Introduction to fcore rule

작성자 : 김준용(tombraid@snu.ac.kr)
작성일 : 2019.09.01

1. 개요

본 문서는 fcore에서 사용하는 룰에 대한 간략한 설명을 다룬다. fcore는 farmos의 실시간 룰엔진으로 수집된 센서데이터를 바탕으로 명령을 생성하거나 새로운 데이터를 생성하는 일을 담당한다. (실시간 룰엔진이라고 썼지만 이벤트기반 엔진은 아니고 비교적 짧은시간에 작동하는 배치엔진이라고 볼 수 있다. 작동주기는 룰별로 설정이 가능하다.)

2. 룰 vs 룰 템플릿

fcore를 이해하기 위해서는 룰과 룰 템플릿을 구분할 수 있어야 한다. 간단히 말하면 룰 템플릿은 룰의 틀에 해당한다.

예를 들어 일사량을 이용해서 적산 일사를 계산하는 룰이 있다고 가정해보자. 그런데, 어떤 농장에 일사량계가 2개 있다면, 적산 일사를 계산하는 룰은 어떻게 작동해야 할까?

여기에 룰 템플릿의 필요성이 있다. farmos에는 장비를 플러그앤플레이 방식으로 연결할 수 있기 때문에 위와 같은 상황이 발생할 수 있다. (기존 스마트팜에는 정해진 장비만 들어가기 때문에 유연한 룰을 지원할 필요가 없다.)

룰 템플릿은 연구자 혹은 개발자가 작성하는 룰이라고 생각하면 좋다. 어떤 종류의 센서가 필요한지를 결정하고 어떻게 계산할지를 정하는 방식으로 작성한다. 룰 템플릿 작성은 룰의 작동방식을 정확히 이해하고, json과 간단한 프로그램 작성 능력이 필요하다

사용자 혹은 관리자는 룰 템플릿을 사용하여 룰을 생성한다. 필요한 센서를 적용하면 룰 생성이 완료된다. 룰을 적용할때는 설치된 센서의 정확한 명칭만 알면 특별한 지식없이 처리가 가능하다.

3. 시간대 룰 vs 계산 룰

시간대 룰은 하루를 원하는 구간으로 나누는 룰이다. 일반적으로 온실은 심야, 일출전, 일출후, 낮시간, 일몰전, 일몰후 정도의 시간대에 따라서 다른 환경설정으로 동작시키는 것이 일반적이다. 따라서 이러한 시간대를 구분하는 것이 필요하고 시간대 룰은 이런 역할을 수행한다.

계산 룰은 일반 룰 이라고 이해하면 된다. 어떤 계산을 수행할지를 정하고 있는 룰이다.

사실 일반 룰은 시간대룰과 계산 룰을 모두 가져야 한다. 하지만 시간대 룰의 경우 여러 룰에서 공통적으로 사용이 가능하기 때문에 별도로 분리하여 공통사용이 가능하도록 정리한 것이다.

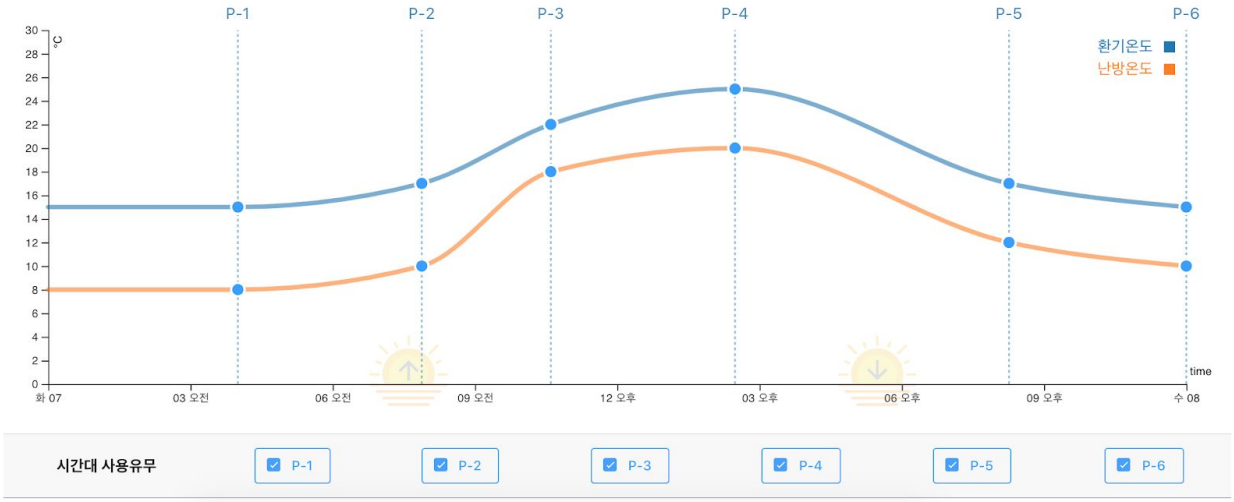
4. 시간대 룰

시간대 룰은 고정형 시간대와 일출일몰시를 사용한 시간대로 구분할 수 있다. 고정형 시간대는 7시, 8시 이렇게 정해진 시간대역을 사용하고, 일출 일몰시는 일출 1시간 전, 일몰 2시간 후 이런식으로 시간대를 계산한다.

fcore에서는 일출일몰시를 사용하면 고정형 시간대도 사용할 수 있기 때문에 대부분의 룰을 일출일몰시를 기준으로 동작한다고 보면 된다.

시간대 룰은 임계치를 포함한다. 특정 시간대에 관리하고자 하는 값을 임계치라고 한다. 특정 시간대에서 환기 온도와 난방온도를 정한다면 이런 값이 임계치가 될 수 있다.

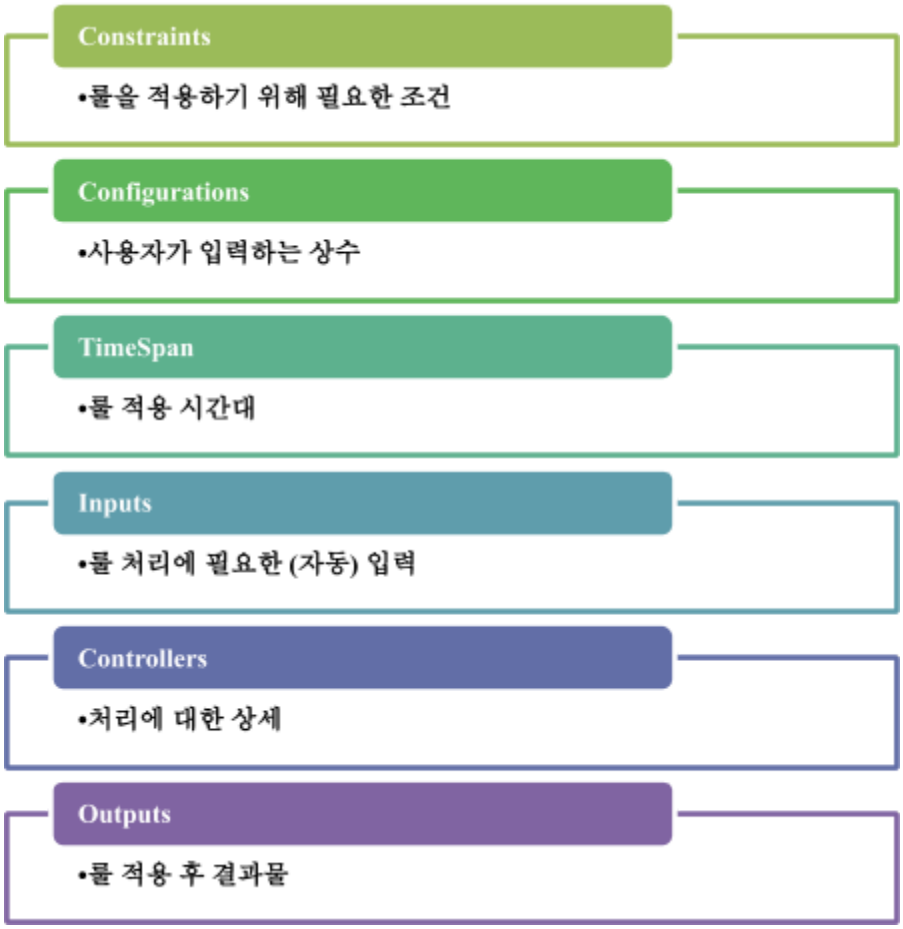
아래 이미지는 farmos에서 시간대를 관리하는 UI이다. 6개의 시간대 (P-1 ~ P-6) 를 나누고 있고, 각 시간대의 환기온도와 난방온도를 설정하고 있다.



5. 계산 룰 (일반 룰)

정해진 시간대에 특정한 계산을 수행해서 구동기를 구동하거나 (명령을 만들거나) 새로운 지표값을 계산하는 역할을 담당한다.

계산 룰의 작동을 위해서는 다음의 7가지가 정리되어야 한다.



fcore 에서 룰은 디비에서 데이터값을 읽는 것으로 시작한다. 읽혀진 데이터값은 컨트롤러에서 처리가 되고, 새로운 데이터값(지표, 가상센서값 등), 명령(과 인자) 으로 만들어져 전달되거나 디비에 저장된다.

- 센싱값이라고 하지 않고 데이터값이라고 하는 이유는 센서 이외의 경로로 발생되는 데이터들도 처리하기 위함이다.
- current_observations 라는 테이블에서만 데이터를 획득하는 것으로 한다.

룰 수행의 결과물은 항상 **float형 숫자의 배열**이다. 다만, 이 숫자를 해석하는 방법은 결과물의 종류에 따라 약간씩 다를 수 있다. 지표(가상센서값)인 경우에는 그 값을 그대로 해석하면 된다.

6. 간단한 예시

다음은 내외부온도차를 구하는 룰 템플릿의 예시이다. 룰 템플릿에서는 Constraints가 중요하다. (룰이 적용되는 시점에 Inputs가 자동으로 설정되기 때문에 템플릿에서는 Inputs를 다룰 필요가 없다.)

a. Constraints (제약조건) : 어떤 장비가 필요한지를 결정한다.
내외부온도차를 구하기 위해서는 내부 온도 센서와 외부 온도 센서가 필요하다. 아래의 예는 본 룰에서 온도센서를 2개 사용해야 한다는 사실을 지정한다.

```
{
  "target": "field",
  "devices": [
    {
      "class": "sensor",
      "type": "temperature-sensor",
      "desc": "내부온도센서를 선택해주세요.",
      "inputs": {
        "key": "#intemp",
        "codes": [
          0,
          1
        ]
      },
      "name": "내부온도센서"
    },
    {
      "class": "sensor",
      "type": "temperature-sensor",
      "desc": "외부온도센서를 선택해주세요.",
      "inputs": {
        "key": "#outtemp",
        "codes": [
          0,
          1
        ]
      },
      "name": "외부온도센서"
    }
  ]
}
```

여기서 inputs 파트가 중요하다. inputs는 특정 온도센서가 선택되었을때 해당 센서의 정보를 입력으로 사용하기 위한 정보를 다룬다. 여기서는 사용자가 어떤 내부온도센서를 선택하면 해당 센서의 정보가 intemp라는 변수명에 저장된다는 의미가 된다.

codes 데이터 postfix에 해당하는데 여기서는 구체적으로 다루지 않는다. 그냥 0이면 장비의 상태, 1이면 관측치라고 생각하면 된다.

간단히 정리하면 intemp0 이라는 변수에는 내부온도센서의 상태가, intemp1이라는 변수에는 내부온도의 관측치가 저장된다.

b. Configuration (설정) : 룰을 작동시킬때 필요한 사용자 입력값이다.
내외부온도차를 구하기 위해서 별도의 사용자 입력은 필요하지 않다. 만약 이동평균을 구하기 위한 룰이었다면 몇개의 데이터로 이동평균을 구할지 등의 설정이 필요할 수 있다.
다만, 룰의 우선순위와 작동주기는 항상 포함되어야 한다.


```
{
  "basic": [],
  "advanced": [
    {
      "key": "priority",
      "name": "우선순위",
      "value": 2,
      "minmax": [
        0,
        5
      ],
      "description": "룰의 우선순위"
    },
    {
      "key": "period",
      "name": "기간",
      "value": 180,
      "description": "룰의 작동주기"
    }
  ],
  "timespan": {
    "id": 0,
    "used": [
      true
    ]
  }
}
```

c. TimeSpan (시간대) : 룰이 사용할 시간대를 지정한다.
내외부온도차는 특정한 시간대에 구애받을 필요없이 동작하면 된다. 따라서 시간대 id를 0으로 설정한다. 룰의 샘플은 Configuration의 샘플 하단을 참고한다.

d. Inputs (입력) : 룰 템플릿에서는 별도로 다룰 필요가 없다. (사용자가 장비를 선택하는 시점에서 자동으로 설정된다.)

e. Controllers (컨트롤러) : 실제 계산을 수행할 방법을 지정한다.
컨트롤러는 트리거와 프로세서로 나뉠 수 있다. 트리거는 프로세서를 작동시켜도 되는지를 평가한다. 간단한 예를 들면 온도센서가 고장이라면 굳이 내외부온도차를 계산하지 않아도 된다.
트리거와 프로세서는 동일한 구조의 컨트롤러를 사용하는데, 다양한 모드의 컨트롤러가 존재하지만 여기서는 eq모드에 대해서 다룬다. eq모드는 정해진 수식을 평가하는 방식으로 동작한다.

```
{
  "trigger": {
    "type": "eq",
    "eq": "intemp0 == 0 and outtemp0 == 0"
  },
  "processors": [
    {
      "type": "eq",
      "eq": "intemp1 - outtemp1",
      "outputs": [
        "#inoutdiff"
      ]
    }
  ]
}
```

추가적인 설명을 위해 위의 예에 대해서 설명한다.

트리거의 모드(타입)은 eq이고, 사용할 수식(eq)는 “intemp0 == 0 and outtemp0 == 0” 이다. 여기서 intemp0은 내부온도센서의 상태이고, outtemp0은 외부온도센서의 상태인데, 둘다 정상(0)이라면 트리거가 참이된다.

프로세서는 여러개 설정될 수 있는데, 여기서는 한개만 설정되어 있다. 프로세서의 모드(타입) 역시 eq이고, 사용할 수식(eq)는 "intemp1 - outtemp1” 이다. 즉 내부온도센서의 관측치 에서 외부온도센서의 관측치를 뺀값이 된다. 이 값은 inoutdiff 라는 변수에 저장된다.

f. Outputs (출력) : 룰의 실행된 결과를 다룬다.

를의 실행결과는 명령이거나 데이터가 된다. 여기서 내외부온도차만 계산한 것이므로 이경우 룰의 실행 결과는 데이터가 된다. 아래의 예시는 해당 데이터에 대한 정보를 다룬다.

```
{
  "data": [
    {
      "name": "내외부온도차",
      "outputs": "#inoutdiff",
      "outcode": 21,
      "unit": "℃"
    }
  ]
}
```

최종적으로 계산된 inoutdiff는 farmos의 디비에 내외부온도차라는 이름으로 단위는 ℃ 로 저장되게 된다.

여기서 outcode값은 다른 룰에서 내외부온도차를 활용할때 의미가 있다. 간단히 설명하면 위에서 본 예시에서 0, 1의 code 를 확인하였다. 0은 장비의 상태, 1은 관측치를 의미한다. 이와 같은 방식으로 사용자가 내외부온도차를 쉽게 룰에서 사용하기 위해 설정할 수 있는 값정도로 이해하면 된다.