Thing+ Embedded Guide

Thing+ Embedded는 다양한 IoT 기기들이 Thing+ Cloud에 연결될 수 있도록 프로토콜을 정의하고 있으며, IoT 기기(thing) 제조사들이 Thing+ 연동을 쉽게 할 수 있도록 라이브러리를 제공하고 있습니다. 또한, IoT 기기와 Thing+ Cloud 연결과 하드웨어에 맞는 부가적인 기능이 포함된 Thing+ Gateway 프로그램도 포함이 됩니다.

0. Thing+ Overview

홈페이지 참조

1. Thing+ Embedded Overview

Thing+ Embedded는 Thing+ MQTT 프로토콜, Thing+ Embedded SDK, Thing+ Gateway로 구성됩니다.
Thing+ MQTT 프로토콜은 Thing+ Cloud와 하드웨어 사이에 사용하는 MQTT 토픽과 메시지를 정의하고있습니다. Thing+ Embedded SDK는 Thing+ MQTT를 구현한 라이브러이입니다(개발중). Thing+ Gateway는 Daliworks에서 개발한 Thing+ MQTT 프로토콜을 따르는 소프트웨어입니다.

Thing+와 하드웨어를 연동하는 방법은 3가지가 있습니다.

1. Thing+ MQTT 프로토콜을 참고하여 소프트웨어 작성
    - 하드웨어 업체는 Thing+ MQTT 프로토콜 문서와 C로 작성된 예제 코드를 참고하여 IoT 기기에 구동되는 Thing+ 연동 소프트웨어를 개발하실 수 있습니다. 개발해야 할 항목은 Thing+ Cloud 연결 및 데이터 송수신, MQTT 메시지 생성 및 분석, 시간동기 등이 있습니다. 개발 자유도가 가장 높으며, 하드웨어에 최적화 된 소프트웨어를 작성하실 수 있습니다.

2. Thing+ Embedded SDK 라이브러리 이용하며 소프트웨어 작성
    - **(준비중)**

3. Thing+ Gateway를 사용하고, Device Agent 작성
    - Thing+ Gateway는 Thing+ Cloud와의 연결을 담당하고, Device Agent는 센서값을 읽고 액추에이터를 동작시키는 일을 합니다. 하드웨어 업체든 Device Agent 부분을 작성하시면 됩니다. **단, Thing+ Gatweay를 사용하기 위해선 하드웨어에 Node.js가 설치되어야 하며, RTOS, Micro OS, Firmware를 이용하는 하드웨어는 사용할 수 없습니다.** Thing+ Gateway와 Device Agent 사이의 프로토콜 문서와 예제코드를 참고하실 수 있습니다.

1.0 Guidelines for things

Thing+와 연동하는 IoT 기기(thing)는 다음사항을 충족시켜야 합니다.

1. 하드웨어는 SSL, TLS, MQTT 이용해 Thing+ Cloud에 접속이 가능해야 합니다.
2. 하드웨어의 로컬시간은 Thing+에서 서버시간과 동기화되어야 합니다.
3. 하드웨어와 Thing+ Cloud간 접속이 끊어졌을 경우 하드웨어는 3시간동안 센서값을 저장하여 데이터 유실을 최소화 해야 합니다.
4. 네트워크가 재연결되면 저장한 센서값을 Thing+ Cloud에 전송해야 합니다.
5. 하드웨어는 주기적으로 기기의 상태와 센서의 상태를 Thing+ Cloud에 전송해야 합니다.
6. 하드웨어에 시리즈 센서가 있을 경우, 전송 주기를 지켜야합니다. 최소 전송 주기는 1분입니다.
7. 전송주기가 변경될 경우, 하드웨어는 새로운 전송 주기에 맞쳐 센서값을 올려야 합니다.
8. 하드웨어는 MQTT 구독을 Thing+에서 정한 토픽만 해야합니다.

1.1 Thing+ Embedded에서 정의한 개념 설명

1.1.1 Event Sensor, Series Sensor, Report Interval

Thing+는 센서의 종류에 따라 이벤트(Event) 센서와 시리즈(Series) 센서로 구분하고 있습니다. 이벤트 센서는 센서값이 변할 때 센서값을 즉시 Thing+에 전송하는 센서입니다. 시리즈 센서는 하드웨어에서 센서값을 주기적으로 전송하는 센서입니다.
리포트 인터벌(Report Interval)은 시리즈 센서의 전송 주기를 의미하며, 최소 리포트 인터벌은 60초입니다.

1.1.2 Sensor, Actuator lists

Thing+는 지원하는 센서와 액추에이터를 정의하고 있습니다. 센서, 액추에이터 타입은 Thing+에서 정한 고유의 문자열이며 IoT 기기 제조사들은 Thing+에서 정한 타입을 사용해야 합니다. 만약 잘못된 타입을 사용하면, Thing+ Portal에 아이콘, 그래프 등이 잘못 표시될 수 있습니다.

대표적인 센서의 목록은 아래와 같습니다.

> **센서목록**

Type	Category	Type	Category
number	series	onoff	event
string	series	motion	event
percent	series	countEvent	event
temperature	series	door	event
humidity	series	accelerometer	series

액추에이터는 타입뿐만 아니라 명령어도 Thing+가 정의하고 있습니다.

> **액추에이터 목록**

Type	Command	Type	Command
led	on off blink	lcd	print clear
powerSwitch	on off	buzzer	on off
camera	snapPicture		

Thing+가 지원하는 센서, 액추에이터의 전체 목록은 센서,엑츄에이터 정의파일(Thing+ 포털 로그인 후 확인 가능)에서 확인할 수 있습니다.

만약 사용하시는 센서, 액추에이터가 목록에 빠져있으면 아래 사이트에 가셔서 양식을 작성하시기 바랍니다.
TODO : 센서 등록 폼 링크 삽입

1.1.3 Gateway ID, APIKEY

게이트웨이 아이디(Gateway ID)는 thing을 구분하기 위한 12자리 문자열입니다. 모든 IoT 기기는 고유의 게이트웨이 아이디를 정의해야하며,

일반적으로 유선랜의 MAC어드레스를 사용합니다.
APIKEY는 Thing+에서 발급하는 KEY로 MQTT 인증에 사용됩니다. Thing+ 포털에 게이트웨이 아이디를 등록하면 APIKEY를 발급하실 수 있으며, **발급된 APIKEY는 해당 thing에만 유효합니다. 하드웨어는 발급받은 APIKEY를 저장하고 있어야하며, MQTT 접속 시 인증 비밀번호로 APIKEY를 사용해야 합니다.**

2. Thing+ Embedded 프로토콜

Thing+는 IoT 기기와 주고 받는 데이터 형식을 사전에 정의해 두었으며, 이를 Thing+ Embedded 프로토콜이라고 합니다. thing은 Thing+에서 정의한 형태로 데이터를 구성하여 Thing+로 전송해야 하며, Thing+도 Embedded 프로토콜에서 정의한 형식에 따라 데이터를 전송합니다.

Thing+와 IoT 기기들 사이 사용하는 네트워크 프로토콜은 MQTT와 HTTP입니다. MQTT는 IoT 기기에서 수집한 센서값, 액츄에이터 명령어 등이 전송이 되며, HTTP로는 thing에서 사용하는 디바이스, 센서를 추가할 때 사용됩니다. Thing+ Portal에서 디바이스와 센서를 추가하는 게이트웨이는 HTTP 프로토콜은 사용 안하셔도 됩니다.

2.1 MQTT

MQTT(Message Queuing Telemetry Transport)는 경량 메시지 프로토콜로 낮은 대역폭과 낮은 전력을 사용하는 IoT 기기와 Thing+ Cloud 사이에 사용되는 프로토콜입니다. MQTT는 Publish/Subscribe 구조로 되어으며, TCP/IP를 통해 구현됩니다. SSL 및 TLS를 사용하여 데이터 보안을 할 수 있고, USERNAME/PASSWORD 기반의 인증방법을 제공하고 있습니다.

MQTT 브로커(Broker)는 다양한 클라이언트들이 메시지를 주고 받을 수 있도록 메시지를 전달하는 역할을 합니다. Thing+ Cloud는 MQTT 브로커를 제공하고 있으며, IoT 기기는 Thing+ Cloud가 제공하는 MQTT 브로커에게 센서값을 전송(Publish)하고, 액추에이터 명령을 수신(Subscribe)하여 명령에 맞는 동작하면 됩니다.

메시지를 생성하는 클라이언트(Publisher)는 메시지 앞에 토픽(Topic)을 붙여 발행하고, 메시지를 구독하는 클라이언트(Subscriber)는 수신할 토픽을 브로커에 등록합니다. MQTT 브로커는 토픽을 기준으로 메시지를 중계합니다. 토픽은 계층이 있으며, 토픽 구분을 위해 슬래시(/)를 사용합니다.

```
토픽 예)
HOME0/BEDROOM/TEMERTATURE
HOME0/BEDROOM/HUMIDITY
HOME0/LIVING/CO2_LEVEL
HOME1/KITCHEN/DUST_LEVEL
```

MQTT는 3단계의 QoS(Quality of Service)를 제공합니다.

- QoS0 : 메시지는 한 번만 전송되며, 전송의 성공, 실패를 확인하지 않습니다. 메시지 유실 가능성이 있습니다.
- **QoS1** : 메시지는 한 번 이상 전송될 수 있습니다. 메시지는 항상 정확하게 전달되며, 전달과정 중 중복으로 전달될 수 있습니다. **Thing+ Cloud는 QoS1을 사용하고 있습니다.**
- QoS2 : 메시지는 정확히 한번만 전송이 되며, 메시지 유실 및 중복 전송을 유발하지 않습니다. 하지만 메시지 전송에 많은 네트워크 대역폭이 요구됩니다.

MQTT는 클라이언트 접속이 끊어졌을 경우, 다른 클라이언트에게 전송할 메시지를 저장할 수 있는 Last Will and Testament(LWT) 기능을 제공합니다. 클라이언트는 Will 메시지를 정의할 수 있으며, 브로커는 이 메시지를 저장하고 있습니다. 클라이언트가 에러에 의해 접속이 끊어졌을 경우 브로커는 Will 메시지를 다른 클라이언트에게 전송합니다. Thing+는 thing의 에러 상태를 Will 메시지로 사용하고 있으며, thing에 에러가 발생 시 Thing+ Cloud는 즉각 기기의 에러를 알 수 있습니다.

2.2 Thing+ MQTT Protocol

Thing+에서 사용하는 MQTT 토픽과 데이터 포멧에 대해서 설명합니다. **Thing+와 연동을 원하는 하드웨어 업체는 이 장에서 정의한 토픽과 데이터 포멧에 맞게 센서값을 전송해야하며, Thing+도 이 프로토콜에 기반하여 액추에이터 명령어를 전송합니다.**

- MQTT 메시지는 **QoS1**을 사용하며, WiLL 메시지를 제외한 모든 메시지의 RETAIN은 사용하지 않습니다.
- 시간 값은 UTC를 사용합니다.
- 하드웨어는 MQTT 연결 상태, 게이트웨이 상태, 센서 상태 메시지를 전송하며 상태를 정의하는 값은 아래와 같습니다.

> | 상태 | 값(문자열) |
> | --- | --- |
> | 켜짐 | on |
> | 꺼짐 | off |
> | 에러발생 | err |

2.2.1 MQTT 연결 설정

Thing+가 사용하는 MQTT 버전은 3이며, 포트는 8883입니다. **thing이 설치되는 네트워크에 8883 포트가 막혀있으면 Thing+에 접속할 수 없습니다.**

MQTT 접속 아이디는 게이트웨이 아이디, 비밀번호는 APIKEY입니다. WiLL 메시지의 토픽은 *v/a/g/{gateway_id}/mqtt/status*이고, 메시지 내용은 'err'이고, Retain은 True 입니다. 재접속 시 세션은 새로 생성하도록 설정해야합니다.

MQTT Connection SPEC	**Thing+ Definition**
MQTT Version	3
PORT	8883
MQTT Client ID	{gateway_id}
Clean Settion	TRUE
MQTT ID	{gateway_id}
MQTT Passworkd	{APIKEY}
Will Topic	v/a/g/{gateway_id}/mqtt/status
Will Message	err
Will Message Retain	TRUE
Keep Alive[sec]	{report_interval} x 2 (**Recommend**)

2.2.2 MQTT 상태 전송

하드웨어는 MQTT 접속에 성공한 경우 MQTT의 상태가 정상임을 전송해야합니다.

> MQTT 접속 상태 전송

```
{
    "TOPIC": "v/a/g/{gateway_id}/mqtt/status"
    "MESSAGE" : "on"
}
```

2.2.3 하드웨어 상태 전송

하드웨어는 주기적으로 하드웨어 상태와 상태의 유효시간을 전송해야합니다. 만약, Thing+가 유효시간이 내에 하드웨어 상태를 재수신하지 못하면 해당 thing에 에러가 발생하였다고 판단합니다.

> 하드웨어 상태 전송

```
{
    "TOPIC": v/a/g/{gateway_id}/status
    "MESSAGE": {hw_status},{valid_time}
}
```

하드웨어 상태 전송 메시지에 센서의 상태를 붙여서 전송 할 수도 있습니다. 하나의 토픽을 이용하여 하드웨어 상태와 센서의 상태를 같이 전송할 수 있기 때문에, 네트워크 비용이 비싼 thing은 이 방법을 이용하면 네트워크 비용을 줄일 수 있습니다. 만약 이 방법을 사용하였다면, 개별 센서의 상태는 별도로 전송하지 않아도 됩니다.

> 하드웨어, 센서 상태 전송

```
{
    "TOPIC": v/a/g/{gateway_id}/status
    "MESSAGE": {hw_status},{valid_time},{sensor_id},{sensor_status},{valid_time},...,{repeat for
 sensors}
}
```

2.2.4 센서 상태 전송

하드웨어는 주기적으로 센서 상태와 상태의 유효시간을 전송해야합니다. 만약, Thing+가 유효시간이 내에 센서 상태를 재수신하지 못하면 해당 센서에 에러가 발생하였다고 판단합니다.

> 센서 상태 전송

```
{
    "TOPIC": v/a/g/{gateway_id|/s/{sensor_id}
    "MESSAGE": {sensor_status},{valid_time}
}
```

2.2.5 센서값 전송

하드웨어는 개별 센서의 값을 전송할 수 있습니다. 하드웨어는 센서값과 값을 읽은 시간을 쌍으로 전송해야하며, 한개의 센서에 대해 여러개의 센서값을 한꺼번에 전송할 수도 있습니다. 여러개의 센서값 전송 시 시간순으로 정렬이 되어있어야 합니다. 센서값과 시간을 배열로 묶은 형태도 전송이 가능합니다.

> 센서 값 전송

```
{
    "TOPIC": v/a/g/{gateway_id}/s/{sensor_id}
    "MESSAGE": {time},{value},{time},{value},...,{repeats}
}
```

> 센서 값 전송 (시간, 값을 배열로 묶음)

```
{
    "TOPIC": v/a/g/{gateway_id}/s/{sensor_id}
    "MESSAGE": [{time},{value},{time},{vaule},...,{repeats}]
}
```

2.2.6 N개 센서의 센서값 전송

하드웨어에 연결된 여러개의 센서 값을 한꺼번에 전송할 수 있습니다. 각각의 센서에 대해서 여러개의 센서값을 한꺼번에 전송할 수 있으며, 센서값은 시간순으로 정렬이 되어있어야 합니다.

> N개 센서의 센서값 전송

```
{
    "TOPIC": v/a/g/{gateway_id}
    "MESSAGE": {"{sensor_id}":[{time},{value},...,(repeats)],"{sensor_id}":[{time},{value},...,(repeats)],...  }
}
```

2.2.7 Thing+가 요청한 작업 결과 전송

Thing+는 액추에이터 명령어, 하드웨어 환경 설정 등을 위해 MQTT 메시지를 전송합니다. IoT 기기는 Thing+가 요청한 작업을 수행해야하며 수행결과를 Thing+에게 알려줘야 합니다.

> Thing+ 요청 작업 결과

```
{
    "TOPIC": v/a/g/{gateway_id}/res
    "MESSAGE IF SUCCESS": {"id":{message_id},"result":"{result}"}
    "MESSAGE IF FAILED ": {"id":{message_id},"error":{"code":{err_code}, "message":{error_message}}}
}
```

메시지 아이디는 작업을 요청한 메시지의 아이디로 자세한 내용은 다음절을 참고하시며 됩니다.
Thing+는 작업 성공 시에 수신해야 될 값을 사전에 정의하고 있으며, 센서,엑츄에이터 정의파일(Thing+ 포털 로그인 후 확인 가능)에서 확인하실 수 있습니다. 작업 성공 시 IoT 기기는 성공값을 {result} 에 적어주면 됩니다.
에러코드는 JSONRPC의 에러코드 규칙을 따르며, 자세한 에러원인은 {error_message} 에 적어주면 됩니다.

2.2.8 하드웨어가 수신하는 메시지

하드웨어는 Thing+가 발행하는 MQTT 메시지 수신을 위하여 아래 토픽을 구독해야합니다. 각 하드웨어는 자신에게 오는 MQTT 메시지만 구독할 수 있습니다.

> 하드웨어가 구독하는 토픽, 전송받는 메시지

```
{
    "TOPIC": v/a/g/{gateway_id}/req
    "MESSAGE": "id":"{msg_id}","method":"{method}","params":"{params}"
}
```

메시지는 메시지 아이디(msg_id), 메쏘드(method), 파라미터(params)로 구성되어 있습니다.
메시지 아이디는 Thing+에서 정의하는 값으로 메시지마다 고유의 값을 가집니다. 하드웨어가 명령어 수행 후 수행결과를 Thing+에 알려줄 때 이 메시지 아이디를 사용해야합니다.
메쏘드는 IoT 기기가 수행해야 할 일의 목록이며, 파라미터는 메쏘드에 따라 달라집니다. 메쏘드는 다음과 같이 정의하고 있습니다.

> Method List

Method	Description
timeSync	시간 동기화
controlActuator	액추에이터 실행
setProperty	Thing+와 연관된 하드웨어 환경 설정
poweroff	하드웨어 종료
reboot	하드웨어 재시작
restart	소프트웨어 재시작
swUpdate	소프트웨어 버전 업그레이드
swInfo	소프트웨어 버전 정보

위 메쏘드 중 timeSync, setProperty는 의무적으로 구현해야 하며, 나머지는 필요한 것들만 구현하시면 됩니다.

timeSync 메쏘드는 Thing+에서 하드웨어 시간을 설정할 때 사용합니다. thing의 로컬시간이 틀릴 경우, 센서값을 읽은 시간에 오류가 발생합니다. 하드웨어가 MQTT 접속 상태를 전송하면, Thing+는 하드웨어가 현재 시간을 동기화 할 수 있도록 서버시간을 전송합니다. 하드웨어는 수신한 시간을 기준으로 로컬시간을 재설정해야합니다.

> timeSync params

```
{
    params:{"time":{UTC}}
}

Example)
{
    "TOPIC": v/a/g/1928dbc93871/req
    "REQUEST MESSAGE": {"id":"e1kcs13b9","method":"timeSync","params":{"time":1372874401865}}
    "SUCCESS RESPONSE MESSAGE" : {"id":"e1kcs13b9","result":""}
    "ERROR REPONSE MESSAGE" : {"id":"e1kcs13b9","error":{"code": -32000, "message": "invalid opt
ions"}}
}
```

setProperty 메쏘드는 리포트 인터벌을 전달할 때 사용합니다. 리포트 인터벌의 단위는 msec입니다. Thing+ 포털에서 리포트 인터벌을 변경할 수 있습니다.

> setProperty params

```
{
    params:{"reportInterval":{interval}
}

Example
{
    "TOPIC": v/a/g/1928dbc93781/req
    "REQUEST MESSAGE": {"id":"e1kcs13bb","method":"setProperty","params":{"reportInterval":"6000
0"}}
    "SUCCESS RESPONSE MESSAGE": {"id":"e1kcs13bb","result":""}
    "ERROR RESPONSE MESSAGE": {"id":"e1kcs13bb","error":{"code":-32000,"message":"invalid interv
al"}}
}
```

controlActuator 메쏘드는 액추에이터에 명령을 내릴 때 사용합니다. Thing+ 포털, 규칙에서 액추에이터 명령을 내릴 수 있으며, 각 하드웨어는 명령에 맞는 동작을 해야합니다. 각 액추에이터의 명령어 및 옵션는 Thing+에서 정의하고 있으며, 메시지의 파라미터로 전송이 됩니다.

> 대표적인 액추에이터 명령어 및 명령어 옵션

Actuator	Command	Option
led	on	duration
led	off	
led	blink	duration interval

Actuator	Command	Option
powerSwitch	on	duration
powerSwitch	off	

전체목록은 센서,엑츄에이터 정의파일(Thing+ 포털 로그인 후 확인 가능)에서 확인하실 수 있습니다.

> controlActuator params

```
{
    params:{"id":{actuator_id},"cmd":{cmd},"options": {"cmd options"}}
}

Example : Led On
{
    "TOPIC": v/a/g/1928dbc93781/req
    "REQUEST MESSAGE": {"id":"46h6f8xp3","method":"controlActuator","params":{"id":"led-1928dbc9
3781-r","cmd":"on","options":{"duration":3000}}}
    "SUCCESS RESPONSE MESSAGE": {"id":"46h6f8xp3","result":""}
    "ERROR RESPONSE MESSAGE": {"id":"46h6f8xp3","error": {"code":-32000,"message":"invalid optio
ns"}}
}
```

powerOff는 하드웨어 종료시키는 메쏘드로 파라미터는 없습니다.
reboot은 하드웨어를 재시작하는 메쏘드로 파리미터는 없습니다.
restart은 IoT 기기의 소프트웨어를 재시작하는 메쏘드로 파리미터는 없습니다.
swUpdate은 IoT 기기의 소프트웨어를 업데이트 시키는 메쏘드로 파리미터는 없습니다.
swInfo은 IoT 기기의 소프트웨어 버전 정표를 가져가는 메쏘드로 파리미터는 없습니다.

2.3 Thing+ HTTP Protocol

Thing+ HTTP Protocol은 thing이 사용하는 REST API에 관한 프로토콜입니다. 디스커버 기능 구현 시 IoT 기기는 자신에게 연결된 디바이스와 센서의 정보를 REST API를 통해 Thing+에게 알려줍니다. 디스커버 기능은 사용하기 위해선 Thing+ HTTP Protocol은 구현은 필수입니다.

2.3.1 디바이스 등록 과정

1. 게이트웨이 정보를 얻어옵니다. (2.3.4절 참고)
    - 얻어 온 게이트웨이 정보에서 디스커버가 가능한지 판별합니다.
        - autoCreateDiscoverable 참조
            - **디스커버 하지 않다면, thing은 디바이스를 등록할 수 없습니다.**
        - 게이트웨이 정보의 model은 게이트웨이 모델을 얻어올 때 사용됩니다.
2. 게이트웨이 모델을 얻어옵니다. (2.3.5절 참고)
    - 게이트웨이 모델의 deviceModels 배열에서 사용할 디바이스 모델을 선택합니다.
    - 디바이스 모델에서 정의한 idTemplate은 디바이스 등록 시 사용됩니다.
3. 등록할 디바이스 정보를 만들어 전송합니다. (2.3.6절 참고)
   데이터 포멧은 아래와 같습니다.

```
{
  reqId: '<Device ID>',
  name: '<Device Name>',
  model: '<Device Model>
}
```

    - reqId : 디바이스 모델에 있는 idTemplate 형식에 맞게 ID를 생성합니다.
        - 일반적으로 idTemplate은 {gatewayID}-{deviceAddress}입니다.
            - gatewayID : 게이트웨이 아이디
            - deviceAddress : 게이트웨이 내에 디바이스를 구분하기 위한 값으로 게이트웨이 내에서 중복이 되면 안됩니다. 사용자 정의
    - name : 디바이스 이름. 사용자 정의
    - model : 사용할 디바이스 모델의 이름. 게이트웨이 모델 정보에서 사용할 디바이스 모델 이름입니다.

2.3.2 센서 등록 과정

1. 게이트웨이 정보를 얻어옵니다.(2.3.4절 참고)
    - 얻어 온 게이트웨이 정보에서 디스커버가 가능한지 판별합니다.

        - autoCreateDiscoverable 참조
            - **디스커버 하지 않다면, thing은 센서를 등록할 수 없습니다.**
        - 게이트웨이 정보의 model은 게이트웨이 모델을 얻어올 때 사용됩니다.

2. 게이트웨이 모델을 얻어옵니다. (2.3.5절 참고)
    - 게이트웨이 모델의 deviceModels 배열에서 사용할 디바이스 모델을 선택합니다.
    - 디바이스 모델의 sensors 배열은 디바이스에서 사용할 수 있는 센서 모델 목록 입니다.
    - 사용할 수 있는 센서 모델 중 network, driverName, model, type, category는 센서 등록 시 사용이 됩니다.
    - 또한, driverName은 센서 드라이버를 얻어올 때 필요합니다.

3. 센서 드라이버를 가지고 옵니다.
    - 센서 드라이버에서 정의한 idTemplate은 센서 등록 시 사용됩니다.

4. 등록할 센서 정보를 만들어 전송합니다. (2.3.7절 참고)
   데이터 포멧은 아래와 같습니다.

```
{
  reqId: 'abcdefghijkl-0-humidity',
  category: 'sensor',
  type: 'humidity',
  model: 'jsonrpcHumi',
  driverName: 'jsonrpcSensor',
  network: 'jsonrpc',
  name: 'My Camera',
  owner: 'abcdefghijkl',
  ctime: 1456297274325,
  deviceId: 'abcdefghijkl-0'
}
```

- reqId : 센서 드라이버에 있는 idTemplate 형식에 맞게 ID를 생성합니다.
    - 일반적으로 idTemplate은 {gatewayID}-{deviceAddress}-{type}-{sequence}입니다.
        - gatewayID : 게이트웨이 아이디
        - deviceAddress : 게이트웨이 내에 디바이스를 구분하기 위한 값으로 센서가 속한 디바이스의 어디레스를 적어줘야합니다.
        - type : Thing+에서 정의한 센서 타입
        - sequence : 한 디바이스 안에 동일한 종류의 센서가 2개 이상 있을 때 구분하기 위한 값입니다. 한 개만 있으면 생략하셔도 됩니다. 사용자 정의
- category : 등록 할 센서의 카테고리로 센서 모델에 정의되어 있습니다.
- type : 센서 타입으로 센서 모델에 정의되어 있습니다.
- model : 센서 모델의 이름으로 센서 모델에 정의되어 있습니다.
- driverName : 센서가 사용할 드라이버 이름으로 센서 모델에 정의되어 있습니다.
- network : 센서가 사용하는 네트워크로 센서 모델에 정의되어 있습니다.
- name : 센서 이름입니다. 사용자 정의
- owner : 센서가 속해 있는 게이트웨이 아이디입니다.
- ctime : 센서가 생성 된 시간입니다. UTC
- deviceId : 센서가 속해있는 디바이스의 아이디입니다.

2.3.3 기본 설정 및 인증

REST API의 URL은 다음과 같습니다.

`https://api.thingplus.net`

인증을 위해 BODY에 username과 apikey를 채워주시면 됩니다.

```
{
  username: <GATEWAY_ID>
  apikey: <APIKEY>
}
```

> GATEWAY_ID: 게이트웨이 아이디
> APIKEY: Thing+ Portal에서 발급받은 APIKEY

2.3.3 에러코드

Error Code	Description
401	Unauthorized
403	Forbidden
404	Not Found
409	Post Item Error
471	Billing Error

2.3.4 사용중인 게이트웨이 정보 가지고 오기

사용중인 게이트웨이의 정보를 가지고 오는 API입니다.

Resource URL

`GET https://api.thingplus.net/gateways/<GATEWAY_ID>?fields=model&fiedlds=autoCreateDiscoverable`

> **GATEWAY_ID** 게이트웨이 아이디

Request Example

`GET https://api.thingpus.net/gateways/abcdefghijkl?fields=model&fields=autoCreateDiscoverable`

Response Example

```
{
  id: "abcdefghijkl",
  model: "34",
  autoCreateDiscoverable: "y",
}
```

> **model** 게이트웨이 모델 번호
> **autoCreateDiscoverable** 디스커버 기능 지원 여부

2.3.5 게이트웨이 모델 가지고 오기

Thing+에서 정의한 게이트웨이 모델을 가지고 오는 API입니다. 게이트웨이 정보에 있는 모델 번호를 사용하여, 게이트웨이 모델을 가지고 옵니다.

Resource URL

`GET https://api.thingplus.net/gatewayModels/<MODEL_NUMBER>`

> **MODEL_NUMBER** 게이트웨이 모델 번호

Request Example

`GET https://api.thingplus.net/gatewayModels/34`

Response Body Format and Example

Body Format

```
{
  ctime: "<Gateway Model 생성 시간>",
  model: "<Gateway Model 이름>",
  deviceMgmt: {
    reportInterval: {
      show: "<리포트 인터벌이 Thing+ Cloud에 표시 여부. y or n>",
      change: "<리포트 인터벌 변경 가능 여부. y or n>"
    },
    DM: {
      poweroff: {
        support: "<Thing+를 통해 전원 끄기 기능 지원. y or n>"
      },
      reboot: {
        support: "<Thing+를 통해 재시작 기능 지원. y or n>"
      },
      restart: {
        support: "<Thing+를 통해 어플리케이션 프로그램 재시작 기능 지원. y or n>"
      },
      swUpdate: {
        support: "<Thing+를 통한 소프트웨어 업데이트 기능 지원. y or n"
      },
      swInfo: {
        support: "<Thing+에서 소프트웨어 버전 정보를 읽어가는 기능 지원. y or n"
      }
    }
  },
  id: "<Model ID>",
  vendor: "<Vendor Name>",
  mtime: "<수정된 시간. UTC>",
  deviceModels: [
    {
      id: "<디바이스 모델 아이디>",
      displayName: "<디바이스 모델 이름>",
      idTemplate: "<디바이스 아이디 형식>",
      discoverable: "<디스커버 가능 여부. y or n>",
      sensors: [
        {
          network: "<센서가 사용하는 네트워크>",
          driverName: "<센서 드라이버 이름>",
          model: "<센서 모델>",
          type: "<센서 타입>",
          category: "<카테고리. sensor or actuator>"
        },
        ...,
      ],
      max: <사용할 수 있는 디바이스 개수>
    }
  ],
  displayName: "<게이트웨이 이름>"
}
```

> **deviceModels** : 게이트웨이가 가질 수 있는 디바이스에 대한 모델
>
> > **discoverable** : 디바이스 디스커버 가능 여부
> > **idTemplate** : 디바이스 아이디의 형식 정의

> 디바이스 등록 시 idTemplate 형식으로 디바이스 아이디를 만들어서 등록해야 한다.

Example

```
{
  ctime: "1456122659103",
  model: "openHardwareCustom",
  deviceMgmt: {
    reportInterval: {
      show: "y",
      change: "y"
    },
    DM: {
      poweroff: {
        support: "n"
      },
      reboot: {
        support: "n"
      },
      restart: {
        support: "y"
      },
      swUpdate: {
        support: "y"
      },
      swInfo: {
        support: "y"
      }
    }
  },
  id: "34",
  vendor: "OPEN SOURCE HARDWARE",
  mtime: "1456122659103",
  deviceModels: [
    {
      id: "jsonrpcFullV1.0",
      displayName: "Open Source Device",
      idTemplate: "{gatewayId}-{deviceAddress}",
      discoverable: "y",
      sensors: [
        {
          network: "jsonrpc",
          driverName: "jsonrpcSensor",
          model: "jsonrpcNumber",
          type: "number",
          category: "sensor"
        },
        {
          network: "jsonrpc",
          driverName: "jsonrpcSensor",
          model: "jsonrpcString",
          type: "string",
          category: "sensor"
        }
      ],
```

```
      max: 1
    }
  ],
  displayName: "Open Source Gateway"
}
```

2.3.6 센서 드라이버 가지고 오기

Thing+에서 정의한 센서 드라이버를 가지고 오는 API입니다.

Resource URL

`GET https://api.thingplus.net/sensorDrivers/?filter[id]=<driverName>`

> **driverName** 센서 드라이버 이름

Request Example

`GET https://api.thingplus.netsensorDrivers/?filter[id]=jsonrpcSensor`

Response Example

```
{
  discoverable: "true",
  ctime: "1456122653281",
  id: "jsonrpcSensor",
  displayName: "jsonrpc Sensor",
  models: [
    "jsonrpcNumber",
    "jsonrpcString",
    ...,
    "jsonrpcReader"
  ],
  supportedNetworks: [
    "jsonrpc"
  ],
  mtime: "1456122653281",
  category: "sensor",
  addressable: "false",
  dataTypes: {
    jsonrpcNumber: [
      "number"
    ],
    jsonrpcString: [
      "string"
    ],
    ...,
    jsonrpcReader: [
      "reader"
    ]
  },
  driverName: "jsonrpcSensor",
  idTemplate: "{gatewayId}-{deviceAddress}-{type}-{sequence}"
}
```

> **discoverable** 센서의 디스커버 가능 여부

2.3.7 디바이스 등록하기

Resource URL

`POST https://api.thingplus.net/devices`

Post Parameters

parameter	description
reqId	디바이스 아이디 게이트웨이 모델에서 정한 idTemplate 형식으로 생성해야 한다. idTemplate은 게이트웨이 모델의 deviceModels 배열의 사용하는 디바이스의 idTemplate을 사용하면 된다.
name	디바이스 이름
model	디바이스가 사용하는 모델 아이디 게이트웨이 모델의 deviceModels 배열 중 thing이 사용 할 디바이스의 ID를 넣어준다.

Request Body Example

```
{
  reqId: 'abcdefghijkl-0',
  name: 'My Device0',
  model: 'jsonrpcFullV1.0' }
}
```

Response Example

```
{
  name: 'My Device0',
  model: 'jsonrpcFullV1.0',
  owner: 'abcdefghijkl',
  mtime: 1456297274619,
  ctime: 1456297274619,
  id: 'abcdefghijkl-0'
}
```

Error

Error Code	Description
401	body의 게이트웨이 아이디 또는, APIKEY가 틀렸음.
404	등록이 안된 게이트웨이에 디바이스 추가를 시도 함.
471	요금제에 의해 디바이스 추가를 할 수 없음.

2.3.8 센서 등록하기

Resource URL

`POST https://api.thingplus.net/gateways/<GATEWAY_ID>/sensors`

> **GATEAY_ID** 센서가 속한 게이트웨이의 아이디

Post Parameter

parameter	description
network	네트워크 이름
driverName	센서가 사용하는 드라이버 이름
type	센서타입
카테고리	센서 or 액츄에이터
name	센서이름
reqId	센서 아이디 센서 드라이버에서 정한 idTemplate 형식으로 생성해야 함
owner	센서가 속한 게이트웨이
ctime	센서가 생성된 시간(UTC)
deviceId	센서가 속한 디바이스의 아이디

Request Body Example

```
{
  network: 'jsonrpc',
  driverName: 'jsonrpcActuator',
  model: 'jsonrpcCamera',
  type: 'camera',
  category: 'actuator',
  reqId: 'abcdefghijkl-0-camera',
  name: 'My Camera',
  owner: 'abcdefghijkl',
  ctime: 1456297274325,
  deviceId: 'abcdefghijkl-0'
}
```

Response Example

```
{
  network: 'jsonrpc',
  driverName: 'jsonrpcActuator',
  model: 'jsonrpcCamera',
  type: 'camera',
  category: 'actuator',
  name: 'My Camera',
  address: '0',
  options: {},
  deviceId: 'abcdefghijkl-0',
  owner: 'abcdefghijkl',
  mtime: 1456297274458,
  ctime: 1456297274458,
  id: 'abcdefghijkl-0-camera'
}
```

Error

Error Code	Description
401	body의 게이트웨이 아이디 또는, APIKEY가 틀렸음.
404	등록이 안된 게이트웨이에 디바이스 추가를 시도 함.
471	요금제에 의해 디바이스 추가를 할 수 없음.

3. Thing+ Embedded SDK

준비중

4 Thing+ Gateway

Thing+ gateway는 Daliworks에서 만든 Thing+ MQTT 프로토콜을 따르는 소프트웨어입니다. Thing+ Gateway는 하드웨어를 Thing+ Cloud에 연결하며, 게이트웨이 상태 및 센서값을 전송하고, 시간 동기, 센서값 재전송, 연결된 센서/액추에이터 탐색, 원격 업데이트 기능을 제공합니다.

디바이스 에이전트(Device Agent)는 센서값, 액추에이터 동작을 시키는 소프트웨어 모듈로 이 부분은 하드웨어 업체가 직접 작성해야 합니다. Thing+ Gateway는 디바이스 에이전트에게 센서값 읽기, 액추에이터 동작을 요청하며, JSONRPC 프로토콜을 사용합니다.

Thing+ Gateway를 사용하면 Thing+ MQTT 프로토콜을 직접 구현하는 것 보다 간편하고 빠르게 하드웨어를 Thing+와 연동할 수 있습니다.

4.1 Hardware Requirement

Thing+ 게이이트웨이는 하드웨어에서 실행되는 프로그램이며, Node.js로 작성되어 있습니다. **Node.js가 동작하지 않는 시스템에서는 사용할 수 없습니다.**

> Node.js 실행 환경

- CPU : arm, ia32, x86, x86_64
- Memory : 128MB 이상
- OS : linux, win, mac, freeebsd, openbsd, android
    - **RTOS, MICRO OS에는 사용할 수 없습니다.**

Thing+ Gateway 구동을 위해 필요한 저장공간 사용량은 아래와 같습니다.

> Storage Requirement

Category	Size
Thing+ Gateway	11 MB
Node.js(binary)	9 MB
Node.js Modules	5 MB
StoreDb, Log	5 MB

4.2 Features

Thing+ Gateway에는 아래 기능이 있습니다.

> Thing+ Gateway Features

Feature	Description
Connection	Thing+ Cloud 연결
Discover	연결된 센서, 액추에이터 탐색
TimeSync	시간 동기
Store DB	네트워크 미연결 시, 센서값 저장
SW Update	원격에서 게이트웨이 소프트웨어 업데이트

- Connection
  Thing+ Gateway는 하드웨어를 Thing+ Cloud에 연결합니다. Thing+ Gateway 최초 실행 시 APIKEY만 알려주면 됩니다. Thing+ Gateway는 Thing+ MQTT 프로토콜에서 정의한 Conection SPEC에 맞쳐 MQTT 패킷을 전송하며, MQTT 상태, 게이트웨이 상태, 시리즈 센서의 상태를 리포트 인터벌에 맞처 전송합니다.

- Discover
  Thing+ Gateway는 하드웨어에 연결 된 센서, 액추에이터 정보를 Thing+ Cloud에 전송하며 합니다. 게이트웨이 모델에 속한 센서, 액추에이터 목록 중 현재 사용중인 센서, 엑추에이터만 표시할 수 있습니다. 또한, 센서, 액추에이터가 추가되면, Thing+ Gateway가 자동으로 이 정보를 Thing+ Cloud로 전송합니다. 만약, Discover 기능을 사용하지 않는다면, 센서, 액추에이터가 추가될 때 마다 Thing+ 포털에서 수동으로 추가를 해야합니다.

- TimeSync
  thing의 시간이 서버시간과 다를경우 thing의 로컬시간은 서버시간과 동기화 되어야합니다. Thing+ Gateway는 하드웨어를 서버시간으로 동기화하는 기능이 있습니다. Thing+ Gateway를 사용하는 하드웨어 업체는 thing의 시간 설정에 대해서 해야 할 일은 없습니다.

- Store DB
  네트워크 연결이 끊어질 경우, Thing+ Gateway는 센서값을 하드웨어의 스토리지에 자동으로 저장합니다. 네트워크가 복구되면 저장한 센서값을 Thing+ Cloud로 한꺼번에 전송을 합니다. 따라서, Thing+ Gateway는 데이터 유실을 막습니다.

4.3 Device Agent

디바이스 에이전트는 센서값, 액추에이터를 구동하는 소프트웨어로 Thing+ Gateway 독립적으로 실행되는 프로그램입니다. 하드웨어 구성에 맞게 작성되어야 하며, 이 부분은 하드웨어 업체가 직접 작성을 해야합니다.
디바이스 에이전트는 Thing+ Gateway에게 센서값을 읽기, 액추에이터 동작, 연결된 센서 탐색 서비스를 제공해야 합니다.

4.3.1 JSON RPC 연결

Thing+ Gateway와 디바이스 에이전트는 JSONRPC 프로토콜을 사용합니다. 디바이스 에이전트는 JSONRPC 서버이며, 포트 50800를 열어두어야 합니다. 또한, Thing+ Gateway에서 사용할 서비스를 공개(expose)해야합니다. Thing+ Gateway는 JSONRPC 클라이언트며 해당 포트로 접속하여, 디바이스 에이전트가 제공하는 서비스를 사용합니다.

4.3.2 Device Agent Services

디바이스 에이전트는 Thing+ Gateway를 위해 두 개의 서비스를 제공해야 합니다. 첫번째는 센서상태, 센서값을 읽고 엑추에이터를 구동시키는 서비스로 서비스 이름은 "sensor"로 정의되어야 합니다. 두 번째는 와 하드웨어에 연결된 센서, 액추에이터를 탐색하는 서비스로 "discover"로 정의되어야 합니다.

4.3.3 "sensor" Service

"sensor" 서비스는 세 가지 함수가 있어야 합니다. 첫 번째는 Thing+ Gateway가 센서값을 읽어갈 수 있는 "get", 두 번째는 Thing+ Gateway에서 액추에이터를 동작시킬 수 있는 "set", 마지막은 Thing+ Gateway가 이벤트 센서를 받을 준비가 되었을 때 호출하는 "setNotification" 입니다.

Thing+ Gateway는 시리즈 센서의 값을 읽기 위하여 리포트 인터벌 주기마다 "get"함수를 호출합니다. Thing+ Gateway는 값을 읽을 센서의 아이디와 디바이스 에이전트에서 센서값을 전달할 수 있는 콜백함수를 "get"함수의 인자로 전달합니다. 디바이스 에이전트는 센서값을 읽은 후 콜백함수를 호출해야하며, 콜백함수에 센서값과 센서의 상태를 인자로 전달해야 합니다.

```
"sensor" 서비스의 "get" 함수

 - Description : Thing+ Gateway가 센서값을 읽을 때 사용한다.
 - Parameter
    - id : 센서 아이디
    - result(err, {value: {value}, status: 'on'}) : 센서값을 읽은 후 호출 할 콜백함수
        - err: 에러 발생 시 'err', 에러 없으면 null
        - {value:{value}, status: 'on'} : {value}에 센서값을 적는다.
 - Return value : None
```

Thing+ Gateway가 액추에이터 명령어를 Thing+ Cloud로 부터 받으면 "set"함수를 호출하여 액추에이터를 구동시킵니다. 액추에이터 명령어와 옵션, 수행 결과를 전달할 수 있는 콜백함수를 인자로 전달합니다. 디바이스 에이전트는 액추에이터를 구동한 후, 콜백함수를 호출해야하며, 수행한 명령어를 인자로 전달해야 합니다.

```
"sensor" 서비스의 "set"함수

 - Description : Thing+ Gateway가 액추에이터를 구동시킬 때 사용한다.
 - Parameter
    -  id : 액추에이터 아이디
    -  cmd : 액추에이터 명령어
    -  options : 명령어 옵션
    -  result(err, cmd} : 액추에이터 동작 후 호출 할 콜백함수
        - err: 에러 발생 시 'err', 에러 없으면 null
        - cmd : 액추에이터를 구동 시킨 명령어
    - Return value : None
```

Thing+ Gateway는 이벤트 센서를 받을 준비가 되면 "setNotification"함수를 호출하여 디바이스 에이전트에게 알려줍니다. 디바이스 에이전트는 "setNotification" 함수가 호출된 이후 이벤트 센서의 값을 전송하면 됩니다.

```
"sensor" 서비스의 "setNotificatin" 함수

 - Description : Thing+ Gateway가 이벤트 센서값을 받을 준비가 되었을 때 호출한다.
 - Parameter
    - id : 센서 아이디
    - result(err) : 처리 결과 전송
        - err : 에러 발생 시 'err', 에러 없으면 null
- Return value : None
```

4.3.4 "discover" Service

"discover" 서비스는 단일 함수로 구성됩니다. Thing+ Gateway가 디바이스 에이전트의 JSONRPC 서버에 접속하면 "discover" 서비스를 이용해 하드웨어가 사용하는 센서, 액추에이터 목록을 얻어 옵니다. 디바이스 에이전트가 목록을 전달할 수 있도록 콜백함수를 인자로 전달합니다.

```
"discover" 서비스

 - Description : Thing+ Gateway가 센서, 액추에이터 목록을 가지고 올때 사용한다.
 - Parameter
    - result(err, {sensor list}): 디바이스 에이전트에서 목록을 전달할 때 호출하는 콜백함수
        - err : 에러 발생 시 'err', 에러 없으면 null
        - {sensor list} : 센서 목록
 - Return Value : None
```

> 센서 목록

```
{
    deviceAddress: {device address},
    {deviceAddress}: {
        "sensors": [{
            "id":{sensor id},
            "type":{sensor type},
            "name":{sensor name},
            "notification":{true if event sensor}
        },
        ...
        {
        }]
    }
}
```

4.3.5 Event sensor 전송

시리즈 센서는 리포트 인터벌에 맞쳐 Thing+ Gateway가 센서값과 상태를 읽어가지만, 이벤트 센서는 센서값이 바뀔 때 마다 디바이스 에이전트에서 전송해야 합니다. 또한, 리포트 인터벌 주기로 센서 상태도 전송해야합니다.

센서값과 상태를 전송할 때 사용하는 JSON 형식은 다음과 같습니다.

> 센서값 전송

```
{
  "method": "sensor.notification",
  "params": [{sensor_id}, {"value": {value}}]
}
```

> 센서 상태 전송

```
{
  "method": "sensor.notification",
  "params": [{sensor_id}, {"status": {status}}]
}
```

5. How to Test Example Code

하드웨어에서 구현해야 할 Thing+ 연동 예제 코드에 대한 설명입니다.

thing에서 선택할 수 있는 연동 방법은 세 가지가 있습니다.

1. Thing+ Embedded Protocol 직접 구현
    - Thing+에서 정의한 MQTT, HTTP로 데이터 형식에 맞쳐 Thing+로 정의
2. Thing+ Embedded SDK 사용
    - Thing+에서 제공하는 SDK를 사용하여 데이터 전송
3. Thing+ Gateway를 사용하고 Device Agent 구현
    - Thing+ Gateway를 사용하고, 데이터 전송에 관해선 Thing+ Gateway가 담당
    - 센서, 액츄에이터를 구동하는 코드만 적성

Thing+는 하드웨어 업체를 위하여 각 방법에 대한 예제코드 제공하고 있습니다.

5.1 Thing+ Embedded Protocol

- 소스위치 : example/protocol/
- OS : Linux
- 라이브러리
    - Paho : MQTT(Daliworks에서 TLS를 위해 포트 수정 함)
    - cJSON : JSON 파싱
- 게이트웨이 모델 : Linux - Device
    - 센서
        - Series : 온도, 습도
        - Event : 온오프
    - 액츄에이터 : LED

센서는 시뮬레이션을 하였습니다. 온도, 습도 센서는 랜덤한 값을 생성하도록 하였으며, 온오프 센서는 랜덤한 시간에 센서 값이 변하도록 하였습니다.

5.1.1 예제 프로그램 실행 방법

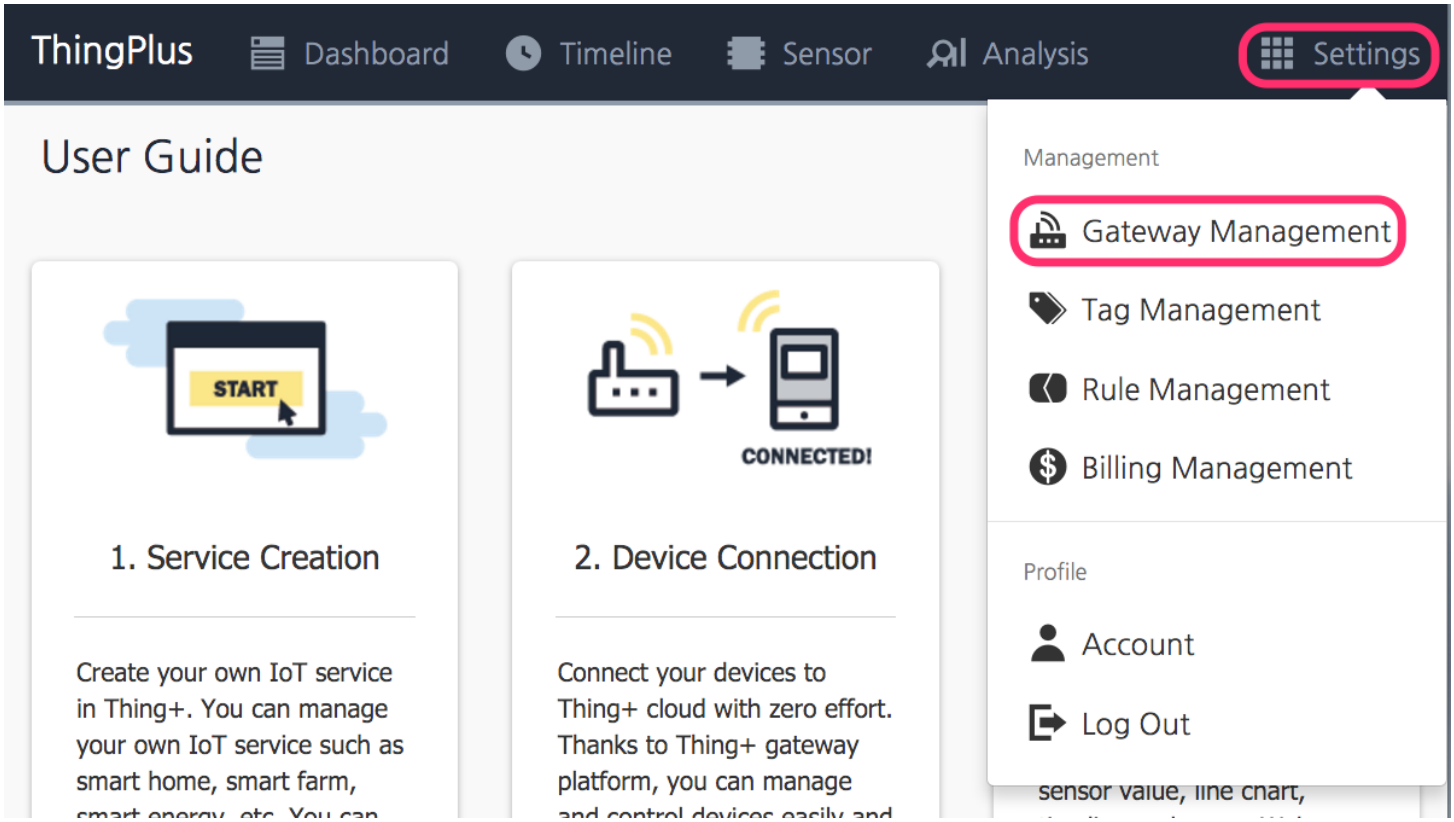
1. 예제 코드를 실행 시킬 하드웨어를 Thing+ Portal에서 게이트웨이로 등록합니다.

2. 예제 코드의 config.json에 MAC 주소와 APIKEY를 설정합니다.
3. 예제 코드를 빌드한 후 실행시킵니다.

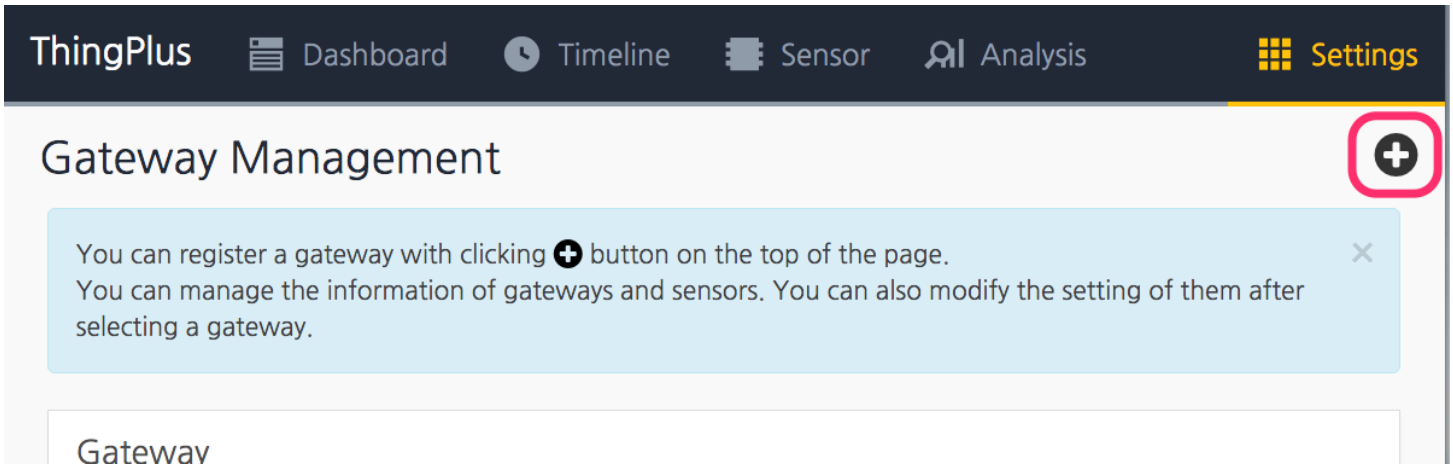
5.1.2 게이트웨이 등록 방법

1. Thing+ 회원 가입 후 로그인

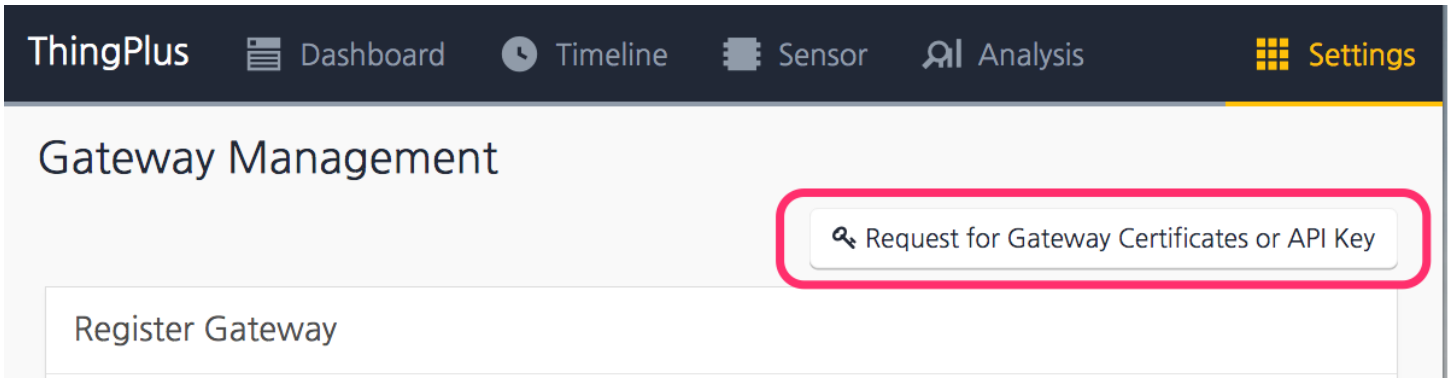
2. 오른쪽 상단의 "설정" -> "게이트웨이 관리"로 이동



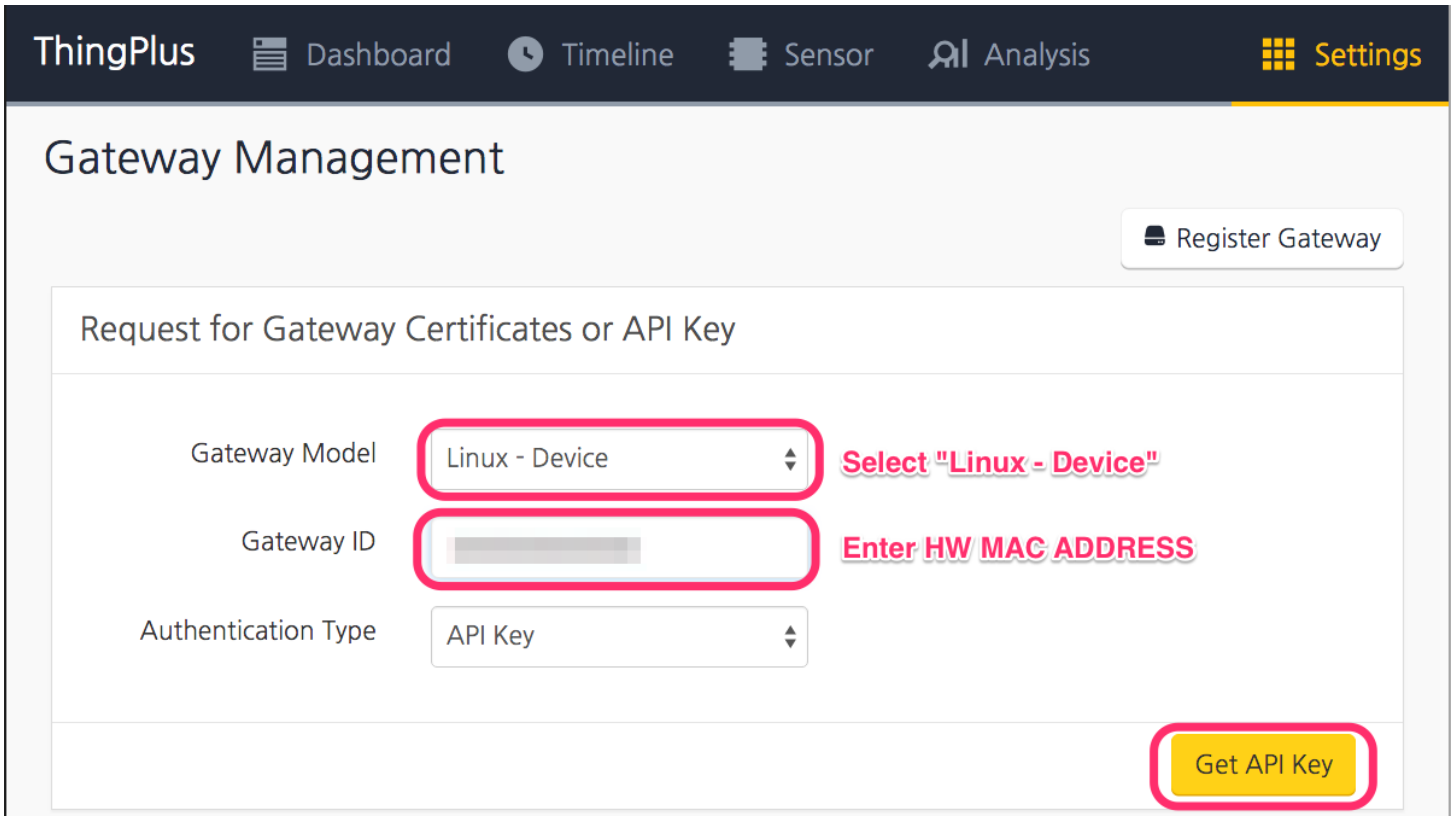
3. 게이트웨이 추가를 위해 "+" 아이콘 누름



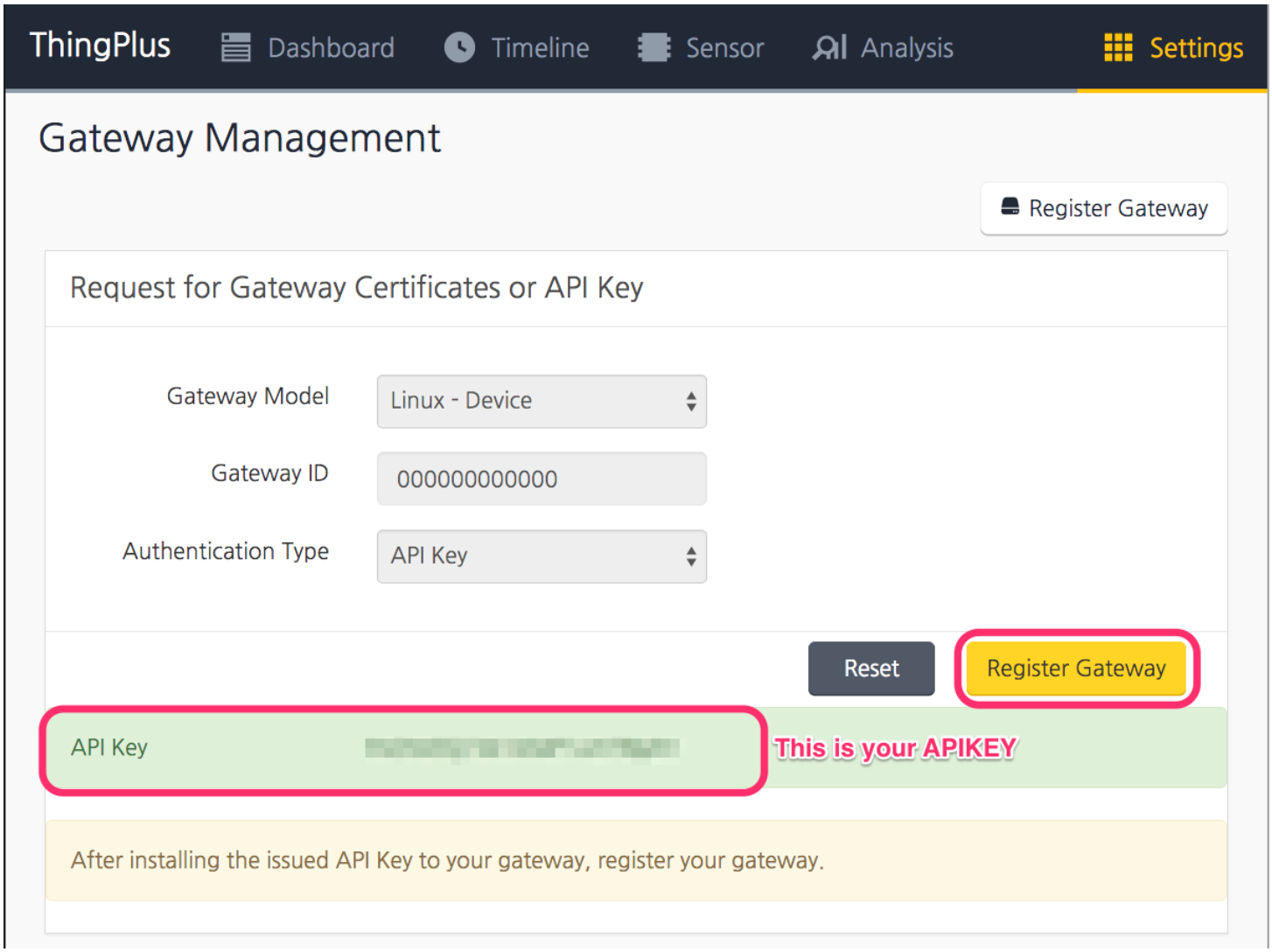
4. APIKEY 발급을 위하여 "게이트웨이 인증서/API키 발급급기"로 이동



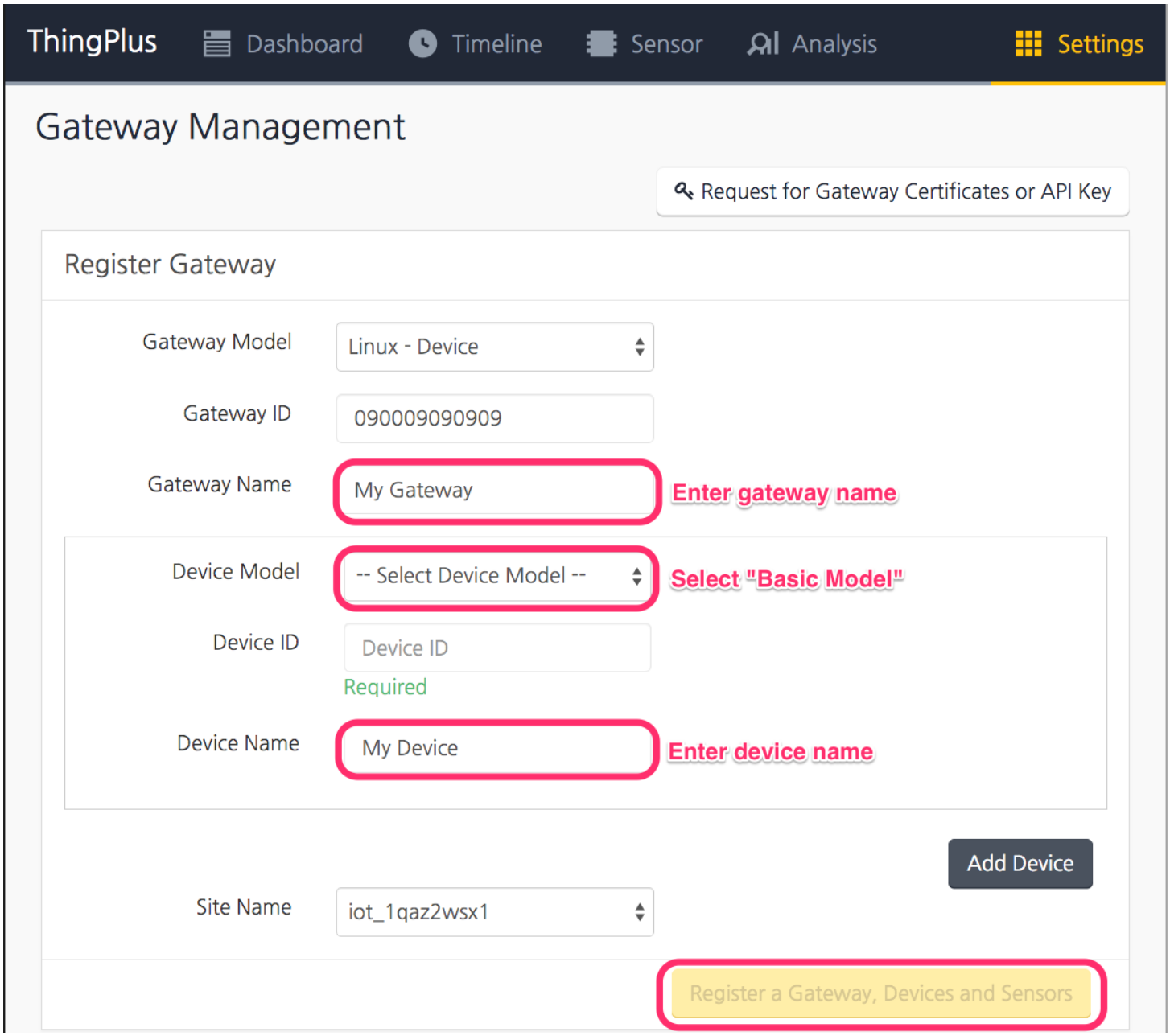
5. 게이트웨이 모델은 "Linux - Device"로 선택하고, MAC 어드레스 입력



6. APIKEY 발급



7. 게이트웨이 이름, 디바이스 이름 설정하고, 디바이스 모델을 "Basic Model"로 선택



8. 게이트웨이 등록 완료

5.1.3 소스의 config.json 설정

```
{
  "gatewayId": "<HW_MAC_ADDRESS>",
  "host": "dmqtt.thingplus.net",
  "port": 8883,
  "username": "<HW_MAC_ADDRESS>",
  "password": "<APIKEY>",
  "caFile": "./cert/ca-cert.pem"
}
```

> **HW*MAC*ADDRESS** : MAC 어드레스
> **APKEY** : Thing+ Portal에서 발급받은 APIKEY

5.1.4 빌드 및 실행방법

- 빌드

```
$ git submodule init
$ git submodule update
$ make
```

  - 디버깅 모드 빌드

```
$ make ENABLE_DEBUG=1
```

- 실행

```
$ ./thingplus
```

- 클린

```
$ make clean
```