

# Nutrient Supply Communicator (nscomm)

## 매뉴얼

version 1.0



---

# Nutrient Supply Communicator (nscomm) Manual

최종 작성일 : 2021.06.01

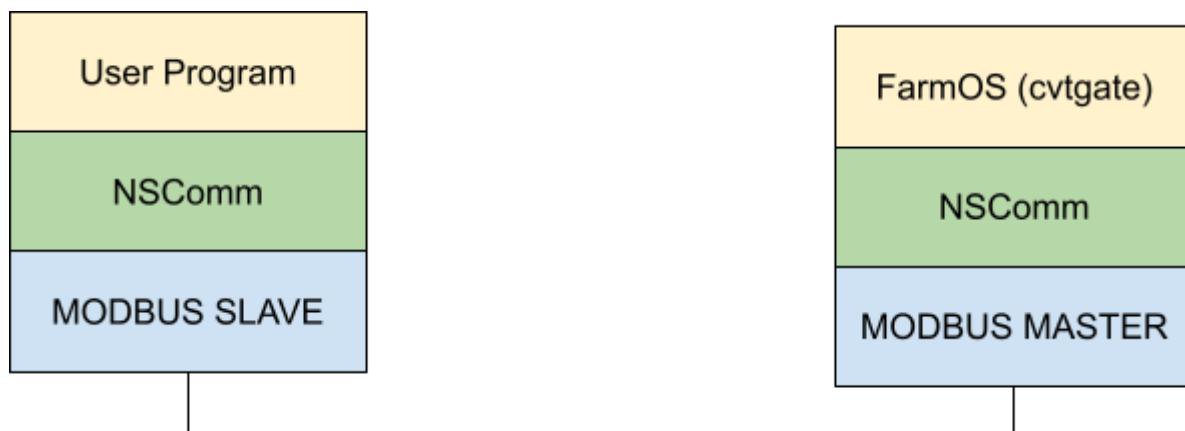
## 0. 개요

본 문서는 주식회사 지농 파모스사업본부에서 제공하는 NSComm 라이브러리에 대해서 설명합니다. NSComm 라이브러리는 양액기 표준 통신을 지원하기 위한 python 라이브러리로 사용자가 쉽게 표준 통신을 수행할 수 있도록 돋습니다.

본 문서에서 설명하는 내용은 아래의 환경에서 테스트 되었음을 알려드립니다.

```
OS : Linux  
Language : Python 3.8.5  
Package : pymodbus twisted enum34
```

NSComm 라이브러리의 전체적인 구조는 아래와 같습니다. 사용자의 프로그램은 NSComm 을 활용해서 모드버스 슬레이브로 동작하고, 통합제어기(FarmOS)는 모드버스 마스터로 동작하게 됩니다.



본 문서는 아래 3개의 표준을 기반으로 작성되었습니다.

KS X 3267 : “스마트 온실 센서/구동기 노드 및 온실 통합 제어기간 RS485 기반 모드버스 인터페이스”

KS 신규 : “스마트 온실의 온실 통합 제어기와 양액기 노드 간 RS485 기반 모드버스 인터페이스”

KS 신규 : “RS485/모드버스 기반 스마트 온실 노드/디바이스 등록 절차 및 기술 규격”

---

## 1. NSComm 라이브러리 설치

NSComm 라이브러리 설치는 파일만 복사하시면 됩니다. 라이브러리 압축파일을 풀고, 폴더 안에 있는 nscomm 폴더를 원하시는 작업경로에 복사하시어 사용하시면 됩니다.

```
# tar xvfz nscomm.tgz  
# cp -r nscomm/nscomm /원하는작업경로
```

필요한 경우 pip 를 사용하여 라이브러리를 추가로 설치해야 할 수 있습니다.

```
# sudo pip3 install pymodbus enum34 twisted
```

---

## 2. NSComm 라이브러리 사용하기

### 2.1 라이브러리 임포트

라이브러리를 사용하기 위해서 필요한 모듈을 임포트 합니다. 양액기의 운용을 위해서는 communicator 모듈을 임포트하면 됩니다.

```
import nscomm.communicator as nss
```

### 2.2 통신을 위한 설정

양액기 표준 모드버스통신은 RS485 기반 rtu 모드버스 통신입니다. 하지만 본 라이브러리는 더 빠른 통신이 가능한 TCP 모드버스기능도 제공합니다. 원하시는 옵션을 선택해서 설정하면 됩니다.

```
option = {"method": "rtu", port: "/dev/ttyUSB0", unit: 1}  
# or  
option = {"method": "tcp", host: "192.168.0.5", port: 2222}
```

method 에서 “rtu” 혹은 “tcp” 를 선택하실 수 있습니다. “rtu” 방식을 사용하는 경우 port와 unit 정보가 필요합니다. port 는 시리얼 통신을 수행할 포트를 선택하면 됩니다. 리눅스 장비이고, USB 시리얼 컨버터를 하나 사용한다면 “/dev/ttyUSB0” 일 가능성이 높습니다. unit 은 슬레이브 아이디라고도 하는데, RS485 라인에 연결된 장비를 식별하기 위한 용도로 사용합니다. 연결된 장비가 따로 없다면 1을 사용하셔도 무방합니다.

“tcp” 방식을 사용하는 경우 IP주소와 포트번호가 필요합니다. 각각 “host”, “port” 에 해당 정보를 입력해주시면 됩니다. 위의 예시는 양액기의 아이피가 192.168.0.5이고, 2222번 포트를 사용해서 통합제어기와 통신을 하겠다라는 의미가 됩니다.

### 2.3 양액기의 스펙 설정

양액기가 통합제어기와 통신을 하기 위해서는 레지스터맵이라는 것을 만들고 서로 공유해야합니다.

KS 표준에서는 디폴트 레지스터 맵을 제공하는데 이를 이용하면 간단히 처리가 됩니다.

```
defmap = nss.get_default_register_map()
```

디폴트 레지스터 맵을 사용하기 위해서는 2가지 사항을 확인할 필요가 있습니다.

a. 사용하는 센서의 종류

디폴트 레지스터 맵에서는 EC센서 3개, pH센서 3 개, 일사 센서 1개, 유량계 13개를 최대로 사용할 수 있습니다. 센서는 위의 순서대로 0부터 시작하는 숫자와 맵핑이

됩니다. EC 1번 센서는 0, EC 2번 센서는 1 과 같습니다. 맨 마지막 유량계는 19번이 됩니다.

여기서 만약 EC센서를 2개만 사용하고, 나머지 센서는 다 사용하겠다고 하면 사용하지 않을 센서를 다음과 같이 표시해주면 좋습니다.

```
defmap.remove_sensor(2)
```

b. 제공하려는 양액기의 레벨

양액기 표준에서는 양액기를 4가지 레벨로 나누고 있습니다.

1. level0 : 작동상태 모니터링은 가능하지만 제어를 할 수 없는 양액기
2. level1 : 1회관수 와 정지만 제공하는 양액기
3. level2 : level1의 기능을 지원하면서 구역관수 기능을 제공하는 양액기
4. level3 : level2의 기능을 지원하면서 파라미터 관수(EC, pH까지 설정) 기능을 제공하는 양액기

원하는 양액기 수준을 선택해줘야 합니다. 디폴트로는 level1 관수기입니다. level1 관수기라면 수정할 필요가 없지만, level2 관수기로 변경하고자 한다면 다음과 같이 수정할 수 있습니다.

```
defmap.change_nutrient_supply(2)
```

아주 특별한 경우 (예를들어, 표준에서 제공하지 않은 센서를 사용해야하는 경우)에는 레지스터맵을 완전히 변경할 필요가 있습니다. 레지스터맵 변경이 필요한경우, “RS485/모드버스 기반 스마트 온실 노드/디바이스 등록 절차 및 기술 규격”을 숙지하시어 defmap을 직접 수정하는 것도 가능합니다만 추천드리지 않습니다. 내용이 복잡해지기 때문에 여기서는 디폴트 레지스터 맵을 사용하는 것으로 설명하도록 하겠습니다.

## 2.4 Nutrient Supply Communicator의 실행

이미 설정된 옵션과 레지스터맵을 이용하여 ns\_communicator 를 실행시킬 수 있습니다. ns\_communicator 는 스레드로 동작하면서 통신과 관련된 작업을 수행하기 때문에, 아래의 함수를 실행 시킨 이후부터는 양액기 제어와 관련된 코드를 넣으시면 됩니다.

```
communicator = nss.start_ns_communicator(option, defmap)
```

communicator의 경우 thread 모드와 process 모드로 동작이 가능합니다. python 에서 thread는 실제 적인 thread가 아니지만 동시성이 중요한 경우가 아니라면 사용에 큰 제약이 있지는 않습니다.

process 모드로 동작하기 위해서는 마지막 인자에 False를 넣어줍니다.

```
communicator = nss.start_ns_communicator(option, defmap, False)
```

- 주의

---

python3.7의 경우 **process** 모드로 동작하는 경우에 문제가 있어서 python3.8을 사용하는 것이 좋습니다. 라즈베리파이의 경우 현재 디폴트로 지원하는 파이썬의 버전이 3.7이기 때문에 **thread** 모드로 사용하시는 것을 추천드립니다. **process** 모드로 동작시키고자 하시면 파이썬 버전을 업그레이드하시기 바랍니다. 파이썬 업그레이드 방법은 3장에서 설명합니다.

---

## 2.5 양액기노드의 제어권 관리

양액기 노드는 3가지 제어상태를 가질 수 있습니다.

- a. 수동제어 (**MANUAL\_CONTROL**) : 사용자가 양액기 앞에서 직접 제어를 하는 상태
- b. 원격제어 (**REMOTE\_CONTROL**) : 통합제어기가 양액기를 통신으로 제어하는 상태
- c. 로컬제어 (**LOCAL\_CONTROL**) : 양액기가 스스로 제어하는 상태

로컬제어가 가장 일반적인 제어상태라고 할 수 있으며, 가장 우선순위가 낮은 상태입니다. 수동제어는 사용자가 양액기를 직접 제어하는 상태로 가장 우선순위가 높습니다.

원격제어는 통합제어기가 제어를 할 수 있는 상태로, 로컬제어의 상태에서 통합제어기의 명령으로 상태 변환이 가능합니다. 하지만 수동제어 상태에서는 통합제어기가 명령을 내리더라도 원격제어 상태를 유지해야합니다.

`get_node_control_info` 함수로 통합제어기로부터 제어권 변경에 대한 요청이 있었는지를 확인할 수 있습니다.

```
ndinfo = nss.get_node_control_info(communicator)
```

`ndinfo`는 {"operation": 2, "opid": 1, "control": 2}와 같은 딕셔너리 형식이거나, None 값을 갖게 됩니다. None 이라면 노드에 대한 (제어권 변경) 명령이 없었다는 의미입니다.

`operation`이 2라면 제어권 변경을 의미합니다. 현재 그외의 명령은 없습니다.

`control`의 2는 **REMOTE\_CONTROL**을 뜻합니다.

`LOCAL_CONTROL`은 1, `MANUAL_CONTROL`은 3의 코드값을 갖습니다.

이 명령을 이용해서 관련된 처리를 수행한 뒤에는 제어권의 상태를 통합제어기에 알려주어야 합니다. 이 때 사용할 수 있는 함수가 `update_node_status` 함수입니다. 노드의 상태가 정상이고, 직전에 받은 명령의 아이디가 1, 제어권 값이 2 이므로 정상적으로 처리되었다면 다음과 같이 정보를 업데이트 합니다.

```
nss.update_node_status(communicator, {"status": 0, "opid": 1, "control": 2})
```

## 2.6 센서 정보의 업데이트

양액기는 수시로 연결된 센서의 정보를 읽어서 통합제어기로 전송해주어야 합니다. 보통 1분에 한번 업데이트를 하지만, 실제로는 더 자주 읽어서 업데이트를 해주는 것이 좋습니다.

센서는 여러개 설치될 수 있기 때문에 설치된 센서의 순서대로 0부터 시작하는 번호를 할당하여 사용합니다. 디폴트 레지스터맵을 사용한다면, 0이 EC 1번 센서가 되고, 1이 EC 2번 센서가 됩니다.

센서의 관측치는 캘리브레이션 된 실제값을 실수형으로 사용하면 되고, 센서의 상태는 정상이라면 `StatCode.READY`를 사용하면 됩니다.

사용할 수 있는 상태 코드의 정보는 다음과 같습니다.

<code>StatCode.READY</code>	0	정상, 준비중, 정지
<code>StatCode.ERROR</code>	1	오류
<code>StatCode.BUSY</code>	2	처리 불능
<code>StatCode.VOLTAGE_ERROR</code>	3	동작 전압 이상
<code>StatCode.CURRENT_ERROR</code>	4	동작 전류 이상
<code>StatCode.TEMPERATURE_ERROR</code>	5	동작 온도 이상
<code>StatCode.FUSE_ERROR</code>	6	휴즈 이상
<code>StatCode.NEED_REPLACE</code>	101	센서 및 소모품 교체 요망
<code>StatCode.NEED_CALIBRATION</code>	102	센서 교정 요망
<code>StatCode.NEED_CHECK</code>	103	센서 점검 필요

숫자 혹은 코드 변수를 사용하실 수 있습니다.

EC 1번 센서의 값이 2.3이고, 센서가 정상이라면 `update_sensor_status` 함수를 다음과 같이 사용하면 됩니다.

```
nss.update_sensor_status(communicator, 0, {"status": 0, "value": 2.3})  
  
# or  
  
nss.update_sensor_status(communicator, 0, {"status": StatCode.READY.value, "value": 2.3})
```

pH 1번 센서의 값이 6.5이고, 센서의 캘리브레이션이 필요하다면 `update_sensor_status` 함수를 다음과 같이 사용하면 됩니다.

```
nss.update_sensor_status(communicator, 3, {"status": 102, "value": 6.5})  
  
# or  
  
nss.update_sensor_status(communicator, 3, {"status": StatCode.NEED_CALIBRATION, "value": 6.5})
```

## 2.7 양액기 제어

통합제어기는 양액기노드의 제어권이 `REMOTE_CONTROL`인 경우에 양액기를 제어할 수 있습니다. 제어할 수 있는 명령은 4가지이며, 양액기의 레벨에 따라 사용할 수 있는 명령이 차이가 있다라는 내용을 위에서 설명한 바 있습니다.

양액기에 전달되는 명령은 `get_nutrient_control_info` 함수로 확인이 가능합니다.

```
ctrlinfo = nss.get_nutrient_control_info(communicator)
```

ctrlinfo 는 딕셔너리 형식 (ex. {operation:402, opid:2, start\_area:1, end\_area:2, on\_sec:50}) 이거나 **None** 의 값을 갖습니다. **None** 이 나오면 새로들어온 명령은 없다고 생각하면 됩니다. 새로운 명령이 왔다면 위의 예와 같은 형식이 됩니다. 402 명령은 구역관수 명령이기 때문에 EC, pH에 대한 값이 없지만 403 명령이었다면 EC, pH 값도 전달됩니다.

명령 코드는 아래와 같습니다.

ON	401	1회 관수 : 양액기에 기 설정된 대로 1회 관수 작동 시작을 요청하는 명령
OFF	0	작동 멈춤 :양액기를 강제로 정지시키고자 하는 명령
AREA_ON	402	구역관수 : 양액기에 지정된 시간과 구역으로 관수를 요청하는 명령
PARAM_ON	403	파라미터 관수: 양액기에 파라미터로 전달되는 EC, pH 값이 반영되어 지정된 시간과 지정된 구역으로 관수를 요청하는 명령

양액기가 구동중인 중에 새로운 명령이 도착했다면, 기본적으로 새로 들어오는 명령을 큐로 관리하는 기능을 제공합니다. 이 경우 중지 명령이 들어오면, 큐에 들어있는 명령들은 모두 삭제되고, 중지명령만 남게 됩니다.

예를 들어 1구역에 30초 관수를 하라는 명령이 들어와서 관수를 진행하고 있는 중에, 2구역에 40초, 1구역에 20초, 2구역에 60초의 관수 명령이 추가적으로 들어온다면 각각의 명령들은 라이브러리에서 제공하는 큐에 저장됩니다. 그래서 `get_nutrient_control_info` 함수를 부를때마다 다음 명령이 나오게 됩니다.

이런 방법으로 양액기는 1구역 30초 관수명령을 완료한 이후에 2구역 40초의 새로운 명령을 큐에서 꺼내서 작업을 진행할 수 있게 됩니다. 이때 큐에는 1구역 20초, 2구역 60초의 관수명령이 남아 있는 상태입니다. 그런데 이상황에서 정지명령이 오게 된다면, 남아있던 2개의 명령은 모두 사라지고, 정지 명령만 큐에 남게 됩니다.

양액기에 전달되는 명령을 확인하는 특별한 함수로 `should_stop` 함수를 제공합니다. 이 함수는 수시로 실행되어야 하는 함수인데, 정지 명령이 도착했는지를 확인하는 용도로 사용됩니다. `get_nutrient_control_info` 는 명령을 처리하고 있지 않을때 다음 명령이 있는지를 확인하는 용도로 사용된다는 차이가 있습니다.

```
opid = nss.should_stop(communicator)
```

opid 가 0 이라면 정지명령이 도착하지 않은 상황이고, 그 외의 값이라면 해당 opid로 정지명령이 도착한 상황입니다.

명령을 확인했다면, 해당 명령을 실행하고, 양액기의 상태를 변경해야 합니다. 양액기의 상태는 다음과 같은 상태를 가질 수 있습니다.

StatCode.READY	0	정상, 준비중, 정지
----------------	---	-------------

---

StatCode.ERROR	1	오류
StatCode.BUSY	2	처리 불능
StatCode.VOLTAGE_ERROR	3	동작 전압 이상
StatCode.CURRENT_ERROR	4	동작 전류 이상
StatCode.TEMPERATURE_ERROR	5	동작 온도 이상
StatCode.FUSE_ERROR	6	휴즈 이상
StatCode.PREPARING(또는 MIXING)	401	준비 중
StatCode.SUPPLYING	402	제공 중 (관수 중)
StatCode.STOPPING	403	정지 중

STOPPING의 경우 정지를 하기 위한 작업을 수행하는 경우인데, 완전히 정지가 되면 0으로 바뀌어야 합니다.

만약 경보를 알려야 할 내용이 있다면 다음과 같은 코드를 사용할 수 있습니다.

AlertCode.NORMAL	0	정상
AlertCode.HIGH_CONCENTRATION_EC	1	고농도 EC
AlertCode.LOW_CONCENTRATION_EC	2	저농도 EC
AlertCode.HIGH_CONCENTRATION_pH	3	고농도 pH
AlertCode.LOW_CONCENTRATION_pH	4	저농도 pH
AlertCode.LOW_FLOW_ALARM	5	저유량 경보
AlertCode.HIGH_TEMPERATURE_ALARM	6	고온 경보
AlertCode.LOW_TEMPERATURE_ALARM	7	저온경보
AlertCode.ABNORMAL	8	앞에 나열된 경우를 제외한 장비 이상

양액기가 EC, pH를 맞추기 위해 믹싱을 시작했고, 경보는 없다면 update\_nutrient\_status 함수를 이용하여 다음과 같이 양액기의 상태를 변경할 수 있습니다.

```
nss.update_nutrient_status(communicator, {"status": 401, "area": 0, "alert": 0, "opid": 2})
```

양액기가 1구역에 공급을 시작했다면 다음과 같이 변경합니다.

```
nss.update_nutrient_status(communicator, {"status": 402, "area": 1, "alert": 0, "opid": 2})
```

위와 같은 방법으로 양액기의 상태가 변경될때마다 적절한 상태로 값을 변경해 update\_nutrient\_status 함수를 호출해주게 되면, 통합제어기가 변경된 상태를 읽어 적절한 상태를 표시해줄 수 있습니다.

## 2.8. 양액기 종료

양액기의 상태를 종료할때는, 다음의 함수를 사용합니다.

```
nss.stop_ns_communicator(communicator)
```

### 3. Python 3.8 설치하기

nscomm을 활용할때 process 모드로 동작시키기 위해서는 파이썬 3.8 버전이 필요합니다. 아래에서는 라즈베리파이에서 3.8 버전을 설치하고 사용하는 방법을 설명합니다.

#### 3.1 라즈베리파이환경 업데이트하기

라즈비안의 apt 패키지를 업데이트하고 필요한 패키지들을 설치합니다.

```
# sudo apt-get update
# sudo apt-get install -y build-essential tk-dev libncurses5-dev
libncursesw5-dev libreadline6-dev libdb5.3-dev libgdbm-dev
libsdl1.2-dev libssl-dev libbz2-dev libexpat1-dev liblzma-dev
zlib1g-dev libffi-dev
```

#### 3.2 Python 을 다운받아 설치하기

필요한 Python 을 다운받고 컴파일 합니다. 이 문서를 작성하는 시점에는 3.8.10이 최신버전입니다.

```
# wget https://www.python.org/ftp/python/3.8.10/Python-3.8.10.tar.xz
# tar xf Python-3.8.10.tar.xz
# cd Python-3.8.10
# ./configure --enable-optimizations
# make -j 4
# sudo make altinstall
```

#### 3.3 Python 설치 마무리 하기

새로 설치된 python이 제대로 설치되었는지 확인합니다. 기존에 설치되어 있던 python3.7은 python3 으로 그대로 사용 가능합니다. python3.8을 구동하고자 할때는 python3.8 이라는 명령을 이용합니다.

```
# python3.8 -V
# python3 -V
```

#### 3.4 pip 설치하기

새로 설치된 python을 위한 패키지를 다운받아 사용하기 위해서는 pip 설치가 필요합니다. python3.8 명령을 사용하여 get-pip.py 를 실행해야합니다.

```
# curl https://bootstrap.pypa.io/get-pip.py
# python3.8 get-pip.py
```

---

pip를 이용해서 패키지를 설치하고자 할때는 pip3.8 명령을 사용합니다.

```
# pip3.8 install pymodbus
```

---

## 4. 활용하기

### 4.1 장비 연결 및 기본 세팅

양액기를 원격으로 제어하기 위한 장비로 슬레이브 장치, 마스터 장치 2개가 제공됩니다.

#### a. 마스터 장치

다른 하나는 마스터 장치로써 통합제어기(FarmOS)를 통하여 슬레이브 장치를 원격으로 제어하는 역할을 수행합니다. 마스터 장치는 주식회사 지농에서 직접 관리합니다. 마스터 장치를 외부망과 연결된 공유기에 연결한 뒤, 웹브라우저를 통해 마스터 장비에 접속하면 양액기를 제어할 수 있습니다. 만약 마스터 장치의 아이피가 192.168.0.11 이라면 <http://192.168.0.11:8081>로 접속할 수 있습니다.

#### b. 슬레이브 장치

모드버스 슬레이브가 설치된 장치는 양액기를 직접 제어하는 역할을 수행하며, 양액기 라이브러리를 사용하여 장치와 연동을 하는 코드를 작성하면 통합제어기(FarmOS)와 통신할 수 있습니다. 슬레이브가 설치된 장치의 아이디/패스워드는 pi/KistJN#2021입니다.

슬레이브 장치와 마스터 장치를 사용하기 위해서는 동일한 공유기에 연결해야 합니다. 슬레이브 장치는 마스터 장치에서 항상 접근 가능해야 하기 때문에, 장치의 IP는 항상 고정IP로 설정하는 것을 권해드립니다.

양액기 장비의 운영체제로 Raspberry pi가 설치되어 있으며, IP 설정방법은 다음과 같습니다.

```
$ sudo vi /etc/dhcpcd.conf
```

다음 항목을 설정에 맞게 수정한 뒤 저장합니다.

```
# Example static IP configuration:  
#interface eth0  
#static ip_address=192.168.0.10/24  
#static ip6_address=fd51:42f8:caae:d92e::ff/64  
#static routers=192.168.0.1  
#static domain_name_servers=192.168.0.1 8.8.8.8 fd51:42f8:caae:d92e:
```

네트워크 인터페이스를 재시작하면, 수정된 IP가 적용됩니다.

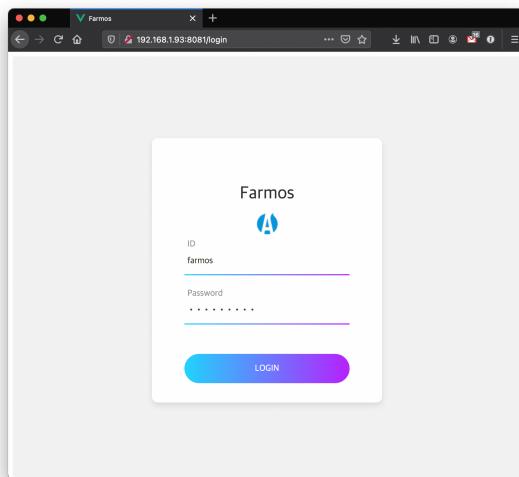
```
service networking restart
```

**[중요]** 슬레이브 장치의 아이피를 고정아이피 혹은 유동아이피로 설정이 하셨다면, 해당 아이피를 주식회사 지농에 알려주시면 저희가 마스터 장치를 세팅해 드립니다.

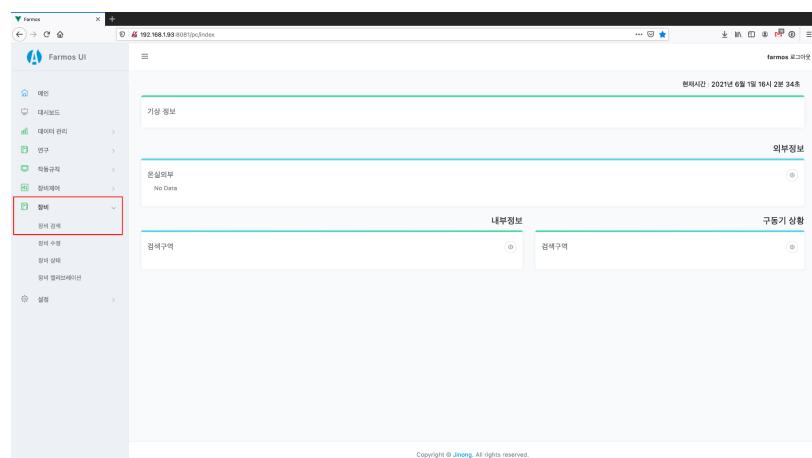
## 4.2 양액기 자동 인식 및 장비 등록

4.1 과정이 종료되었다면, 웹에서 마스터 장치를 통하여 양액기에 접근할 수 있습니다. 로그인 후에 양액기의 장비를 탐색을 수행하고 구역과 연결하면 양액기를 제어 및 모니터링을 수행할 수 있습니다.

우선 웹브라우저를 열고, 마스터 장치의 아이피에 8081 포트로 접속합니다. 아래 예에서 보면 192.168.1.93이 마스터 장치의 아이피입니다. 이 경우 <http://192.168.1.93:8081>이 됩니다. 로그인을 위해 아이디와 패스워드를 입력합니다. 아이디는 **farmos**이고, 패스워드는 **farmosv2@**입니다.

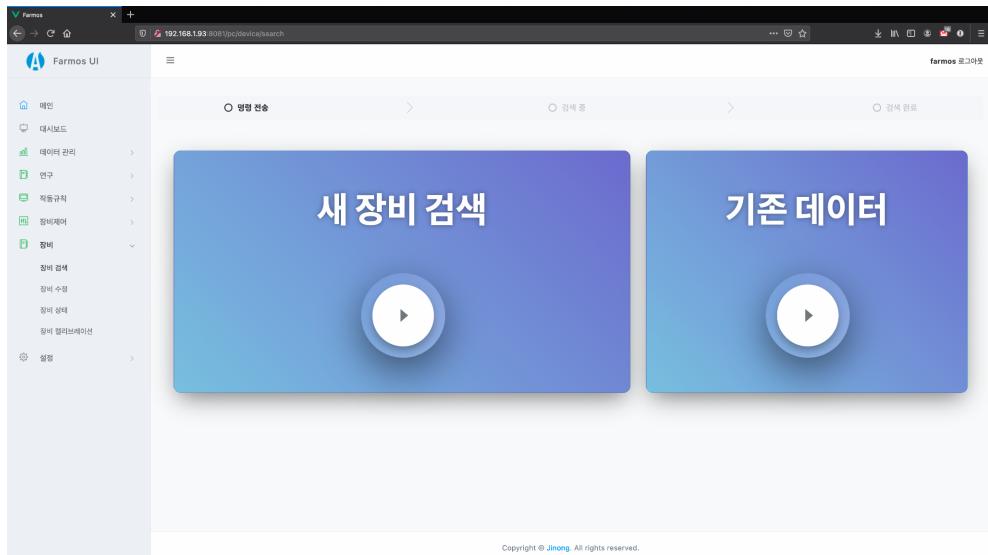


FarmOS의 사용자 인터페이스를 통해 양액기를 인식할 수 있습니다. UI화면 왼쪽에 있는 장비 - 장비검색 버튼을 클릭하여 탐색 요청을 수행하면 양액기 장치에서 작동하는 슬레이브 장치에 접근하여 현재 작동하는 양액기의 장비를 탐색합니다.

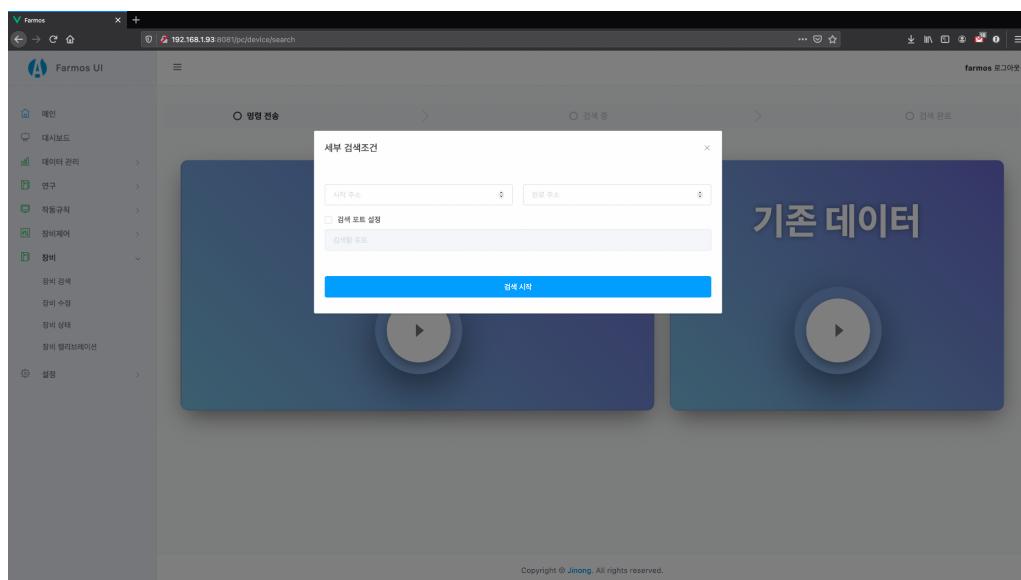


마스터 장치는 슬레이브 장치의 모드버스 서버의 레지스터에 저장된 장비 코드들을 탐색하고, 웹 서버로 탐색한 결과를 전송합니다.

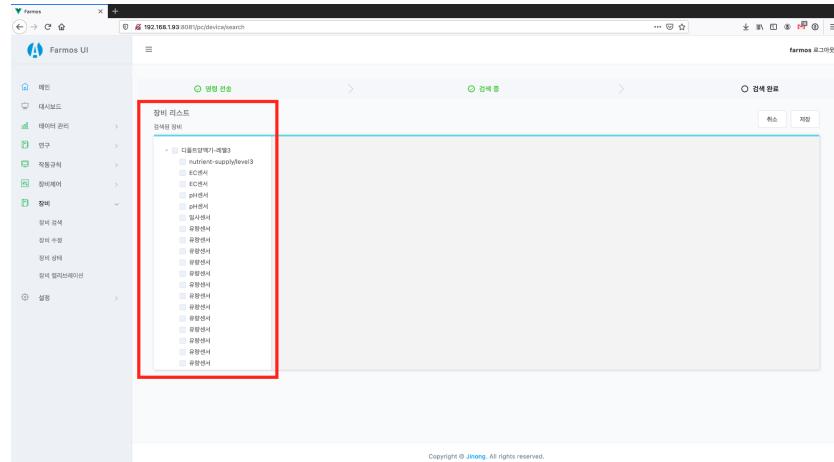
장비 검색메뉴를 선택하면 표시되는 아래 화면에서 새 장비 검색 버튼을 클릭합니다.



아래 화면에서 별다른 조건을 입력하지 않고, 검색 시작 버튼을 클릭합니다. RS485 기반 모드버스를 사용하는 경우 슬레이브 아이디의 범위를 주어 탐색할 수 있지만, TCP 모드버스를 사용하는 경우 슬레이브 아이디가 하나만 존재하기 때문에 범위를 지정하지 않습니다.

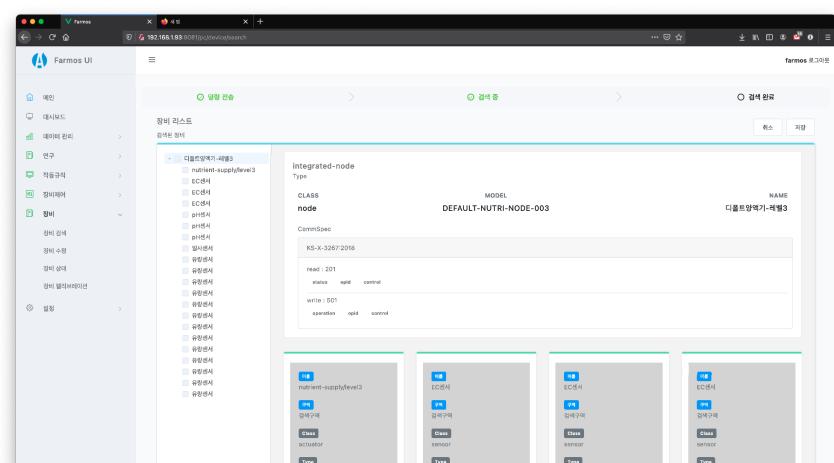


약간의 시간을 기다리면, 아래 화면과 같이 검색된 항목들을 확인할 수 있습니다. 검색된 항목은 양액기에 설치된 센서의 종류와 양액기의 레벨에 따라 다르게 나타날 수 있습니다.

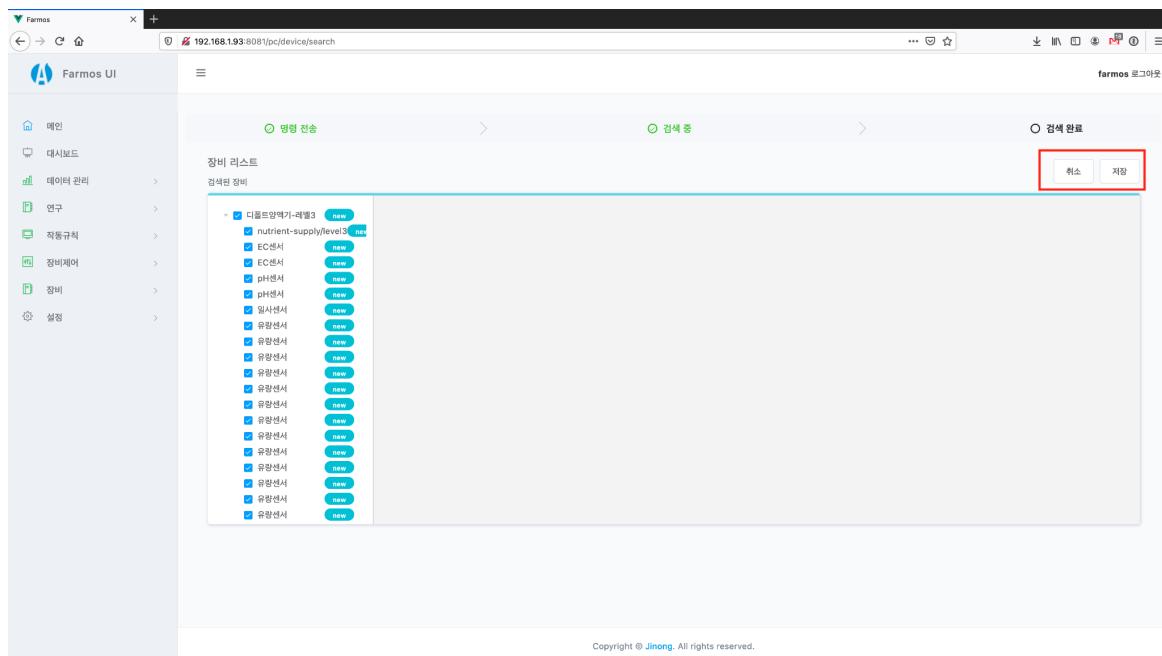


왼쪽의 장비 리스트의 항목의 디폴트 양액기 - 레벨 3 혹은 하위 아이템을 클릭하면 양액기와 관련된 정보들을 확인할 수 있습니다. 아래 화면에 있는 각 목록의 내용은 다음과 같습니다.

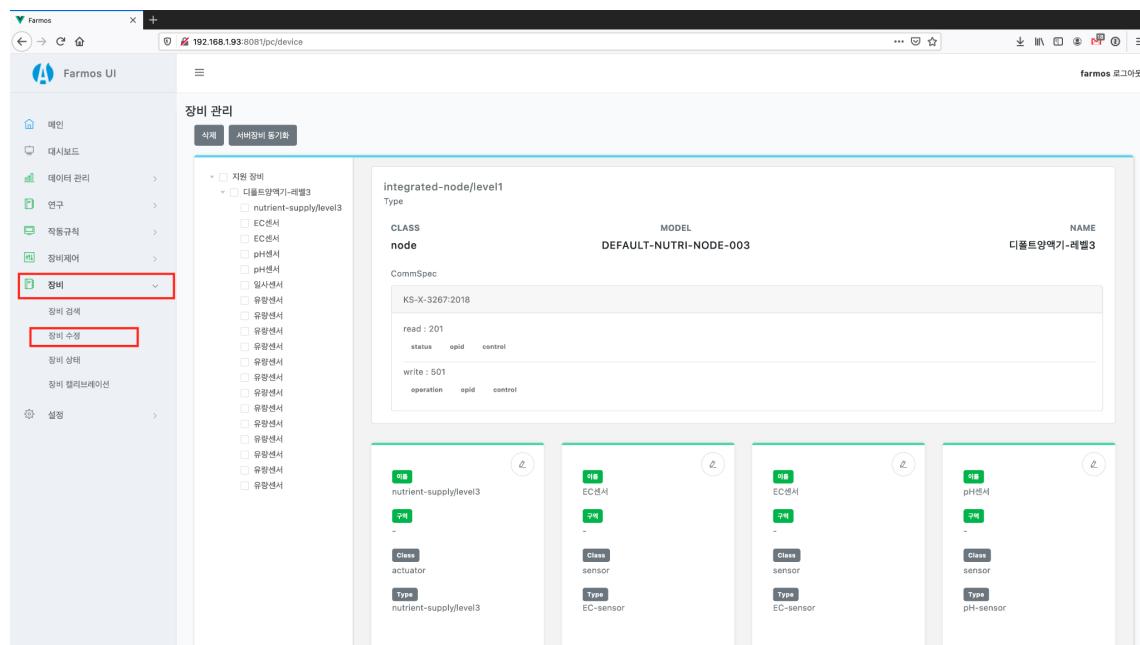
- **class** : 장치가 어떤 역할을 하는지 나타냅니다. 주요 값은 node, actuator, sensor 등을 나타냅니다.
- **type** : 장치의 유형을 나타냅니다. 주요값은 nutrient-supply/level3, flowmeter, pH-sensor, pyranometer 등이 있습니다.
- **model** : 노드의 모델명을 표시합니다. 아래 예시에서 양액기 노드의 모델 명은 DEFAULT-NUTRI-NODE-003입니다.
- **name** : 노드의 디폴트 이름을 나타냅니다.
- **CommSpec** : 읽기 및 쓰기 주소 영역을 표시합니다.



검색된 장비들 중에 사용하고자 하는 장비의 체크박스를 클릭한 뒤에, 우측 상단에 있는 저장 버튼을 클릭하면 장비가 등록됩니다.

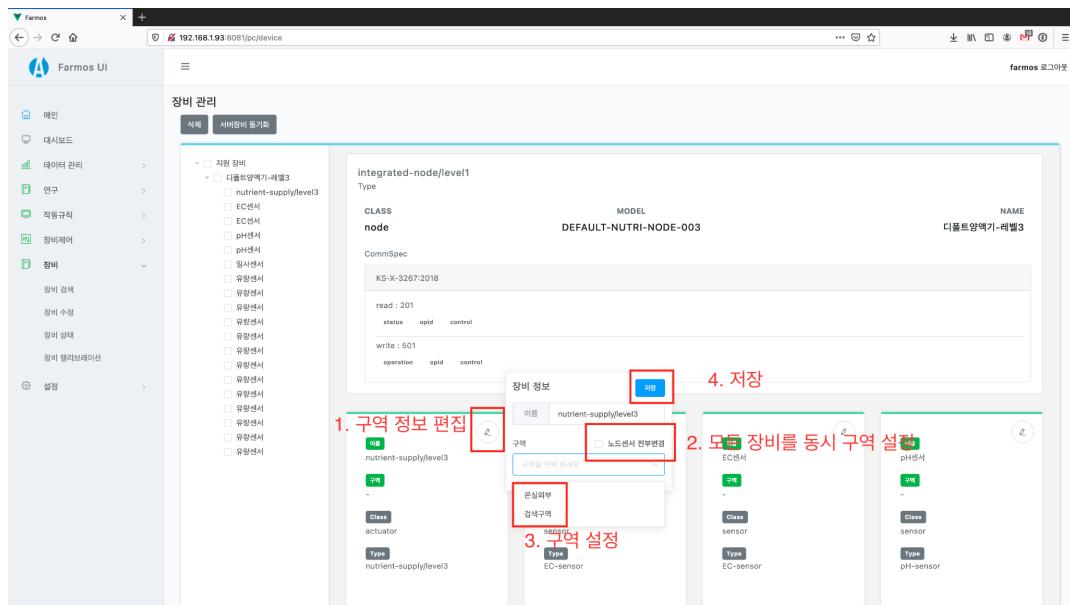


장비가 등록된 뒤, 구역을 할당하여 제어할 수 있습니다. 왼쪽 메뉴에 장비 - 장비 수정을 클릭합니다.



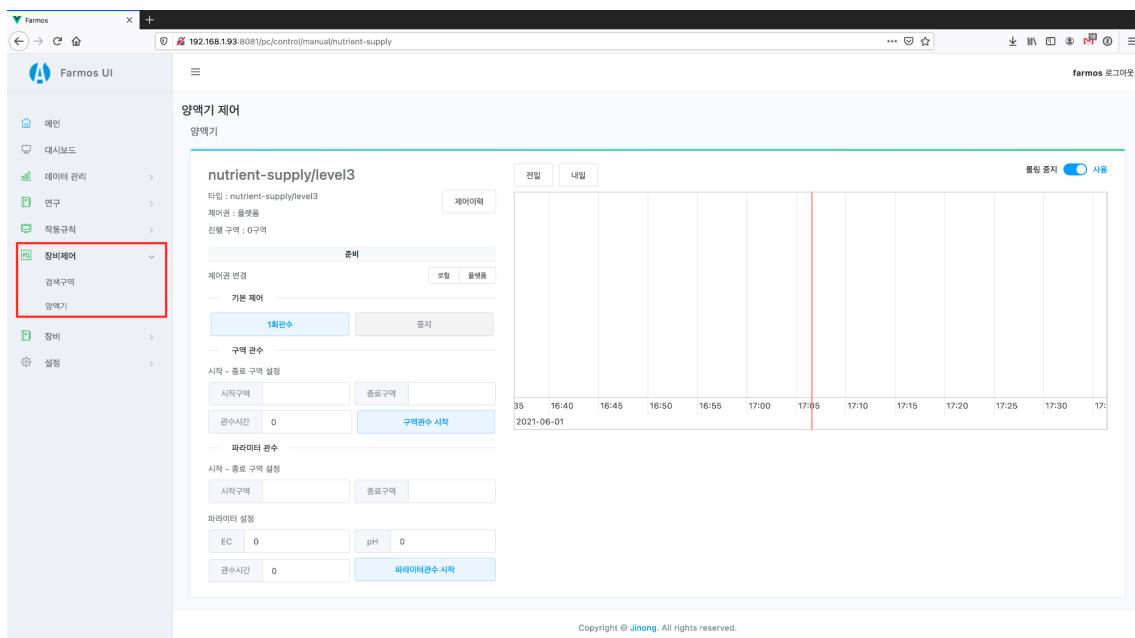
양액기 장치목록이 나오고 장치의 오른쪽 위에 연필 버튼을 클릭하면 팝업 창이 뜹니다. 노드 센서 전부 변경을 클릭하고 구역 설정을 누르면 모든 노드 센서에 선택한 구역이 할당됩니다. 마지막으로 저장 버튼을 누르면 변경한 구역이 적용됩니다.

양액기의 경우 특정 구역에 설치되는 장비가 아니기 때문에 온실외부로 구역을 선택하는 것을 추천합니다. 하지만 원한다면 다른 구역으로 선택해도 아무런 문제가 없습니다.



#### 4.3 양액기 센서 데이터 확인 및 수동 제어

양액기 제어 홈페이지에 접속한 뒤, 장비제어 - 양액기(다른 이름일 수 있습니다.)를 클릭하면 아래와 같은 화면이 나옵니다. 우측 그림판에서는 양액기의 작동이력을 확인할 수 있습니다.



센서 데이터를 확인하려면, 메인 버튼을 클릭하고 오른쪽 화면을 스크롤하면 검색구역에서 센서 및 장치의 데이터를 확인할 수 있습니다.

검색구역		검색구역	
<b>EC센서</b>	<b>pH센서</b>	<b>검색구역</b>	<b>검색구역</b>
<b>준비</b>	<b>준비</b>	<b>준비</b>	<b>준비</b>
관측치 2.29 dS/m Raw값 0 장비기동률 0 추정관측치 0 일간 개수 0 일간 평균 0 일간 최고 0 일간 최저 0 일간 표준편차 0 일간 변동편차 0	관측치 2.29 dS/m Raw값 0 장비기동률 0 추정관측치 0 일간 개수 0 일간 평균 0 일간 최고 0 일간 최저 0 일간 표준편차 0 일간 변동편차 0	관측치 0 dS/m Raw값 0 장비기동률 0 추정관측치 0 일간 개수 0 일간 평균 0 일간 최고 0 일간 최저 0 일간 표준편차 0 일간 변동편차 0	관측치 0 Raw값 0 장비기동률 0 추정관측치 0 일간 개수 0 일간 평균 0 일간 최고 0 일간 최저 0 일간 표준편차 0 일간 변동편차 0
<b>pH센서</b>	<b>일사센서</b>	<b>유량센서</b>	
<b>준비</b>	<b>준비</b>	<b>준비</b>	
관측치 0 Raw값 0 장비기동률 0 추정관측치 0 일간 개수 0	관측치 0 W/m <sup>2</sup> Raw값 0 장비기동률 0 추정관측치 0 일간 개수 0	관측치 0 L Raw값 0 장비기동률 0 추정관측치 0 일간 개수 0	

기간별로 데이터를 조회하고 싶을 때에는, 왼쪽 메뉴에서 데이터 관리 - 데이터 조회버튼을 클릭합니다.

검색 조건

구역

기간

2021-05-31 ~ 2021-05-31

데이터

장비 / EC센서 / 상태
장비 / EC센서 / 관측치
장비 / EC센서 / Raw값
장비 / EC센서 / 장비기동률
장비 / EC센서 / 추정관측치
장비 / EC센서 / 일간 개수
장비 / EC센서 / 일간 평균
장비 / EC센서 / 일간 최고
장비 / EC센서 / 일간 최저
장비 / EC센서 / 일간 표준편차
장비 / EC센서 / 일간 변동편차

비교 조건

구역

기간

2021-05-31 ~ 2021-05-31

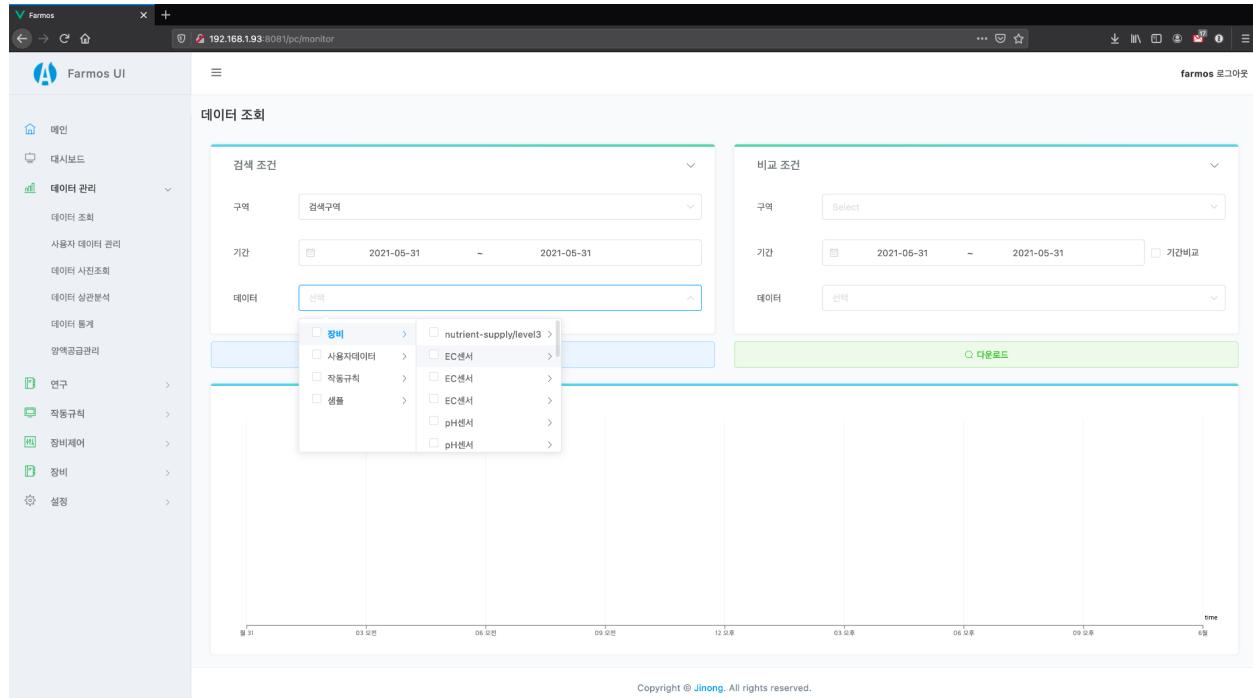
기간비교

데이터

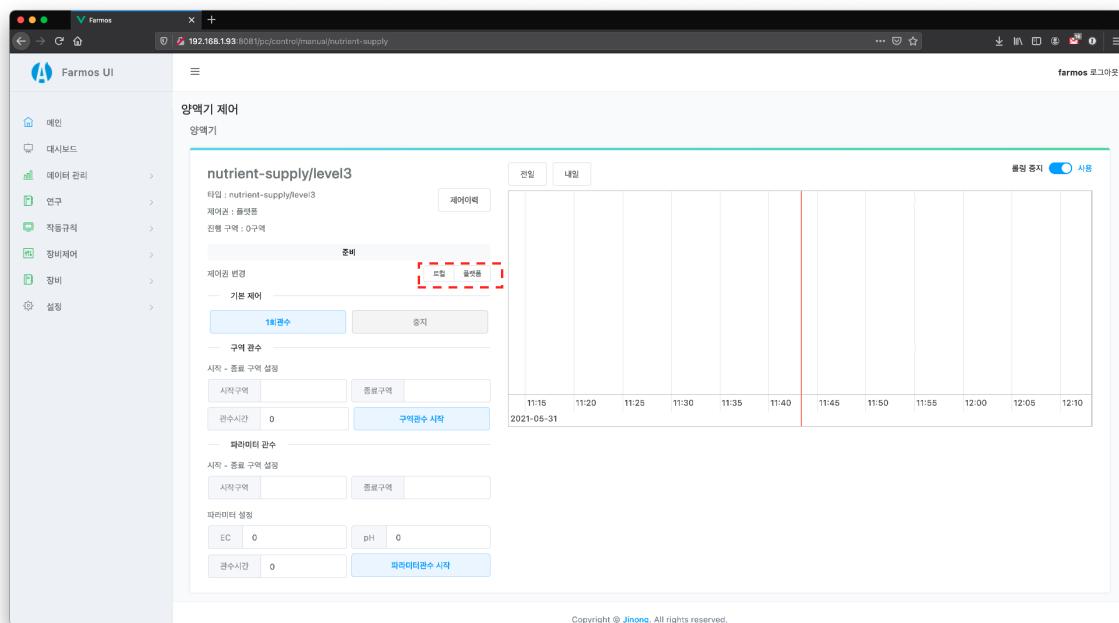
선택

▷ 비교조회
▷ 다운로드

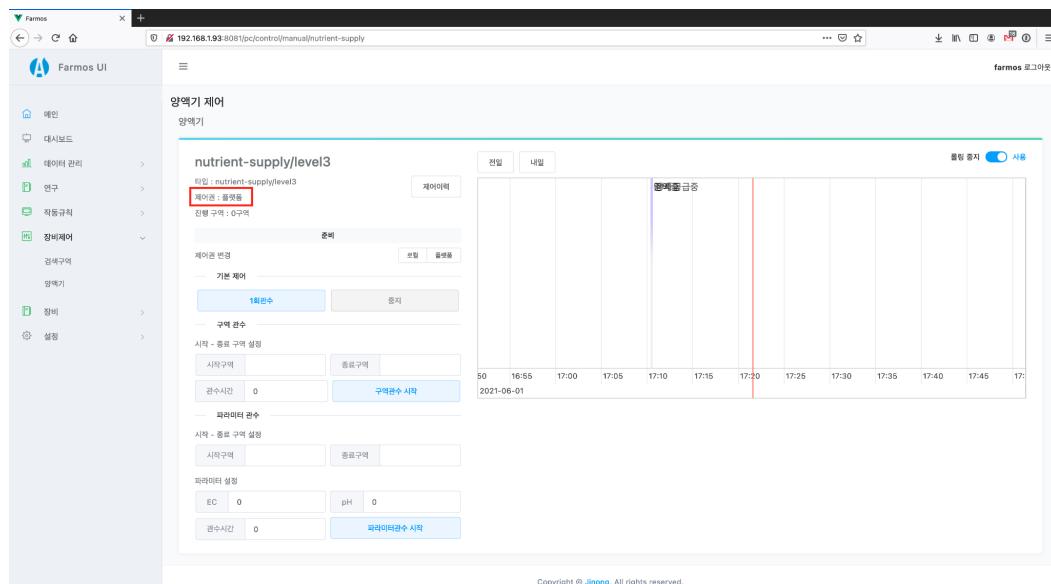
데이터 조회 버튼을 클릭한 뒤에는 다음과 같이 구역과 데이터를 지정하여 원하는 센서의 값을 가져올 수 있습니다.



양액기의 제어 모드는 로컬 모드 및 플랫폼 모드 2가지로 제공됩니다. 로컬 모드에서는 FarmOS에서 장비를 제어할 수 없습니다. 제어 명령의 버튼이 동작하지 않습니다. FarmOS에서 양액기를 제어하기 위해서는 플랫폼 제어 모드로 변경하여야 합니다. 양액기를 플랫폼으로 제어하기 위해서 붉은색으로 표시된 부분에 플랫폼을 클릭합니다. 클릭한 뒤 잠시 기다리면 모드가 변경되어 제어가 가능해집니다.



아래 빨간 표시와 같이 제어권이 플랫폼 제어로 바뀌어있다면, 플랫폼 제어 모드로 변환된 것입니다.



위의 상태에서 1회관수, 중지 등의 명령을 수행할 수 있습니다.

#### 4.4 양액기 자동 제어하기

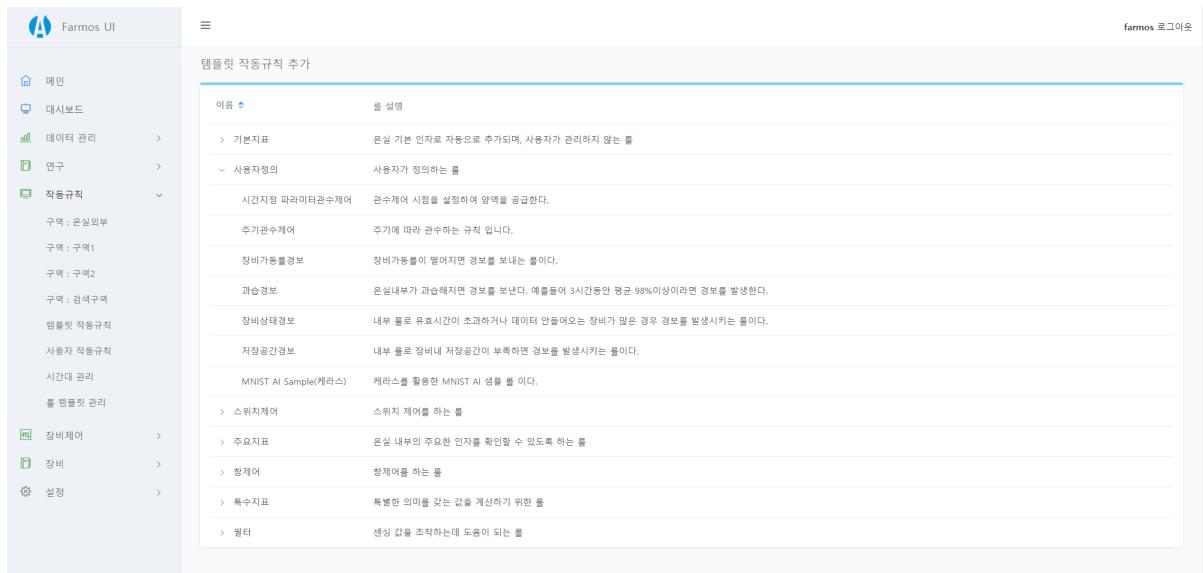
FarmOS는 장비의 자동제어를 위해 작동규칙을 적용하여 자동제어를 합니다. 작동규칙을 생성하는 방법은 크게 템플릿 작동규칙 생성과 사용자 작동규칙 생성이 있습니다. 아래 예시를 통해 작동규칙을 생성하여, 자동제어하는 방법을 확인할 수 있습니다.

##### a. 템플릿 작동규칙 생성방법

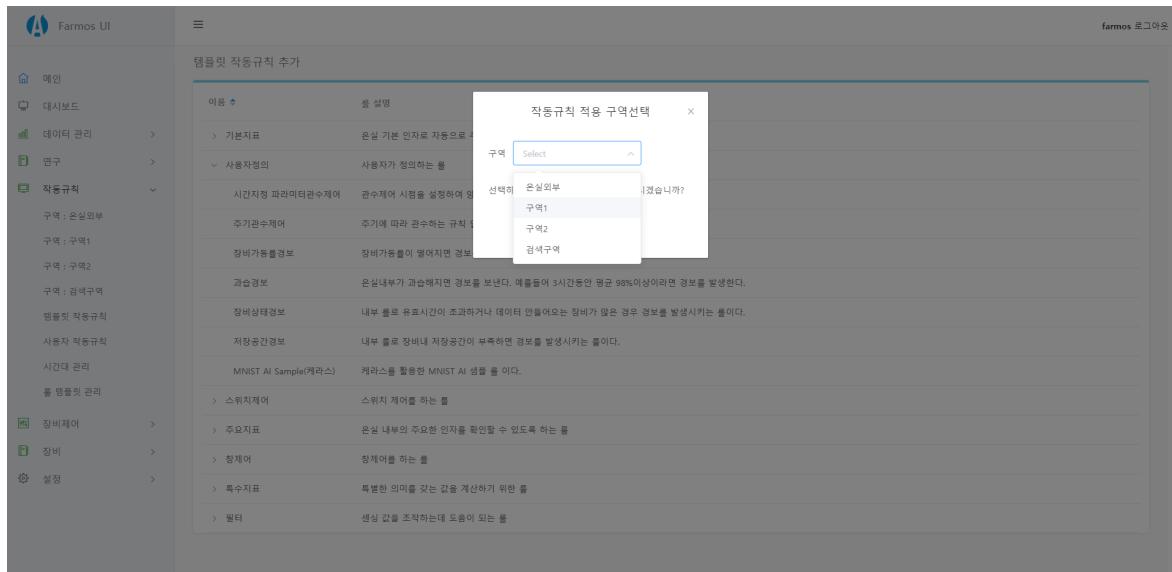
템플릿 작동규칙은 FarmOS에 기존에 세팅되어있는 규칙 포맷을 가져와 사용하는 방법입니다. 매일 설정한 시간에 파리미터 관수 명령을 내리는 작동규칙을 예시로 템플릿 작동규칙 생성법을 알아보겠습니다.



## 메뉴에서 작동규칙 > 템플릿 작동규칙을 클릭합니다.



미리 세팅되어 있는 여러가지 규칙 템플릿이 있으며, 이중 사용할 규칙을 선택하면 설정값만 입력해 규칙을 이용할 수 있습니다. 위 화면에서 사용자 지정의 시간지정 파라미터관수제어 규칙을 선택해 특정 시간에 관수를 수행할 수 있습니다. 자동관수를 사용하기 위해서는 양액기의 제어권이 미리 원격제어(플랫폼)으로 변경되어 있어야 합니다.



사용할 템플릿을 클릭하면 위와 같이 적용구역을 선택합니다.

구역을 선택하면 위와 같이 규칙설정화면이 나타납니다.

환경설정에 양액기를 선택하고 저장버튼을 누르면, 위의 설정값 화면을 이용할 수 있습니다.

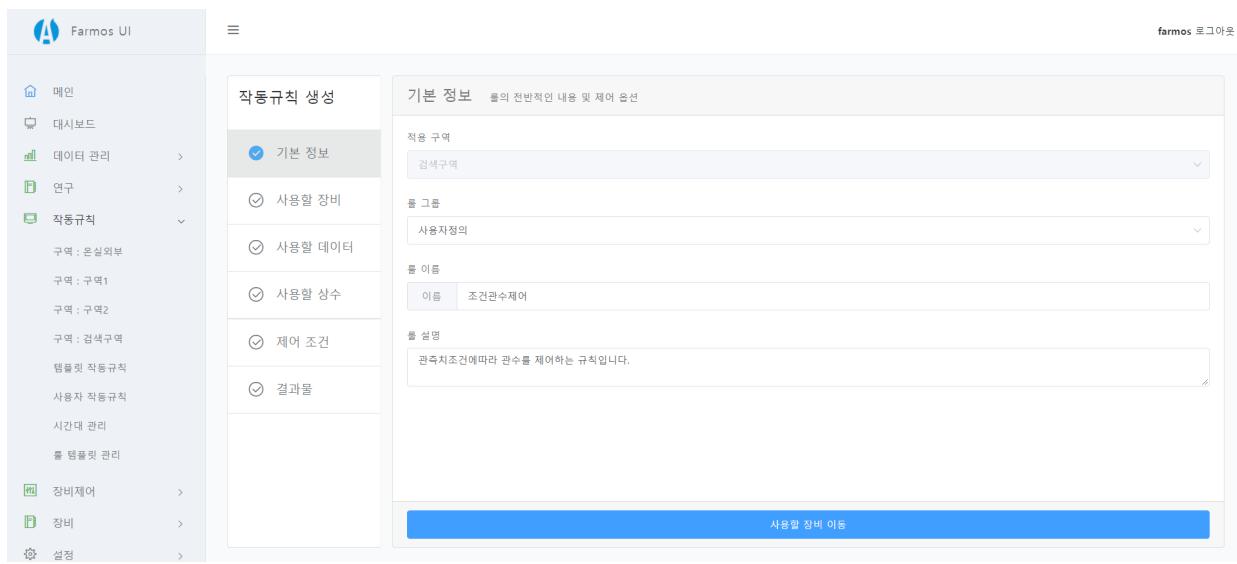
스케줄설정에서는 매일 특정시간에, 어떤 명령과 어떤 파라미터로 양액기를 자동제어할 것인지 입력이 가능합니다. 복수의 스케줄을 생성할 수 있으며, 설정을 완료한뒤엔 저장버튼클릭과 적용스위치를 눌러줍니다. 이 과정까지 진행하셨다면 작동규칙이 FarmOS에 적용된 것이며 규칙의 목적대로 ‘매일 설정된 스케줄마다 파라미터 관수제어’를 명령합니다.

적용한 작동규칙은 메뉴의 작동규칙 > 조회하고싶은 구역 을 클릭하면 구역에 적용된 규칙과 동작상태를 확인할 수 있습니다.

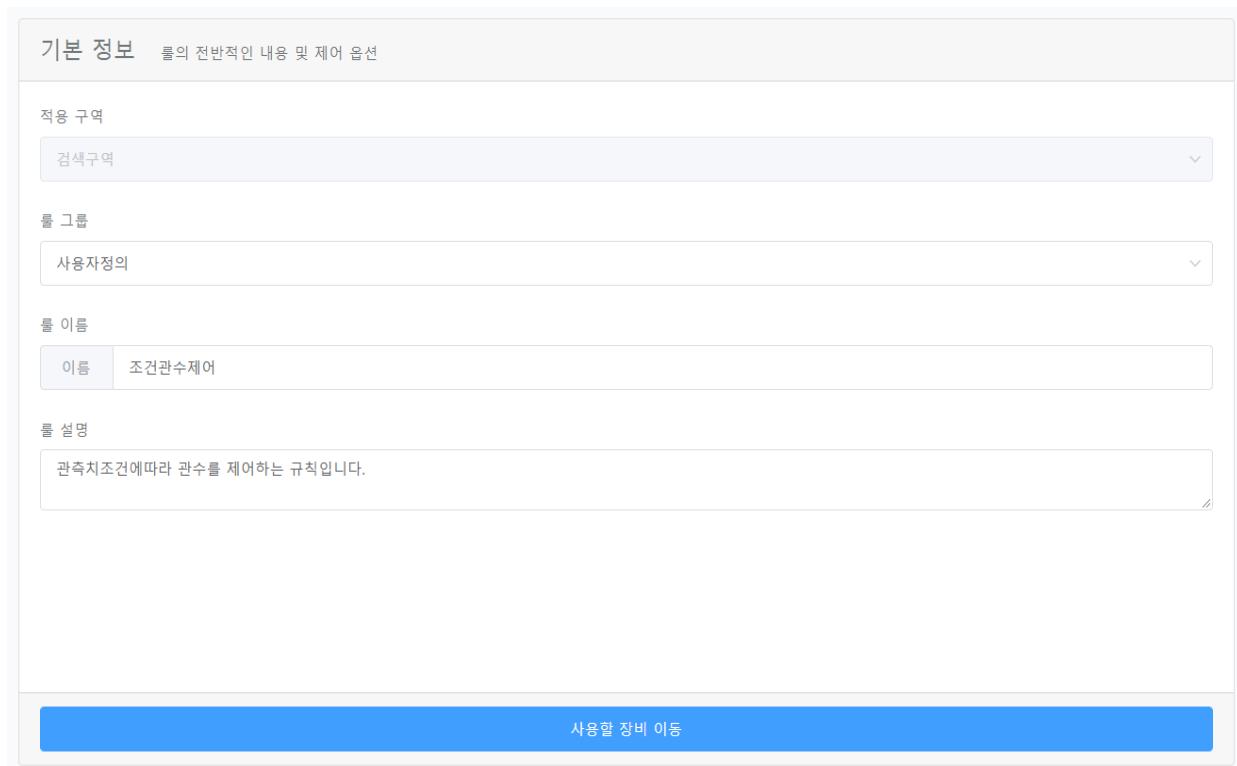
### b. 사용자 작동규칙 생성방법

사용자 작동규칙은 각 단계를 통해 원하는 설정등을 부여하여 사용자가 원하는 규칙을 직접만들 수 있는 기능입니다. 센서 관측치(예시 : EC 센서관측치)의 값에 따라 파라미터 관수 명령을 내리는 작동규칙을 예시로 사용자 작동규칙생성법을 알아보겠습니다.

메뉴에서 작동규칙 > 사용자 작동규칙을 클릭합니다.



사용자 작동규칙 메뉴에 들어오면, 위 사진과 같이 6단계가 표시되며 이 단계를 통해 사용자는 자신이 원하는 작동규칙을 생성할 수 있습니다.



1 단계는 기본정보 입력입니다. 규칙을 적용할 구역과 규칙에 해당되는 를 그룹을 선택하고, 를 이름과 설명을 작성합니다. 양액기가 설치된 구역을 적용구역으로 해야 합니다.

사용할 장비 를의 전반적인 내용 및 제어 옵션

+

장비 추가

장비 옵션 여부

nutrient-supply/level3

장비 데이터

데이터 선택

데이터 개수

삭제

장비 옵션 여부

EC센서

장비 데이터

관측치

데이터 개수

삭제

기본 정보 이동

사용할 데이터 이동

2 단계는 사용할 장비 선택화면입니다. 장비추가를 클릭하여 사용을 원하는 장비를 선택합니다. 위 사진과 같이 제어를 할 양액기와, 제어 조건을 구성하는데 필요한 EC센서를 선택했고, 관측치를 통해 조건을 구성하기 위해 장비데이터의 관측치를 선택하였습니다.

작동규칙 생성

<input checked="" type="checkbox"/> 기본 정보
<input checked="" type="checkbox"/> 사용할 장비
<input checked="" type="checkbox"/> 사용할 데이터
<input checked="" type="checkbox"/> 사용할 상수
<input checked="" type="checkbox"/> 제어 조건
<input checked="" type="checkbox"/> 결과물

사용할 데이터 를의 전반적인 내용 및 제어 옵션

데이터 추가

+

데이터 선택

+

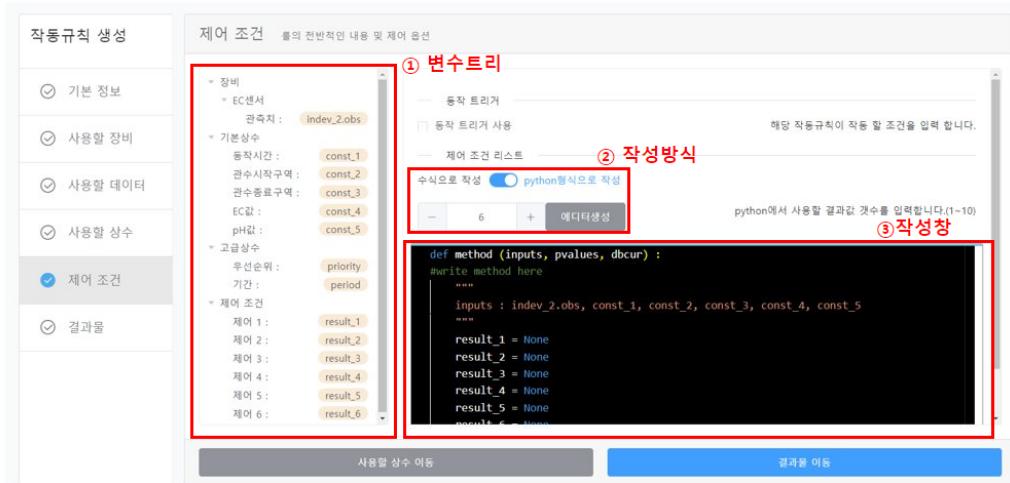
Copyright © Jinong. All rights reserved.

3 단계는 사용할 데이터 선택화면입니다. 주로 FarmOS에서 연산 또는 설정을 통해 생성되는 데이터를 설정합니다. 규칙 생성에 필요할 경우 필요한 데이터를 선택합니다. (현재 예시로 세팅한 룰에서는 사용하지 않습니다.)

Copyright © Jinong. All rights reserved.

4단계는 사용할 상수를 설정합니다. 상수추가를 설정하고 이름과 설명, 원하는 타입을 선택합니다. 여기서 설정하게 된 상수값은 규칙을 모두 생성한 뒤에 입력값을 받을 수 있는 설정값이 됩니다. 현재 규칙에서는 양액기의 파라미터 관수를 위해 필요한 동작시간, 관수시작구역, 종료구역, EC, pH값을 입력받을 수 있도록 상수를 추가한 모습입니다.

위와 같이 상수값을 설정하여 룰을 작성하게 되면, 룰을 작동시킬 때 사용자가 원하는 값을 입력할 수 있습니다. 아래의 이미지는 작동규칙을 모두 생성한 뒤에, 규칙설정화면에서 원하는 값을 입력할 수 있도록 자동으로 UI가 구성된 모습입니다. 룰을 적용하고자 하는 시점에서 결정되어야 할 설정값이 있다면 여기에 포함하는 것이 좋습니다.



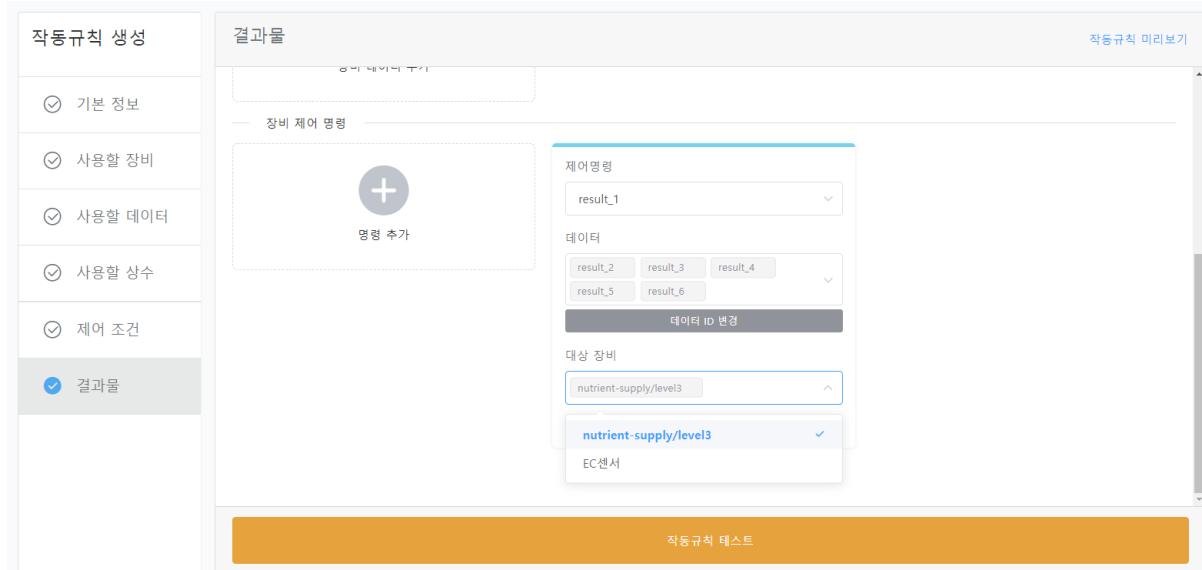
5단계는 제어조건 입력화면입니다. ① 변수트리는 지금까지 생성과정의 단계에서 설정한 값을 조합하고, ③작성창에서 제어조건을 입력하기 위한 변수포맷을 보여줍니다. ②작성방식은 수식작성, python코드형식 작성이 있으며 예시에서는 python 형식을 이용했습니다. 단순한 수식으로 처리가 가능하다면 수식으로 작성할 선택합니다.

파이썬으로 작성할 경우 result를 이용하기 위해 필요한 result갯수를 입력 후 ‘에디터 생성’버튼을 클릭 한뒤에 아래 규칙을 작성합니다.

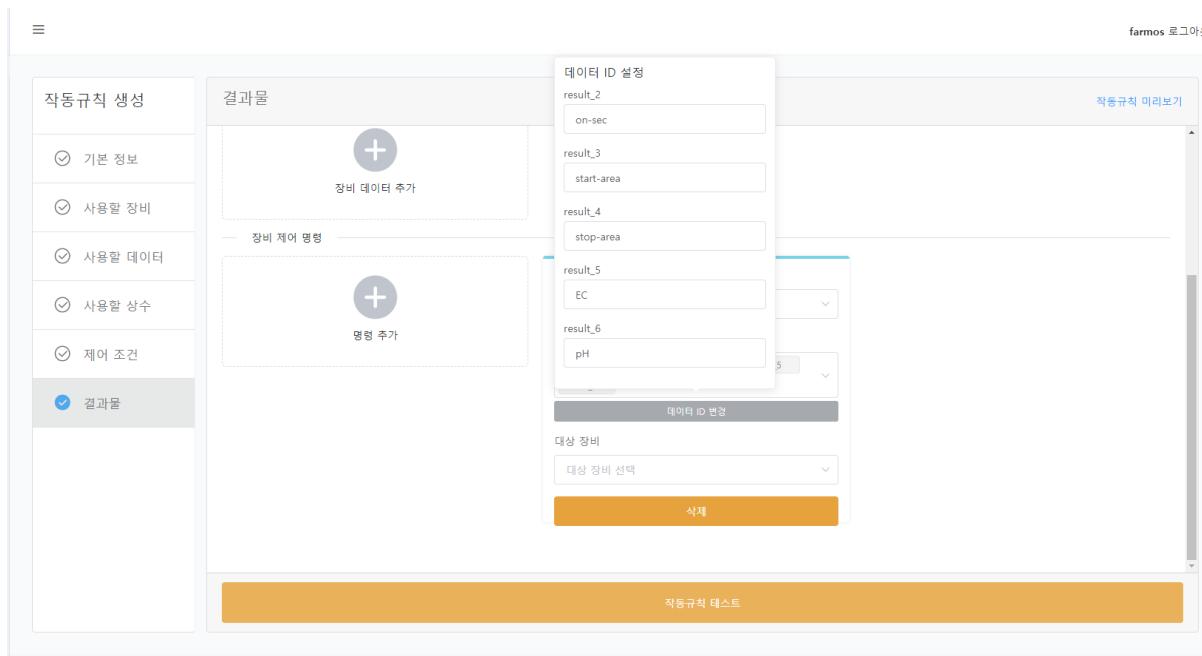
양액기 제어 명령에 필요한 인자는 동작시간(on-sec), 시작구역(start-area), 종료구역(stop-area), EC값 (EC), pH값 (pH)로 5가지입니다. 여기에 원하는 제어명령까지 총 6개의 결과물을 생성해야 합니다. 제어명령으로는 0 혹은 403을 사용합니다. 0은 정지, 403은 파라미터 관수에 해당합니다. 아래의 파이썬 코드를 보면 6개의 result를 확인할 수 있습니다.

```
def method (inputs, pvalues, dbcur) :
    #write method here
    """
    inputs : indev_2.obs, const_1, const_2, const_3, const_4, const_5
    """
    result_1 = None
    result_2 = None
    result_3 = None
    result_4 = None
    result_5 = None
    result_6 = None
    ec_obs = inputs["#indev_2.obs"].getvalue()
    on_sec = inputs["#const_1"].getvalue()
    start_area = inputs["#const_2"].getvalue()
    stop_area = inputs["#const_3"].getvalue()
    EC = inputs["#const_4"].getvalue()
    pH = inputs["#const_5"].getvalue()
    if ec_obs > 2 :
        result_1 = 403
        result_2 = on_sec
        result_3 = start_area
        result_4 = stop_area
        result_5 = EC
        result_6 = pH
    else :
        result_1 = 0
    return (RetCode.OK, [result_1,result_2,result_3,result_4,result_5,result_6])
```

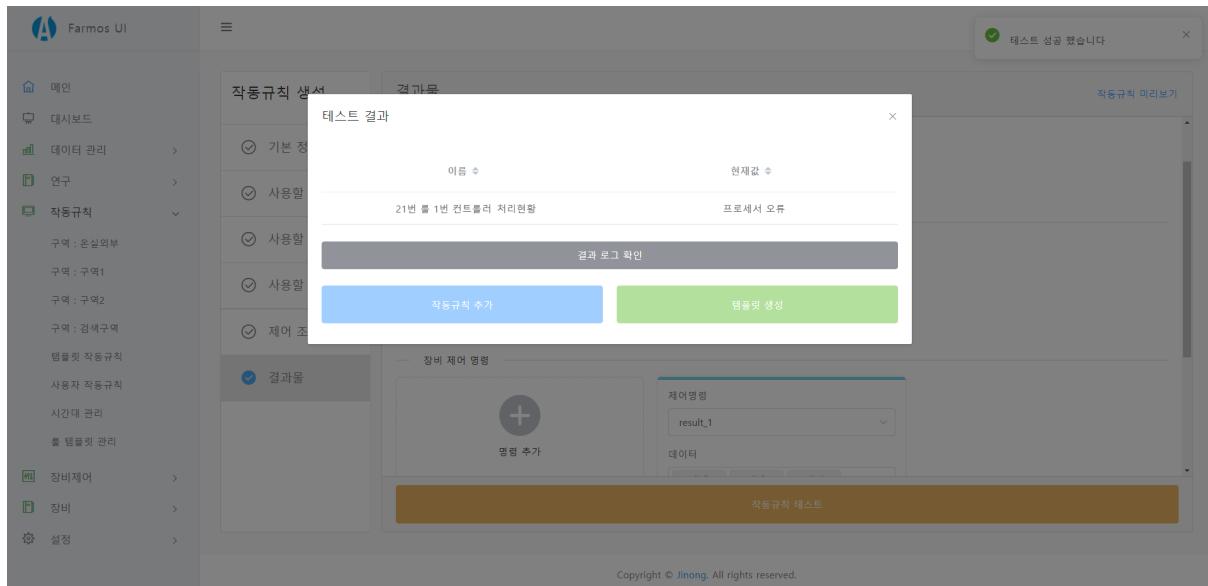
변수트리에 표시된 EC센서의 관측치 혹은 상수값들은 `input["변수명"].getvalue()`를 통해서 획득할 수 있습니다. 위의 예시는 사실 특별한 내용은 없습니다. EC관측치가 2보다 크면 파라미터 관수명령을 전송하고, EC관측치가 2보다 작거나 같으면 정지하는 명령을 전송합니다.



제어조건을 원하는 제어에 맞게 작성한 뒤에 결과물로 이동합니다. 이 화면은 만들어진 결과물을 어떻게 활용할지를 결정하는 화면입니다. 제어조건에서 작성한 결과를 양액기를 제어하기 위한 명령을 생성했기 때문에, 장비 제어 명령을 추가합니다. 명령은 위에 설명한 것처럼 `result_1`이며, 함께 보내는 파라미터 데이터는 `result_2~6`이니 그에 맞게 설정합니다. 명령을 받을 대상 장비는 양액기로 설정합니다.

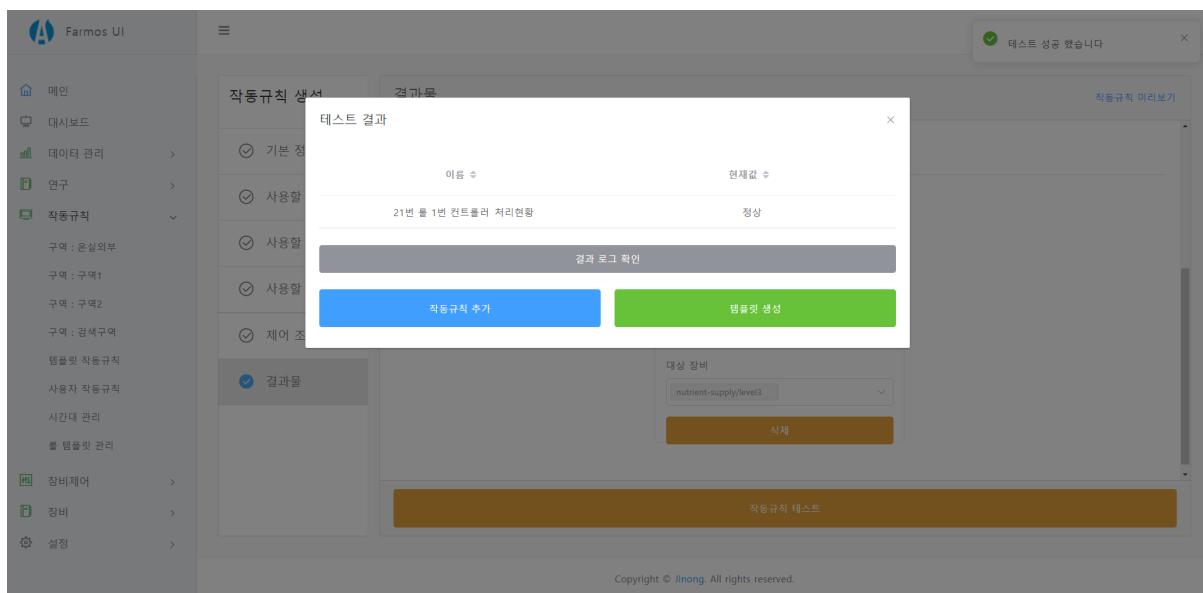


마지막으로 만들어진 각각의 `result`가 어떤 파라미터를 의미하는지 알려줘야 합니다. 이를 위해 데이터 ID변경 버튼을 눌러서 양액기에 보내는 파라미터에 맞게 아이디를 설정해줍니다. 대소문자에 주의해서 입력을 해야 합니다.

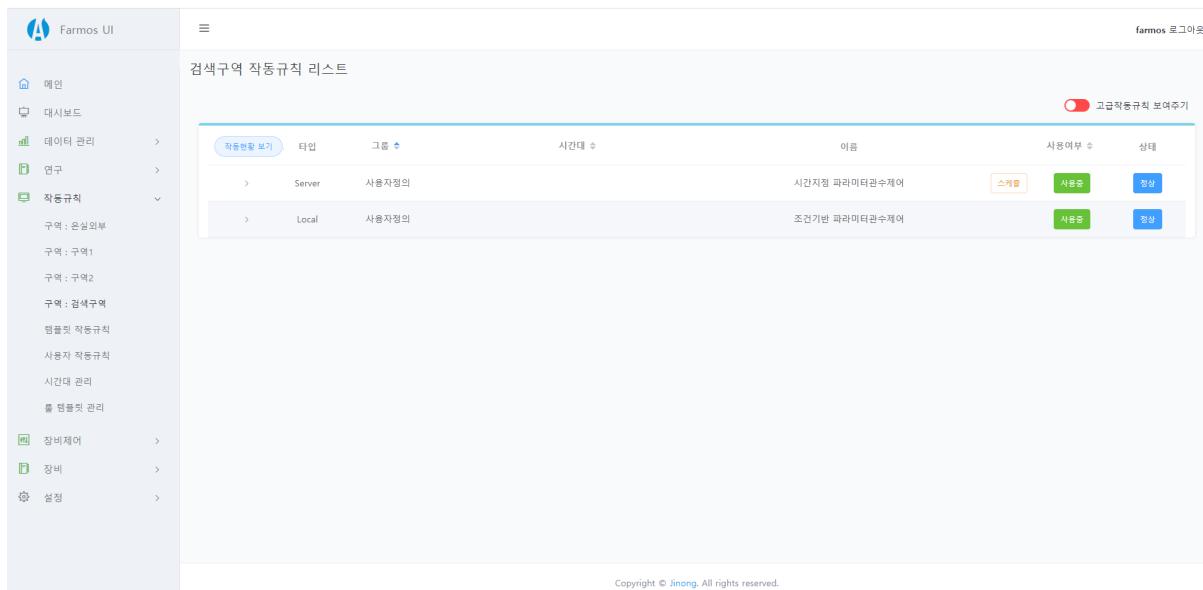


세팅이 완료되었다면 자동규칙 테스트 버튼을 클릭합니다. 규칙을 생성하여 저장하기 전에 FarmOS가 테스트를 실행해보고, 동작을 확인합니다. 결과 로그 확인 버튼을 통해 어디서 오류가 발생했는지 확인할 수 있으며, 이를 확인하여 규칙을 수정합니다. 작성한 파이썬 코드에 문제가 있다면 아래와 같이 오류가 발생합니다.





최종적으로 테스트결과가 정상이라면, 작동규칙 추가 버튼을 눌러 작동규칙을 FarmOS에 적용하도록합니다.



적용한 작동규칙은 메뉴의 작동규칙 > 조회하고싶은 구역 을 클릭하면 구역에 적용된 규칙과 동작상태를 확인할 수 있습니다.

한가지 아쉬운 점은 이미 적용된 작동 규칙의 내용을 변경하기 위해서 다시 코드를 확인하는 기능은 UI로 제공되지 않습니다. 기존의 작동규칙을 정지 혹은 삭제하시고, 새로운 사용자 작동규칙을 처음부터 다시 만드셔야 합니다. 따라서 완성되지 않은 작동규칙이라면 별도로 코드를 저장해 놓으시기를 추천드립니다.

## 5. Nscomm 을 활용한 양액기 샘플코드

```
#!/usr/bin/env python3
#
# -*- coding: utf-8 -*-
#
# Copyright (c) 2021 JiNong, Inc.
# All right reserved.
#
# User Sample
# 아래의 코드는 적당히 양액기처럼 동작하기 위한 샘플로 실제 구동과는 차이가 있습니다.
#
import time
import nscomm.communicator as nss
from nscomm.log import log

def get_kist_register_map():
    defmap = nss.get_default_register_map()
    defmap.remove_sensor(2)      # not use EC 3 sensor
    defmap.remove_sensor(5)      # not use pH 3 sensor
    defmap.change_nutrient_supply(3)    # use nutrient-supply / level3
    return defmap

option = {"method": "tcp", "host": "192.168.1.76", "port": 2222}
defmap = get_kist_register_map()

sencnt = 0
control = 1
opid = 0
supplyingstatus = 0      # 0 ready  401 preparing  402 supplying  403
stopping

communicator = nss.start_ns_communicator(option, defmap)

# 초기화
nss.update_node_status(communicator, {"status": 0, "opid": 0, "control": control})

while True:
    log.info("loop" + str(control) + str(supplyingstatus))
    # 센서 정보 획득

    # 센서 데이터 업데이트
    if sencnt % 60 == 0:
        for idx in range(2):
            nss.update_sensor_status(communicator, idx, {"status": 0,
"value": 2.3})
    sencnt = sencnt + 1

    ctrlinfo = nss.get_node_control_info(communicator)
    if ctrlinfo:
        # 노드 명령 처리
        print ("node control info", ctrlinfo)
        # 노드 상태 업데이트
```

```

control = ctrlinfo["control"]
nss.update_node_status(communicator, {"status": 0, "opid" :
ctrlinfo["opid"], "control": control})
time.sleep(1)

if nss.should_stop(communicator) > 0: # 중지명령이 있다면

    ctrlinfo = nss.get_nutrient_control_info(communicator)
    if control == 2 and ctrlinfo: # 원격제어 상태에서 명령을 받았다면
처리함. 그렇지 않으면 무시함.
        print ("nutrient supply stop control info", ctrlinfo)
        # 양액기 상태 업데이트
        supplyingstatus = 0
        opid = ctrlinfo["opid"]
        nss.update_nutrient_status(communicator,
{"status":supplyingstatus, "opid":opid, "area" : 0, "alert" : 0})

    elif supplyingstatus == 0: # 공급중이 아니라면
        ctrlinfo = nss.get_nutrient_control_info(communicator)
        if control == 2 and ctrlinfo : # 원격제어 상태에서 명령을 받았다면
처리함.
            supplyingstatus = 401
            print ("nutrient supply control info", ctrlinfo)
            # 양액기 상태 업데이트 - 실제로 이렇게 구현하면 안됨.
            nss.update_nutrient_status(communicator,
{"status":supplyingstatus, "opid":opid, "area" : 0, "alert" : 0})
            time.sleep(3)

    else:
        # 중지명령도 없고, 양액기가 동작중이라면, 할일을 계속함.
        print("nutrient supply is working")
        if supplyingstatus == 401:
            supplyingstatus = 402
            nss.update_nutrient_status(communicator,
{"status":supplyingstatus, "opid":opid, "area" : 1, "alert" : 0})
            time.sleep(5)

        elif supplyingstatus == 402:
            supplyingstatus = 403
            nss.update_nutrient_status(communicator,
{"status":supplyingstatus, "opid":opid, "area" : 0, "alert" : 0})
            time.sleep(3)

        elif supplyingstatus == 403:
            supplyingstatus = 0
            nss.update_nutrient_status(communicator,
{"status":supplyingstatus, "opid":opid, "area" : 0, "alert" : 0})

    time.sleep(1)

nss.stop_ns_communicator(communicator)

```