



AgroSense_Barometric Pressure Sensor LoRaWAN® Manual V1.2

Author: Yuki

Time: 2024.11.01

Contents

1 Product Description	1
1.1 Introduction	1
1.2 Feature	1
1.3 Parameter	2
2 Technical route	3
2.1 System Framework	3
2.2 Regional frequency band	4
3 Usage	5
3.1 TTN and ThingSpeak	5
3.1.1 Network Server configuration	5
3.1.2 Decoder	8
3.1.3 Application Server configuration	10
3.1.4 Connect the Network Server and Application Server	11
3.1.5 Change Time Interval (5-1440min)	12
3.2 Datacake	14
3.2.1 Change Time Interval (5-1440min)	21

AgroSense_Barometric Pressure Sensor LoRaWAN®

Date	Versions	Description	Author
2024.7.16	V1.0	Introduction to Use & Function	Yuki
2024.10.16	V1.1	1. Changing the use of TTN to a new page. 2. LoRaWAN version: LoRaWAN specification 1.0.2 updated to 1.0.3. 3. Add downlink function.	Yuki
2024.11.01	V1.2	Add use in datacake	Yuki

1 Product Description

1.1 Introduction

AgroSense LoRaWAN® Barometric Pressure Sensor measures the barometric pressure in the atmosphere at the range of 300 to 1100 hPa, -40°C to 85°C with accuracy ± 0.12 hPa and resolution 0.01 hPa respectively, also with highly waterproof performance tested to IP68, making it widely applicable in agricultural environmental sensing scenarios to support the smart agricultural production.

The sensor benefits from LoRaWAN , which ensures stability and reliability. It is capable of covering a long transmission range while maintaining low power consumption. Unlike wireline devices, it is battery-powered, reducing the workload and complexity of deployment, design and development for end-users that can work via powering it, and setting the configuration in the cloud server, for LoRaWAN® remote monitoring. It monitors the barometric pressure and report every 1 hour.



1.2 Feature

- Includes a **high precision** sensor.
- Compatible with Worldwide **LoRaWAN® Networks**: Support the universal frequency bands EU868/ US915.
- LoRaWAN version: LoRaWAN Specification **1.0.3**.
- **Long Range**: Up to 2 kilometers in the city, up to 10 kilometers in the wilderness, receive sensitivity -137dBm , transmit power up to 21dBm.

- **Ultra-low power** consumption design, traditional AAA alkaline dry battery can be used for one year.
- **Data encryption:** Provide end-to-end secure communication, including device authentication and network data encryption, to ensure the security of data transmission and prevent data theft and malicious attacks.
- **High stability and reliability:** good stability in noisy environments, able to penetrate buildings and obstacles, so it can maintain good communication quality in urban and suburban environments.
- Suitable for **Harsh Environments:** Can work normally under the temperature of -40°C ~ 85°C, IP68 waterproof, suitable for outdoor use in harsh conditions, high UV, dusty, heavy rain and other bad weather.
- Monitor data and upload **real-time** data regularly.
- Modify the product parameters through **AT commands**.
- Support **downlink** to modify the time interval (5min-1440min).

1.3 Parameter

1. General Parameters

Product Model	AGLWBP01
Measurement Range	300-1100hPa
Measurement Accuracy	1hPa
Resolution	0.01hPa

2. Wireless Parameters

Communication Protocol	Standard LoRaWAN® protocol
Network Access/Operating Mode	OTAA Class A
MAX Transmit Power	21dBm
Receiver Sensitivity	-137dBm/125kHz SF=12
Frequency Band	EU868/US915

3. Physical Parameters

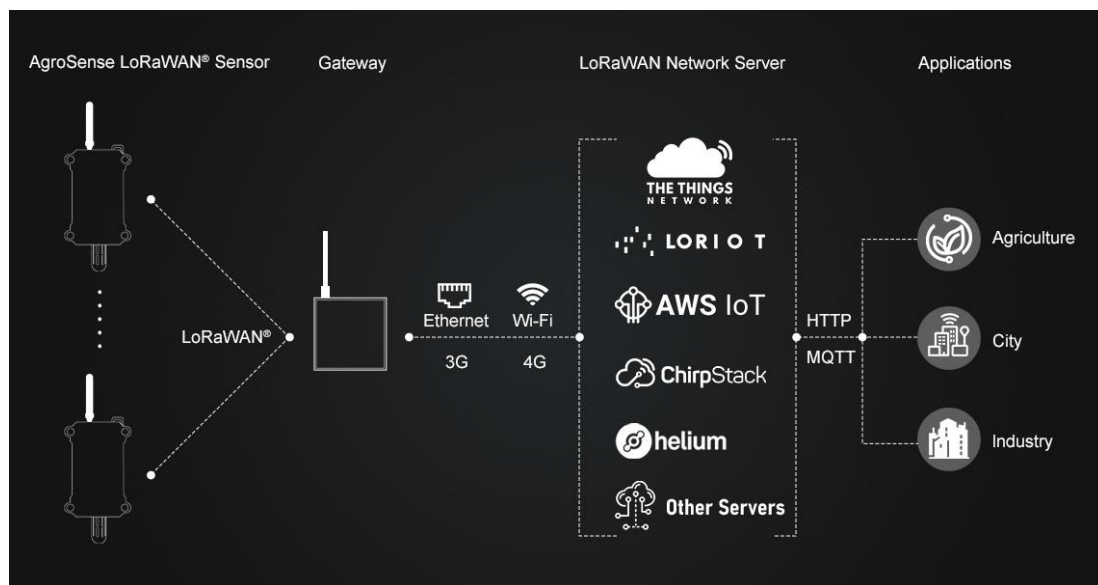
Power Supply	2 x AAA 1.5V batteries
Operating Temperature	-40°C ~85°C
Protection Class	IP68
Dimensions	131 × 62.7 × 27.5 mm
Mounting	Wall Mounting

2 Technical route

2.1 System Framework

AgroSense_Barometric Pressure Sensor uses LoRaWAN technology, and its network architecture includes four parts: End Nodes, Concentrator/Gateway, Network Server and Application Server.

End Nodes	It is responsible for collecting sensing data and then transmitting it to Gateway via the LoRaMAC protocol.
Concentrator/Gateway	It is mainly responsible for transmitting node data to the server.
Network Server	Organize the data into JSON packets and decode them.
Application Server	Display the data.



The steps to achieve the detection of Barometric Pressure is:

1. Collect the Barometric Pressure data by sensor, and send the data from End Node to Gateway.
2. The Gateway packages node data and transmits it to the Network Server.
3. The Network Server decodes the data and sends it to the Applications.
4. Finally, user can monitor the Barometric Pressure data in the APP.

2.2 Regional frequency band

At the present moment, our product solely accommodates compatibility with the US915 and EU868.

area	frequency band	center frequency
China	470-510MHz	CN486MHz
America	902-928MHz	US915MHz
Europe	863-870MHz	EU868MHz
Korea	920-923MHz	KR922MHz
Australia	915-928MHz	AU923MHz
New Zealand	921-928MHz	NZ922MHz
Asia	920-923MHz	AS923MHz

3 Usage

We use The Things Network as our Network Server, we need to configuration the country/ area frequency, inputting DEV EUI/ APP EUI/ APP Key, decodes, and connect to ThingSpeak.

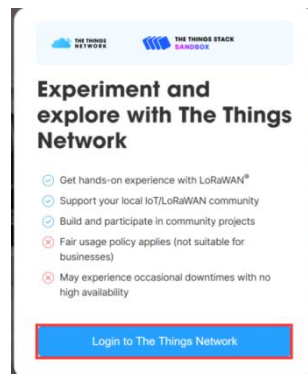
DEV EUI	Unique identification of device, authorized by IEEE
APP EUI	Unique identification of application
APP Key	One of the join network parameters on OTAA mode, calculated by DE EUI

- End Nodes and Gateway: AgroSense_Barometric Pressure Sensor LoRaWAN®. (The AgroSense series is applicable)
- Network Server: The Things Network. (Loriot, AWS IoT, ChirpStack, ect)
- Application Server: ThingSpeak.(Datacake, Blockbox, akenza, ect)

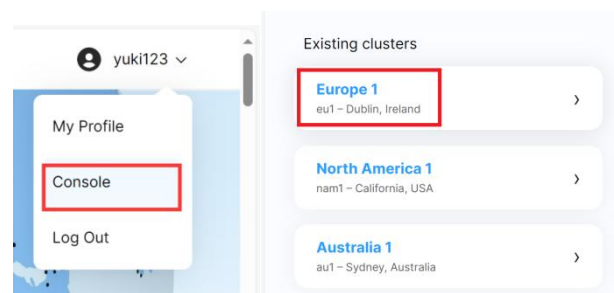
3.1 TTN and ThingSpeak

3.1.1 Network Server configuration

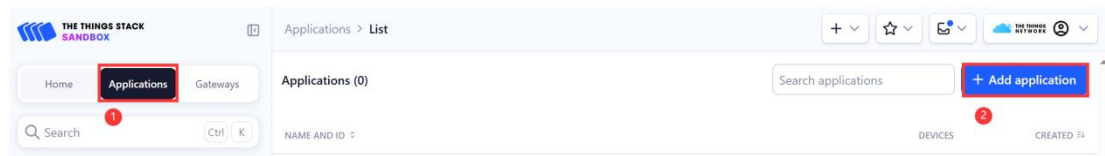
- Open The Things Network in your browser and login it. (Or register an account)



- Click “Console” and select clusters. (we take the European region for example.)



- Click “Go to applications” --> “+ Create application”.



- Write the Application ID and click “Create application”.

Application ID *

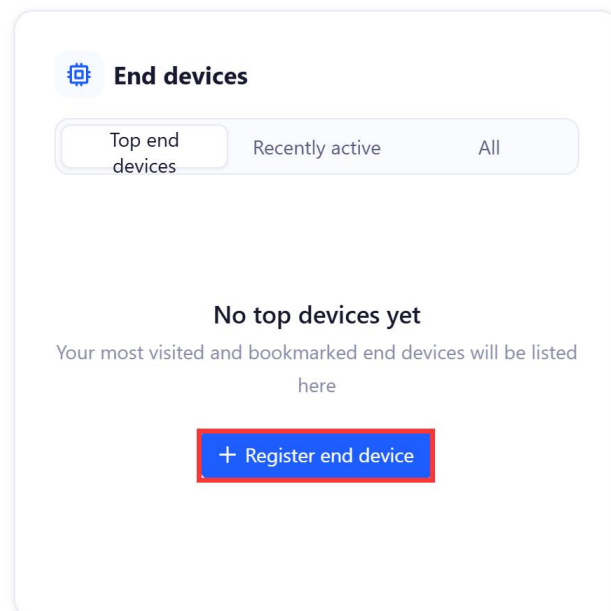
Application name

Description

Optional application description; can also be used to save notes about the

Create application

- Click “+ Register and device”.



- Following the steps, and input the DEV EUI/ APP EUI/ APP Key (notice: JoinEUI=APP EUI) and subsequently click on "Register end device" to complete the registration process.

End device type

Input method ⓘ

☐ Select the end device in the LoRaWAN Device Repository

☒ Enter end device specifics manually 1

Frequency plan ⓘ *

Europe 863-870 MHz (SF9 for RX2 - recommended) | v

LoRaWAN version ⓘ *

LoRaWAN Specification 1.0.3 | v

Regional Parameters version ⓘ *

RP001 Regional Parameters 1.0.3 revision A | v 2

[Show advanced activation, LoRaWAN class and cluster settings](#)

Provisioning information

JoinEUI ⓘ *

48 FF 00 00 00 00 01 65

Confirm

3 Continue, please enter the JoinEUI 4 and device so we can determine onboarding options

Provisioning information

JoinEUI ⓘ *

48 FF 00 00 00 00 01 65 Reset

This end device can be registered on the network

DevEUI ⓘ *

48 E6 63 FF FE 30 01 65 Generate 0/50 used

AppKey ⓘ *

4A 35 62 6B 95 AB 5B 4D 3F 3B DE 12 71 B1 6F 2A Generate

End device ID ⓘ *

eui-48e663fffe300165

This value is automatically prefilled using the DevEUI

After registration

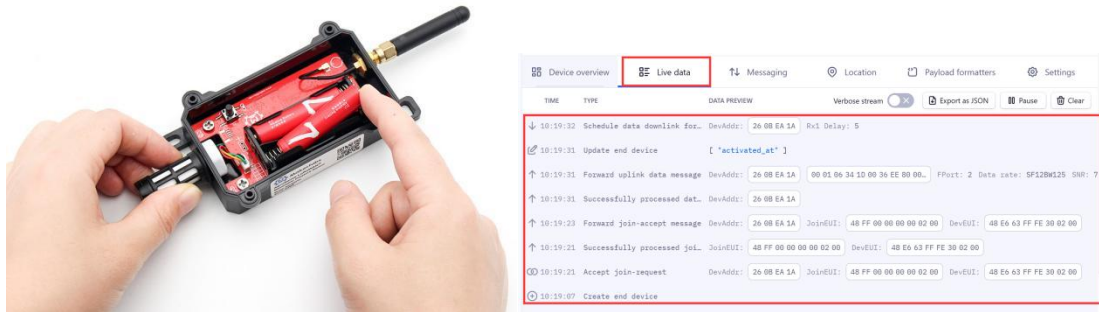
☒ View registered end device

☐ Register another end device of this type

Register end device

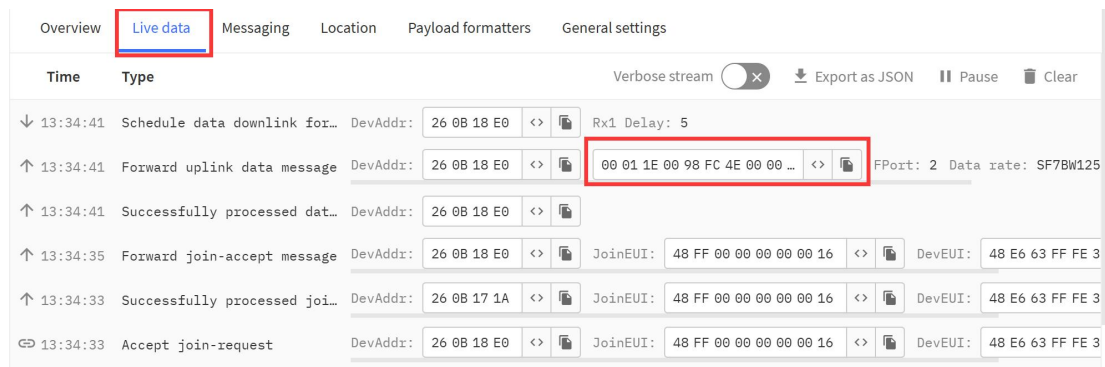


- Plug the battery and press RES button, you can see the device is connected successfully in the TTN.



3.1.2 Decoder

- Now, we need to decode the data.



Data length	Data description	Value range	Explanation
byte 0	Data packet sequence number high 8 bits	0-0xFFFF	Counting starts from 0 and increments, resetting back to 0 after reaching 65535
byte 1	Data packet sequence number low 8 bits		
byte 2	Battery voltage		The value is obtained by amplifying the data by 10 times, and the actual value needs to be divided by 10 to convert to the actual battery voltage. The purpose of multiplying by 10 is to retain one decimal place of the voltage value. For example, if the value is 0x21 = 33, then the battery voltage is 3.3V.
byte 3	Pressure sensor bits 24 to 31		This data is enlarged 100 times, the real value needs to be divided by 100 to get the actual atmospheric pressure value, the unit Pa. The purpose of multiplying by 100 is to retain the value of atmospheric pressure after 2 decimal places. For example, the Bit23 to Bit0 of the value is 0x0098bb53 = 10009427, divided by 100, it is 100094.27hPa.
byte 4	Pressure sensor bits 16 to 23		
byte 5	Pressure sensor bits 8 to 15		
byte 6	Pressure sensor		

	bits 0 to 7		
byte 7	NC		
byte 8	NC		

Example: 0x00, 0x02, 0x1C, 0x00, 0x9A, 0x7E, 0xAA, 0x00, 0x00

Data parsing:

Battery voltage is 2.8 V.

Atmospheric pressure is 101249.70 Pa.

- Know how to decode it after, we need to write it in code. (you can check it out on [Github](#))

```
function decodeUplink(input) {
  // var num = input.bytes[0] * 256 + input.bytes[1]
  var bat = input.bytes[2] / 10.0
  var press = (input.bytes[3] * 16777216 + input.bytes[4] * 65536 + input.bytes[5] * 256 + input.bytes[6]) /
100000.0
  var temperature = (input.bytes[7] * 16777216 + input.bytes[8] * 65536 + input.bytes[9] * 256 +
input.bytes[10]) / 100.0;
  return {
    data: {
      field1: bat,
      field2: press,
      field3: temperature
    },
  };
}
```

- Select “Payload formatters” and follow the steps.

The screenshot shows the 'Payload formatters' configuration page. The 'Setup' section is active, showing 'Formatter type' as 'Custom Javascript formatter'. The 'Formatter code' field contains the following JavaScript code:

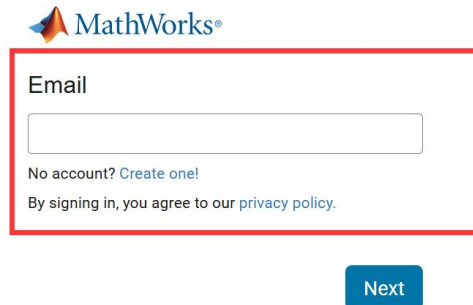
```
function decodeUplink(input) {
  // var num = input.bytes[0] * 256 + input.bytes[1]
  var bat = input.bytes[2] / 10.0
  var press = (input.bytes[3] * 16777216 + input.bytes[4] * 65536 + input.bytes[5] * 256 + input.bytes[6]) / 100000.0
  var temperature = (input.bytes[7] * 16777216 + input.bytes[8] * 65536 + input.bytes[9] * 256 + input.bytes[10]) / 100.0;
  return {
    data: {
      field1: bat,
      field2: press,
      field3: temperature
    },
  };
}
```

A 'Save changes' button is located at the bottom of the configuration area.

3.1.3 Application Server configuration

In the Application Server configuration, we need to create ThingSpeak channel and get Channel ID and API Key, this is the key to our connection to TTN.

- Login to the ThingSpeak. (Or register an account)



MathWorks®

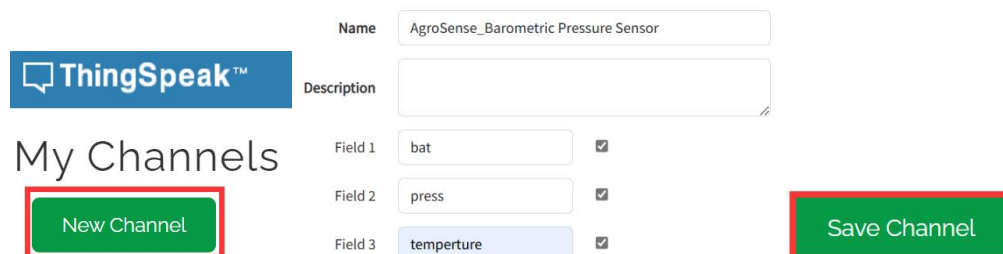
Email

No account? [Create one!](#)

By signing in, you agree to our [privacy policy](#).

Next

- Click “New Channel”, fill in the Channel name and field names and click “Save Channel”.



ThingSpeak™

My Channels

New Channel

Name: AgroSense_Barometric Pressure Sensor

Description:

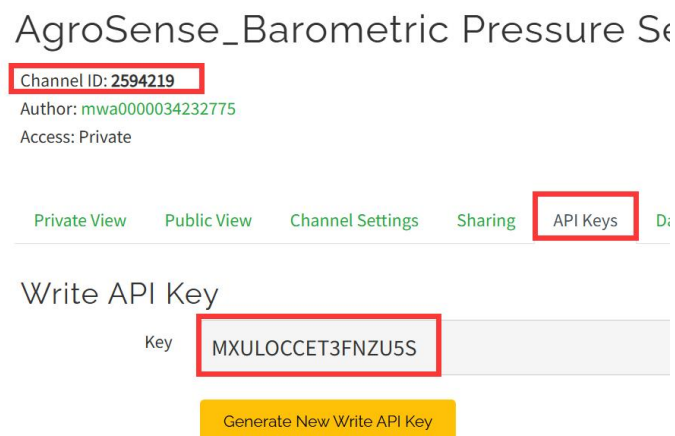
Field 1: bat ☒

Field 2: press ☒

Field 3: temperture ☒

Save Channel

- After successful creation, copy the Channel ID and API Key.



AgroSense_Barometric Pressure Sensor

Channel ID: 2594219

Author: mwa0000034232775

Access: Private

Private View Public View Channel Settings Sharing API Keys

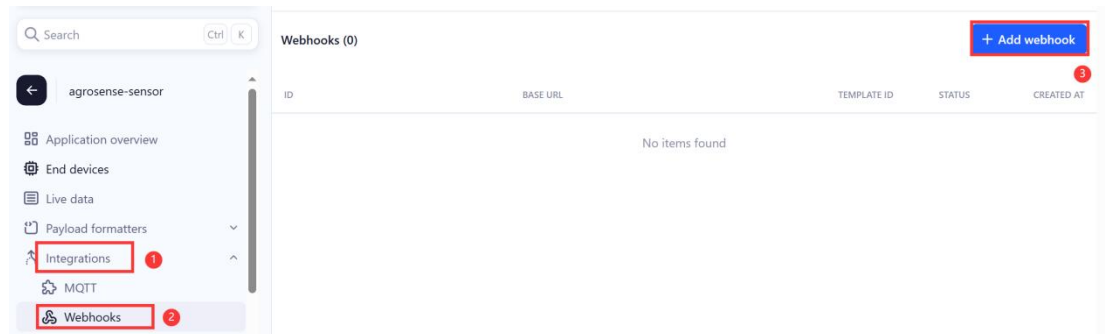
Write API Key

Key: MXULOC CET3FNZU5S

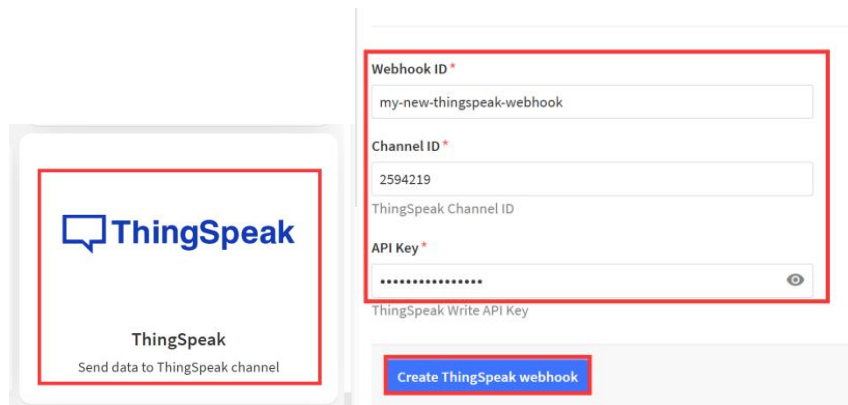
Generate New Write API Key

3.1.4 Connect the Network Server and Application Server

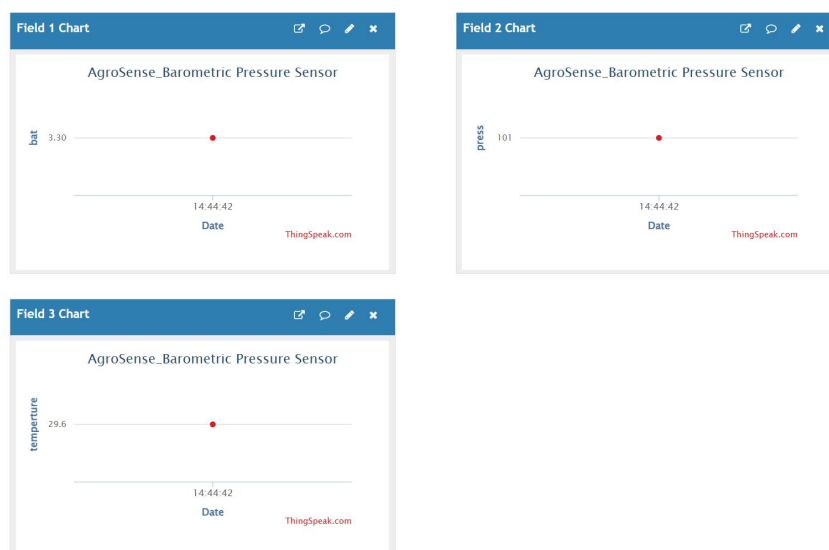
- In the TTN, click “integrations” --> “Webhooks” --> “+ Add webhook”.



- Select “ThingSpeak”, Fill in the Webhook ID and paste the Channel ID and API Key, click “Create ThingSpeak Webhook”.



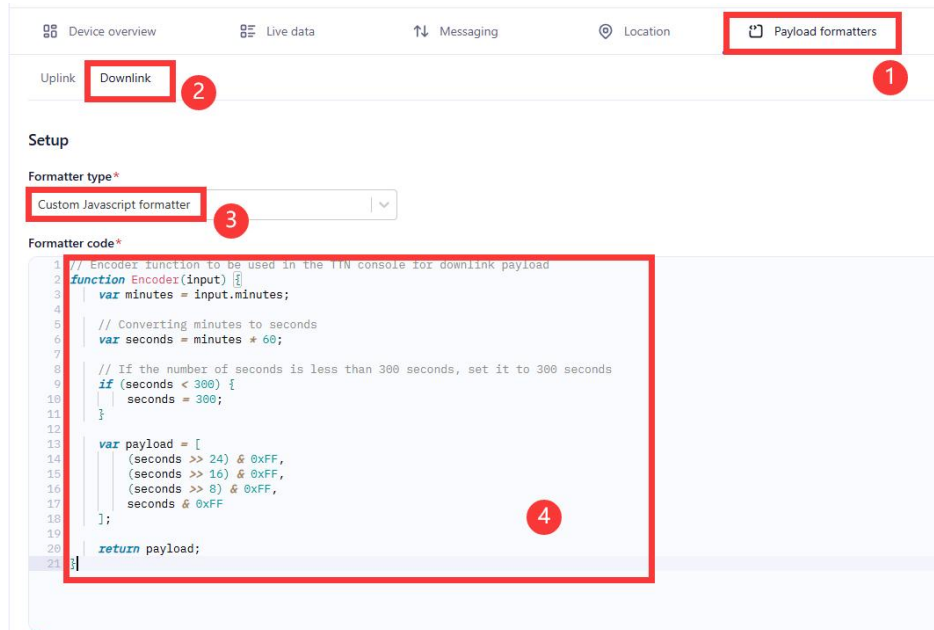
- Press RES button, wait about a minute, you will successfully see the data in ThingSpeak.(You will receive the data every hour.)



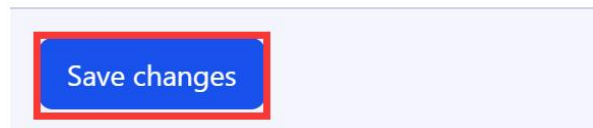
3.1.5 Change Time Interval (5-1440min)

1、If you need to change time Interval (Default 60 minutes), you can click “Payload formatters-->Downlink” and follow the steps.

Formatter code you can find in [Github](#).



2、Click “Save changes”.



3、Click “Messaging-->Schedule downlink”.

Note: you must use this format:

```

{
  "minutes": 5
}

```

Device overview Live data Messaging

Schedule downlink Simulate uplink

Schedule downlink

Insert Mode

☒ Replace downlink queue

☐ Push to downlink queue (append)

FPort*

1

Payload type

☐ Bytes ☒ JSON

Payload

```
{  
  "minutes": 180  
}
```

The decoded payload of the downlink message

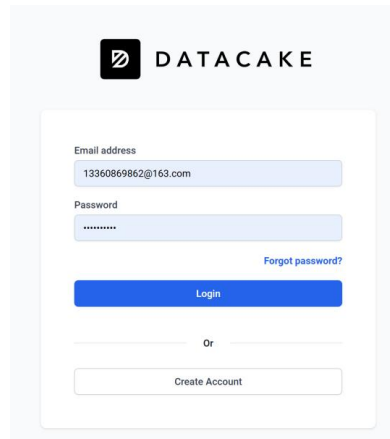
☐ Confirmed downlink

Schedule downlink

4、The modified interval will be updated after the next data upload.

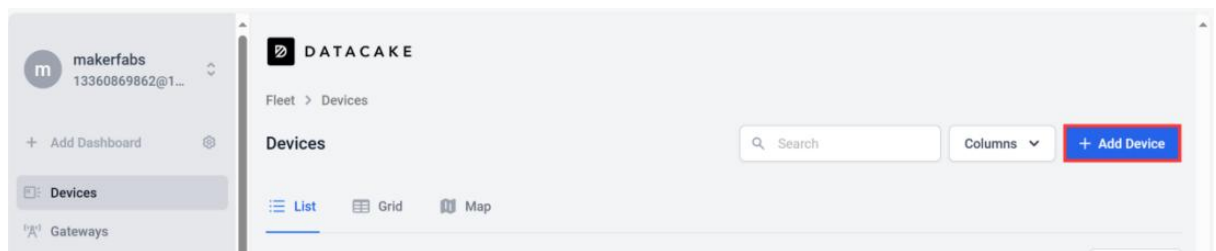
3.2 Datacake

1、Login datacake or Create Account

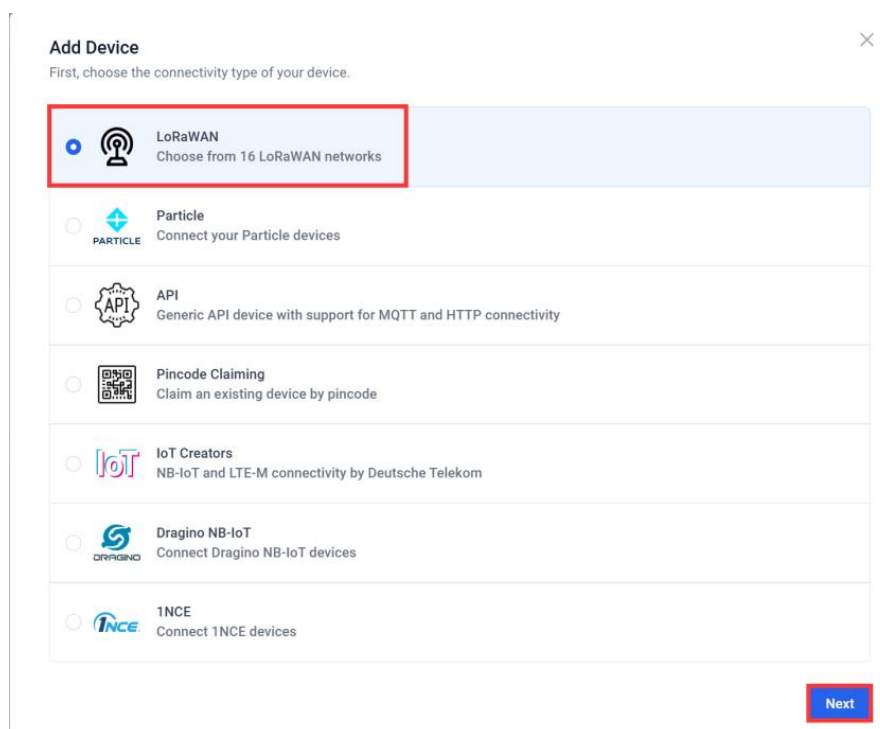


The image shows the Datacake login and registration interface. At the top is the Datacake logo. Below it is a form with two input fields: 'Email address' containing '13360869862@163.com' and 'Password' with masked characters. A 'Forgot password?' link is next to the password field. Below the inputs are two buttons: a blue 'Login' button and a white 'Create Account' button separated by an 'Or' text.

2、Click “Add Device”



3、Select LoRaWAN and click “Next”



The image shows the 'Add Device' dialog box. It has a title 'Add Device' and a subtitle 'First, choose the connectivity type of your device.' Below this is a list of connectivity options, each with a radio button and an icon:

- LoRaWAN** (selected, highlighted with a red box): Choose from 16 LoRaWAN networks
- Particle: Connect your Particle devices
- API: Generic API device with support for MQTT and HTTP connectivity
- Pincode Claiming: Claim an existing device by pincode
- IoT Creators: NB-IoT and LTE-M connectivity by Deutsche Telekom
- Dragino NB-IoT: Connect Dragino NB-IoT devices
- 1NCE: Connect 1NCE devices

 At the bottom right of the dialog is a blue 'Next' button, also highlighted with a red box.

4、Select a Product based on your needs, take "Create new empty product" as an example.

Add LoRaWAN Device

You can add individually billed devices.

STEP 1

STEP 2

STEP 3

STEP 4

Product

Network Server

Devices

Plan

Datacake Product

You can add devices to an existing product on Datacake, create a new empty product or start with one of the templates. Products allow you to share the same configuration (fields, dashboard and more) between devices.

New Product from template
Create new product from a template

Existing Product
Add devices to an existing product

New Product
Create new empty product

New Product

If your device is not available as a template, you can start with an empty device. You will have to create the device definition (fields, dashboard) and provide the payload decoder in the device's configuration.

Product Name

Agrosense sensor

Back

Next

5、Select "Datacake LNS"

Add LoRaWAN Device

STEP 1

STEP 2

STEP 3

STEP 4

Product

Network Server

Devices

Plan

Network Server

Please choose the LoRaWAN Network Server that your devices are connected to.

Datacake LNS

AUTOMATIC SETUP

Start and scale easily with a managed LNS

Uplinks
Downlinks

☐
The Things Stack V3
TTN V3 / Things Industries

Uplinks
Downlinks

☐
Helium
Use your own console

Uplinks
Downlinks

☐
LORIoT

Uplinks
Downlinks

☐
ChirpStack

Uplinks
Downlinks

☐
Activity

Uplinks
Downlinks

☐
KPN

Uplinks
Downlinks

Showing 1 to 6 of 15 results

Previous

Next

Back

Next

6、Enter DEVEUI、APPEUI、APPKEY、FREQUENCY(take 915 for example) and DEVICE CLASS.

7、Choose the type according to your needs, and click “Add 1 device”.

8、Click to go to the device you just added.

DATA CAKE

Fleet > Devices

Devices

Search Columns + Add Device

List Grid Map

Actions

DEVICE	PRIMARY	SECONDARY	DEVICE SIGNAL	DEVICE BATTERY	
AgroSense_Air Temperature and Humidity Sensor	40.2	25	-48	2.5	👁️ ⋮ ⌵
AgroSense-carbon dioxide (CO2) sensor	0	N/A	-86	3.3	👁️ ⋮ ⌵
agrosense sensor	N/A	N/A	N/A	N/A	👁️ ⋮ ⌵

Showing 1 to 3 of 3 results

50 per page Previous Next

9、Click “Configuration”, enter Decoder and click “Save”.(You can check it out on [Guihub](#))

The screenshot shows the Datacake interface for the 'agrosense sensor'. The 'Configuration' tab is selected and highlighted with a red box. Below the configuration tab, the 'Payload Decoder' section is visible. The code editor shows a JavaScript function for decoding the payload, which is also highlighted with a red box. The function is as follows:

```
function Decoder(payload, port) {
  var input = {
    bytes: payload
  };

  // var num = input.bytes[0] * 256 + input.bytes[1];
  var bat = input.bytes[2] / 10.0;
  var press = (input.bytes[3] * 16777216 + input.bytes[4] * 65536 + input.bytes[5] * 256 + input.bytes[6]) / 100000.0;
  var temperature = (input.bytes[7] * 16777216 + input.bytes[8] * 65536 + input.bytes[9] * 256 + input.bytes[10]) / 100.0;

  var decoded = {
    bat: bat,
    press: press,
    temperature: temperature,
  };

  // Test for LoRa properties in normalizedPayload
  try {
    console.log('normalizedPayload:', normalizedPayload); // Log to check normalizedPayload structure
    decoded.lora_payload = normalizedPayload;
    decoded.lora_payload.gateway = Array.isArray(normalizedPayload.gateway) ? normalizedPayload.gateway.length : 0;
    decoded.lora_payload.gateway_id = Array.isArray(normalizedPayload.gateway) ? normalizedPayload.gateway[0].id : 0;
    decoded.lora_payload.gateway_name = Array.isArray(normalizedPayload.gateway) ? normalizedPayload.gateway[0].name : 0;
    decoded.lora_payload.gateway_address = normalizedPayload.gateway_address || 'not microcontroller';
    console.log('Error occurred while decoding LoRa properties: ' + error);
  } catch (error) {
    console.log('Error occurred while decoding LoRa properties: ' + error);
  }

  return {
    #bat: bat, value: decoded.bat,
    #press: press, value: decoded.press,
    #temperature: temperature, value: decoded.temperature,
    #lora_payload: decoded.lora_payload,
    #lora_gateway: decoded.lora_gateway,
    #lora_gateway_name: decoded.lora_gateway_name,
    #lora_gateway_address: decoded.lora_gateway_address,
  };
}
```

Below the code editor, there are sections for 'Payload', 'console.log Output', and 'Recognized measurements'. The 'Payload' section shows the input payload. The 'console.log Output' section shows the output of the console.log statement. The 'Recognized measurements' section shows the decoded measurements.

```
function Decoder(payload, port) {
  var input = {
    bytes: payload
  };

  // var num = input.bytes[0] * 256 + input.bytes[1];
  var bat = input.bytes[2] / 10.0;
  var press = (input.bytes[3] * 16777216 + input.bytes[4] * 65536 + input.bytes[5] * 256 + input.bytes[6]) / 100000.0;
  var temperature = (input.bytes[7] * 16777216 + input.bytes[8] * 65536 + input.bytes[9] * 256 + input.bytes[10]) / 100.0;

  var decoded = {
    bat: bat,
    press: press,
    temperature: temperature,
  };

  // Test for LoRa properties in normalizedPayload
  try {
    console.log('normalizedPayload:', normalizedPayload); // Log to check normalizedPayload structure
  } catch (error) {
    console.log('Error occurred while decoding LoRa properties: ' + error);
  }

  return {
    #bat: bat, value: decoded.bat,
    #press: press, value: decoded.press,
    #temperature: temperature, value: decoded.temperature,
    #lora_payload: decoded.lora_payload,
    #lora_gateway: decoded.lora_gateway,
    #lora_gateway_name: decoded.lora_gateway_name,
    #lora_gateway_address: decoded.lora_gateway_address,
  };
}
```

```

        decoded.lora_rssi =
            (normalizedPayload.gateways
                Array.isArray(normalizedPayload.gateways) && normalizedPayload.gateways.length >
                0 && normalizedPayload.gateways[0].rssi) || 0;
        decoded.lora_snr =
            (normalizedPayload.gateways
                Array.isArray(normalizedPayload.gateways) && normalizedPayload.gateways.length >
                0 && normalizedPayload.gateways[0].snr) || 0;
        decoded.lora_datarate = normalizedPayload.data_rate || 'not retrievable';
    } catch (error) {
        console.log('Error occurred while decoding LoRa properties: ' + error);
    }
    return [
        { field: "bat", value: decoded.bat },
        { field: "press", value: decoded.press },
        { field: "temperature", value: decoded.temperature },
        { field: "lora_rssi", value: decoded.lora_rssi },
        { field: "lora_snr", value: decoded.lora_snr },
        { field: "lora_datarate", value: decoded.lora_datarate }
    ];
}

```

10、 Follow the steps to add a field. (Every fields is the same way)

Fields

Fields describe the data the device will store.

+ Add Field

Add Field

Fields define the schema of the data the device stores.

Type

Float

Name

Bat

Identifier

BAT

The field identifier is a unique string that can consist of uppercase letters, numbers and underscores. Once a field has been created, the identifier can not be changed.

Unit

Optional

Role

None

You can define the role of a field, which are unique per product and can be used to add context to global visualisations and reports.

Formula

Optional

Formulas can be used to perform calculations on values from other fields. Fields that have a formula can not be written to from a decoder or via the API.

☐ Use Formula

Cancel

Add Field

NAME	IDENTIFIER	TYPE	ROLE
Bat	BAT	Float	N/A
Press	PRESS	Float	N/A
Temperature	TEMPERATURE	Float	N/A

11、Press RST button, wait until the sensor connects to the gateway successfully, you will see the data the sensor is currently reading.

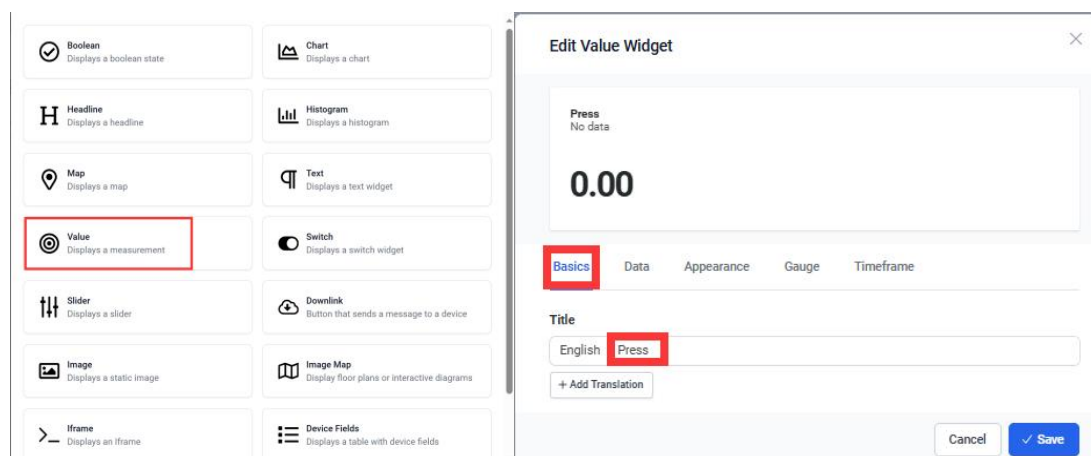
NAME	IDENTIFIER	TYPE	ROLE	CURRENT VALUE	LAST UPDATE
Bat	BAT	Float	N/A	2.9	1 second ago
Press	PRESS	Float	N/A	1,456	16 seconds ago
Temperature	TEMPERATURE	Float	N/A	25.7	9 seconds ago

12、To get a better look at the data, we can add widget.

Click “Dashboard-->switch-->+ Add Widget”.



13、Select “Value” and set Title, Field and presentation form as well as the interval color.



Edit Value Widget

Press
6 minutes ago

1,456.00

Basics **Data** Appearance Gauge Timeframe

Field
Field
Press

Unit
English

+ Add Translation ☒ Sync Translations With Other Widgets

Decimal Places
2

Cancel **Save**

Edit Value Widget

Press
7 minutes ago

Basics Data Appearance **Gauge** Timeframe

Gauge Type
☐ None ☐ Linear ☐ Vertical ☒ **Circular** ☐ Fill Level ☐ Compass

Values	Color
0	#f56565
1500	#48bb78
3000	#f56565

Add

Cancel **Save**

14、Select Chart and set Title, Field, Kind, Line Thickness and click “save”.

Edit Chart Widget

Press
No data

Basic **Data** Appearance Axes Timeframe Reference Lines

Title
English **Press**

+ Add Translation

Cancel **Save**

Edit Chart Widget

Press
10 minutes ago

Basics **Data** Appearance Axes Timeframe Reference Lines

Field
Press

Values
Absolute Change

Label
Press

Kind
☐ Line Chart ☒ **Area Chart** ☐ Bar Chart

Color
#F5A623

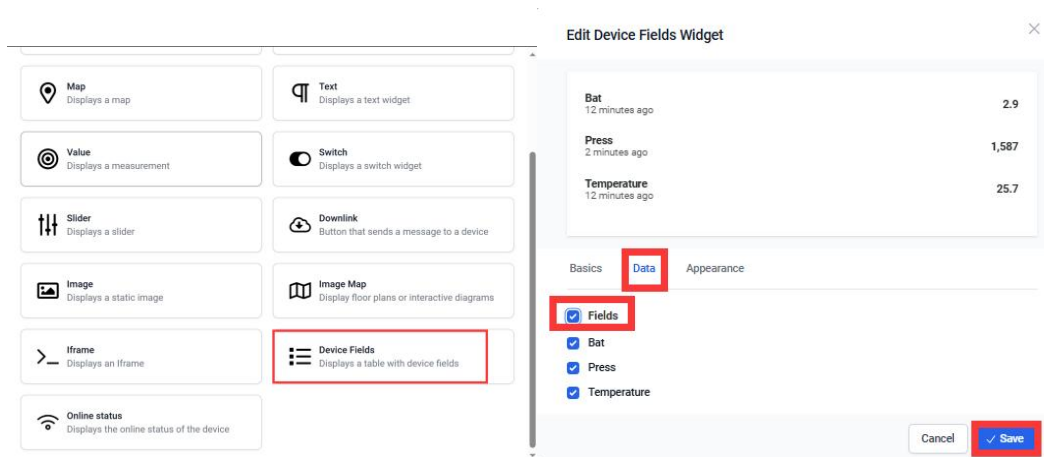
Interpolation Type
Linear

Y Axis
Axis 1

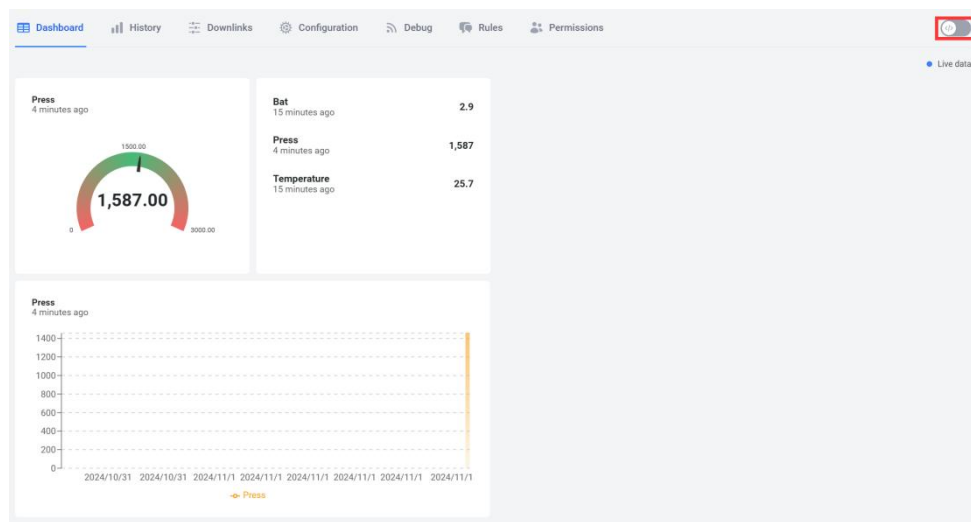
+ Add

Cancel **Save**

15、Select Device Fields, check “Fields” and click “Save”.



16、Click the switch to save, and you can see the data visually.

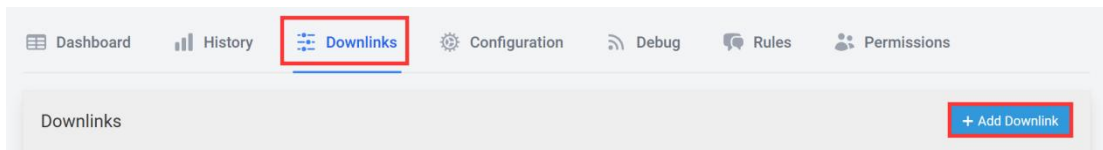


3.2.1 Change Time Interval (5-1440min)

1 、 If you need to change time Interval (Default 60 minutes), you can click “Configuration-->Fields-->+Add Field”

The screenshot shows the 'Add Field' dialog. The 'Type' is set to 'Integer', the 'Name' is 'Sending Time Interval', and the 'Identifier' is 'SENDING_TIME_INTERVAL'. The 'Add Field' button is highlighted with a red box.

2、Click "Downlink-->Add Downlink".



Enter name、description、fields used and payload encoder respectively.

Name: Set User-Defined Sending Time Interval

Description: Set the user-defined report transmission interval and store it in the configuration variable.(5Min-1440Min)

Payload Encoder: copy in [Github](#).

Configure Downlink

Name

Description

Fields used

If your encoder function takes input from the device's fields, you can specify them here. They will be used to create the form for the downlink generator.

☐ Trigger on measurements

If activated, each time the device records a measurement in one of the fields used, the downlink will be sent automatically.

Port

Payload Encoder

```

1 function Encoder(measurements, port) {
2   var interval = measurements["SENDING_TIME_INTERVAL"].value * 60;
3   if (interval < 300) {
4     interval = 300;
5     console.log("Interval < 300 Seconds / 5 Minutes not allowed!");
6   }
7   // Convert to hexadecimal only from interval
8   return interval.toString(16).padStart(4, '0').match(/.{2}/g).map(function(f) {return parseInt(f, 16)});
9 }
10
11 /**
12  * String.prototype.padStart() polyfill
13  * https://github.com/uxitten/polyfill/blob/master/string.polyfill.js
14  * https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/padStart
15  */
16 if (!String.prototype.padStart) {
17   String.prototype.padStart = function padStart(targetLength,padString) {
18     targetLength = targetLength>>0; //truncate if number or convert non-number to 0;
19     padString = String((typeof padString !== 'undefined' ? padString : ' '));
20     if (this.length > targetLength) {
21       return String(this);
22     }
23     else {
24       targetLength = targetLength-this.length;
25       if (targetLength > padString.length) {
26         padString += padString.repeat(targetLength/padString.length); //append to original to en

```

3、Click "Dashboard-->switch-->+ Add Widget".

Select "Downlink" and setting as follow image.

The image shows two side-by-side screenshots of the 'Edit Downlink Widget' dialog box. The left screenshot shows the 'Basics' tab with the title 'User-Defined Time Interval(5Min-1440Min)' highlighted. The right screenshot shows the 'Data' tab with the 'Downlink' dropdown set to 'Set User-Defined Sending Time Interval' and the 'Save' button highlighted.

4、Click the switch to save, and you can click to change your time Interval.

The image shows two screenshots. The top screenshot shows the 'User-Defined Time Interval(5Min-1440Min)' widget. The bottom screenshot shows a 'Sending Time Interval' dialog box with the value '1' in the input field and the 'Save measurements and send downlink' button highlighted.