

튜토리얼: Java SDK로 라즈베리파이 연결하기

IoTmakers를 통하여 센서 값을 수집/제어 하고 이벤트를 생성하는 방법을 소개한다.

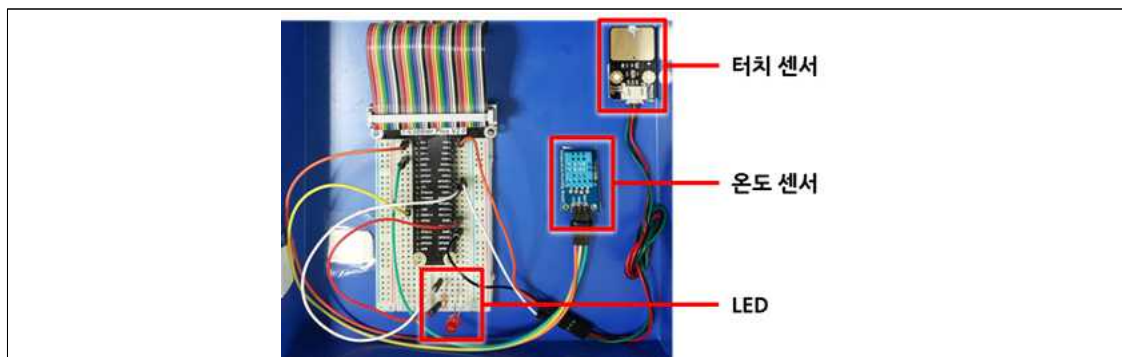
1. 튜토리얼 목표

- IoTmakers의 Java SDK(TCP)를 사용하여 라즈베리파이 연동하기
- IoTmakers를 통해 LED 제어하기 (On/Off)
- IoTmakers를 통해 온도/터치 센서 값 수집하기
- IoTmakers를 통해 이벤트 등록하기

2. 준비사항

RASPBERRY PI, 스타터 키트/입출력 키트, LED, DHT11-11 센서, 디지털 접촉식 터치센서, 노트북, (키보드, 마우스, 모니터) 등

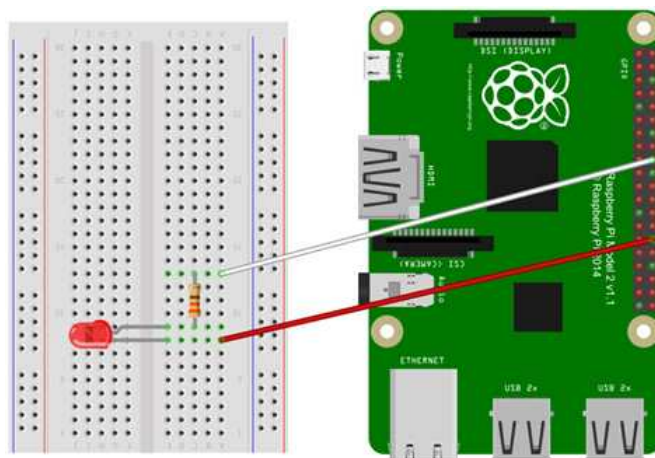
3. 하드웨어 구성



<전체 구성 >

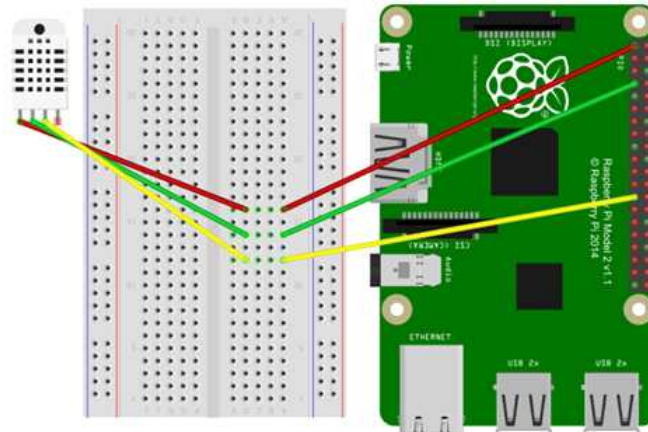
3.1 LED 제어

- 부품 : LED, Resistor Anything between 330 to 1k ohms, 점퍼 케이블
- GPIO Pin 설정 : GPIO Pin 5 (BCM_24)
GND



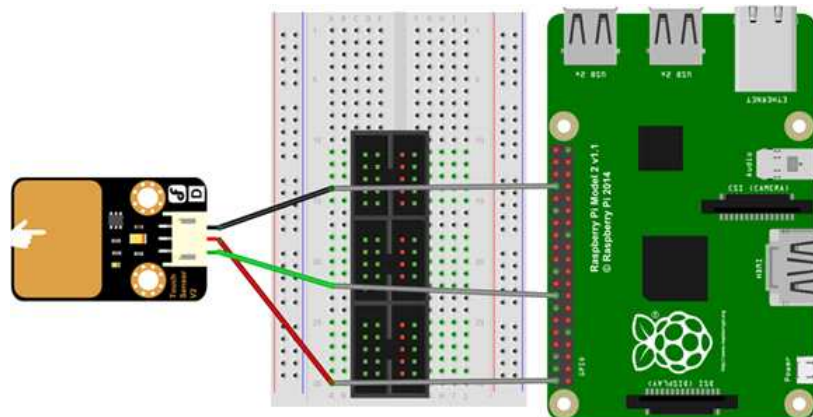
3.2 DHT-11 온습도 센서

- 부품 : DHT 11 Humidity & Temperature Sensor
- GPIO Pin 설정 : BCM_GPIO 4 : Data out (GPIO 7)
Power : 3.3V
GND



3.3 디지털 접촉식 터치센서(DFR0030)

- 부품 : 디지털 접촉식 터치센서(DFR0030)
- GPIO Pin 설정 : BCM_GPIO 23 : Data input
Power : 5V
GND



4. 환경 구성

4.1 GPIO(General Purpose Input Output) 설정

라즈베리파이는 범용적인 목적의 입/출력을 담당하는 GPIO(General Purpose Input/Output) 포트를 가지고 있다. 라즈베리파이에서 GPIO를 사용하는 방법은 여러가지가 있는데 그 중 Pi4J라는 것을 이용해서 GPIO를 사용해보자. Pi4J는 라즈베리파이에서 각종 기능들을 쉽게 접근할 수 있도록 만들어준 Java 기반 GPIO 사용 라이브러리이다.

○ Pi4J Library 설치

- Pi4J Project : pi4j.com/download.html

- 다운로드 : pi4j-1.0.zip
 ※ 사용 방법은 아래에 소개할 'IoT Makers를 통해서 센서 태그 정보 수집하기'에서 설명함

4.2 DHT-11 온습도 센서 for Python Library

DHT-11 센서는 라즈베리파이에서 일반적으로 사용하는 protocol과 다르게 "Manchester-esque"라 불리는 protocol을 사용하고 있다. 이는 Adafruit Library를 이용하여 쉽게 사용할 수 있다.

- Adafruit's DHT sensor library 설치
 - working directory : /home/pi/Downloads/DHT/Python/

```
$ cd /home/pi/Downloads/DHT/Python/
$ git clone https://github.com/adafruit/Adafruit_Python_DHT.git
$ cd Adafruit_Python_DHT
$ sudo apt-get update
$ sudo apt-get install build-essential python-dev
$ sudo python setup.py install
```

- 설치 확인

```
$ cd examples/
$ sudo ./AdafruitDHT.py 11 4
```

```
pi@raspberrypi: ~/Downloads/DHT/Python/Adafruit_Python_DHT/examples
파일(F) 편집(E) 탭(T) 도움말(H)
pi@raspberrypi: ~/Downloads/DHT/Python/Adafruit_Python_DHT/examples $ sudo ./AdafruitDHT.py 11 4
Temp=27.0° Humidity=31.0%
pi@raspberrypi: ~/Downloads/DHT/Python/Adafruit_Python_DHT/examples $ sudo ./AdafruitDHT.py 11 4
Temp=27.0° Humidity=31.0%
pi@raspberrypi: ~/Downloads/DHT/Python/Adafruit_Python_DHT/examples $ sudo ./AdafruitDHT.py 11 4
Temp=29.0° Humidity=30.0%
pi@raspberrypi: ~/Downloads/DHT/Python/Adafruit_Python_DHT/examples $ sudo ./AdafruitDHT.py 11 4
Temp=29.0° Humidity=30.0%
pi@raspberrypi: ~/Downloads/DHT/Python/Adafruit_Python_DHT/examples $
```

※ 사용 방법은 아래에 소개할 'IoT Makers를 통해서 센서 태그 정보 수집하기'에서 자세히 설명함

5. 디바이스 등록

실습을 위한 준비를 마쳤다면 IoT Makers에 접속해서 디바이스를 등록해보자. 처음 가입하면 다음과 같이 디바이스 등록 화면을 볼 수 있다.

나의 디바이스

아직 등록된 디바이스가 없습니다. 첫번째 디바이스를 등록 해보세요!

디바이스 생성

디바이스 명을 입력하고 프로토콜 유형을 선택한다. 그리고 IoTmakers의 Java TCP SDK를 사용하여 라즈베리파이를 연동할 것이므로 다음과 같이 kt 표준 인터페이스/TCP를 선택한다.

디바이스 등록

디바이스 정보 등록

기본정보 (+탈수입력)

디바이스 명

edu raspberry pi

디바이스 아이디

XXXXXXXXXX

수정

디바이스 패스워드

XXXXXXXXXX

수정

프로토콜 구분

☒ 플랫폼 표준 ☐ 사설

프로토콜 유형

kt 표준 인터페이스 TCP(Stream)

상세정보 (선택입력)

취소

디바이스 생성

6. 센서 태그 등록

디바이스 목록에서 생성한 디바이스를 선택하면 다음과 같이 상세정보를 볼 수 있다. 이제 센서 태그를 등록하기 위해서 태그 스트림을 생성하자.

나의 디바이스

디바이스 상세 정보

디바이스 목록

디바이스명 : edu raspberry pi

수정 삭제 삭제

디바이스 이미지

디바이스 위치보기

디바이스 아이디

XXXXXXXXXX

디바이스 패스워드

XXXXXXXXXX

사용자 정의 모델명

XXXXXXXXXX

제조사명

정보 없음

프로토콜 유형

나트론 인터페이스 / TCP(Stream)

Gateway 연결 ID

XXXXXXXXXX

카테고리

기타

생성일 / 최근 사용일

2016-04-14 / 정보 없음

공개여부 / 사용여부

비공개 / 사용

Tag Stream

SDK

Event

아직 등록된 Tag Stream이 없습니다. 첫번째 Tag Stream을 등록 해보세요!

디바이스를 통해 수집된, 특정한 속성(습도, 온도 등)의 데이터 타입으로 구성된 데이터 집합이 Tag Stream입니다.

사용자의 디바이스가 사용하는 목적에 따라 추출할 수 있는 Tag Stream을 숫자와 그 외의 데이터들로 등록하고 관리할 수 있습니다.

Tag Stream 생성

총 3개의 태그스트림을 추가해보자. 이번 실습에서는 온도/터치 센서를 모니터링 하고 LED를 제어할 것이므로 다음과 같이 태그스트림을 등록한다. (Unit은 필수 입력 값이 아니다.)

센서	Tag Stream ID	Tag Stream Type	Value Type
LED	LED	제어	문자열
온도 센서	Temp	수집	숫자형식
터치 센서	Touch	수집	숫자형식

Tag Stream

SDK

Event

태그 스트림 생성

Tag Stream ID

Unit(% ,℃ , \$, W)

Tag Stream Type

Value Type

LED

제어

문자열

생성

등록을 완료하면 아래와 같이 태그스트림 목록에 생성한 태그 스트림을 확인할 수 있다.



7. Java SDK(TCP) 다운로드

디바이스와 태그스트림 등록이 모두 끝났으니 이제 C SDK를 다운받아 보자.

IoTmakers SDK(Software Development Kit)는 IoT 디바이스가 IoTmakers 플랫폼에 연동할 때에 필요한 API(Application Programming Interface)를 제공한다. 이 API를 통하여 플랫폼에 접속 및 디바이스를 인증한 후, 수집데이터를 전송하고 플랫폼으로부터 제어데이터를 수신할 것이다. SDK는 아래와 같이 디바이스 상세정보 페이지 내 SDK 탭에서 다운로드 받을 수 있다.



다운로드 받은 SDK의 압축을 푼 후, IoTmakers 플랫폼에 디바이스를 연동하기 위하여 등록된 디바이스 정보를 기준으로 IoTSDK.properties 파일을 설정하자.

- 파일 위치 : /home/pi/Downloads/JSDK/SDK_JAVA/IoTSDK.properties

디바이스 정보는 아래와 같이 디바이스 상세 정보에서 확인할 수 있다. 예시와 같이 연동하고자 하는 디바이스 정보를 입력한다.

나의 디바이스

디바이스 상세 정보

디바이스 목록

디바이스명 : raspberry pi

수정 삭제 제거

디바이스 위치보기

디바이스 아이디	XXXXXXXXXX	제조사명	정보 없음
디바이스 패스워드	XXXXXXXXXX	Gateway 연결 ID	XXXXXXXXXX
사용자 정의 모델명	XXXXXXXXXX	카테고리	- 기타
프로토콜 유형	키트론 인터페이스 / TCP(Stream)	생성일 / 최근 사용일	2016-04-14 / 정보 없음
공개여부 / 사용여부	비공개 / 사용		

```

2 # Server IP
3 sdk.serverIp= XXX.XX.XXX.XX
4 # Server Port
5 sdk.serverPort= XXXXX
6 # Service ID
7 sdk.externalSysId= Gateway 연결 ID 입력
8 # Device ID
9 sdk.deviceId= 디바이스 아이디 입력
10 # Password
11 sdk.password= 디바이스 패스워드 입력

```

플랫폼 정보

디바이스 정보

8. IoTMakers를 통해서 센서 태그 정보 수집하기

8.1 pi4j jar 파일 추가

먼저 '환경 구성'에서 다운로드 받았던 pi4j 라이브러리를 사용하기 위하여 pi4j-core.jar 파일을 SDK 폴더에 추가한다.

8.2 Python Library 사용하기 (for DHT-11 온습도 센서)

이제 '환경 구성'에서 다운로드 받았던 Adafruit's DHT sensor library 를 활용하여 온습도 센서 값을 읽어오는 코드를 작성해보자. 소스코드는 다운로드 받은 라이브러리의 simpletest.py 샘플 코드를 활용해보자

- 파일 위치 : /home/pi/Downloads/DHT/Python/Adafruit_Python_DHT/examples/

파일명을 dhtlib.py로 변경하고 다음 위치에 파일을 위치시킨 후, 아래와 같이 소스코드를 수정해보자.

- 파일 위치: ./pi/Downloads/JSDK/SDK_JAVA/dhtlib.py


```

13 # The above copyright notice and this permission notice shall be included in all
14 # copies or substantial portions of the Software.
15
16 # THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
17 # IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
18 # FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
19 # AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
20 # LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
21 # OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
22 # SOFTWARE.
23 import Adafruit_DHT
24
25 # Sensor should be set to Adafruit_DHT.DHT11,
26 # Adafruit_DHT.DHT22, or Adafruit_DHT.AM2302.
27 sensor = Adafruit_DHT.DHT11
28
29 # Example using a Beaglebone Black with DHT sensor
30 # connected to pin P8_11.
31 # pin = 'P8_11'
32
33 # Example using a Raspberry Pi with DHT sensor
34 # connected to pin 23.
35 pin = 4
36
37 # Try to grab a sensor reading. Use the read_retry method which will retry up
38 # to 15 times to get a sensor reading (waiting 2 seconds between each retry).
39 humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
40
41 # Note that sometimes you won't get a reading and
42 # the results will be null (because Linux can't
43 # guarantee the timing of calls to read the sensor).
44 # If this happens try again!
45 if humidity is not None and temperature is not None:
46     print '{0:0.1f}'.format(temperature, humidity)
47 else:
48     print 'Failed to get reading. Try again!'
49

```

센서 정보

pin number

입력이 완료되면 다음과 같이 테스트 해보자. 현재 온도가 표시될 것이다.

```

$ sudo chmod a+x dhtlib.py (a : all user, x : Execute)
$ sudo ./dhtlib.py

```

이제 최종적으로 dhtlib.py를 import하여 온습도 센서를 읽어오는 자바 함수를 구현하는 것이 필요하다. 이는 아래 소스코드와 같이 구현해보자. (getValue() 함수 참고)

8.3 소스코드 작성

IoTmakers를 통해 센서 태그 정보를 수집하기 위한 소스코드를 작성해보자. 소스코드는 SDK내에 있는 샘플코드를 활용한다

- 파일 위치: ./pi/Downloads/JSDK/SDK_JAVA/Sample.java

기존 샘플코드는 IoTmakers플랫폼에 접속 후에 1초 간격으로 태그스트림 데이터를 전송하는 예제이며, 이번 튜토리얼에서는 IoTmakers와 연동하여 데이터를 송신하고 제어데이터를 수신하여 디바이스에 전달하는 코드를 작성할 것이다.

raspberry.java 라는 파일을 생성한 후 다음과 같은 소스코드를 작성해보자.


```

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.util.Date;
import java.util.Map;

import com.kt.smcp.gw.ca.comm.exception.SdkException;
import com.kt.smcp.gw.ca.gwfrwk.adap.stdsys.sdk.tcp.BaseInfo;
import com.kt.smcp.gw.ca.gwfrwk.adap.stdsys.sdk.tcp.IMCallback;
import com.kt.smcp.gw.ca.gwfrwk.adap.stdsys.sdk.tcp.IMTcpConnector;
import com.kt.smcp.gw.ca.util.IMUtil;
import com.pi4j.io.gpio.GpioController;
import com.pi4j.io.gpio.GpioFactory;
import com.pi4j.io.gpio.GpioPinDigitalOutput;
import com.pi4j.io.gpio.PinState;
import com.pi4j.io.gpio.RaspiPin;
import com.pi4j.io.gpio.GpioPin;
import com.pi4j.io.gpio.GpioPinDigitalInput;
import com.pi4j.io.gpio.PinPullResistance;

public class raspberry extends IMCallback
{
    private GpioController gpio = null;
    private GpioPinDigitalOutput pin1 = null;
    private static GpioPinDigitalInput pin2 = null;

    public raspberry() {
        // get a handle to the GPIO controller...
        gpio = GpioFactory.getInstance();

        // creating the pin with parameter PinState.HIGH...
        // will instantly power up the pin...
        pin1 = gpio.provisionDigitalOutputPin(RaspiPin.GPIO_05, "PinLED",
        PinState.HIGH);
        pin2 = gpio.provisionDigitalInputPin(RaspiPin.GPIO_04, "PinTC",
        PinPullResistance.PULL_DOWN);
    }

    public static void main(String[] args) throws Exception {
        // callback fuction call...
        raspberry callback = new raspberry();
        IMTcpConnector tcpConnector = new IMTcpConnector();
        BaseInfo baseInfo = null;

        Long transID;
        Long timeOut = (long)3000;

        try {
            baseInfo = IMUtil.getBaseInfo("IoTSDK.properties");
            tcpConnector.init(callback, baseInfo);

            tcpConnector.connect(timeOut);
            tcpConnector.authenticate(timeOut);

            while(true) {

```

```

        transID = IMUtil.getTransactionLongRoundKey4();

        // Temp Teg value send...
        tcpConnector.requestNumColecData("Temp",    getValue(),    new
Date(), transID);
        Thread.sleep(1000);

        if(pin2.isLow()) {
            // System.out.println("Not Detect !!");
        } else {
            // System.out.println("Detected !!");

            tcpConnector.requestNumColecData("Touch", (Double)1.0, new
Date(), transID);
        }
    }

    } catch(SdkException e) {
        System.out.println("Code   :" + e.getCode() + "   Message   :" +
e.getMessage());
    }
}

// Temp sensor value...
private static Double getValue() throws Exception {
    // run 'Python process' and get Temp value...
    Runtime run = Runtime.getRuntime();
    Process proc= run.exec("sudo python dhtlib.py");
    BufferedReader      stdInput      =      new      BufferedReader(new
InputStreamReader(proc.getInputStream()));

    Double temperature;

    // read the output from the command...
    String s = null;
    String sOut = "";

    while((s = stdInput.readLine()) != null) {
        sOut = sOut + s;
    }

    if(!(sOut.contains("ERR_RANGE") || sOut.contains("ERR_CRC"))) {
        temperature = Double.parseDouble(sOut);
        return temperature;
    } else {
        return null;
    }
}

@Override
public void handleColecRes(Long transId, String respCd) {
    System.out.println("Collect Response. Transaction ID   :" + transId + "
Response Code : " + respCd);
}

```



```

1 #####
2 # This is a project standard makefile..
3 #####
4 JAVAC = javac
5 JAVA = java
6
7 #####
8 # IOTMAKERS_SDK_HOME
9 #####
10 IOTMAKERS_SDK_HOME = ..
11 IOTMAKERS_SDK_LIBNAME = $(IOTMAKERS_SDK_HOME)/dist/JAVA_TCP_SDK_2.0.0.jar
12 THIRDPARTY_LIB_PATH = $(IOTMAKERS_SDK_HOME)/lib
13
14 #####
15 # FLAGS
16 #####
17 OUTDIR = .
18 JFLAGS = -g -Xlint
19 JDPATH = -d $(OUTDIR)
20
21 JCPATH = -classpath $(OUTDIR):$(IOTMAKERS_SDK_LIBNAME):$(THIRDPARTY_LIB_PATH)/*:./pi4j-core.jar
22
23 #####
24 # Compile
25 #####
26 .SUFFIXES: .java .class
27
28 .java.class:
29     $(JAVAC) $(JFLAGS) $(JCPATH) $(JDPATH) $*.java
30
31 #####
32 # SOURCE TREE
33 #####
34 JAVA_SOURCE = \
35     ./raspberry.java \
36
37 #####
38 # BUILD
39 #####
40 default: classes
41 classes: $(JAVA_SOURCE:.java=.class)
42
43 #####
44 # Util
45 #####
46 clean:
47     $(RM) *.class
48
49 run:
50     $(JAVA) $(JCPATH) raspberry

```

pi4j jar 파일 추가

raspberry 파일명 변경

raspberry 파일명 변경

작성을 완료하였으면 이제 소스를 Compile 해보자

```

$ make clean
$ make
$ sudo make run

```

컴파일 하면 다음과 같이 RASPBERRY PI에서 IoTMakers로 전송/수신한 로그를 확인할 수 있다.

```

pi@raspberrypi: ~/Downloads/JSDK/SDK_JAVA/Test
파일(F) 편집(E) 탭(T) 도움말(H)
pi@raspberrypi: ~/Downloads/JSDK/SDK_JAVA/Test $ sudo make run
java -classpath ../../dist/JAVA_TCP_SDK_2.0.0.jar:../lib/*:./pi4j-core.jar raspbe
rry
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further detail
s.
Collect Response. Transaction ID : 1457417280001 Response Code : 100
Collect Response. Transaction ID : 1457417290002 Response Code : 100
Collect Response. Transaction ID : 1457417290003 Response Code : 100
Collect Response. Transaction ID : 1457417290004 Response Code : 100
Collect Response. Transaction ID : 1457417290005 Response Code : 100
Collect Response. Transaction ID : 1457417290006 Response Code : 100
Handle Control Request Called. Transaction ID : 1457417304037
0 Number Type controls. 1 String Type controls.
Tag Stream : LED Value: ON
LED ON
Collect Response. Transaction ID : 1457417300007 Response Code : 100
Collect Response. Transaction ID : 1457417300008 Response Code : 100
Collect Response. Transaction ID : 1457417300009 Response Code : 100
Collect Response. Transaction ID : 1457417300010 Response Code : 100
Collect Response. Transaction ID : 1457417300011 Response Code : 100
Collect Response. Transaction ID : 1457417310012 Response Code : 100

```


9. 결과 확인

또한 IoT Makers 포털 화면에서 라즈베리파이로부터 수집된 정보를 텍스트, Json 로그나 차트로 실시간으로 확인할 수 있다.

- ‘ON/OFF’ 문자를 통한 LED 제어

[illegible]

- Temp (온도) 값의 수신



The screenshot shows the 'Stream' configuration page in the 'Stream Studio' application. The page is titled 'Stream' and has a 'Tap Stream' button. The 'Stream ID' is 'Stream1'. The 'Stream Type' is 'Tap Stream'. The 'Value Type' is 'Integer'. The 'Stream' table shows a list of values ranging from 10 to 100. The 'Value' column shows the corresponding values, and the 'Stream' column shows the stream name.

- Touch(=1) 값의 수신

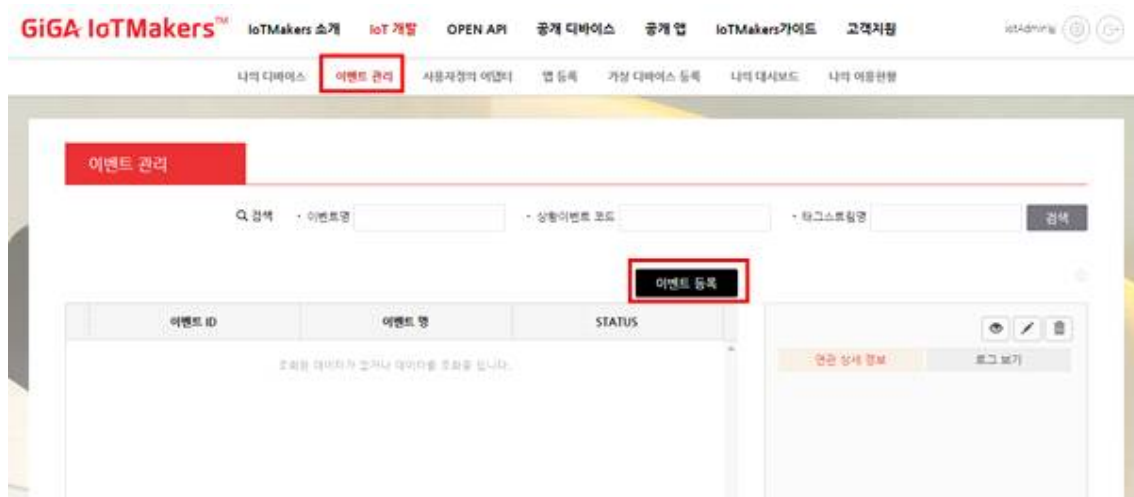
[illegible]

10. 이벤트 등록

센서로부터 값을 수신하였으면 마지막으로 이벤트를 등록해보자.

이벤트란 디바이스의 태그스트림에서 올라오는 값 중 사용자가 설정한 특정 조건에 부합할 때 발생하는 사건을 말한다. IoTmakers를 통해 등록한 디바이스의 Tag Stream을 활용하여 이벤트를 생성하고, 이벤트 발생 시 액션을 취하게끔 나만의 시나리오를 작성할 수 있다. 본 튜토리얼에서는 온도 센서로부터 수집된 값이 10도 보다 높으면 LED를 ON하는 이벤트를 만들어 볼 것이다.

이제 IoT개발> 이벤트 관리 페이지에서 이벤트를 등록해보자.



10.1 이벤트 등록

이벤트 등록 버튼을 누르면 다음과 같은 이벤트 에디터 화면을 볼 수 있다.



10.2 '디바이스 데이터' 선택

디바이스 데이터를 캔버스에 drag&drop 한 후, 이벤트를 발생시킬 디바이스를 선택한다.



10.3 '이벤트 정의' 등록

다음과 같이 이벤트 정의를 등록한다.



10.4 디바이스와 이벤트의 연결

디바이스로부터 데이터를 수집하여 이벤트를 발생시키기 위하여 디바이스 박스와 이벤트 정의박스를 연결시킨다.



10.5 ‘이벤트 발생 조건’ 추가/생성

이벤트 발생 조건은 이벤트 정의 박스 내에 위치시켜야 한다. ‘온도가 10도 보다 높으면 ’이라는 이벤트 조건을 만들기 위해 논리식 입력기를 사용하여 다음과 같은 논리식을 만들어 보자.

디바이스명.태그스트림명 => 10

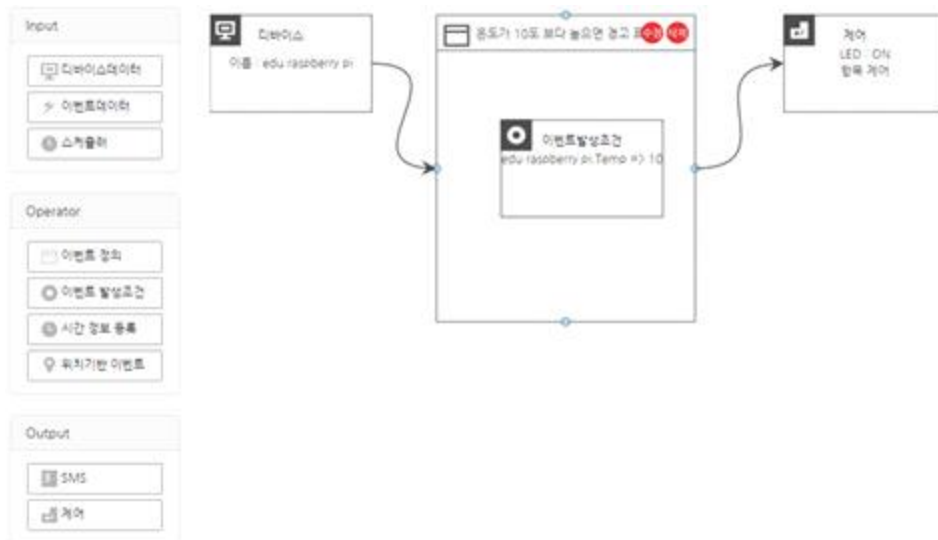
10.6 제어 조건 추가/저장

온도가 10도 보다 높으면 LED를 ON하기 위하여 제어 조건을 추가한다.



10.7 이벤트 등록 완료/저장

이벤트가 발생하면 LED를 제어하기 위하여 이벤트 정의 박스와 제어 박스를 연결시킨다. 아래와 같이 완성한 후 저장 버튼을 클릭하면 이벤트 등록이 완료된다.



10.8 이벤트 리스트 로그 확인

목록에서 해당 이벤트를 클릭한 후 로그 보기 탭을 선택하면 이벤트 발생 로그를 확인할 수 있다. 온도가 10도 이상일 때 이벤트가 발생하며 동시에 LED가 ON되는 것을 확인할 수 있다.

이벤트 관리

🔍 검색

• 이벤트명

• 담당이벤트 코드

• 태그소속팀명

검색

이벤트 등록

	이벤트 ID	이벤트 명	STATUS
1	001PTL001D10001371	온도가 10도 보다 높으면 경고 표시	ON
2	001PTL001D10001369	온도 온도가 10이상이면 이벤트는 발생 시	ON

온도가 10도 보다 높으면 경고 표시

받은 상태 정보

로그 보기

온도가 10도 보다 높으면 경고 표시[이벤트]가
발생하였습니다
2018-03-08 15:42:31.990

온도가 10도 보다 높으면 경고 표시[이벤트]가
발생하였습니다
2018-03-08 15:42:31.990

온도가 10도 보다 높으면 경고 표시[이벤트]가
발생하였습니다
2018-03-08 15:42:31.990

온도가 10도 보다 높으면 경고 표시[이벤트]가
발생하였습니다
2018-03-08 15:42:31.990