1. Thing+ Embedded Overview

Thing+ Embedded is composed of the Thing+ Embedded Protocol, Thing+ Embedded SDK and Thing+ Gateway. The Thing+ Embedded Protocol defines the format of data flowing between hardware and the Thing+ Cloud server. The Thing+ Embedded SDK is a C library that implements our Thing+ Embedded Protocol. The Thing+ Gateway is a Node.js application program that implements Thing+ Embedded Protocol.

1.1 Hardware application requirements

1.1.1 Thing+ Embedded Protocol

The Thing+ Embedded Protocol can be implemented in hardware ranging from the low end such as WiFi chips to the high end. Firmware or any OS can be used. The ability to connect a MQTT server, sense sensors, actuate actuators, compose MQTT, and send HTTP messages are required in order to function correctly.

Category	**Description**
Application requirement	Sensing sensors Actuating actuators MQTT, HTTP connection Composing MQTT, HTTP message Sending status and values periodically.
Target hardware	Low end to high end
Dependency	None

1.1.2 Thing+ Embedded SDK

The Thing+ Embedded SDK is a C library that can be implemented typically in middle end to high end hardware. The SDK's depencies include libmosquitto, libcurl, libjson-c, openssl. The SDK's primary purpose is to connect hardware to the Thing+ Cloud server and compose MQTT and HTTP messages. Sensing sensors, actuating actuators, sending status messages and values are required features of any hardware used.

Category	**Descriptions**
Application requirement	Sensing sensors Actuating actuators Sending status and value periodically.
Target hardware	Middle end to high end
Dependency	libcurl libjson-c libmosquitto openssl C or C++ application

1.1.3 Thing+ Gateway

The Thing+ Gateway can be implemented on high end hardware. The Thing+ Gateway is a Node.js application program. Whatever hardware is used must have the ability to run node.js. There is another protocol between the Thing+ Gateway and hardware/device

applications we call the "Device Agent". The Thing+ Gateway connects hardware to the Thing+ Cloud server and composes MQTT & HTTP messages with the purpose of sending status and data/values to the server. Sensing sensors, actuating actuators and running a JSONRPC server must be possible on any hardware you use to set up a Thing+ gateway.

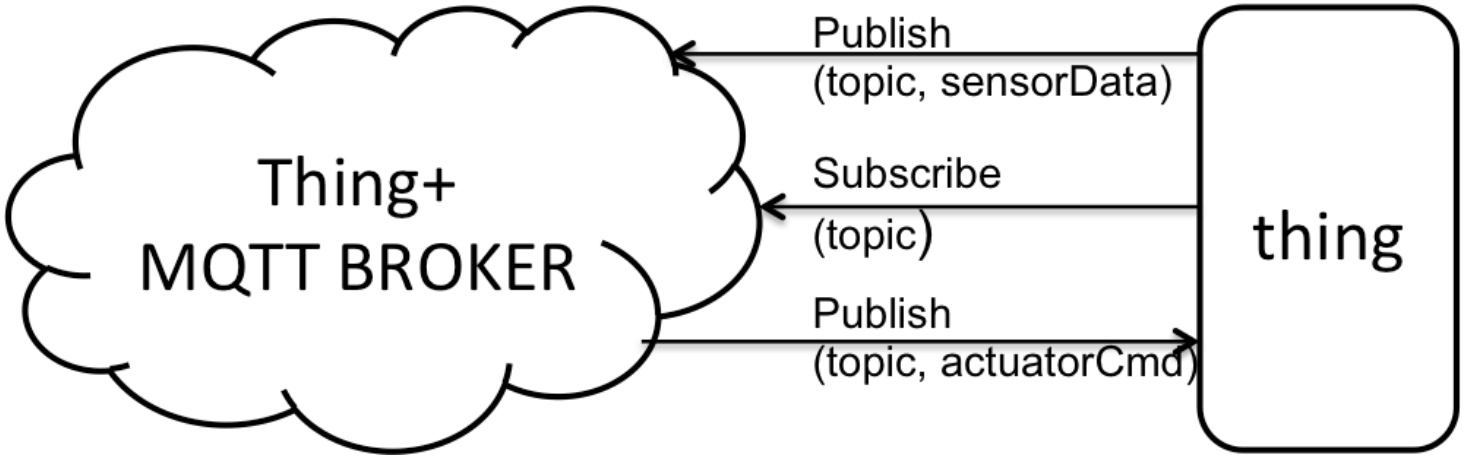
Category	Descriptions
Application requirement	Sensing a sensors Actuating actuators Sending sensor event values JSONRPC Server
Target hardware	High end
Dependency	Node.js

2. Thing+ Embedded Protocol

The Thing+ defined message protocol used to communicate between hardware and the Thing+ cloud is called the "Thing+ Embedded Protocol". Thing+ Embedded Prococol is based on HTTP and MQTT. Sensor values and actuator commands are sent via MQTT between sensors and devices - data coming out of devices to a gateway is passed via HTTP.

2.1 MQTT

Message Queuing Telemetry Transport (MQTT) is a lightweight messaging protocol used between an IoT device that uses low bandwidth and low power and the Thing + Cloud. MQTT has a publish / subscribe structure and is implemented via TCP / IP. You can use SSL and TLS to secure your data and provide a USERNAME / PASSWORD based authentication method.



The MQTT broker is responsible for delivering messages to various clients for sending and receiving messages. Thing + Cloud provides an MQTT broker. IoT devices send sensor values to the MQTT broker provided by the Thing+ Cloud, receive actuators commands, and operate according to commands.

The client that generates the message publishes the message with a Topic before it, and the subscriber that subscribes to the message registers the topic to be received with the broker. The MQTT broker relays messages based on topics. Topics have a hierarchy and use slashes (/) to separate them.

```
Topic example)
HOME0 / BEDROOM / TEMERTATURE
HOME0 / BEDROOM / HUMIDITY
HOME0 / LIVING / CO2_LEVEL
HOME1 / KITCHEN / DUST_LEVEL
```

MQTT provides three levels of quality of service (QoS).

- QoS0: The message is sent only once and does not check the success or failure of the transmission. Possible message loss.
- ** QoS1 **: Messages can be sent more than once. Messages are always delivered accurately and can be delivered in duplicates during the delivery process. ** Thing + Cloud uses QoS1 **
- QoS2: The message is transmitted exactly once and does not cause message loss and duplicate transmission. However, a large amount of network bandwidth is required for message transmission.

MQTT provides a Last Will and Testament (LWT) function that can save messages to other clients when a client connection is lost. The client can define a Will message, which the broker is storing. If the client is disconnected due to an error, the broker sends a Will message to the other client. Thing + uses the error status of the thing as a Will message. When an error occurs in the thing, Thing + Cloud can immediately detect the error of the device.

2.2 Thing+ MQTT Protocol

This chapter describes the MQTT Topic and data format used by Thing+ Platform. Any hardware to be integrated with the Thing+ Cloud Platform should follow the given Thing+ MQTT Specification. When the Thing+ Cloud Platform sends a command to an actuator, the command uses the same Thing+ MQTT Specification.

- Thing+ MQTT Message uses QoS1. 'RETAIN' of all messages except that the 'Will' Message is not used.
- All the time value of messages is Unix time and it should be converted msec.
- A thing transmits data regarding the status of the MQTT connection, gateway and sensor.
- The value for each status is defined as below,

Status	**Value(strings)**
turned-on	on
turned-off	off
Error occurred	err

2.2.1 MQTT connection.

The version of MQTT used by the Thing+ Platform is 3 and the 8883 port is used for MQTT communication. 8883 port should be opened before communicating with the Thing+ Platform.

When connecting MQTT to a Thing+ broker, you must use the **mqtts** protocol. Plain MQTT connections are not allowed.

** Details of the MQTT Connection Spec **

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 Transmission of the MQTT Connection status data

Hardware should transmit the status of the MQTT Connection to the Thing+ Platform after the MQTT connection has been created successfully.

> Transmit the MQTT Connection Success Status

```
TOPIC: v/a/g/__GATEWAY_ID__/mqtt/status
MESSAGE: on
```

Example

```
TOPIC: v/a/g/000011112222/mqtt/status
MESSAGE: on
```

2.2.3 Transmission of the H/W status data

Hardware should transmit its status and the valid duration of the status data periodically. When Thing+ fails to get the status data of a specific piece of H/W within the valid duration, Thing+ defines its status using the error status

> Transmit the H/W Status Data

```
TOPIC: v/a/g/__GATEWAY_ID__/status
MESSAGE: __HW_STATUS__,__VALID_DURATION__

__HW_STATUS__: "on" or "off"
__VALID_DURATION__: Unit is sec
```

Example

```
TOPIC: v/a/g/000011112222/status
MESSAGE: on,90
```

2.2.4 Transmission of the sensor status data

Hardware should transmit the status of a sensor attached to it and the valid duration of the status data periodically. When Thing+ fails to get the status data within the valid duration, Thing+ defines its status using the error status

> Transmit the sensor status data

```
TOPIC: v/a/g/__GATEWAY_ID__/s/__SENSOR_ID__/status
MESSAGE: __SENSOR_STATUS__,__VALID_DURATION__

__SENSOR_STATUS__ : "on" or "off"
__VALID_DURATION__: Unit is sec
```

Example

```
TOPIC: v/a/g/000011112222/s/000011112222-temperature-0/status
MESSAGE: on,90
```

2.2.5 Transmission of the H/W status and sensor's status data

Sensor Status Data can be transmitted along with the H/W status data. It is an efficient way to send the sensor status data combined with the H/W status data in one topic in order to save on network bandwidth cost. When hardware has multiple sensors and/or actuators, each sensor/actuator should have a unique value for identifying it from other sensors/actuators on the hardware. You can also define it by yourself. If you use this method, you can skip chapter 2.2.4

> Transmit H/W status and sensors status data

```
TOPIC: v/a/g/__GATEWAY_ID__/status
MESSAGE: __HW_STATUS__,__VALID_DURATION__,__SENSOR_ID__,__SENSOR_STATUS__,__VALID_DURATION__, ...(REPEAT
FOR SENSOR), __SENSOR_ID__,__SENSOR_STATUS__,__VALID_DURATION__
}

__HW_STATUS__ : "on" or "off"
__VALID_DURATION__: Unit is sec
__SENSOR_STATUS__ : "on" or "off"
```

Example

```
TOPIC: v/a/g/000011112222/status
MESSAGE: on,90,000011112222-onoff-0,on,90,000011112222-temperature-0,off,90
```

2.2.6 Transmission of the sensor value data

A single sensor value and multiple sensor values for a specific sensor can be transmitted in one go. A sensor value should be paired with the time value when the hardware reads it. When multiple sensor values are transmitted at a time, they should be ordered by time. An array of multiple sensor values is allowed.

> Transmit of the sensor value data

```
TOPIC: v/a/g/__GATEWAY_ID__/s/__SENSOR_ID__
MESSAGE: __TIME__,__VALUE__, ...(REPEAT FOR VALUES), __TIME__,__VALUE__

__GATEWAY_ID__: Gateway ID
__SENSOR_ID__: Sensor ID
__TIME__: Sensing time. It is Unix time and its unit is msec.
__VALUE__: Sensor value
```

Example

```
TOPIC: v/a/g/000011112222/s/000011112222-temperature-0
MESSAGE: 1461561631000,26.5,1461561632000,27.5,1461561633000,30
```

> Transmit the sensor value data as array

```
TOPIC: v/a/g/__GATEWAY_ID__/s/__SENSOR_ID__
MESSAGE: [__TIME__,__VALUE__, ...(REPEAT FOR VALUES)__TIME__,__VALUE__]

__GATEWAY_ID__: Gateway ID
__SENSOR_ID__: Sensor ID
__TIME__: Sensing time. It is Unix time and its unit is msec.
__VALUE__: Sensor value
```

Example

```
TOPIC: v/a/g/000011112222/s/000011112222-temperature-0
MESSAGE: [1461561631000,26.5,1461561632000,27.5,1461561633000,30]
```

2.2.7 Transmission of the sensor value data for multiple sensors

Multiple sensor values for multiple sensors can be transmitted at a time. At this time, the sensor value should, for a single sensor, be grouped and ordered by time

> Transmit multiple sensor values for multiple sensors

```
TOPIC: v/a/g/__GATEWAY_ID__
MESSAGE: {"__SENSOR_ID__":[__TIME__,__VALUE__,...(REPEAT FOR VALUES),__TIME__,__VALUE__],"__SENSOR_ID__":
[__TIME__,__VALUE__,...,__TIME__,__VALUE__], ...(REPEAT FOR SENSORS), "__SENSOR_ID__":[__TIME__,__VALUE__
,...(REPEAT FOR VALUES),__TIME__,__VALUE__]}

__GATEWAY_ID__: Gateway ID
__SENSOR_ID__: Sensor ID
__TIME__: Sensing time. It is Unix time and its unit is msec.
__VALUE__: Sensor value

Messsage includes {, }, [, ], ""
```

Example

```
TOPIC: v/a/g/000011112222
MESSAGE: {"000011112222-temperature-0":[1461563978000,27.5,1461563978000,28.5],"000011112222-humidity-0":
[1461561631000,30,1461561632000,35,1461561633000,40]}
```

2.2.8 Transmission of the result of a request from Thing+

Hardware should be able to report the result of a request from the Thing+ Platform. the request can be the command for an actuator or a configuration request.

> Transmit the result of a request from Thing+ - Success Case

```
TOPIC: v/a/g/__GATEWAY_ID__/res
MESSAGE: {"id":__MESSAGE_ID__,"result":__RESULT__}
```

> Transmit the result of a request from Thing+ - Failed Case

```
TOPIC: v/a/g/__GATEWAY_ID__/res
MESSAGE: {"id":__MESSAGE_ID__,"error":"code":__ERR_CODE__, "message":__ERR_MSG__}}
```

In case of *errCode*, it is the error code of the failed case. Thing+ follows the JSONRPC Error Code Rule. *errMessge* should have the details about the root cause of the error.

2.2.9 MQTT Messages from Thing+ Platform

A hardware should subscribe below topic for getting the MQTT Message published by Thing+ Platform. Each hardware can subscribe a MQTT message only for it.

> MQTT Topic and Message subscribed by a hardware

```
TOPIC : v/a/g/__GATEWAY_ID__/req
MESSAGE: {"id": __MESSAGE_ID__, "method": __METHOD__, "params": __PARAMS__}

__MESSAGE_ID__ : It is a unique id for identifying each message and reporting the result of each request.
__METHOD__ : List of requests from Thing+ Platform.
__PARAMS__ : : Parameters for a method. each method has its own parameters List of Methods defined by the
 Thing+ platform as below,
```

Example

```
TOPIC: v/a/g/000011112222/req
```

2.2.10 List of Methods and Parameters from the Thing+ Platform

> Method List

Method	Description	Parameters	Param Description
timeSync	Synchronize the local time of a hardware with Thing+ server time	{"time":__TIME__}	__TIME__ : current time in UTC
controlActuator	execute a command on an actuator	{"id":__SENSOR_ID__,"cmd":__CMD__,"options",__OPTIONS__}	__SENSOR_ID__ : ID of an actuator __CMD__ : Command for an actator __OPTIONS__ : Options for a command
setProperty	Environment configuration	{"reportInterval}":__INTERVAL__	__INTERVAL__ : frequency for reporting sensor value in msec
poweroff	Turn off the device	None	None
reboot	Reboot the H/W	None	None
restart	Restart the embedded software	None	None
swUpdate	Upgrade the embedded software	None	None
swInfo	Provide the version of embedded software	None	None

The mandatory methods which should be implemented for embedded software integrated with the Thing+ Platform include both timeSync and setProperty. Others are optional.

timeSync

When hardware gets an MQTT message including this method, it should reset the local time on it as the received time value.

> timeSync params

```
TOPIC: v/a/g/__GATEWAY_ID__/req

REQUEST MESSAGE: {"id":"__MESSAGE_ID__", "method":"timeSync","params":{"time":__SERVER_TIME__}}

RESPONSE IF SUCCESS : {"id":"__MESSAGE_ID__","result":""}
RESPONSE IF FAILED : {"id":"__MESSAGE_ID__","error":{"code":__ERR_CODE__, "message":"__ERR_MSG__"}}

__SERVER_TIME__ : UTC
```

Example

```
TOPIC: v/a/g/1928dbc93871/req

REQUEST MESSAGE : {"id":"e1kcs13b9","method":"timeSync","params":{"time":1372874401865}}

RESPONSE IF SUCCESS : {"id":"e1kcs13b9","result":""}
REPONSE IF FAILED : {"id":"e1kcs13b9","error":{"code": -32000, "message": "invalid options"}}
```

setProperty

This method is for changing the report interval. The unit of reporting interval values is msec. The Thing+ Portal provides the UI for changing the report interval.

> setProperty params

```
TOPIC : v/a/g/__GATEWAY_ID__/req

REQUEST MESSAGE: {"id":"__MESSAGE_ID__","method":"setProperty","params":{"reportInterval":__INTERVAL__}}

RESPONSE IF SUCCESS : {"id" : "__MESSAGE_ID__", "result"}
RESPONSE IF FAILED : {"id":"__MESSAGE_ID__","error":{"code":__ERR_CODE__, "message":"__ERR_MSG__"}}
```

Example

```
TOPIC: v/a/g/1928dbc93781/req
REQUEST MESSAGE : {"id":"e1kcs13bb","method":"setProperty","params":{"reportInterval":"60000"}}

RESPONSE IF SUCCESS : {"id":"e1kcs13bb","result":""}
RESPONSE IF ERROR : {"id":"e1kcs13bb","error":{"code":-32000,"message":"invalid interval"}}
}
```

controlActuator

This method is for sending commands to an actuator. Commands can be sent from the Thing+ Portal UI and an actuator should execute the delivered command from the Thing+ Platform. Commands and options for each actuator are defined by Thing+. The examples of commands and options for some actuators are below,

> Commonly used actuator commands and options

Actuator	Command	Option
led	on	duration
led	off	
led	blink	duration interval
powerSwitch	on	duration
powerSwitch	off	None

> controlActuator params

```
TOPIC: v/a/g/__GATEWAY_ID__/req
REQUEST MESSAGE: {"id":"__MESSAGE_ID__", "method":"controlActuator", "params":{"id":__SENSOR_ID__,"cmd":_
_CMD__, "options":{__OPTIONS__}}

RESPONSE IF SUCCESS : {"id": "__MESSAGE_ID__", "result":""}
RESPONSE IF FAILED : {"id":"__MESSAGE_ID__","error":{"code":__ERR_CODE__, "message":"__ERR_MSG__"}}
```

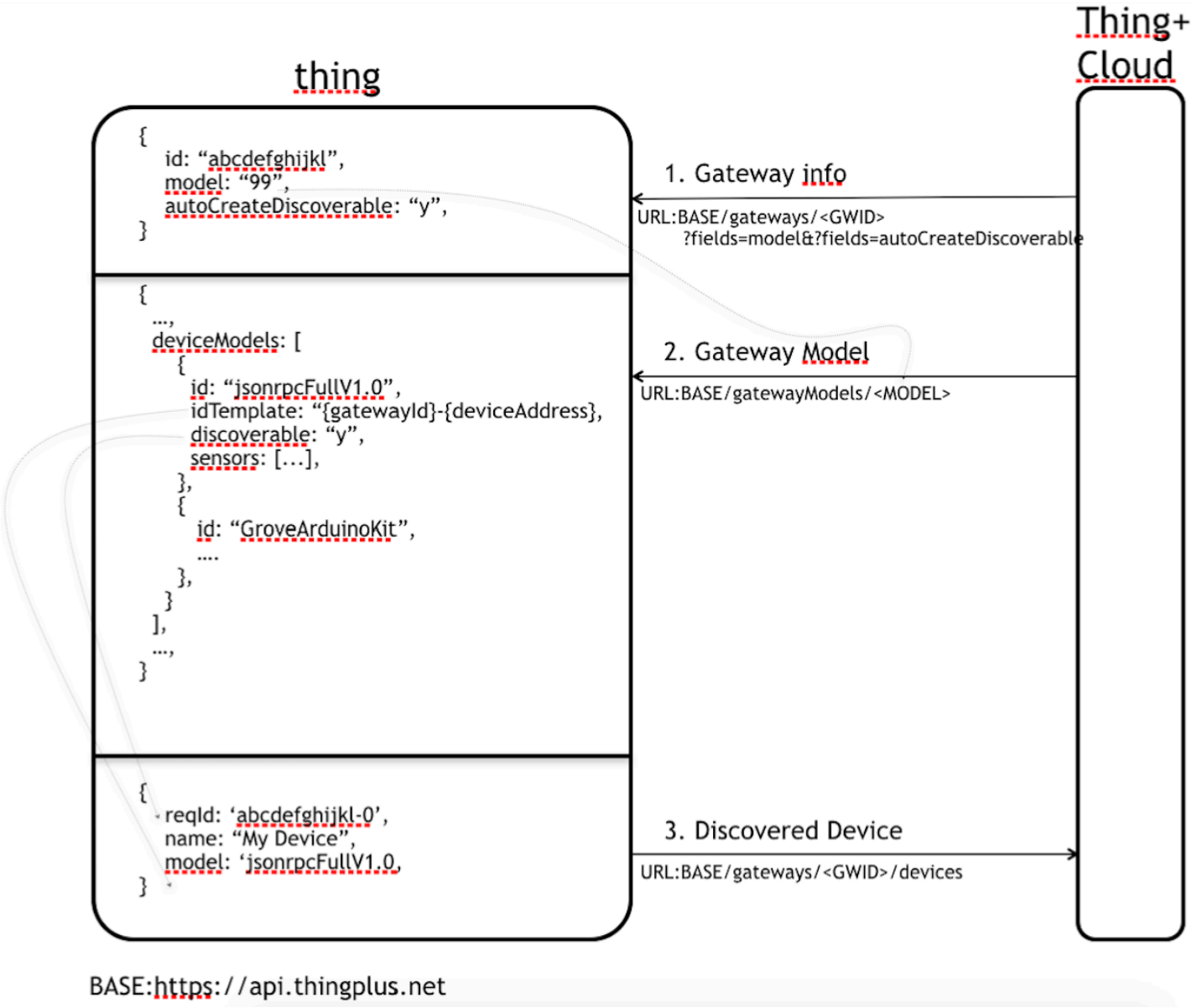
Example: LED ON

```
TOPIC: v/a/g/1928dbc93781/req
REQUEST MESSAGE : {"id":"46h6f8xp3","method":"controlActuator","params":{"id":"led-1928dbc93781-r","cmd":
"on","options":{"duration":3000}}}
RESPONSE IF SUCCESS: {"id":"46h6f8xp3","result":""}
RESPONSE IF FAILED: {"id":"46h6f8xp3","error": {"code":-32000,"message":"invalid options"}}
}
```

2.3 Thing + HTTP Protocol

The Thing + HTTP Protocol is the main protocol used for the Thing+ REST API. When the Discover function is implemented, the IoT device informs Thing+ of the device and sensor information connected to it through the REST API. The Thing+ HTTP Protocol implementation is required to use the Discover function.

2.3.1 Device Registration Process

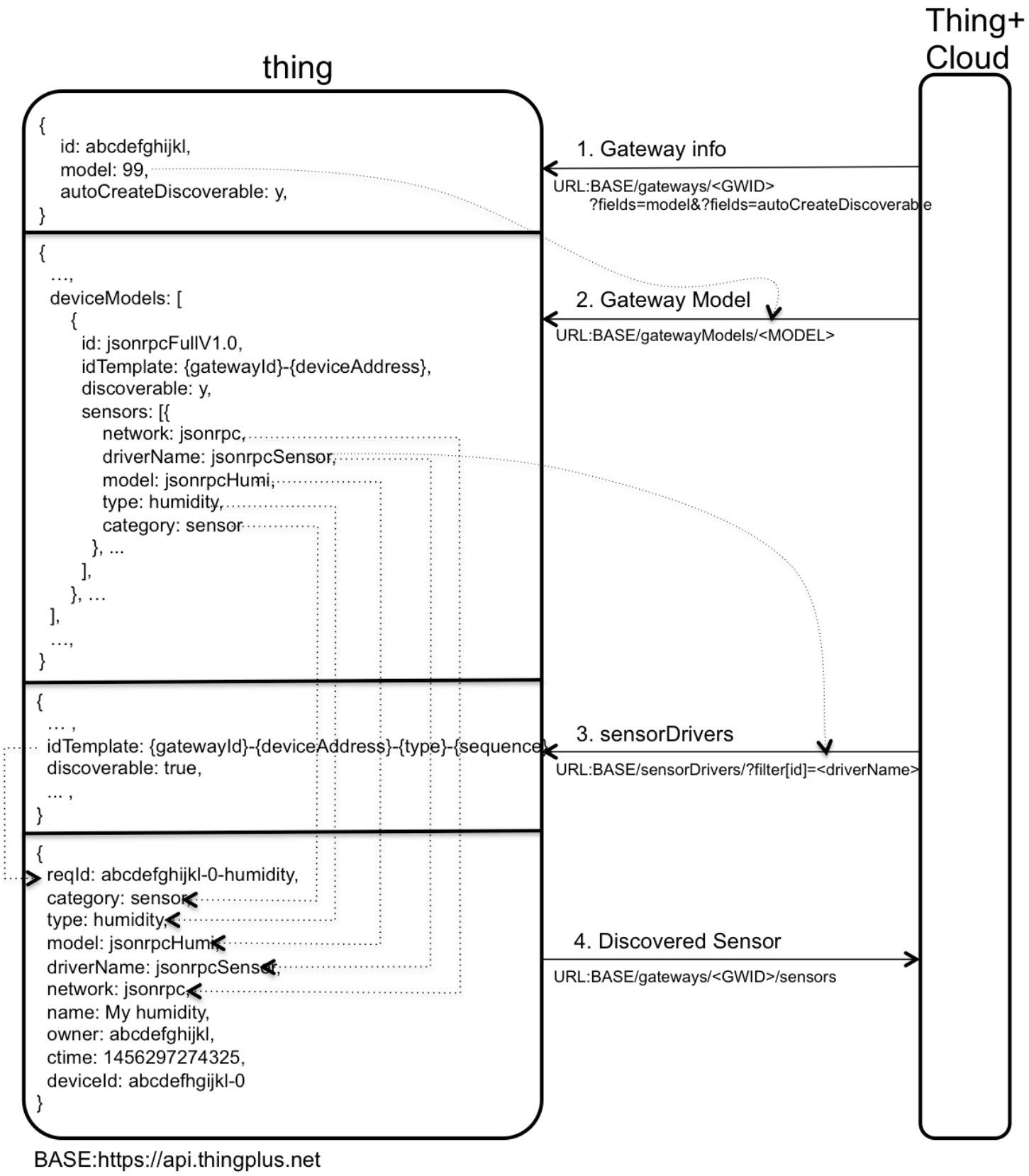


1. Obtaining gateway information. (See Section 2.3.4)
   - Determine if discovery is possible from the obtained gateway information.
     - Refer to autoCreateDiscoverable
       - ** If not dismounted, thing+ can not register a device **
     - The "gateway information model" is used to get the correct gateway model.
2. Get the gateway model. (See Section 2.3.5)
   - Select the device model to use in the deviceModels array of the gateway model. Note: These are Gateways that currently are supported by Thing+.
   - IdTemplate defined in the device model is used when registering the device.
3. Create and transmit device information in order to register with Thing+. (See Section 2.3.6)
   The data format is as follows.

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

- ReqId: Creates an ID for the idTemplate type in the device model.
  - Typically, the idTemplate is {gatewayID} - {deviceAddress}.
    - GatewayID: Gateway ID
    - DeviceAddress: This value is used to identify the device in the gateway. It should not be duplicated within the gateway. Custom
- Name: The device name. Custom
- Model: The name of the device model to use. The device model name to be used in the gateway model information.

2.3.2 Sensor Registration Process



1. Obtain gateway information (see Section 2.3.4).
    - Determine if discovery is possible from the obtained gateway information.
        - See autoCreateDiscoverable
            - ** Without discovery, thing+ can not register sensors **
        - The "gateway information model" is used to get the correct gateway model.

2. Get the gateway model. (See Section 2.3.5)
    - Select the device model to use in the deviceModels array of the gateway model. Note: These are Gateways that currently are supported by Thing+.
    - The sensors array in the device model is a list of sensor models available on the device.
    - Among available sensor models, the network, driverName, model, type and category are used for sensor registration.
    - Also, driverName is needed when getting a sensor driver.

3. Grab the sensor driver.
    - The idTemplate defined by the sensor driver is used when registering the sensor.

4. Create and send sensor information for registration. (See Section 2.3.7)
   The data format is as follows.

```
{
  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: Creates an ID according to the idTemplate type in sensor driver.
    - Typically, idTemplate is {gatewayID} - {deviceAddress} - {type} - {sequence}.
        - GatewayID: Gateway ID
        - DeviceAddress: You must specify the device where the sensor belongs (is attached to) as the value for identifying the device in the gateway.
        - Type: The sensor type defined by Thing+
        - Sequence: It is a value to distinguish when there are two or more sensors of the same type in one device. If you only have one, you can omit it. Custom

- Category: The category of the sensor to be registered. It is defined in the sensor model.
- Type: Sensor type, which is defined in the sensor model.
- Model: The name of the sensor model, which is defined in the sensor model.
- DriverName: The driver name to be used by the sensor, which is defined in the sensor model.
- Network: The network used by the sensor, which is defined in the sensor model.
- Name: The name of the sensor. Custom
- Owner: The gateway ID to which the sensor belongs.
- Ctime: Time at which the sensor was created. UTC
- DeviceId: The ID of the device to which the sensor belongs.

2.3.3 Preferences and Authentication

The URL used for testing and usage of the REST API is:
`https://api.thingplus.net`

To authenticate, just fill in your username and apikey in the header.

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

> GATEWAY_ID: Gateway ID
> APIKEY: APIKEY from Thing+ Portal (Gateway registration section)

The content type should be application / json.

```
Content-type: application / json
```

Error Codes

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

2.3.4 Getting Your Gateway Information

This is an API that brings in information of the gateway in use.

Resource URL

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

> ** GATEWAY_ID ** Gateway ID

Request Example

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

Response Example

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

> ** model ** Gateway model number
> ** autoCreateDiscoverable ** Whether the disk function is supported

2.3.5 Getting the Gateway Model

This is an API that brings in a gateway model defined by Thing+. Using the model number in the gateway information, you can grab the gateway model.

Resource URL

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

> ** MODEL_NUMBER ** Gateway model number

Request Example

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

Response Body Format and Example

Body Format

```
{
  ctime: "<Gateway Model creation time>",
  model: "<Gateway Model name>",
  deviceMgmt: {
    reportInterval: {
      show: "<Whether the report interval is visible in Thing + Cloud, y or n>",
      change: "<report interval changeable y or n>"
    },
    dM: {
      poweroff: {
        support: "<Supports power off via thing + y or n>"
      },
      reboot: {
        support: "<Restart support via Thing +. Y or n>"
      },
      restart: {
        support: "<Support for restarting application programs via Thing+. Y or n>"
      },
      swUpdate: {
        support: "<Software support for Thing +. Y or n"
      },
      swInfo: {
        support: "<Thing + supports reading software version information from y or n"
      }
    }
  },
  id: "<Model ID>",
  vendor: "<Vendor Name>",
  mtime: "<Modified time in UTC>",
  deviceModels: [
    {
      id: "<device model ID>",
      displayName: "<device model name>",
      idTemplate: "<device id format>",
      discoverable: "<Discoverability, y or n>",
      sensors: [
        {
          network: "<network used by the sensor>",
          nriverName: "<Sensor Driver Name>",
          model: "<sensor model>",
          type: "<sensor type>",
          category: "<category. Sensor or actuator>"
        },
        ...,
      ],
      max: <number of available devices>
    }
  ],
  displayName: "<gateway name>"
}
```

> ** deviceModels **: Model for devices that the gateway can connect with
>
> > ** discoverable **: Whether the device is discoverable
> > ** idTemplate **: Defining the format of the device id
> >
> > > When you register the device, you need to create and register the device ID in idTemplate format.

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 Getting the Sensor Driver

This is an API that handles sensor drivers as defined by Thing+.

Resource URL

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

> ** driverName ** Sensor driver name

Request Example

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

Response Example

```
{
  ciscoverable: "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 ** Whether the sensor is discoverable

2.3.7 Registering a Device

Resource URL

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

Post Parameters

Parameter	description
ReqId	Device ID The idTemplate must be created in the gateway model. The idTemplate is the idTemplate of the device in the deviceModels array of the gateway model.
Name	device name
Model	The model ID used by the device. Of the deviceModels array of the gateway model, insert the ID of the device to use.

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	The gateway ID of the body, or APIKEY is incorrect.
404	Attempt to add device to unregistered gateway.
471	Device can not be added by plan.

2.3.8 Registering a sensor

Resource URL

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

> ** GATEAY_ID ** The ID of the gateway to which the sensor belongs

Post Parameter

Parameter	description
Network	network name
DriverName	The name of the driver used by the sensor
Model	sensor model
Type	Sensor type
Category	Sensor or actuator
ReqId	Sensor ID
Name	sensor name
Owner	Gateway to which the sensor belongs
Ctime	Time the sensor was created (UTC)
DeviceId	The ID of the device to which the sensor belongs

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	The gateway ID of the body, or APIKEY is incorrect.
404	Attempt to add device to unregistered gateway.
471	Device can not be added by plan.

3. Thing + Embedded SDK

The Thing+ Embedded SDK is a library created to help easily use Thing+'s Embedded Protocol. It provides sensor values, sensor status transmissions, and sensor and device registration functions. With the Thing+ Embedded SDK, Thing+ can be easily interfaced with and API calls can be used without directly configuring MQTT and HTTP messages.

3.1 Software Requirements

The Thing+ Embedded SDK is written in C and uses openssl, libmosquitto, libjson-c, and libcurl. Before installing the SDK, the library must be installed on the target board.

- Preinstalled Software
  - openssl (https://www.openssl.org/)
  - libmosquitto (https://mosquitto.org/)
  - libjson-c (https://github.com/json-c/json-c)
  - libcurl (https://curl.haxx.se/libcurl/)

3.2 Installation

- Repository: https://github.com/daliworks/thingplus-embedded

```
git clone https://github.com/daliworks/thingplus-embedded
cd thingplus-embedded / library
cmake.
make
make install
```

3.3. API

3.3.1 thingplus_init

```
- Prototype: void * thingplus_init (char * gw_id, char * apikey, char * mqtt_url, char * restapi_url);
- Description: Initialize the Thinglus Embedded SDK.
- Parameters
  - gw_id: Gateway ID
  - apikey: apikey from Thing + Portal
  - mqtt_url: The MQTT server address to connect to. In general, use "mqtt.thingplus.net" and use "dmqtt.
thingplus.net" for non-SSL.
  - restapi_url: HTTPS server address to connect to. Use "https://api.thingplus.net".
- Return Value
  -! NULL: Success. Returns the SDK instance.
  - NULL: Error
```

3.3.2 thingplus_cleanup

```
- Prototype: void thingplus_cleanup (void * t);
- Description: Exit the Thing + Embedded SDK. Must be called before program termination.
- Parameters
  - t: SDK instance
```

3.3.3 thingplus*callback*set

```
- Prototype: void thingplus_callback_set (void * t, struct thingplus_callback * callback, void * callback
_arg);
- Description: Sets the callback function to be called from the SDK.
- Parameters
  - t: SDK instance
  - callback: Callback functions to be called from the SDK. The struct thingplus_callback structure is de
fined in thingplus_types.h.
  - callback_arg: Argument to receive when calling callback function
```

3.3.4 thingplus_connect

```
- Prototype: int thingplus_connect (void * t, char * ca_file, int keepalive);
- Description: Attempt to connect to Thing + server. Asynchronous function, callback function set by thin
gplus_callback_set function is called when connected.
- Parameters
  - t: SDK instance
  - ca_file: SSL certificate. If NULL, connect with non-SSL. Non-SSL connections are only possible when m
qtt_url is "dmqtt.thingplus.net".
  - keepalive: Keepalive time. Unit is seconds
- Return Value
  - 0: Success. Success does not mean that the server connection was successful, but it means that you tr
ied to connect to the server.
        The callback function should check whether the connection of the server is successful.
  - <0: Failed
```

3.3.5 thingplus_disconnect

```
- Prototype: int thingplus_disconnect (void * t)
- Description: Thing + disconnects the server. Asynchronous function, callback function set by thingplus_
callback_set function is called when a server connection is disconnected.
- Parameters
  - t: SDK instance
  - Return Value
    - 0: Success. Success does not mean that the connection is broken, but it means that the attempt to d
isconnect is successful.
          The result of the disconnection must be confirmed by the callback function.
    - <0: Failed
```

3.3.6 thingplus*status*publish

```
- Prototype: int thingplus_status_publish (void * t, int nr_status, struct thingplus_status * status)
- Description: Transmits the status of the gateway, sensor, and actuator.
- Parameters
  - t: SDK instance
  - nr_status: Number of states to transmit
  - status: status to transmit
- Return Value
  - 0: Success
  - <0: Failed
```

3.3.7 thingplus*value*publish

```
- Prototype: int thingplus_value_publish (void * t, int nr_value, struct thingplus_value * values)
- Description: Transmit the sensor value to Thing +.
- Parameters
  - t: SDK instance
  - nr_value: number of sensor values to transmit
  - values: the sensor value to send
- Return Value
  - 0: Success
  - <0: Failed
```

3.3.8 thingplus*device*register

```
- Prototype: int thingplus_device_register (void * t, char * name, int uid, char * device_model_id, char
device_id [THINGPLUS_ID_LENGTH]
- Description: Register the device on Thing + server.
- Parameters
  - t: SDK instance
  - name: the device name
  - uid: The unique ID of the device in the gateway. It should not overlap with other devices.
  - device_model_id: ID of the device model specified in the gateway model
  - device_id: The device ID used by Thing +. If the device registration succeeds, the device ID used by
Thing + is filled in the corresponding array.
- Return Value
  - 0: Success
  - <0: Failed
```

3.3.9 thingplus*sensor*register

```
- Prototype: int thingplus_sensor_register (void * t, char * name, int uid, char * type, char * device_id
, char sensor_id [THINGPLUS_ID_LENGTH]
- Description: Register sensor in Thing +.
- Parameters
  - t: SDK instance
  - name: Sensor name
  - uid: The unique ID of the sensor in the device. It should not overlap with other sensors.
  - type: the type of sensor specified in the gateway model
  - device_id: ID of the device to which the sensor belongs. You must use the device ID issued by Thing +
.
  - sensor_id: Sensor ID used by Thing +. If the sensor registration succeeds, the sensor ID used by Thin
g + is filled in the corresponding array.
- Return Value
  - 0: Success
  - <0: Failed
```

3.3.10 thingplus_gatewayinfo

```
- Prototype: int thingplus_gatewayinfo (void * t, struct thingplus_gateway * info)
- Description: Thing + retrieves the gateway information registered in the server.
- Parameters
  - t: SDK instance
  - info: Structure to hold gateway information
- Return Value
  - 0: Success
  - <0: Failed
```

3.3.11 thingplus_deviceinfo

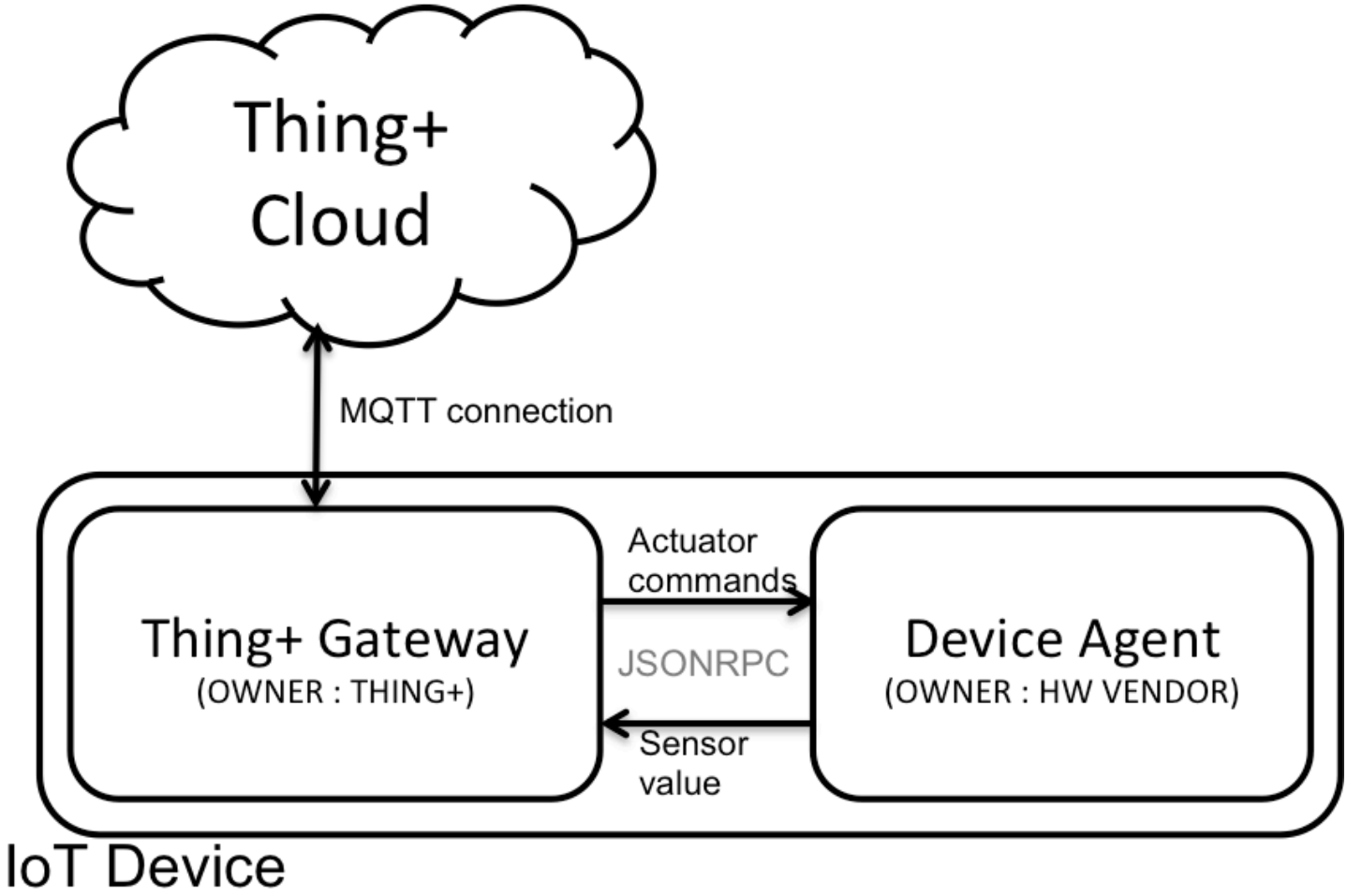
```
- Prototype: int thingplus_deviceinfo (void * t, struct thingplus_device * info)
- Description: Thing + brings up device information registered in server.
- Parameters
  - t: SDK instance
  - info: Structure to hold device information
- Return Value
  - 0: Success
  - <0: Failed
```

4. Thing+ Gateway

The Thing+ Gateway is a program implementing the Thing+ MQTT Protocol, distributed by Daliworks. The Thing+ Gateway connects hardware to the Thing+ Cloud Platform and sends sensor status and values, supports server time syncronization, discover sensors and actuators, and performs remote software updates.

The device Agent is a program made by hardware vendors for collecting sensor values and actuating actuators. The Thing+ Gateway requests sensor values and actuating actuators to the Device Agent and the Thing+ Gateway already defined protocol between Thing+ Gateway and Device Agent. The protocol is based on JSONRPC.

Using a Thing+ Gateway is easier and faster than manually implementing all of the Thing+ Embedded Protocols by hand.



4.1 Hardware Requirement

Thing+ Gateways run on embedded hardware, and is written via Node.js. Your hardware MUST have the ability to run Node.js.

> Node.js Running Environment

- CPU : ARM, ia32, x86, x86_64
- Memory : Minimum 128MB
- OS : LINUX, WINDOWS, MAC, FREEBSD, OPENBSD, android
  - Can NOT support RTOS, MICRO OS

> 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+ Gateway Features

Feature	Description
Connection	Connecting hardware and Thing+ Cloud Server
Discover	Discover connected sensors and actuators and register it.
TimeSync	Syncronize system time automatically
Store DB	Save sensors value if network is disconnected
SW Update	Upgrade Thing+ Gatweway software from remote

- Connection
  The Thing+ Gateway connects hardware to the Thing+ Cloud. One thing you need to do is launch the Thing+ Gateway with APIKEY once. The Thing+ Gateway sends sensors status and values periodically based on the report interval.

- Discover
  The Thing+ Gateway sends a connected sensors and actuators list to the Thing+ Cloud. If sensors and actuators are hot plugged, the Thing+ Gateway sends an updated list to Thing+ Cloud. If not using Discover features, the user MUST add new sensors and actuators manually via the Thing+ Portal.

- TimeSync
  Hardware time MUST be syncronized with server time. The Thing+ Gateway can, however, syncronize time automatically. If you use the Thing+ Gateway, theres nothing required for you to do related to time syncronization. The Thing+ Gateway handles everything.

- Store DB
  If your network is disconnected, the Thing+ Gateway saves sensors values to local storage. The saved sensor values will be published after the network is reconnected.

4.3 Device Agent

Device Agent is a program that reads sensors values and actuates actuators. It is programed directly by hardware vendors. The Thing+ Gateway connects to the Device Agent with JSONRPC. The Thing+ Gateway is a JSONRPC Client and the Device Agent acts as a JSONRPC Server with a port number of 50800.

Server	Device Agent
PORT	**50800**
Client	Thing+ Gateway

4.3.1 JSONRPC Protocol

JSONRPC is light-weight remote procedure call protocol. It uses JSON as it's data format.(WIKI)

- Request format

property	description
method	Name of executino method or service
params	parameters
id	Request ID. Use this id when reponse.

- Response

property	description
result	The result of executed service or method
error	error object. Null if no error
id	The ID from request message

** Every message MUST end with NEW LINE(\n)**

- Error codes

Code	message
-32700	Parse error
-32600	Invald Request
-32601	Method not found
-32602	Invald params
-32603	Internal Error
-32000 to -32099	Server error

4.3.2 Device Agent method

Method	Description
discover	Request connected sensors and actuator lists.
sensor.get	Request series sensors value
sensor.set	Request actuating actuators
sensor.setNotification	Request using as event sensor
sensor.notification	Response Event sensors value and status.

4.3.2.1 "discover" method

Thing+ Gateway requests a sensor and actuator list

```
Device Agent <-- Thing+ Gateway
- Request Method : discover
- Request params : N/A

Device Agent --> Thing+ Gateway
- Response Result : [{"deviceAddress": DEV_ID, deviceModelId: DEVICE_MODEL_ID, "sensors":["id":ID, "type"
:TYPE, "notification": TRUE or FALSE}, ..., {"id":ID, "type": TYPE, "notification":true or false}]}]
  - deviceAddress: Device ID defined by Device Agent. Each device has unique ID.

  - deviceModelId : Device Model ID defined by Thing+
  - sensors: Sensors and actuator lists
    - id: ID
    - name: NAME
    - type : Sensor types defined by Thing+
    - notification : True if event sensor.
```

- Request Example (Thing+ Gateway)

```
{"id":1,"method":"discover","params":[]}\n
```

- Response Example (Device Agent)

```
{"id":1,"result":[{"deviceAddress":"0a0b0c0d0e00", "deviceModelId": "PowerOutlet", "sensors":[{"id":"0a0b
0c0d0e00-temperature-0","type":"temperature","name":"temp0"},{"id":"0a0b0c0d0e00-temperature-1","type":"t
emperature","name":"temp1", "notification": true},{"id":"0a0b0c0d0e00-humidity-0","type":"humidity","name
":"humi0"},{"id":"0a0b0c0d0e00-onoff-0","type":"onoff","name":"di0"},{"id":"0a0b0c0d0e00-powerSwitch-0","
type":"powerSwitch","name":"do0"}]}],"error":null}\n
```

4.3.2.2 "sensor.get" method

Request a series of sensor values

```
Device Agent <-- Thing+ Gateway
- Request Method : sensor.get
- Request params : [SENSOR ID]

Device Agent --> Thing+ Gateway
- Response Result :
                    {"value": VALUE} or
                    {"status": "on"|"off"|"err"} or
                    {"status": "err", "message": ERROR REASONE}

  - value : Sensor value
  - status : Sensor status.("on"|"off"|"err"). If the sensor value is exists and status is empty, the sta
tus is gussing it being "on"
  - message : If status is error, additional message (for error reasons(optional)).
```

- Request Example (Thing+ Gateway)

```
{"id":2,"method":"sensor.get","params":["0a0b0c0d0e00-
temperature-0"]}\n
```

- Response Example (Device Agent)

```
{"id":2,"result":{"value":5.63},"error":null}\n
{"id":2,"result":{"status":"off"},"error":null}\n
{"id":2,"result":{"status":"err","message":"initializing"},"error":null}\n
```

4.3.2.3 "sensor.set" method

Request actuating actuator

```
Device Agent <-- Thing+ Gateway
- Request Method : sensor.set
- Request Params : [Actuator ID, Command, Command Options]

Device Agent --> Thing+ Gateway
- Response Result : Result
```

- Request Example(Thing+ Gateway)

```
{"id":3,"method":"sensor.set","params":["0a0b0c0d0e00-powerSwitch-0","on",null]}\n
```

- Response Example (Device Agent)

```
{"id":3,"result":"on","error":null}\n
```

4.3.2.4 "sensor.setNotification" method

Request to set a sensor event

```
Device Agent <-- Thing+ Gateway
- Request Method : sensor.setNotification
- Request Params : [Sensor ID]

Device Agent --> Thing+ Gateway
- Response Result : "success" if success
```

- Request Example (Thing+ Gateway)

```
{"id":5,"method":"sensor.setNotification","params":["0a0b0c0d0e00-onoff-0"]}\n
```

- Response Example(Device Agent)

```
{"id":5,"result":"success","error":null}\n
```

** If using as an event sensor, the Device Agent MUST send it's sensor status on each report interval **

4.3.2.5 "sensor.notification" method

Response event with sensor values and status

```
Device Agent --> Thing+ Gateway
- Request Method : sensor.notificaion
- Request Params :
                   [Sensor ID, {"value": value}]
                   [Sensor ID, {"status": "on"|"off"|"err"}]
                   [Sensor ID, {"status": "err", "message":"ERROR REASON"}]
- Request Id : 없음

Device Agent <-- Thing+ Gateway
- Response : NONE
```

- Request Example (Device Agent)

```
{"method":"sensor.notification","params":["0a0b0c0d0e00-onoff-0",{"value": 1}]}
{"method":"sensor.notification","params":["0a0b0c0d0e00-onoff-0",{"status": "off"}]}
{"method":"sensor.notification","params":["0a0b0c0d0e00-onoff-0",{"status": "err","message":"initializing
"}]}
```

4.3.2.6 "ping" method

Check if the Device Agent is working or not

```
Device Agent <-- Thingp+ Gateway
- Request Method : sensor.setNotification
- Request Params : None

Device Agent --> Thing+ Gateway
- Response Result : "success"
```

- Request Example(Thing+ Gateway) `{"id":6,"method":"ping","params":[]}\n`
- Response Example(Device Agent) `{"id":6,"result":"success","error":null}\n`

Appendix

A. Gateway, Sensor Registration Request Form

If you need to register your gateway and sensor model in Thing +, please fill out the following document form and send it to us.

Gateway, Sensor Registration From

```
* Gateway infomation
1. Vendor name :
2. Gateway name :
3. The value to use on the gateway ID : MAC, IMEI or UUID
   (The gateway ID is a value that distinguishes each hardware and can not be duplicated with other gatew
ays.)
   (Please contact us if you can not use the above three items.)
4. Report interval (Minium is 60 second) :
   4.1. Whether Thing + Portal displays the report interval : Yes or No
   4.1. Whether the report interval can be changed : Yes or No
5. Maximum number of devices :

* Device information
1. Device name :
2. Whether using 'Discover' function : Yes or No
   (Discover is a function to find and register the sensors installed in the hardware.)
3. Sensor infomation
   - type :
   - data format :
   - unit :
4. Actuator infomation
   - Command :
     - Parameter :
     (In the parameter, specify the content of the required value, unit, and whether it is required.)
```

Examples

```
- Gateway information
1. Vendor name : Libelium
2. Gateway name : Meshlium
3. The value to use on the gateway ID: MAC
4. Report interval : 60 sec
   4.1. Whether Thing + Portal displays the report interval : Yes
   4.1. Whether the report interval can be changed : Yes
5. Maximum number of devices : 3

- Device information
1. Device name : Plug and Sensor
2. Whether using 'Discover' function : Yes
3. Sensor infomation
   - type : Temperature, format : Number, Unit : ℃
   - type : Humidity, format : Number, unit : %
   - type : Position, format : {x:number, y:number, z:number}, unit : m/s²
   - type : String Sensor, format : String, unit : None
4. Actuator information
   - type : LED
     - command : On
       - parameter
         1. Duration. unit : sec. Optional.
            Description : The LED will turn off after the duration. The parameter is optional; if not, th
e LED remains on.
     - command : Blink
       - parameter
         - 1. Interval. unit : sec. necessary
              Description : LED blinking is the value to control the speed. It lights up for interval tim
e and turns off for interval time.
         - 2. Duration. unit : sec. Optional.
     - command : Off
       - parameter : None
```

```
- Gateway information
1. Vendor name : Dell
2. Gateway name : Edge 5000
3. ID to use on the gateway: IMEI
4. Report interval : 90 seconds
   4.1. Whether Thing + Portal displays the report interval : No
   4.1. Whether the report interval can be changed : No
5. Maximum number of devices : 5

- Device information
1. Device name : Temerature Monitor
2. Whether using 'Discover' function : Yes
3. Sensor infomation
   - type : Temperature, unit : ℃

- Device information
1. Device name : Power Switch Controller
2. Whether using 'Discover' function : Yes
4. Actuation infomation
   - type : Power Switch
     - command : On
       - parameter : None
     - command : Off
       - parameter : None
```