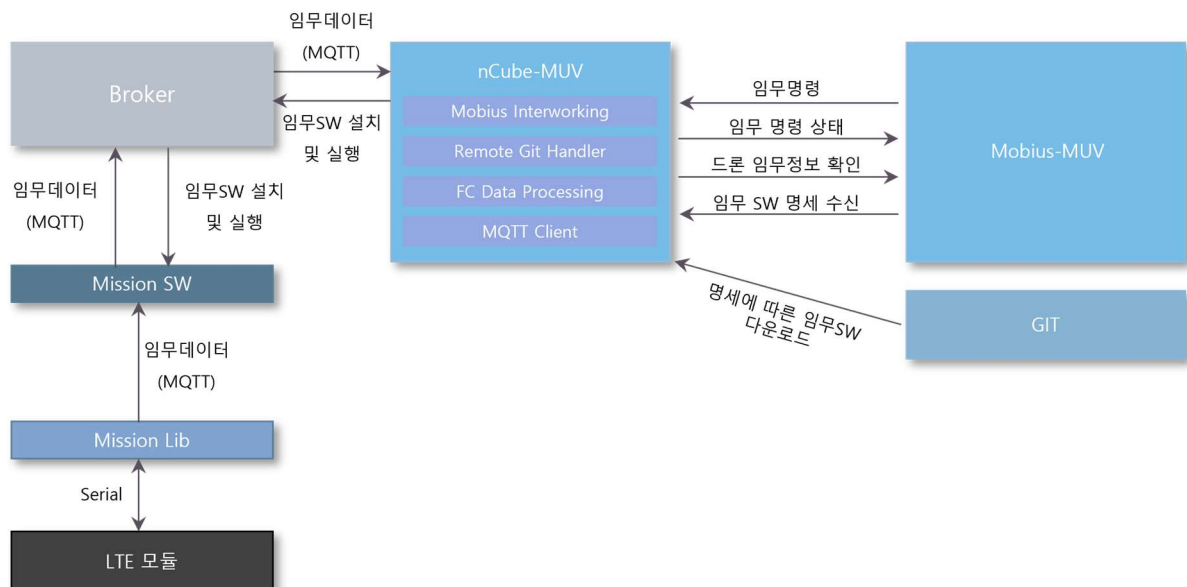
1	개요 .....	3
2	LTE 모듈 .....	4
2.1	LTE 모듈이란? .....	4
3	임무Lib 및 임무SW 개발 .....	5
3.1	임무Lib, 임무SW와 개발도구 .....	5
3.2	임무Lib 개발 .....	5
3.3	임무SW 개발 .....	12

## 1 개요

이 문서는 임무 컴퓨터(Mission Computer, MC)에 탑재된 LTE 모듈을 통해 LTE 통신 품질을 측정하는 임무 소프트웨어에 대해 설명하고 임무 소프트웨어를 개발하는 방법에 대해 설명하는 문서이다. 이 문서에서는 LGU 통신사의 USIM을 사용하여 LGU 통신망의 LTE를 사용한다. LTE 모듈은 MC에 탑재되어 보드 내부에서 Serial로 연결되어 LTE 데이터를 수신하고 임무 소프트웨어는 이 데이터를 중앙으로 전송하여 GCS, LTE 커버리지 맵 등 다양한 곳에 사용이 가능하다.

사용자는 MC를 드론에 탑재하여 아래의 아키텍처와 같이 nCube-MUV와 임무 소프트웨어를 동작하여 드론의 FC 데이터와 LTE 모듈의 LTE 데이터를 통해 어떤 위치에서 LTE 품질이 어떤지 알 수 있다. Serial 통신을 통해 MC로 전달된 LTE 데이터를 데이터별로 파싱하여 FC의 위치 데이터와 함께 중앙으로 업로드 한다. 중앙에 업로드 된 LTE 데이터는 다양한 곳에 사용이 가능하다.

이를 위해 LGU LTE 임무에 대해 설명하고 LTE 모듈과 직접적으로 통신하는 임무라이브러리(Library, Lib), 임무 Lib를 포함한 형태로 nCube-MUV와 연동되어 외부와 통신하며 임무 Lib에서 데이터를 전달받는 임무 소프트웨어(Software, SW)에 대해 설명하고 동작하기 위한 방법을 설명한다.



< LGU LTE 측정 임무 소프트웨어(msw\_lgu\_lte) 아키텍처 >

## 2 LTE 모듈

### 2.1 LTE 모듈이란?

LTE 모듈은 MC에 탑재되어 LTE 통신이 가능하도록 지원하고 LTE 네트워크의 신호 레벨 및 품질의 측정값을 수집하는 모듈로 모뎀, AP, 메모리, RF 등의 부품이 포함되어 있는 하나의 하드웨어로 구성된 제품이다.

LTE 모듈은 MC 내부 회로로 MC와 Serial로 연결되고 MC의 USIM 슬롯에 통신사의 USIM을 장착하여 사용한다. 사용자는 MC에서 LTE 모듈과 연결된 Serial을 통해 통신사에서 제공하는 수신신호강도(Received Signal Strength Indicator, RSSI), LTE셀 네트워크에서 수신된 전력레벨-Reference Signal Received Power(RSRP), Reference Signal Received Quality(RSRQ) 등의 무선환경 지표를 확인할 수 있으며 이 지표들로 현재 LTE 네트워크의 전반적인 상태를 확인할 수 있다.

이 임무SW에서는 MC가 드론에 탑재되어 MC 보드가 중앙과 통신하기 위한 네트워크를 지원하고 드론의 위치에 따라 LTE 품질을 측정하는데 사용된다. 측정된 LTE 품질 데이터는 LTE 커버리지 맵을 그리는데 사용된다.



< MC에 LTE모듈이 장착된 모습 >

### 3 임무Lib 및 임무SW 개발

#### 3.1 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 임무 SW 동작이 가능하다.

#### 3.2 임무Lib 개발

LGU LTE 데이터를 수신하기 위해 LTE 모듈과 직접적으로 연결되어 LTE 데이터를 수신할 수 있는 임무Lib를 사전에 개발한다. 임무Lib는 MC에 탑재되어 LTE 모듈과 직접적인 연결 및 LTE 데이터를 파싱하는 역할을 수행한다. 임무Lib에는 아래의 목록이 필수적으로 구현되어야 한다. 개발된 임무Lib는 최종적으로 응용프로그램 형태로 뒤에 설명하는 임무SW에 포함된 형태로 동작한다.

- 임무Lib는 크게 3가지 부분으로 나뉜다.

① MC와 내부적으로 데이터 통신을 위한 MQTT 통신 예시코드

```
def msw_mqtt_connect(broker_ip, port):
    global lib_mqtt_client

    lib_mqtt_client = mqtt.Client()
    lib_mqtt_client.on_connect = on_connect
    lib_mqtt_client.on_disconnect = on_disconnect
    lib_mqtt_client.on_message = on_message
    lib_mqtt_client.connect(broker_ip, port)
    lib_mqtt_client.loop_start()
    return lib_mqtt_client

def on_connect(client,userdata,flags, rc):
    if rc == 0:
        print('[msw_mqtt_connect] connect to ', broker_ip)
    else:
        print("Bad connection Returned code=", rc)

def on_disconnect(client, userdata, flags, rc=0):
    print(str(rc))

def on_message(client, userdata, msg):
    print(str(msg.payload.decode("utf-8")))

if __name__ == '__main__':
    global missionPort

    my_lib_name = 'lib_LGU_lte'

    try:
        lib = dict()
        with open(my_lib_name + '.json', 'r') as f:
            lib = json.load(f)
            lib = json.loads(lib)

    except:
        lib = dict()
        lib["name"] = my_lib_name
        lib["target"] = 'armv6'
        lib["description"] = "[name] [portnum] [baudrate]"
        lib["scripts"] = './' + my_lib_name + ' /dev/ttyUSB1 115200'
        lib["data"] = ['LTE']
        lib["control"] = []
        lib = json.dumps(lib, indent=4)
        lib = json.loads(lib)

        with open('./' + my_lib_name + '.json', 'w', encoding='utf-8') as json_file:
            json.dump(lib, json_file, indent=4)
```

```
lib['serialPortNum'] = argv[1]
lib['serialBaudrate'] = argv[2]

broker_ip = 'localhost'
port = 1883

msw_mqtt_connect(broker_ip, port)

missionPort = None
missionPortNum = lib["serialPortNum"]
missionBaudrate = lib["serialBaudrate"]
missionPortOpening(missionPortNum, missionBaudrate)

while True:
    missionPortData()
```

## ② LTE 모듈과 직접적으로 연결하기 위한 인터페이스(Serial) 연동 예시코드

```
def missionPortOpening(missionPortNum, missionBaudrate):
    global missionPort
    global lteQ
    global lib

    if (missionPort == None):
        try:
            missionPort = serial.Serial(missionPortNum, missionBaudrate, timeout = 2)
            print ('missionPort open. ' + missionPortNum + ' Data rate: ' + missionBaudrate)

        except TypeError as e:
            missionPortClose()
    else:
        if (missionPort.is_open == False):
            missionPortOpen()

        data_topic = '/MUV/data/' + lib["name"] + '/' + lib["data"][0]
        send_data_to_msw(data_topic, lteQ)

def missionPortOpen():
    global missionPort
    print('missionPort open!')
    missionPort.open()

def missionPortClose():
    global missionPort
    print('missionPort closed!')
    missionPort.close()

def missionPortError(err):
    print('[missionPort error]: ', err)
    os.kill(i_pid, signal.SIGKILL)

def lteReqGetRssi():
    global missionPort

    if missionPort is not None:
        if missionPort.is_open:
            atcmd = b'AT@DBG\r'
            missionPort.write(atcmd)

def send_data_to_msw (data_topic, obj_data):
    global lib_mqtt_client

    lib_mqtt_client.publish(data_topic, obj_data)
```



### ③ LGU LTE 데이터를 파싱하기 위한 예시코드

```
def lteQ_init():
    global lteQ
    lteQ['frequency'] = 0
    lteQ['band'] = 0
    lteQ['bandwidth'] = 0
    lteQ['cell_id'] = ""
    lteQ['rsrp'] = 0.0
    lteQ['rssi'] = 0.0
    lteQ['rsrq'] = 0.0
    lteQ['bler'] = 0.0
    lteQ['tx_power'] = 0
    lteQ['plmn'] = ""
    lteQ['tac'] = 0
    lteQ['drx'] = 0
    lteQ['emm_state'] = ""
    lteQ['rrc_state'] = ""
    lteQ['net_op_mode'] = ""
    lteQ['emm_cause'] = 0
    lteQ['esm_cause'] = ""

def missionPortData():
    global missionPort
    global lteQ

    try:
        lteReqGetRssi()
        missionStr = missionPort.readlines()

        end_data = (missionStr[-1].decode('utf-8'))[:-2]

        if (end_data == 'OK'):
            arrLTEQ = missionStr[1].decode("utf-8").split(", ")

            for idx in range(len(arrLTEQ)):
                arrQValue = arrLTEQ[idx].split(':')
                if (arrQValue[0] == '@DBG'):
                    lteQ['frequency'] = int(arrQValue[2])
                elif (arrQValue[0] == 'Band'):
                    lteQ['band'] = int(arrQValue[1])
                elif (arrQValue[0] == 'BW'):
                    lteQ['bandwidth'] = int(arrQValue[1][:3])
                elif (arrQValue[0] == 'Cell ID'):
                    lteQ['cell_id'] = arrQValue[1]
                elif (arrQValue[0] == 'RSRP'):
                    lteQ['rsrp'] = float(arrQValue[1][:3])
```

```

        elif (arrQValue[0] == 'RSSI'):
            lteQ['rssi'] = float(arrQValue[1][: -3])
        elif (arrQValue[0] == 'RSRQ'):
            lteQ['rsrq'] = float(arrQValue[1][: -2])
        elif (arrQValue[0] == 'BLER'):
            lteQ['bler'] = float(arrQValue[1][: -2])
        elif (arrQValue[0] == 'Tx Power'):
            lteQ['tx_power'] = int(arrQValue[1])
        elif (arrQValue[0] == 'PLMN'):
            lteQ['plmn'] = arrQValue[1]
        elif (arrQValue[0] == 'TAC'):
            lteQ['tac'] = int(arrQValue[1])
        elif (arrQValue[0] == 'DRX cycle length'):
            lteQ['drx'] = int(arrQValue[1])
        elif (arrQValue[0] == 'EMM state'):
            lteQ['emm_state'] = arrQValue[1]
        elif (arrQValue[0] == 'RRC state'):
            lteQ['rrc_state'] = arrQValue[1]
        elif (arrQValue[0] == 'Net OP Mode'):
            lteQ['net_op_mode'] = arrQValue[1]
        elif (arrQValue[0] == 'EMM Cause'):
            lteQ['emm_cause'] = int(arrQValue[1])
        elif (arrQValue[0] == 'ESM Cause'):
            lteQ['esm_cause'] = arrQValue[1].split(",")[0]
    else:
        pass

    data_topic = '/MUV/data/' + lib["name"] + '/' + lib["data"][0]
    lteQ = json.dumps(lteQ)

    send_data_to_msw(data_topic, lteQ)

    lteQ = json.loads(lteQ)

except (TypeError, ValueError):
    lteQ_init()

except serial.SerialException as e:
    missionPortError(e)

```

- 임무Lib는 임무SW와 연동을 위해 실행파일 형태로 배포한다. 파이썬으로 개발된 LGU LTE 측정 임무Lib는 아래의 명령어를 통해 응용프로그램 형태로 만들 수 있다.

```
$ python3 -m PyInstaller -F lib_lgu_lte.py
```

- 파이썬으로 개발되어 응용프로그램 형태로 배포된 LTE 측정 임무Lib는 아래 Github 주소에서 확인할 수 있다.

[https://github.com/loTKETI/lib\\_lgu\\_lte.git](https://github.com/loTKETI/lib_lgu_lte.git)

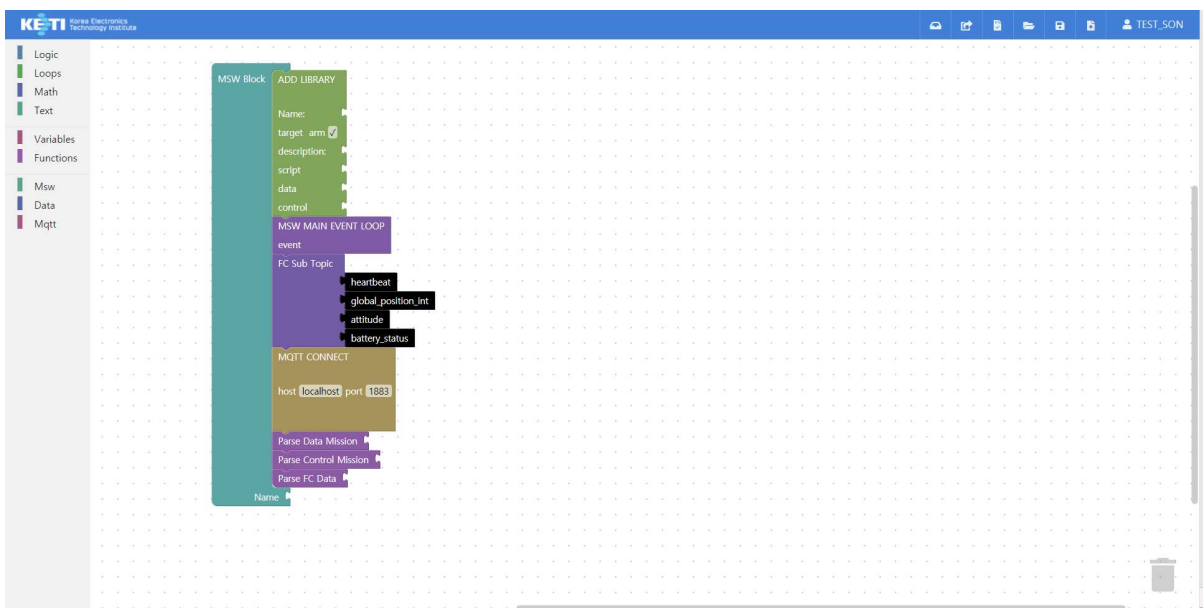
### 3.3 임무SW 개발

#### 3.3.1 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.

#### 3.3.2 임무 S/W 개발과정

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

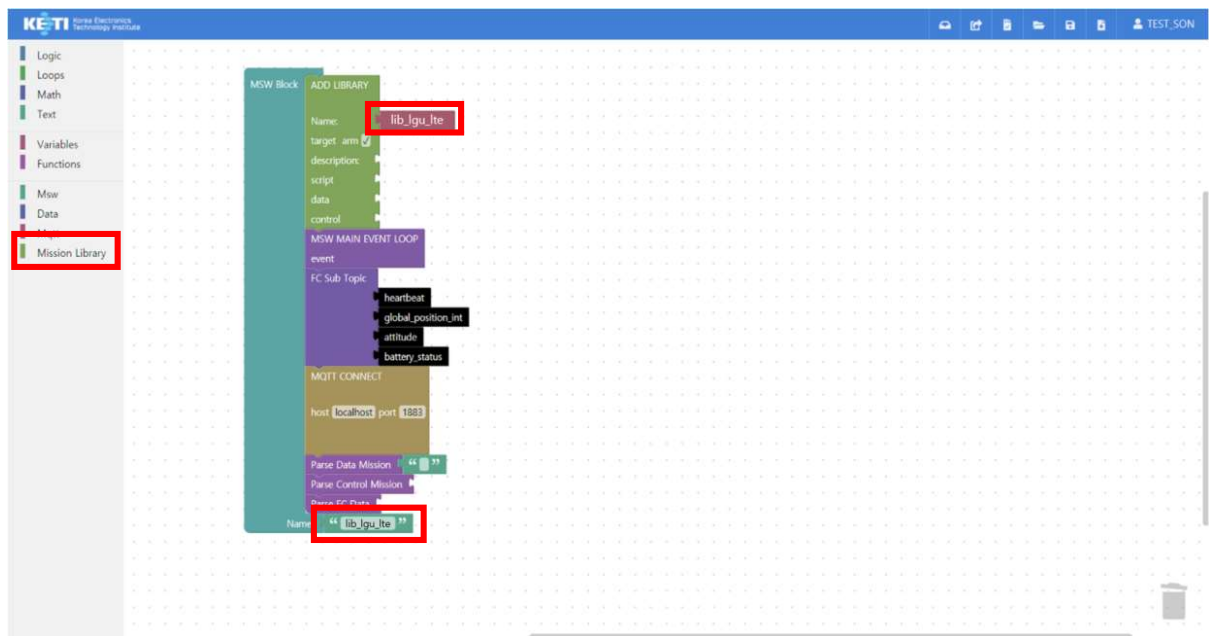


② 사전에 개발해서 배포한 임무Lib를 호출한다.

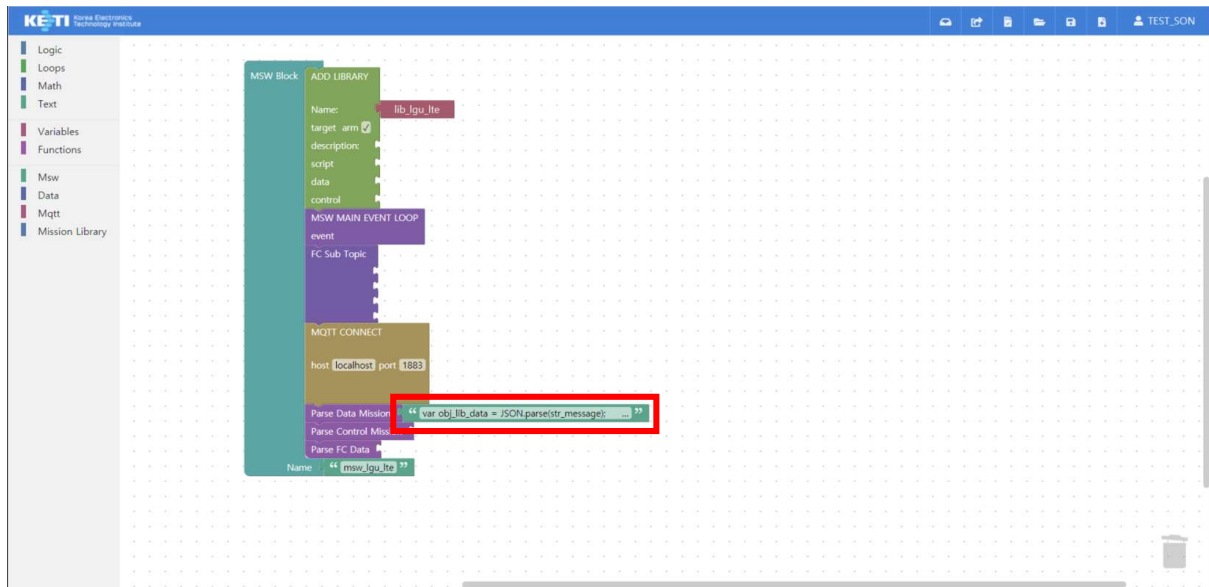
- ✓ 임무Lib 이름: 사전에 응용 프로그램 형태로 배포한 임무Lib의 이름
- ✓ 임무Lib 저장소: Github에 배포된 임무Lib를 가져오기 위한 주소



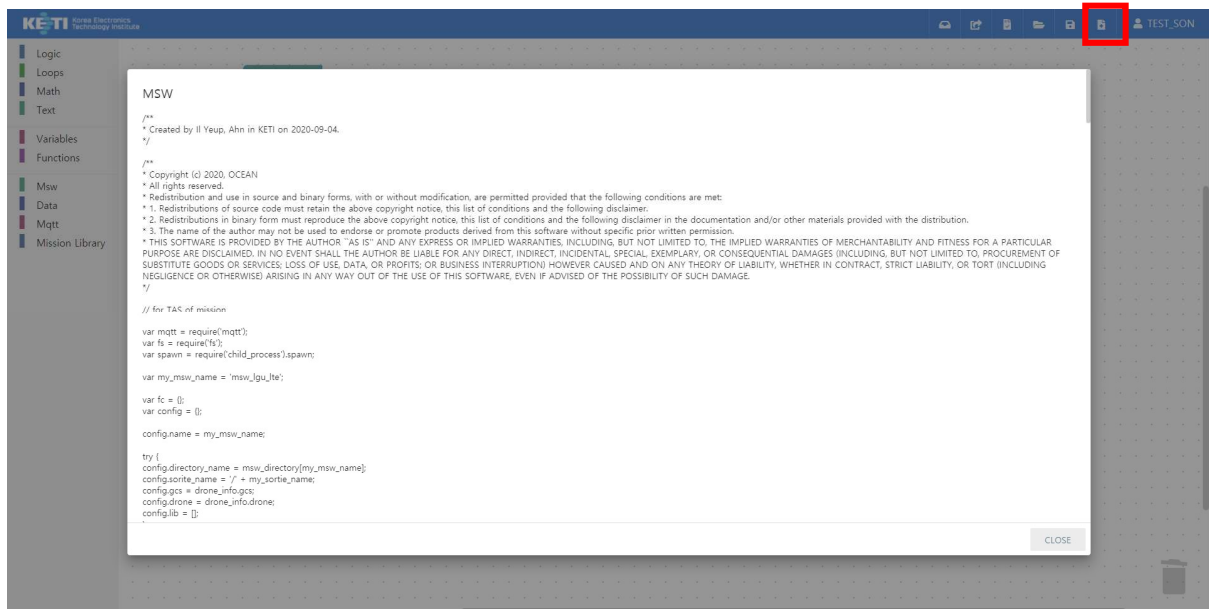
③ 생성된 임무Lib 블록 추가 및 임무SW의 이름 입력



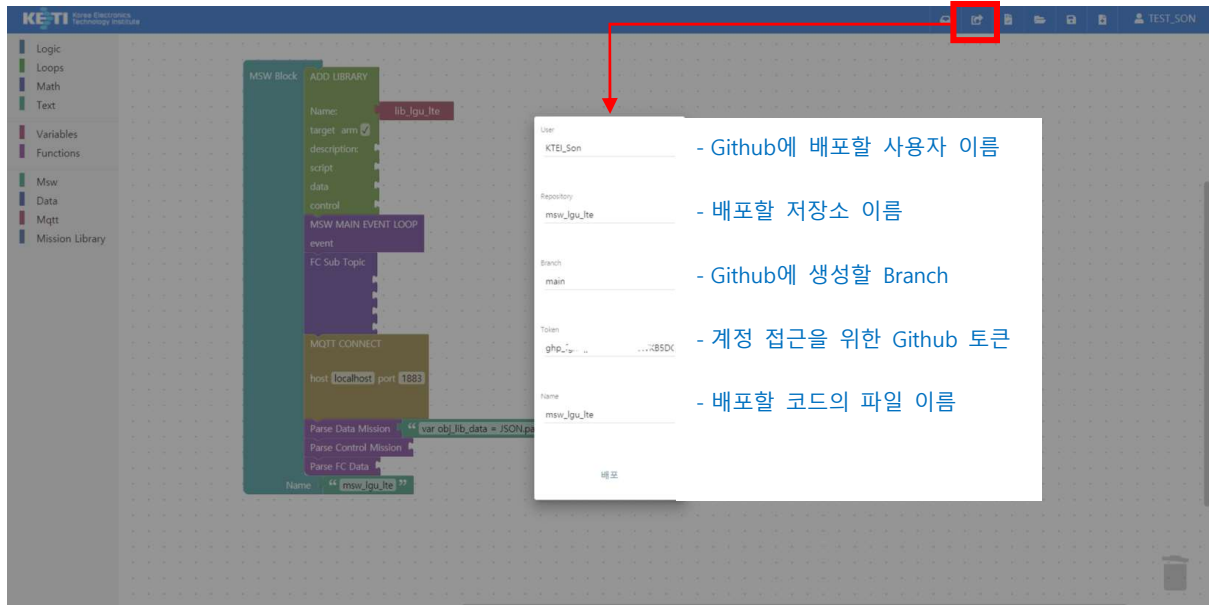
#### ④ LTE 데이터를 파싱하기 위해서 데이터를 정의



#### ⑤ 블록으로 코딩한 임무SW를 배포하기 전 소스코드 형태로 확인



## ⑥ 임무SW 배포에 필요한 정보 입력하여 배포

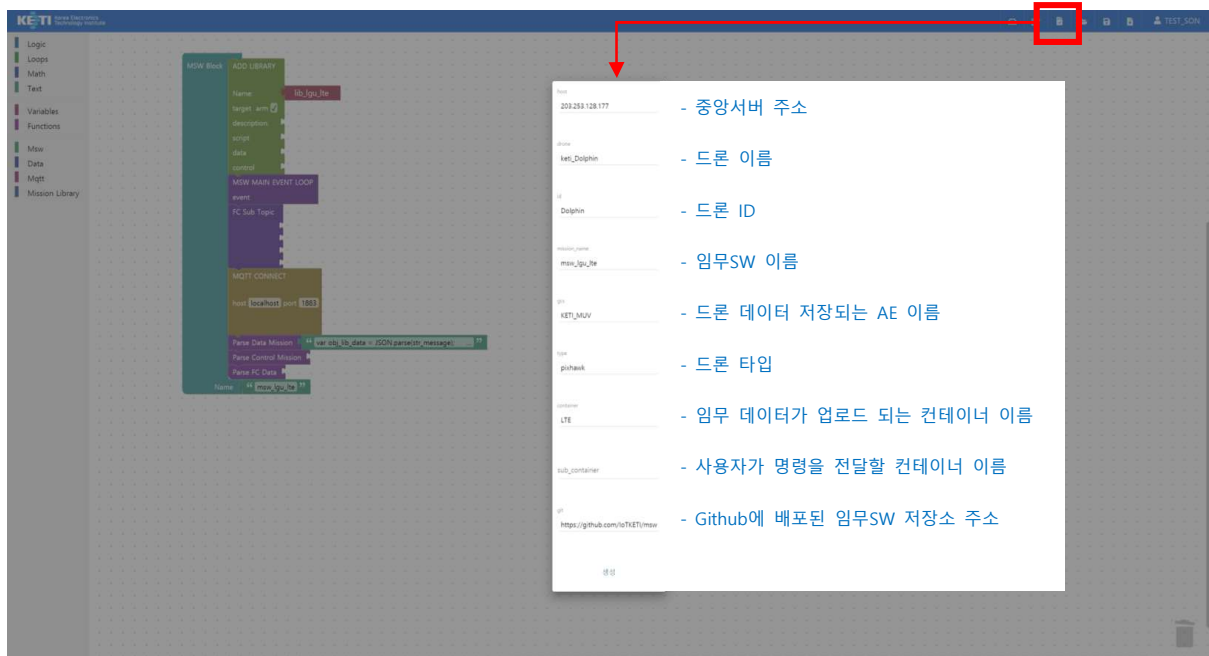


The screenshot shows the KE-TI interface with a mission block diagram on the left and a deployment form on the right. The form is titled '배포' (Deploy) and contains the following fields and instructions:

- User: KTEL\_Son - Github에 배포할 사용자 이름
- Repository: msw\_lgu\_ite - 배포할 저장소 이름
- Branch: main - Github에 생성할 Branch
- Token: ghp\_... - 계정 접근을 위한 Github 토큰
- Name: msw\_lgu\_ite - 배포할 코드의 파일 이름

The '배포' button is located at the bottom of the form.

## ⑦ 드론 탑재를 위한 드론 명세 작성



The screenshot shows the KE-TI interface with a mission block diagram on the left and a drone specification form on the right. The form is titled '드론 명세 작성' (Write Drone Specification) and contains the following fields and instructions:

- Host: 203.253.128.177 - 중앙서버 주소
- Drone: ktel\_dolphin - 드론 이름
- ID: Dolphin - 드론 ID
- Mission name: msw\_lgu\_ite - 임무SW 이름
- AE: KTEL\_MJUV - 드론 데이터 저장되는 AE 이름
- Type: phoenix - 드론 타입
- Container: LTE - 임무 데이터가 업로드 되는 컨테이너 이름
- Sub-container: - 사용자가 명령을 전달할 컨테이너 이름
- URL: https://github.com/kteti/msw - Github에 배포된 임무SW 저장소 주소

The '드론 명세 작성' button is located at the bottom of the form.

- ⑧ MC의 nCube-MUV 폴더에서 flight.json 정보 수정 (nCube-MUV 가이드 참고)
- ✓ flight에 ⑥의 과정에서 업로드한 명세의 드론 ID를 추가한다

```
{ } flight.json U X
{ } flight.json > ...
1  {
2    "approval_gcs": "MUV",
3    "flight": "Dolphin"
4  }
```



⑨ 명세 업로드 확인

The screenshot displays a web-based interface for managing data. On the left, a tree view shows a hierarchy starting with 'MUV', which branches into 'mobiusu ... WEB\_sub' and 'approval'. The 'approval' folder contains a list of sub-folders: 'HiTEC', 'Jan', 'Dolphin' (highlighted with a red box), 'Sun', 'Mercury', 'Venus', 'Dev', 'Gryphon', 'UVINet', 'SYNCTECHNO', 'BZTest', and 'Mars'. To the right, a detailed view of the 'Dolphin' folder is shown, displaying various metadata fields. A red box highlights the 'con' (configuration) section, which includes host, drone, gcs, type, and mission information.

**Folder List:**

- MUV
  - mobiusu ... WEB\_sub
  - approval
    - HiTEC
    - Jan
    - Dolphin**
    - Sun
    - Mercury
    - Venus
    - Dev
    - Gryphon
    - UVINet
    - SYNCTECHNO
    - BZTest
    - Mars

**Detailed View of 'Dolphin' Folder:**

4-20210716065318557

m2m:cin: Object

- pi: "3-20210225124939695973"
- ri: "4-20210716065318557249"
- ty: 4
- ct: "20210716T065318"
- st: 20
- rn: "4-20210716065318557"
- lt: "20210716T065318"
- et: "20230716T065318"
- cs: 201
- cr: sseokgum120@gmail.com

**Configuration (con):**

- host: "209.253.128.177"
- drone: "keti\_Dolphin"
- gcs: "KETI\_MUV"
- type: "pixhawk"
- mission: Object
  - msw\_lgu\_lte: Object
    - container: Array [1]
      - 0: "LTE"
    - sub\_container: Array [1]
      - git: [https://github.com/IoTKETI/msw\\_lgu\\_lte](https://github.com/IoTKETI/msw_lgu_lte)

DELETE CLOSE

⑩ LGU LTE 데이터 확인

- ✓ /Mobius/{GCS 이름}/Mission\_Data/{드론 이름}/msw\_lgu\_lte/LTE 컨테이너 아래에 Content Instance를 생성하여 con값에 LGU LTE 데이터를 파싱한 정보가 저장된다.

