Plat'Home
TECHNOLOGY to serve you.

OpenBlocks IoT Family Data Handling Guide

Ver. 3.1.0

Plat'Home Co., Ltd.

Table of contents

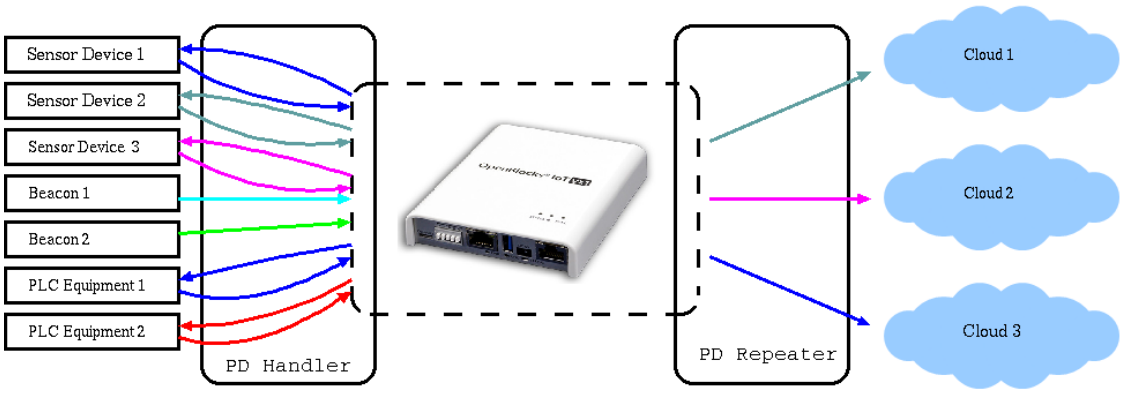
Chapter 1 General

This Guide describes the data handling functions used by the OpenBlocks IoT Family. For setup, a client device (PC, smartphone, tablet PC, etc.) that can use a web browser is required. For the web user interface (hereinafter referred to as "WEB UI"), refer to the OpenBlocks IoT Family WEB UI Set-up Guide.

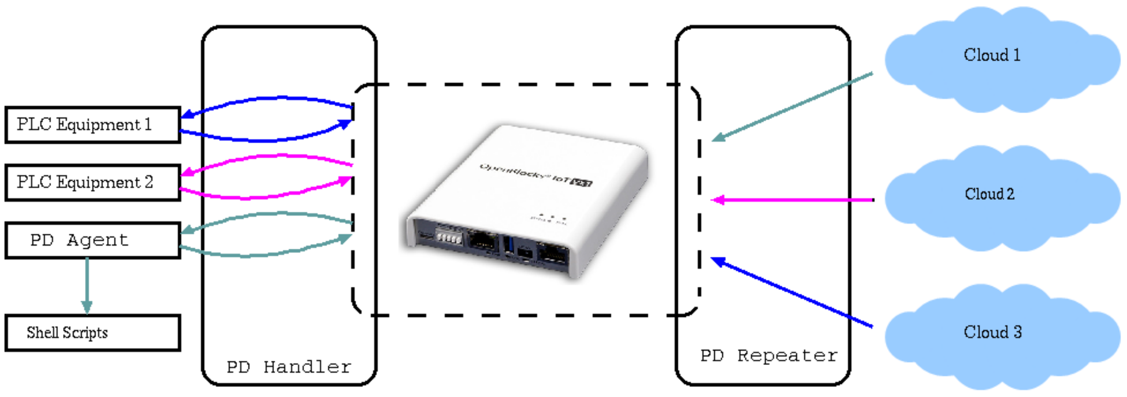
Chapter 2 IoT data control functions

The IoT data control function of the OpenBlocks IoT Family supports a collection function that acquires data from BLE or UART and other sensor devices, in addition to PLC equipment supporting Modbus, and sends information to clouds, etc. and a downstream control function to receive messages from clouds, etc., while controlling PLC equipment supporting Modbus, etc. Refer to our website for all supported sensor devices, etc.

The collection function collects data from devices and sends information to cloud destinations, etc. As data is stored in the OpenBlocks IoT Family as a temporary buffer, data can be safely sent due to the fact that even if a network problem occurs, resending is possible.



The downstream control function receives control messages from clouds and can control PLC supporting Modbus, etc.



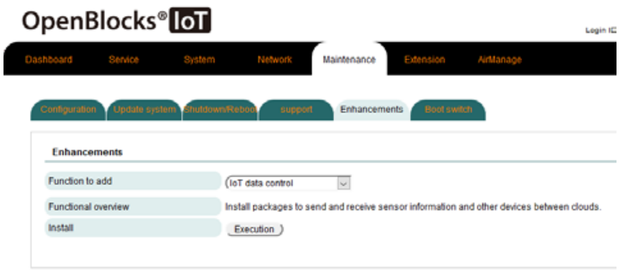
To use the IoT data control function, it is necessary to perform the basic setup for service functions and also install the IoT data control package.

2-1. Basic setup of service functions

To use the IoT data control function, it is necessary to choose the **[Service]-[Basic]** tab in WEB UI and carry out control of BT interface and registration of various devices.
For basic setup of the service function, refer to Chapter 5 of the OpenBlocks IoT Family WEB UI Set-up Guide.

2-2. Installing IoT data control packages

To use the IoT data control function, it is necessary to choose the **[Maintenance]-[Enhancements]** tab and install the IoT data control package.
*IoT data control can work in conjunction with Node-RED. To use this control in conjunction with Node-RED, install Node-RED, too.
*IoT data control can also send data to Azure IoT Edge. To send data to Azure IoT Edge in the same OpenBlocks, install Azure IoT Edge, too.



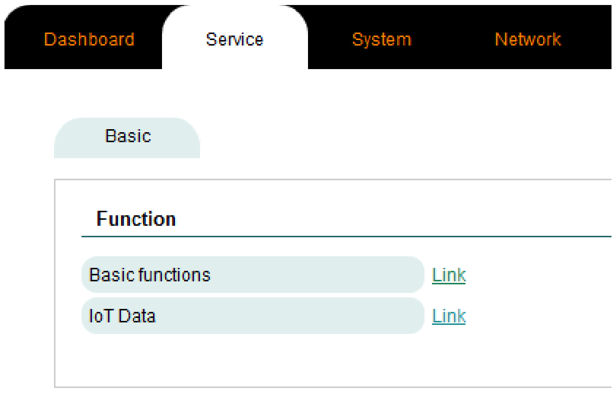
Enhancements

Function to add	IoT data control
Functional overview	Install packages to send and receive sensor information and other devices between clouds.
Install	Execution

Choose the “IoT data control” in pull-down menu of the **[Maintenance]-[Enhancements]** tab and click the “Execution” button. If the IoT data control package has already been installed, the “IoT data control” will be not displayed on pull-down menu.

2-3. Application setup for IoT data control

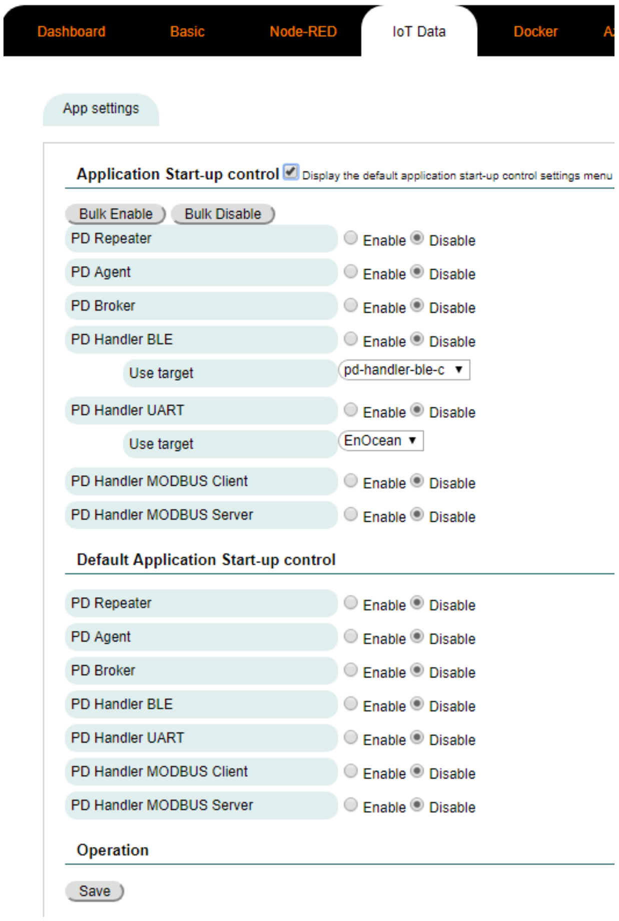
If the IoT data control package has been installed, the link to the **IoT data** will be displayed when choosing the **[Service]-[Basic]** tab.



Choose the **[Service]** tab and then click on the link to the **IoT Data**. The root tab will switch to a combination of the **[Dashboard]**, **[Basic]**, **[IoT Data]**.

*Links shown in the **[Service]-[Basic]** tab of will differ, depending on packages installed by the **[Maintenance]-[Enhancements]** tab.

Choose the **[IoT Data]-[App Setting]** tab to show the App settings screen. Control the startup of applications to use as well as setting up the config setting mode.



Application Start-up control [✓] Display the default application start-up control settings menu

PD Repeater	Enable [✓ Disable]
PD Agent	Enable [✓ Disable]
PD Broker	Enable [✓ Disable]
PD Handler BLE	Enable [✓ Disable]
Use target	pd-handler-ble-c ▼
PD Handler UART	Enable [✓ Disable]
Use target	EnOcean ▼
PD Handler MODBUS Client	Enable [✓ Disable]
PD Handler MODBUS Server	Enable [✓ Disable]

Default Application Start-up control

PD Repeater	Enable [✓ Disable]
PD Agent	Enable [✓ Disable]
PD Broker	Enable [✓ Disable]
PD Handler BLE	Enable [✓ Disable]
PD Handler UART	Enable [✓ Disable]
PD Handler MODBUS Client	Enable [✓ Disable]
PD Handler MODBUS Server	Enable [✓ Disable]

Operation

*Example of screen when "Display the default application start-up control setting" check box is checked

Screen shot on the left shows the default state of the **[IoT Data]- [App settings]** tab.

When the “Display the default application start-up control setting” check box is checked, the Default Application Start-up control setup menu will be displayed. To enable startup control of individual applications, it is necessary to set default settings to “Enable”.

When the “Display the config” check box is checked, the Configuration Settings menu will be displayed to switch between using the configuration set up by this WEB UI or that defined by the user.

The “Bulk Enable” and “Bulk Disable” buttons will set all settings to “Enable” or “Disable”.

PD Repeater : Application to send/receive data or control messages to/from cloud.

PD Agent : Application that receives control message from cloud via the PD Repeater and executes a preset shell script or execution object.

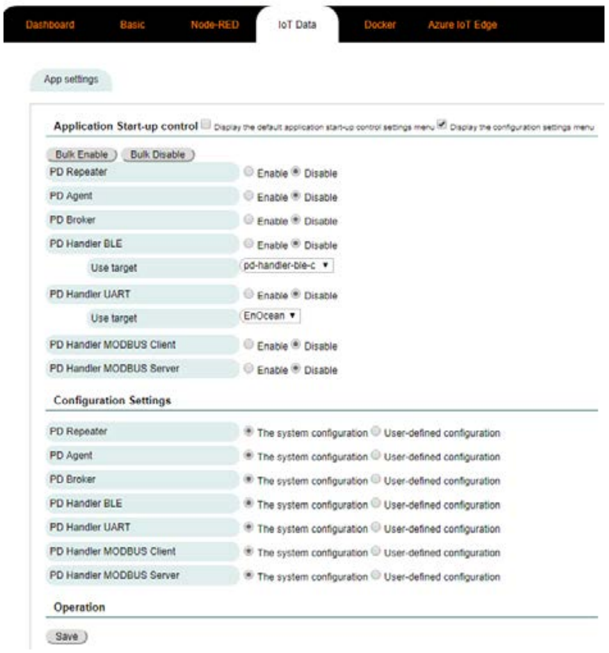
PD Broker : Application that distributes data or control messages to multiple applications.

PD Handler BLE : Application that receives data from BT device sensors or beacons. In the Use Target pull-down menu, it is possible to choose a C-language or a Java script version. The former offers a higher performance but it only supports beacon and connection-less BLE sensors.

PD Handler UART : Application that receives data from URAT connection devices. The Use Target pull-down menu supports the EnOcean.

PD Handler MODBUS Client : Application that reads/writes the register of PLC equipment using the MODBUS protocol.

PD Handler MODBUS Server : Application that receives connections from PLC equipment using the MODBUS protocol.



Application Start-up control [ ] Display the default application start-up control settings menu [✓] Display the configuration settings menu

PD Repeater: Enable [✓ Disable]
PD Agent: Enable [✓ Disable]
PD Broker: Enable [✓ Disable]
PD Handler BLE: Enable [✓ Disable]
Use target: pd-handler-ble-c
PD Handler UART: Enable [✓ Disable]
Use target: EnOcean
PD Handler MODBUS Client: Enable [✓ Disable]
PD Handler MODBUS Server: Enable [✓ Disable]

Configuration Settings

PD Repeater: [✓ The system configuration] User-defined configuration
PD Agent: [✓ The system configuration] User-defined configuration
PD Broker: [✓ The system configuration] User-defined configuration
PD Handler BLE: [✓ The system configuration] User-defined configuration
PD Handler UART: [✓ The system configuration] User-defined configuration
PD Handler MODBUS Client: [✓ The system configuration] User-defined configuration
PD Handler MODBUS Server: [✓ The system configuration] User-defined configuration

Operation

*Example of screen when **Display the config** check box is checked

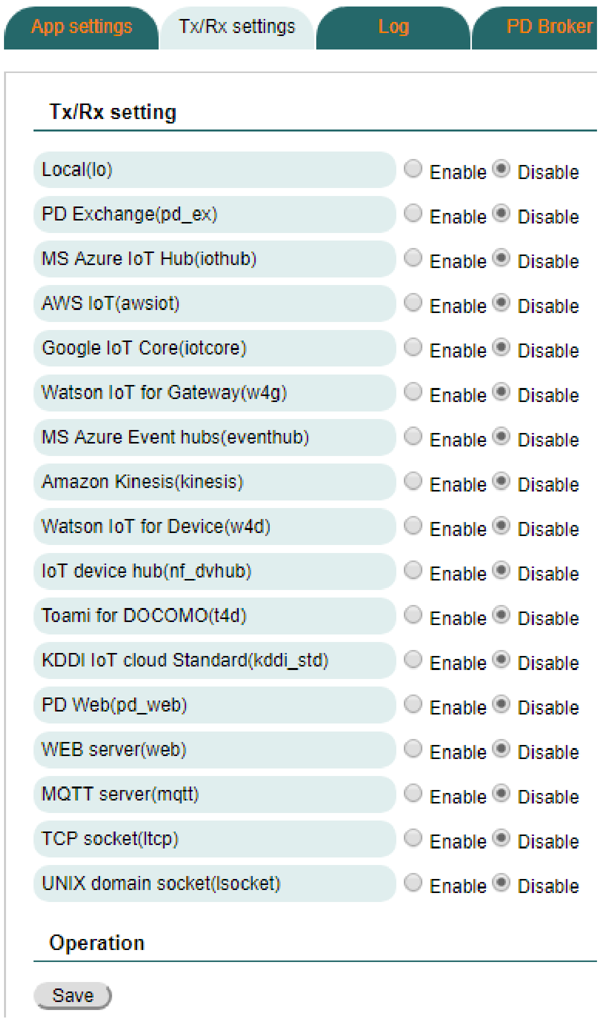
2-4. Send/receive setup (PD Repeater)

Use the **[IoT Data]-[Tx/Rx setting]** tab to set up data destination or control message sender.

Settings for transmission/reception include those independent from individual devices set use the **[IoT Data]-[Tx/Rx setting]** tab and those for individual devices set by the menu for their settings.

2-4-1. Tx/Rx settings

Use the **[IoT Data]-[Tx/Rx setting]** tab to select transmission (Tx)/reception (Rx) destinations and make settings independent of individual devices. Apart from local one (lo), it is possible to choose up to four Tx/Rx destinations.



Tx/Rx setting

Local(lo)	Enable [✓ Disable]
PD Exchange(pd_ex)	Enable [✓ Disable]
MS Azure IoT Hub(iothub)	Enable [✓ Disable]
AWS IoT(awsiot)	Enable [✓ Disable]
Google IoT Core(iotcore)	Enable [✓ Disable]
Watson IoT for Gateway(w4g)	Enable [✓ Disable]
MS Azure Event hubs(eventhub)	Enable [✓ Disable]
Amazon Kinesis(kinesis)	Enable [✓ Disable]
Watson IoT for Device(w4d)	Enable [✓ Disable]
IoT device hub(nf_dvhub)	Enable [✓ Disable]
Toami for DOCOMO(t4d)	Enable [✓ Disable]
KDDI IoT cloud Standard(kddi_std)	Enable [✓ Disable]
PD Web(pd_web)	Enable [✓ Disable]
WEB server(web)	Enable [✓ Disable]
MQTT server(mqtt)	Enable [✓ Disable]
TCP socket(ltcp)	Enable [✓ Disable]
UNIX domain socket(lsocket)	Enable [✓ Disable]

Operation

Local(lo) : Outputs the JSON table of BLE device and temperature/humidity graph to **[IoT Data]-[Data Display]** tab.

PD Exchange(pd_ex) : Selects PD Exchange as Tx/Rx destinations.

MS Azure IoT Hub(iothub) : Selects Microsoft Azure IoT Hub as Tx/Rx destinations. For transmission to Azure IoT Edge, choose this item.

AWS IoT(awsiot) : Selects Amazon AWS IoT as Tx/Rx destinations.

Google IoT Core(iotcore) : Selects Google IoT Core as Tx/Rx

Watson IoT for Gateway(w4g) : Selects IBM Watson IoT for Gateway as Tx/Rx destinations.

MS Azure Event hubs(eventhub) : Selects Microsoft Azure Event hubs as Tx destination.

Amazon Kinesis(kinesis) : Selects Amazon Kinesis as Tx destination.

Watson IoT for Device(w4d) : Selects IBM Watson IoT for Device as Tx/Rx destinations.

IoT device hub(nf_dvhub) : Selects Nifty Cloud IoT Device Hub as Tx/Rx destinations.

Toami for DOCOMO(t4d) : Selects NTT docomo Toami for DOCOMO as Tx destination.

KDDI IoT cloud Standard(kddi_std) : Selects KDDI IoT Cloud Standard as Tx destination.

PD Web(pd_web) : Selects web server of Plat'Home's original specifications as Tx/Rx destinations.

WEB server(web) : Selects general-purpose web server as TX destination.

MQTT server (mqtt) : Selects general-purpose NQTT server as Tx/Rx destinations.

TCP(ltcp): Selects general-purpose TCP server as Tx/Rx destinations.

UNIX domain socket(lsocket) : Selects UNIX domain socket as Tx destination. This option will only be displayed when Node-RED package has been installed.

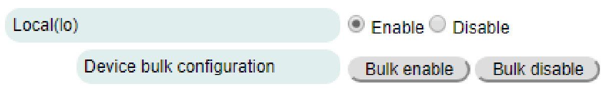
2-4-2. Settings for individual Tx/Rx destinations

This Chapter describes settings, using the **Tx/Rx setting** menu in the **[IoT Data]-[Tx/Rx setting]** tab and an individual device settings menu for each Tx/Rx destination.

2-4-2-1. Local(lo)

Outputs a JSON table and temperature/humidity graph of BLE device to the **[IoT Data]-[Data Display]** tab. This option (lo) only applies to a BLE device.

■Settings at Tx/Rx tab menu

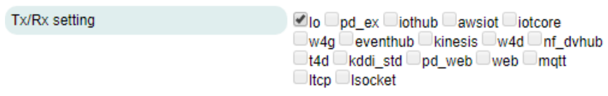

Local(lo) [✓ Enable] Disable

Device bulk configuration

Can only choose “Enable” or "Disable."

Device bulk configuration : To enable/disable all TX destination settings where TX destination is set to “Enable” in the BLE device information transmission settings.

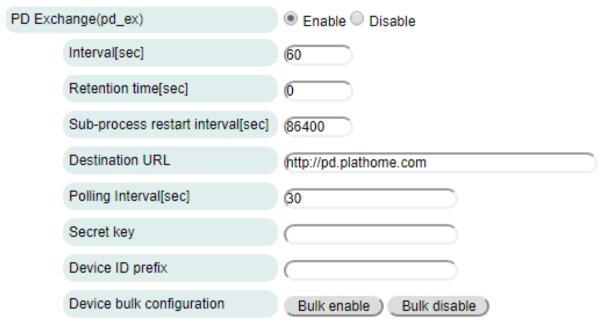
■Settings at BLE device settings menu


Tx/Rx setting [✓ lo] pd_ex iothub awsiot iotcore w4g eventhub kinesis w4d nf_dvhub t4d kddi_std pd_web web mqtt ltcp lsocket

This option enables a check/uncheck of individual check boxes only.

2-4-2-2. PD Exchange(pd_ex)

■Settings at the **Tx/Rx setting** menu



PD Exchange(pd_ex)	● Enable ○ Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
Destination URL	http://pd.plathome.com
Polling Interval[sec]	30
Secret key	
Device ID prefix	
Device bulk configuration	Bulk enable / Bulk disable

Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

Destination URL: Sets URL of PD Exchange to be connected.

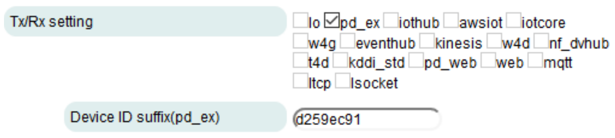
Polling Interval[sec]: Sets interval to read control messages from PD Exchange.

Secret key: Sets secret key of PD Exchange account.

Device ID prefix: Sets device ID prefix for PD Exchange account.

Device bulk configuration: To enable/disable all TX destination settings whose TX destination is set to “Enable” in individual device setup menu.

■Settings for individual device setting menu



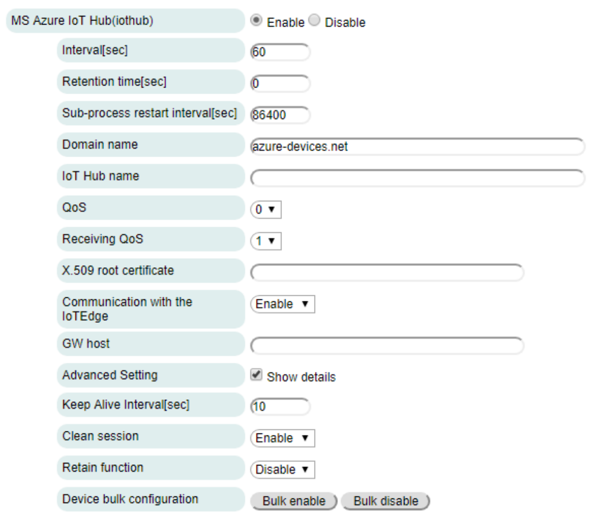
Tx/Rx setting	☐lo ☑pd_ex ☐iothub ☐awsiot ☐iotcore ☐w4g ☐eventhub ☐kinesis ☐w4d ☐nf_dvhub ☐t4d ☐kddi_std ☐pd_web ☐web ☐mqtt ☐ltcp ☐lsocket
Device ID suffix(pd_ex)	d259ec91

Device ID suffix(pd_ex) : Sets device ID suffix for PD Exchange.

2-4-2-3. MS Azure IoT Hub(iothub)

For Azure IoT Edge, please read this section.

■Settings at the **Tx/Rx setting** menu



MS Azure IoT Hub(iothub)	● Enable ○ Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
Domain name	azure-devices.net
IoT Hub name	
QoS	0 ▾
Receiving QoS	1 ▾
X.509 root certificate	
Communication with the IoTEdge	Enable ▾
GW host	
Advanced Setting	✓ Show details
Keep Alive Interval[sec]	10
Clean session	Enable ▾
Retain function	Disable ▾
Device bulk configuration	Bulk enable / Bulk disable

Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

Domain name: Sets domain name of IoT Hub to be connected.

IoT Hub name: Sets name of IoT Hub to be connected.

QoS: Sets QoS of the MQTT protocol for data transmission to IoT Hub.

Receiving QoS: Sets QoS of the MQTT protocol for control message reception from IoT Hub.

X.509 root certificate: Sets path name of root certificate file when X.509 certification is used.

Communication with IoT Edge: Selects use or non-use of IoT Edge.

GW host: Sets gateway host name of IoT Edge (only when IoT Edge is used).

To send data to IoT Edge running on a different host, specify an IP address or FQDN.

Advanced setting: Sets values of the MQTT protocol in detail.

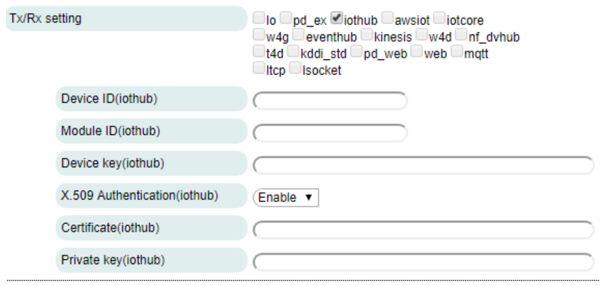
Keep Alive Interval[sec]: Sets keep alive interval of MQTT protocol.

Clean session: Selects if using the clean session of the MQTT protocol.

Retain function: Selects if the retention function of the MQTT protocol is used or not.

Device bulk configuration: To enable/disable all TX destination settings whose TX destination is set to “Enable” in individual device setup menu.

■Settings for individual device setting menu



Tx/Rx setting	☐ lo ☐ pd_ex ☑ iothub ☐ awsiot ☐ iotcore ☐ w4g ☐ eventhub ☐ kinesis ☐ w4d ☐ nf_dvhub ☐ t4d ☐ kddi_std ☐ pd_web ☐ web ☐ mqtt ☐ ltcp ☐ lsocket
Device ID(iothub)	
Module ID(iothub)	
Device key(iothub)	
X.509 Authentication(iothub)	Enable ▼
Certificate(iothub)	
Private key(iothub)	

Device ID(iothub): Sets device ID for IoT Hub.

Module ID(iothub): Sets module ID when IoT Edge is used.

Device key(iothub): Sets device key for IoT Hub.

X.509 Authentication to use(iothub): Choose "Enable" to use X.509 authentication.

Certificate(iothub): Sets path name for the certificate file for X.509 authentication.

Private key(iothub): Sets path name for the private key for X.509 authentication.

*For the gateway host name and device key when IoT Edge is used, use a string given to the environmental variable of the Docker container, as indicated by the Module ID.

*If IoT Edge is not used, upload the X.509 root certificate, certificate and private key using the **[System]-[File Management]** tab.

*To use the X.509 root certificate, certificate and private key created by iotedgectl when using IoT Edge, set the following path names:

X.509 root certificate:

/var/lib/azure-iot-edge/certs/edge-device-ca/cert/edge-device-ca-root.cert.pem

Certificate:

/var/lib/azure-iot-edge/certs/edge-device-ca/cert/edge-device-ca.cert.pem

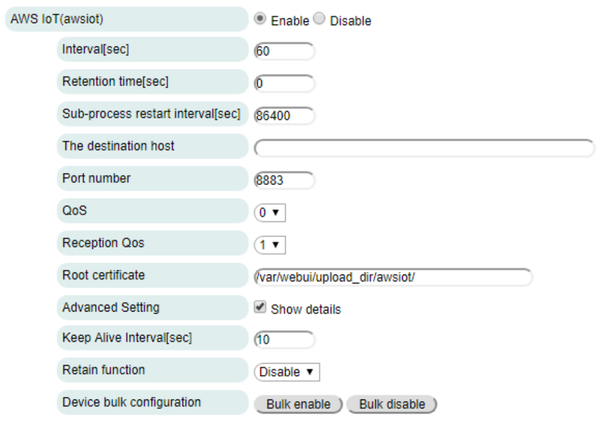
Private key:

```
/var/lib/azure-iot-edge/certs/edge-device-ca/private/edge-device-ca.key.pem
```

*As of the time of preparing this document, Azure IoT Edge is in public preview. Specifications are subject to change at the time of general availability.

2-4-2-4. AWS IoT Hub(awsiot)

■Settings at the **Tx/Rx setting** menu



AWS IoT(awsiot)	● Enable ○ Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
The destination host	
Port number	8883
QoS	0 ▾
Reception Qos	1 ▾
Root certificate	/var/webui/upload_dir/awsiot/
Advanced Setting	✓ Show details
Keep Alive Interval[sec]	10
Retain function	Disable ▾
Device bulk configuration	Bulk enable Bulk disable

Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

Destination host: Sets host name of AWS IoT to be connected.

Port number: Sets port number of AWS IoT to be connected.

QoS: Sets QoS of MQTT protocol when data is sent to AWS IoT.

Reception QoS: Sets QoS of MQTT protocol when a control message is received from AWS IoT.

Root certificate: Sets path name of the root certification file when connected to AWS IoT.

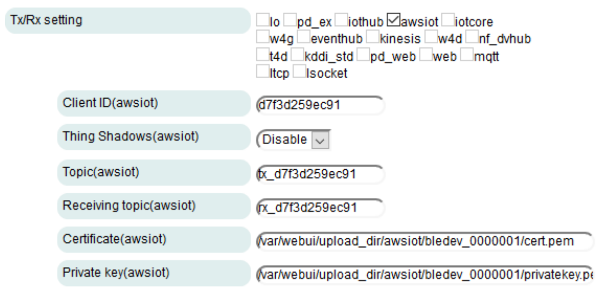
Advanced setting: Sets detailed values of MQTT protocol.

Keep Alive Interval[sec]: Sets keep alive interval of MQTT protocol.

Retain function: Selects if the retention function of the MQTT protocol is used or not.

Device bulk configuration: To enable/disable all TX destination settings whose TX destination is set to “Enable” in individual device setup menu.

■Settings for individual device setting menu



Tx/Rx setting	☐lo ☐pd_ex ☐iothub ☑awsiot ☐iotcore ☐w4g ☐eventhub ☐kinesis ☐w4d ☐nf_dvhub ☐t4d ☐kddi_std ☐pd_web ☐web ☐mqtt ☐ltcp ☐lsocket
Client ID(awsiot)	d7f3d259ec91
Thing Shadows(awsiot)	Disable
Topic(awsiot)	tx_d7f3d259ec91
Receiving topic(awsiot)	rx_d7f3d259ec91
Certificate(awsiot)	/var/webui/upload_dir/awsiot/bledev_0000001/cert.pem
Private key(awsiot)	/var/webui/upload_dir/awsiot/bledev_0000001/privatekey.pe

Client ID(awsiot): Sets client ID for AWS IoT.

Thing Shadows(awsiot): Selects if Thing Shadows function of AWS IoT is used or not.

Topic(awsiot): Sets topic of MQTT protocol used when data is sent to AWS IoT.

Receiving topic(awsiot): Sets topic of MQTT protocol used for waiting and receiving a control message from AWS IoT.

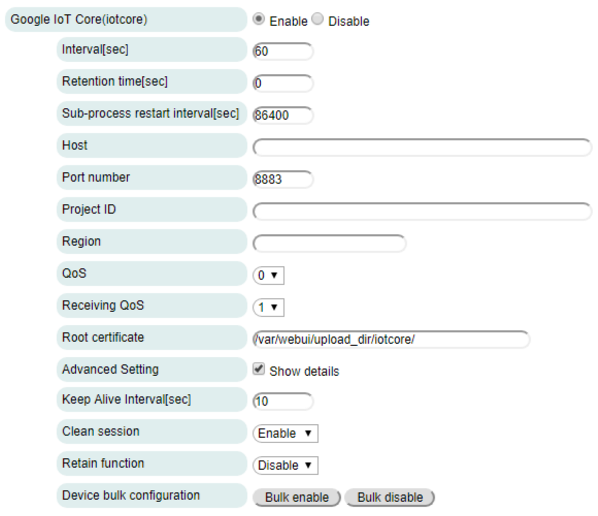
Certificate(awsiot): Sets path name for the certificate file to connect to AWS IoT.

Private key(awsiot): Sets path name for the private key file to connect to AWS IoT.

*Upload the root certificate, certificate and private key using the **[System]-[File Management]** tab.

2-4-2-5. Google IoT Core(iotcore)

■Settings at the **Tx/Rx setting** menu



Google IoT Core(iotcore)	● Enable ○ Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
Host	
Port number	8883
Project ID	
Region	
QoS	0
Receiving QoS	1
Root certificate	/var/webui/upload_dir/iotcore/
Advanced Setting	✓ Show details
Keep Alive Interval[sec]	10
Clean session	Enable
Retain function	Disable
Device bulk configuration	Bulk enable Bulk disable

Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

Host: Sets host name of Google IoT Core to be connected.

Port number: Sets port number of Google IoT Core to be connected.

Project ID: Sets project ID for Google IoT Core.

Region: Sets region code for Google IoT Core.

QoS: Sets QoS of MQTT protocol when data is sent to Google IoT Core.

Receiving QoS: Sets QoS of MQTT protocol when a control message is received from Google IoT Core.

Root certificate: Sets path name of the root certificate file to connect to Google IoT Core.

Advanced Setting: Sets detailed values of MQTT protocol.

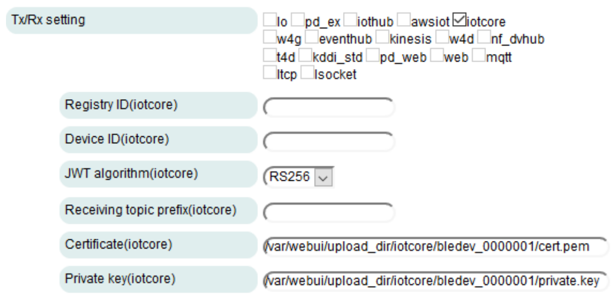
Keep Alive Interval[sec]: Sets keep alive interval of MQTT protocol.

Clean session: Selects if using the clean session of the MQTT protocol.

Retain function: Selects if the retention function of the MQTT protocol is used or not.

Device bulk configuration: To enable/disable all TX destination settings whose TX destination is set to “Enable” in individual device setup menu.

■Settings for individual device setting menu



Tx/Rx setting	☐lo ☐pd_ex ☐iothub ☐awsiot ☑iotcore ☐w4g ☐eventhub ☐kinesis ☐w4d ☐nf_dvhub ☐t4d ☐kddi_std ☐pd_web ☐web ☐mqtt ☐ltcp ☐lsocket
Registry ID(iotcore)	
Device ID(iotcore)	
JWT algorithm(iotcore)	RS256
Receiving topic prefix(iotcore)	
Certificate(iotcore)	/var/webui/upload_dir/iotcore/bledev_0000001/cert.pem
Private key(iotcore)	/var/webui/upload_dir/iotcore/bledev_0000001/private.key

Registry ID(iotcore): Sets client ID for Google IoT Core.

Device ID(iotcore): Sets device ID for Google IoT Core.

JWT algorithm(iotcore): Selects JWT algorithm to connect to Google IoT Core.

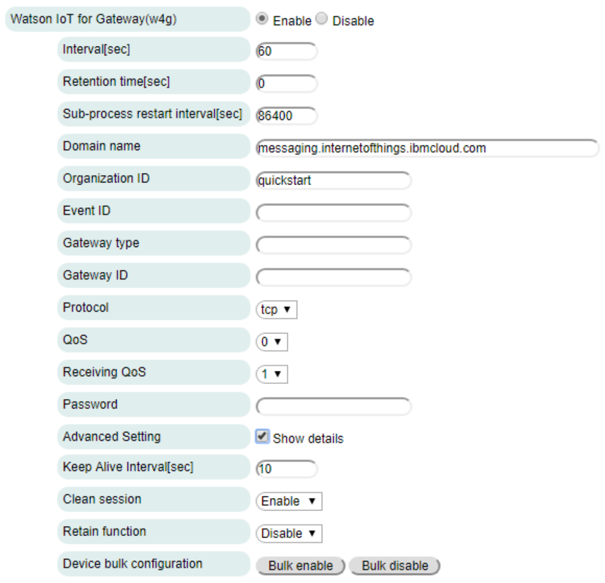
Certificate(iotcore): Sets path name of the certificate file to connect to Google IoT Core.

Private key(iotcore): Sets path name of the private key file to connect to Google IoT Core.

*Upload the root certificate, certificate and private key using the **[System]-[File Management]** tab

2-4-2-6. Watson IoT for Gateway(w4g)

■Settings at the **Tx/Rx setting** menu



Watson IoT for Gateway(w4g)	[✓ Enable] Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
Domain name	messaging.internetofthings.ibmcloud.com
Organization ID	quickstart
Event ID	
Gateway type	
Gateway ID	
Protocol	tcp
QoS	0
Receiving QoS	1
Password	
Advanced Setting	✓ Show details
Keep Alive Interval[sec]	10
Clean session	Enable
Retain function	Disable
Device bulk configuration	Bulk enable / Bulk disable

Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

Domain name: Sets domain name of Watson IoT for Gateway to be connected.

Organization ID: Sets organization ID for Watson IoT for Gateway.

Event ID: Sets event ID for Watson IoT for Gateway.

Gateway type: Sets gateway type of Watson IoT for Gateway.

Gateway ID: Sets gateway ID of Watson IoT for Gateway.

Protocol: Sets protocol to use for connection (TCP/SSL).

QoS: Sets QoS for MQTT protocol to send data to Watson IoT for Gateway.

Receiving QoS: Sets QoS of MQTT protocol when a control message is received from Watson IoT for Gateway.

Password: Sets password used for connection with Watson IoT for Gateway.

Trust Store: Sets path name of the root certificate file to connect to Watson IoT for

Gateway.

Key Store: Sets path name of the client certificate file to connect to Watson IoT for Gateway.

Private key: Sets path name of the private key file to connect to Watson IoT for Gateway.

Advanced Setting: Sets detailed values of MQTT protocol.

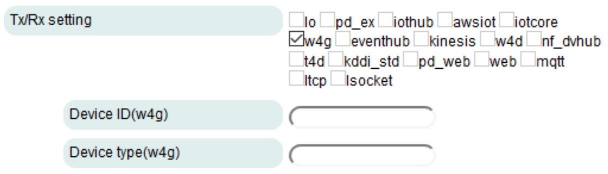
Keep Alive Interval[sec]: Sets keep alive interval of MQTT protocol.

Clean session: Selects if using the clean session of the MQTT protocol.

Retain function: Selects if the retention function of the MQTT protocol is used or not.

Device bulk configuration: To enable/disable all TX destination settings whose TX destination is set to "Enable" in individual device setup menu.

■Settings for individual device setting menu



Tx/Rx setting	☐lo ☐pd_ex ☐iothub ☐awsiot ☐iotcore ☑w4g ☐eventhub ☐kinesis ☐w4d ☐nf_dvhub ☐t4d ☐kddi_std ☐pd_web ☐web ☐mqtt ☐ltcp ☐lsocket
Device ID(w4g)	
Device type(w4g)	

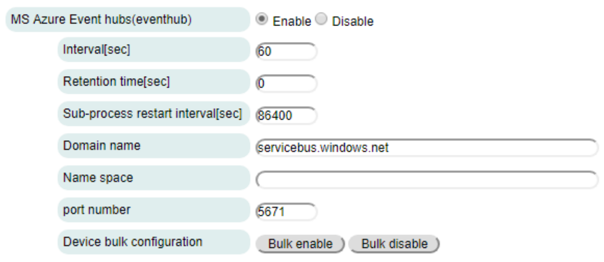
Device ID(w4g): Sets device ID forWatson IoT for Gateway.

Device type(w4g): Sets device type of Watson IoT for Gateway.

*Upload the Trust Store, Key Store and private key using the **[System]-[File Management]** tab.

2-4-2-7. MS Azure Event hubs(eventhub)

■Settings at the **Tx/Rx setting** menu



MS Azure Event hubs(eventhub)	● Enable ○ Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
Domain name	servicebus.windows.net
Name space	
port number	5671
Device bulk configuration	Bulk enable / Bulk disable

Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

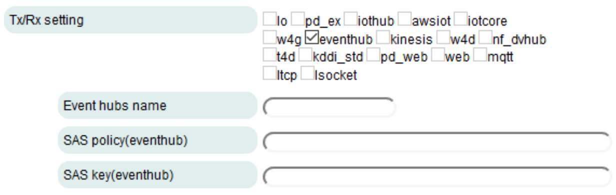
Domain name: Sets domain name of Event hubs to be connected.

Name space: Sets name space of Event hubs.

Port number: Sets port number of Event hubs for destination.

Device bulk configuration: To enable/disable all TX destination settings whose TX destination is set to "Enable" in individual device setup menu.

■Settings for individual device setting menu



Tx/Rx setting	☐lo ☐pd_ex ☐iothub ☐awsiot ☐iotcore ☐w4g ☑eventhub ☐kinesis ☐w4d ☐nf_dvhub ☐t4d ☐kddi_std ☐pd_web ☐web ☐mqtt ☐ltcp ☐lsocket
Event hubs name	
SAS policy(eventhub)	
SAS key(eventhub)	

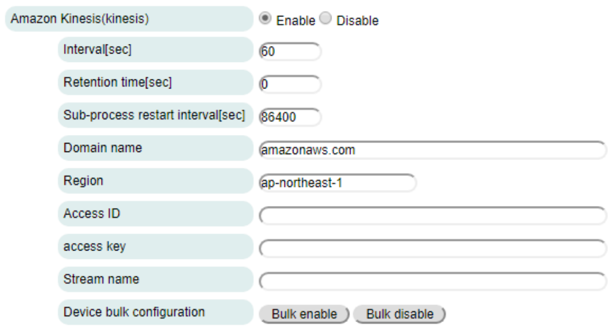
Event hubs name(eventhub): Sets hub name of Event hubs.

SAS policy(eventhub): Sets SAS policy of Event hubs.

SAS key(eventhub): Sets SAS key of Event hubs.

2-4-2-8. Amazon Kinesis(kinesis)

■Settings at the **Tx/Rx setting** menu



Amazon Kinesis(kinesis)	[✓ Enable] Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
Domain name	amazonaws.com
Region	ap-northeast-1
Access ID	
access key	
Stream name	
Device bulk configuration	Bulk enable Bulk disable

Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

Domain name: Sets domain name for Amazon Kinesis of the destination.

Region: Sets region code for Amazon Kinesis of the destination.

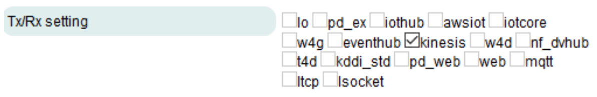
Access ID: Sets access ID for Amazon Kinesis.

Access key: Sets access key for Amazon Kinesis.

Stream name: Sets stream name for Amazon Kinesis.

Device bulk configuration: To enable/disable all TX destination settings whose TX destination is set to "Enable" in individual device setup menu.

■Settings for individual device setting menu

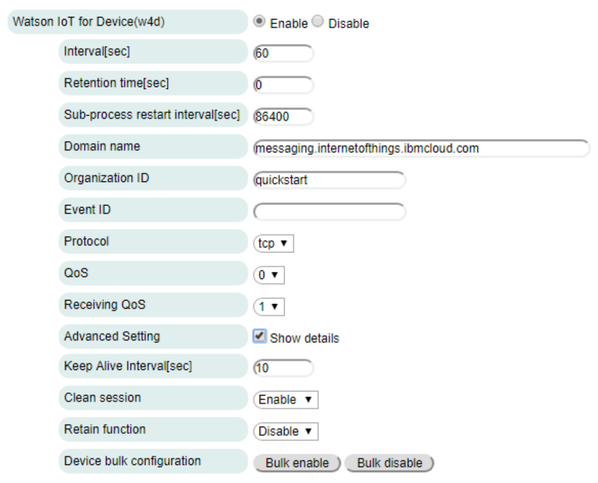


Tx/Rx setting	lo pd_ex iothub awsiot iotcore w4g eventhub [✓ kinesis] w4d nf_dvhub t4d kddi_std pd_web web mqtt ltcp lsocket

No setting item is available other than checking check boxes in Tx/Rx setting.

2-4-2-9. Watson IoT for Device(w4d)

■Settings at the **Tx/Rx setting** menu



Watson IoT for Device(w4d)	[✓ Enable] Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
Domain name	messaging.internetofthings.ibmcloud.com
Organization ID	quickstart
Event ID	
Protocol	tcp
QoS	0
Receiving QoS	1
Advanced Setting	✓ Show details
Keep Alive Interval[sec]	10
Clean session	Enable
Retain function	Disable
Device bulk configuration	Bulk enable / Bulk disable

Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

Domain name: Sets domain name of Watson IoT for Device to be connected.

Organization ID: Sets organization ID for Watson IoT for Device. To use Quickstart, enter "quickstart" here.

Event ID: Sets event ID for Watson IoT for Device.

Protocol: Sets protocol to use for connection (TCP/SSL).

QoS: Sets QoS for MQTT protocol to send data to Watson IoT for Device. To use Quickstart, enter "0" here.

Receiving QoS: Sets QoS of MQTT protocol when a control message is received from Watson IoT for Device.

Trust Store: Sets path name of the root certificate file to connect to Watson IoT for Device.

Advanced Setting: Sets detailed values of MQTT protocol.

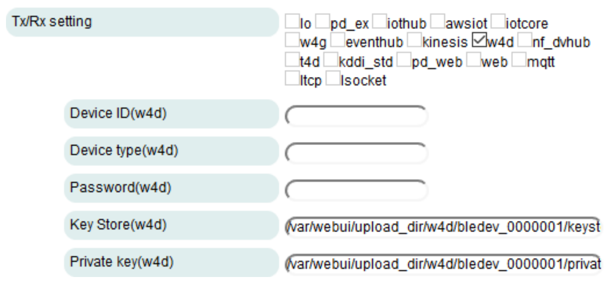
Keep Alive Interval[sec]: Sets keep alive interval of MQTT protocol.

Clean session: Selects if using the clean session of the MQTT protocol.

Retain function: Selects if the retention function of the MQTT protocol is used or not.

Device bulk configuration: To enable/disable all TX destination settings whose TX destination is set to “Enable” in individual device setup menu.

■Settings for individual device setting menu



Tx/Rx setting	☐lo ☐pd_ex ☐iothub ☐awsiot ☐iotcore ☐w4g ☐eventhub ☐kinesis ☑w4d ☐nf_dvhub ☐t4d ☐kddi_std ☐pd_web ☐web ☐mqtt ☐ltcp ☐lsocket
Device ID(w4d)	
Device type(w4d)	
Password(w4d)	
Key Store(w4d)	/var/webui/upload_dir/w4d/bledev_0000001/keyst
Private key(w4d)	/var/webui/upload_dir/w4d/bledev_0000001/privat

Device ID(w4d): Sets device ID for Watson IoT for Device.

Device type(w4d): Sets device type of Watson IoT for Device.

Password: Sets password used for connection with Watson IoT for Device.

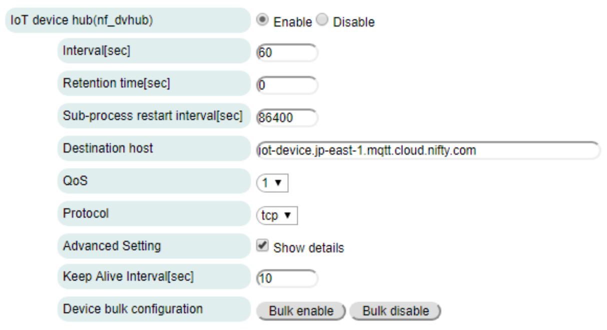
Key Store: Sets path name of the client certificate file to connect to Watson IoT for Device.

Private key: Sets path name of the private key file to connect to Watson IoT for Device.

*Upload the Trust Store, Key Store and private key using the **[System]-[File Management]** tab.

2-4-2-10. IoT device hub(nf_dvhub)

■Settings at the **Tx/Rx setting** menu



IoT device hub(nf_dvhub)	● Enable ○ Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
Destination host	iot-device.jp-east-1.mqtt.cloud.nifty.com
QoS	1
Protocol	tcp
Advanced Setting	✓ Show details
Keep Alive Interval[sec]	10
Device bulk configuration	Bulk enable / Bulk disable

Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

Destination host: Sets host name of the IoT device hub to be connected.

Protocol: Sets protocol to use for connection (TCP/SSL).

QoS: Sets QoS for MQTT protocol to send data to IoT device hub.

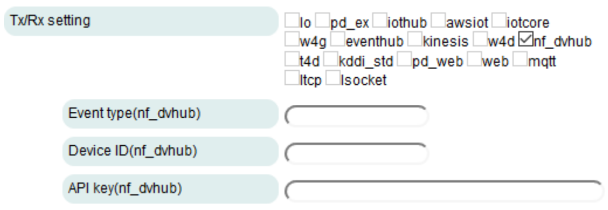
Root certificate: Sets path name of the root certificate file to connect to IoT device hub.

Advanced Setting: Sets detailed values of MQTT protocol.

Keep Alive Interval[sec]: Sets keep alive interval of MQTT protocol.

Device bulk configuration: To enable/disable all TX destination settings whose TX destination is set to “Enable” in individual device setup menu.

■Settings for individual device setting menu



Tx/Rx setting	☐lo ☐pd_ex ☐iothub ☐awsiot ☐iotcore ☐w4g ☐eventhub ☐kinesis ☐w4d ☑nf_dvhub ☐t4d ☐kddi_std ☐pd_web ☐web ☐mqtt ☐ltcp ☐lsocket
Event type(nf_dvhub)	
Device ID(nf_dvhub)	
API key(nf_dvhub)	

Event type(nf_dvhub): Sets event type of the IoT device hub.

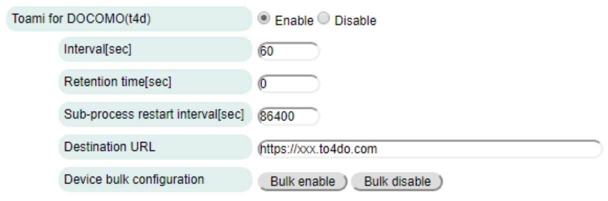
Device ID(nf_dvhub): Sets device ID for the IoT device hub.

API key(nf_dvhub): Sets API key of the IoT device hub.

*Upload the root certificate using the **[System]-[File Management]** tab.

2-4-2-11. Toami for DOCOMO(t4d)

■Settings at the **Tx/Rx setting** menu



Toami for DOCOMO(t4d)	[● Enable] ○ Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
Destination URL	https://xxx.to4do.com
Device bulk configuration	Bulk enable / Bulk disable

Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

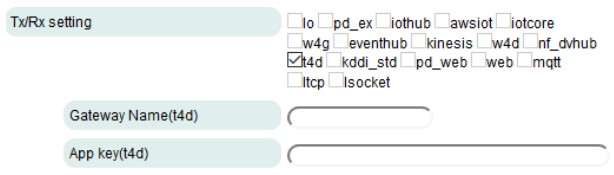
Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

Destination URL: Sets URL of Toami for DOCOMO.

Device bulk configuration: To enable/disable all TX destination settings whose TX destination is set to “Enable” in individual device setup menu.

■Settings for individual device setting menu

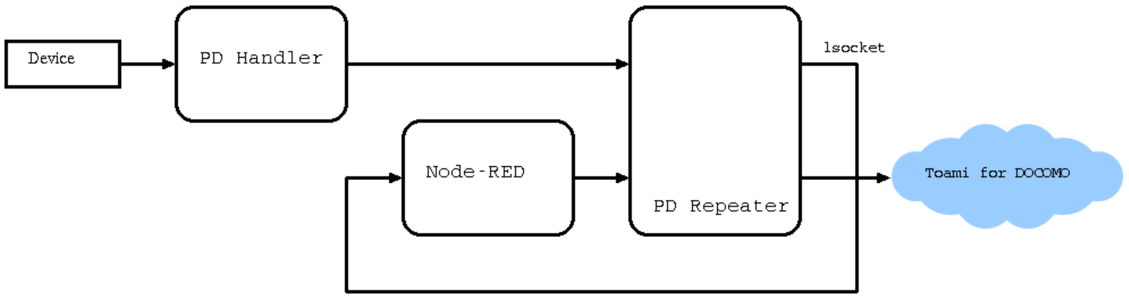


Tx/Rx setting	☐lo ☐pd_ex ☐iothub ☐awsiot ☐iotcore ☐w4g ☐eventhub ☐kinesis ☐w4d ☐nf_dvhub ☑t4d ☐kddi_std ☐pd_web ☐web ☐mqtt ☐ltcp ☐lsocket
Gateway Name(t4d)	
App key(t4d)	

Gateway Name(t4d): Sets gateway name of Toami for DOCOMO.

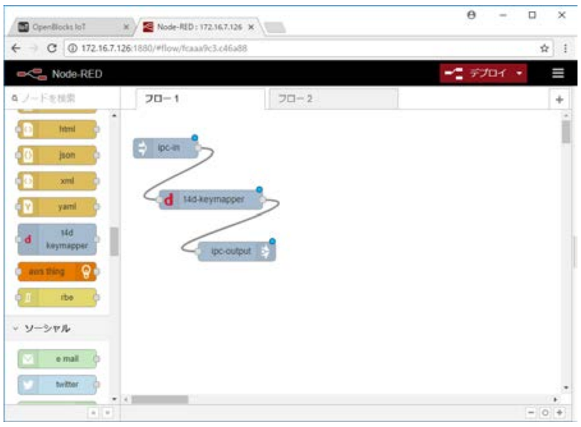
App Key(t4d): Sets gateway name of Toami for DOCOMO.

*With Toami for DOCOMO, as key information of data it can handle (JSON string) is fixed, it is necessary to transfer output of the PD Handler via the PD Repeater to Node-RED, where key information is converted and send this converted information to Toami for DOCOMO via the PD Repeater as shown in the figure below.



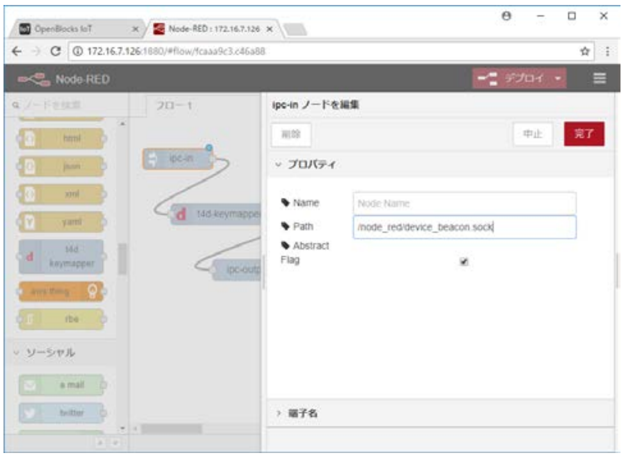
Taking an example of a Beacon device (PD Handler BLE) to explain the set up.

1. Set destination of PD Handler BLE (device #: device_beacon) as domain socket (lsocket).
   For settings of domain socket(lsocket), refer to Chapter 2-4-2-17.
   The socket path prefix is the default value "@/node-red/".
2. Register Node-RED as a user-defined device.
   For registration of user-defined devices, refer to Chapter 2-4-8.
   Suppose that "userdev_0000001" is assigned as the user-defined device.
3. In Node-RED, deploy ipc-in, ipc-out and t4d-keymapper nodes and connect them.
   As to how to use Node-RED, refer to the OpenBlocks IoT Family Node-RED Starter Guide.



Deploy ipc-in, ipc-out and t4d-keymapper nodes and connect them.

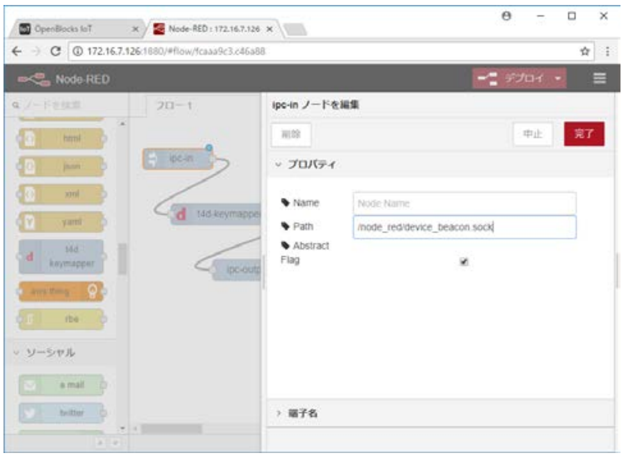
4. Set property of the ipc-in node.



Name: Node Name
Path: /node_red/device_beacon.sock
Abstract Flag: ✓

Set Path in the property to "/node-red/device_beacon.sock".
Check the Abstract Flag check box.

5. Set property of the ipc-out node.



Name: Node Name
Path: /node_red/device_beacon.sock
Abstract Flag: ✓

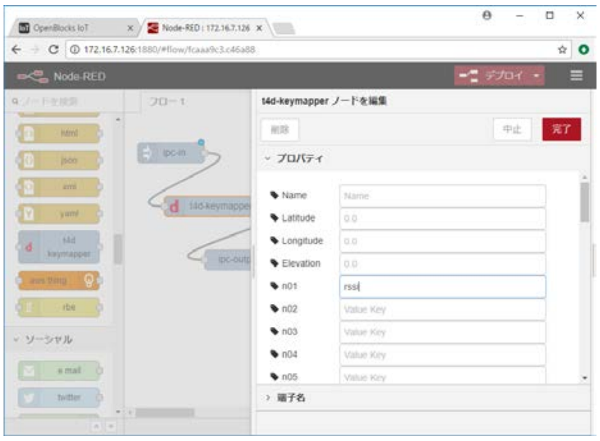
Set Path in the property to "/pd_repeater/userdev_0000001.sock".

Check the Abstract Flag check box.

6. The table below shows JSON keys of the beacon output from the PD Handler BLE and keys following conversion set with this example.

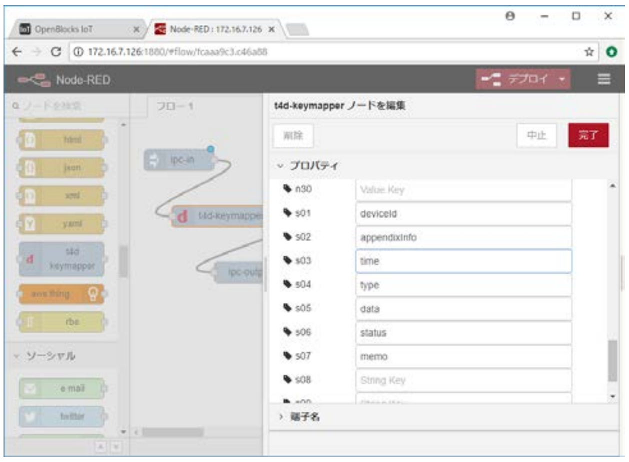
JSON keys	Data format	Description	Keys following conversion
deviceId	String	Device ID	s01
appendixInfo	String	Accompanying information	s02
time	String	Acquisition time of data	s03
rssi	Integer	RX signal strength	n01
type	String	RX signal strength	s04
data	String	Payload data (hex notation)	s05
status	String	Beacon status	s06
memo	String	String set from WEB UI	s07

For the property of t4d-mapper, set up the settings as per the above table.



Name: Name
Latitude: 0.0
Longitude: 0.0
Elevation: 0.0
n01: rssi
n02: Value Key
n03: Value Key
n04: Value Key
n05: Value Key

1. Set n01 to "rssi".



n30	Value Key
s01	deviceId
s02	appendixInfo
s03	time
s04	type
s05	data
s06	status
s07	memo
s08	String Key

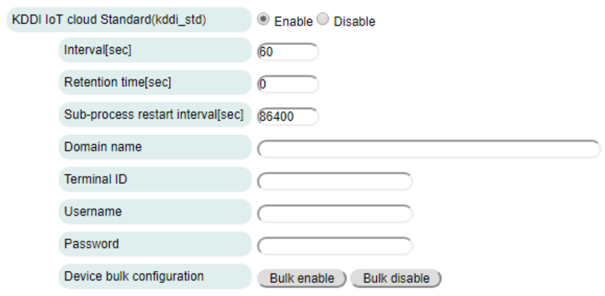
2. Set s01 to "deviceId".
3. Set s02 to "appendixInfo".
4. Set s03 to "time".
5. Set s04 to "type".
6. Set s05 to "data".
7. Set s05 to "data".
8. Set s07 to "memo".

*The data acquisition time set to s03 is handled as a simple string in Toami for DOCOMO. The time information of Toami for DOCOMO (timestamp key value) will be automatically generated and added in the t4d-keymapper node.

*The t4d-keymapper node cannot support devices that output values as arrays or JSON objects such as the PD Handler Modbus Client/Server. For such devices, create a unique function node depending on conversion conditions.

2-4-2-12. KDDI IoT Cloud Standard(kddi_std)

■Settings at the **Tx/Rx setting** menu



KDDI IoT cloud Standard(kddi_std)	[✓ Enable] Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
Domain name	
Terminal ID	
Username	
Password	
Device bulk configuration	Bulk enable / Bulk disable

Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

Domain name: Sets domain name of the KDDI IoT Cloud Standard to be connected.

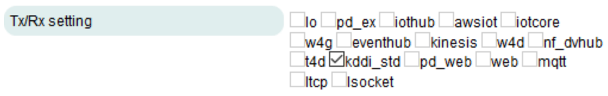
Terminal ID: Sets terminal ID of KDDI IoT Cloud Standard.

Username: Sets username for basic authentication of KDDI IoT Cloud Standard.

Password: Sets password for basic authentication of KDDI IoT Cloud Standard.

Device bulk configuration: To enable/disable all TX destination settings whose TX destination is set to “Enable” in individual device setup menu.

■Settings for individual device setting menu

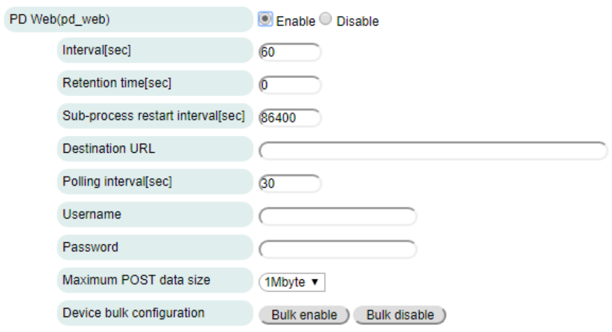


Tx/Rx setting	lo, pd_ex, iothub, awsiot, iotcore, w4g, eventhub, kinesis, w4d, nf_dvhub, t4d, [✓ kddi_std], pd_web, web, mqtt, ltcp, lsocket

No setting item is available other than checking check boxes in Tx/Rx setting.

2-4-2-13. Web server of Plat'Home's original specifications

■Settings at the **Tx/Rx setting** menu



PD Web(pd_web)	[✓ Enable] Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
Destination URL	
Polling interval[sec]	30
Username	
Password	
Maximum POST data size	1Mbyte ▾
Device bulk configuration	Bulk enable / Bulk disable

Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

Destination URL: Sets URL of web server for Plat'Home's original specifications to be connected.

Polling interval[sec]: Sets interval to read control messages from web server.

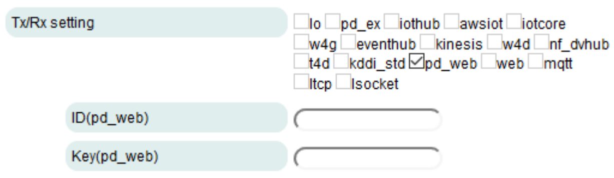
Username: Sets username for basic authentication of web server.

Password: Sets password for basic authentication of web server.

Maximum POST data size: Selects maximum data size to send a single POST method. Select between 1 to 4 Mbytes.

Device bulk configuration: To enable/disable all TX destination settings whose TX destination is set to "Enable" in individual device setup menu.

■Settings for individual device setting menu



Tx/Rx setting	lo, pd_ex, iothub, awsiot, iotcore, w4g, eventhub, kinesis, w4d, nf_dvhub, t4d, kddi_std, [✓ pd_web], web, mqtt, ltcp, lsocket
ID(pd_web)	
Key(pd_web)	

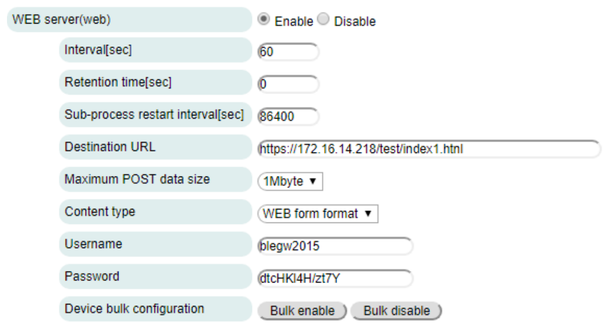
ID(pd_web): Sets ID for token authentication of web server.

Key(pd_web): Sets key for token authentication of web server.

With the data transmission method of the web server for Plat'Home original specifications, Content-Type is posted as "application/json". For further details, refer to Chapter 4-11.

2-4-2-14. WEB server(web)

■Settings at the **Tx/Rx setting** menu



WEB server(web)	[✓ Enable] Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
Destination URL	https://172.16.14.218/test/index1.htnl
Maximum POST data size	1Mbyte
Content type	WEB form format
Username	blegw2015
Password	dtcHKl4H/zt7Y
Device bulk configuration	Bulk enable / Bulk disable

Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

Destination URL: Sets URL of the general-purpose web server to be connected.

Maximum POST data size: Selects maximum data size to send a single POST method. Select between 1 to 4 Mbytes.

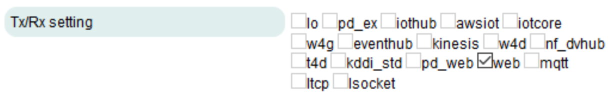
Content type: Selects content type of web server ("web from"/"plain text"/"JSON").

Username: Sets username for basic authentication of web server.

Password: Sets password for basic authentication of web server.

Device bulk configuration: To enable/disable all TX destination settings where TX destination is set to "Enable" in the BLE device information transmission settings.

■Settings for individual device setting menu



Tx/Rx setting	lo, pd_ex, iothub, awsiot, iotcore, w4g, eventhub, kinesis, w4d, nf_dvhub, t4d, kddi_std, pd_web, [✓ web], mqtt, ltcp, lsocket

No setting item is available other than checking check boxes in Tx/Rx setting.

Data is sent to web server with the POST method. If the Tx/Rx format setting is a web form format, **Content-Type** will be "application/x-www-form-urlencoded". Payload (TX data body) will be transmitted as an x-www-form-urlencoded variable containing multiple data called Records.

If the Tx/Rx format is a plain text format, **Content-Type** will be "text/plain". URL safe encoding of payload (TX data body) will not be preformed, and multiple data (presumed to be JSON) will be sent in bulk.

If the Tx/Rx format is a JSON format, **Content-Type** will be "application/json".
URL safe encoding of payload (TX data body) will not be preformed, and multiple data (presumed to be JSON) will be sent in bulk.

■Transmission format setting: Web form format

●Data format

Records＝[{DATA1},{DATA2},{DATA3},…{DATAn}]

●Transmission sample

```
POST / HTTP/1.0
Content-Length: 422
Content-Type: application/x-www-form-urlencoded

Records=[{"deviceId":"b0b448b81105","memo":"cc2650-1","objectTemp":20,"ambientTemp":24.84375,"humidity":47.912
59765625,"temperature":25.32928466796875,"pressure":1016.37,"time":"2016-11-24T16:50:23.431+0900"},{"deviceId":
"b0b448b81105","memo":"cc2650-1","objectTemp":19.75,"ambientTemp":24.875,"humidity":47.91259765625,"temperatu
re":25.3594970703125,"pressure":1016.31,"time":"2016-11-24T16:50:26.459+0900","lux":427.68}]
```

■Transmission format setting: Plain text format

●Data format

[{DATA1},{DATA2},{DATA3},…{DATAn}]

●Transmission sample

```
POST / HTTP/1.0
Content-Type: text/plain
Content-Length: 209

[{"deviceId":"6d4adcb7133f","appendixInfo":"G8H00010","rssi":-90,"time":"2017-12-04T14:07:18.157+09:00"},{"deviceId"
:"e135535f61e4","appendixInfo":"G8H00010","rssi":-74,"time":"2017-12-04T14:07:18.762+09:00"}]
```

■Transmission format setting: JSON format

●Data format

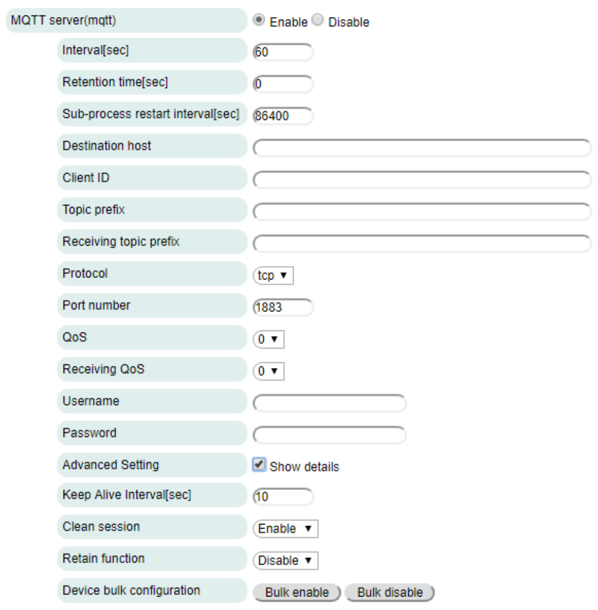
[{DATA1},{DATA2},{DATA3},…{DATAn}]

●Transmission sample

```
POST / HTTP/1.0
Content-Type: application/json
Content-Length: 209

[{"deviceId":"6d4adcb7133f","appendixInfo":"G8H00010","rssi":-90,"time":"2017-12-04T14:07:18.157+09:00"},{"deviceId"
:"e135535f61e4","appendixInfo":"G8H00010","rssi":-74,"time":"2017-12-04T14:07:18.762+09:00"}]
```

2-4-2-15. MQTT server(mqtt)

■Settings at the **Tx/Rx setting** menu



MQTT server(mqtt)	[✓ Enable] Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
Destination host	
Client ID	
Topic prefix	
Receiving topic prefix	
Protocol	tcp
Port number	1883
QoS	0
Receiving QoS	0
Username	
Password	
Advanced Setting	✓ Show details
Keep Alive Interval[sec]	10
Clean session	Enable
Retain function	Disable
Device bulk configuration	Bulk enable Bulk disable

Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

Destination host: Sets host name of general-purpose MQTT server to be connected.

Client ID: Sets client ID to connect to general-purpose MQTT server.

Topic prefix: Sets prefix for the topic of MQTT protocol used when data is sent to general-purpose MQTT server. Its suffix follows "/" to be set as a unique ID.

Receiving topic prefix: Sets prefix for the topic of MQTT protocol used to wait for control messages from general-purpose MQTT server.

Protocol: Sets protocol to use for connection (TCP/SSL).

Port number: Sets port number of destination host.

QoS: Sets QoS for MQTT protocol to send data to general-purpose MQTT server.

Receiving QoS: Sets QoS for MQTT protocol when control message is received from general-purpose MQTT server.

Username: Sets username used for connection with general-purpose MQTT server.

Password: Sets password used for connection with general-purpose MQTT server.

Trust Store: Sets path name of the root certificate file to connect to general-purpose MQTT server.

Key Store: Sets path name of the client certificate file to connect to general-purpose MQTT server.

Private key: Sets path name of the private key file to connect to general-purpose MQTT server.

Advanced Setting: Sets detailed values of MQTT protocol.

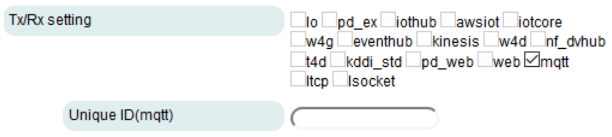
Keep Alive Interval[sec]: Sets keep alive interval of MQTT protocol.

Clean session: Selects if using the clean session of the MQTT protocol.

Retain function: Selects if the retention function of the MQTT protocol is used or not.

Device bulk configuration: To enable/disable all TX destination settings whose TX destination is set to “Enable” in individual device setup menu.

■Settings for individual device setting menu


Tx/Rx setting
☐lo ☐pd_ex ☐iothub ☐awsiot ☐iotcore ☐w4g ☐eventhub ☐kinesis ☐w4d ☐nf_dvhub ☐t4d ☐kddi_std ☐pd_web ☐web ☑mqtt ☐ltcp ☐lsocket

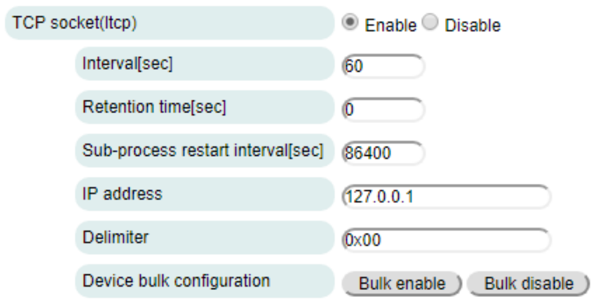
Unique ID(mqtt)

Unique ID(mqtt): Sets unique text string for each device to be used as a suffix of MQTT topics when data is sent to a general-purpose MQTT server or when control message is received. “.” (ASCII code 0x2e） will be converted to “/” (ASCII code 0x2f).

∗Upload the Trust Store, Key Store and private key using the **[System]-[File Management]** tab.

2-4-2-16. TCP(ltcp)

■Settings at the **Tx/Rx setting** menu



TCP socket(ltcp)	[✓ Enable] Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
IP address	127.0.0.1
Delimiter	0x00
Device bulk configuration	Bulk enable Bulk disable

Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

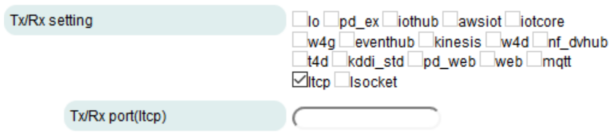
Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

IP address: Sets IP address of general-purpose TCP server to be connected.

Delimiter: Sets one or two-character ASCII code to serve as data border. For example, "0x03" if CR (carriage return) is set as the delimiter, and "0x0d0a" if CRLF (carriage return line feed) is set as the delimiter.

Device bulk configuration: To enable/disable all TX destination settings whose TX destination is set to “Enable” in individual device setup menu.

■Settings for individual device setting menu

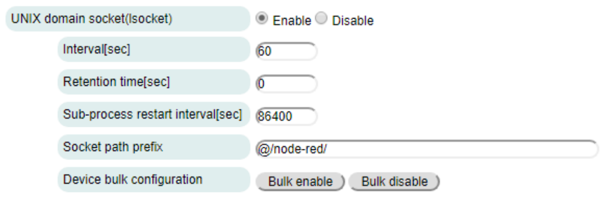


Tx/Rx setting	lo, pd_ex, iothub, awsiot, iotcore, w4g, eventhub, kinesis, w4d, nf_dvhub, t4d, kddi_std, pd_web, web, mqtt, [✓ ltcp], lsocket
Tx/Rx port(ltcp)	

Tx/Rx port(ltcp): Sets port number of general-purpose TCP server to connect between 49152 and 65535.

2-4-2-17 Domain socket(lsocket)

■Settings at the **Tx/Rx setting** menu



UNIX domain socket(lsocket)	[✓ Enable] Disable
Interval[sec]	60
Retention time[sec]	0
Sub-process restart interval[sec]	86400
Socket path prefix	@/node-red/
Device bulk configuration	Bulk enable / Bulk disable

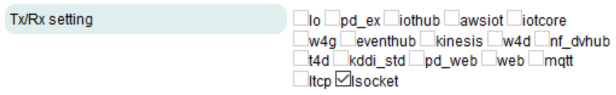
Interval[sec]: Sets time interval between the completion of transmission and the start of next transmission in seconds.

Retention time[sec]: Sets retention time when the PD Repeater cannot send data. If "0" is specified, retention will continue until data transmission is completed.

Sub-process restart interval[sec]: Sets interval to restart a sub-process. Usually, it is not necessary to change the default value (86,400 seconds). If "0" is specified, restart will not be performed.

Socket path prefix: Sets prefix to the path name of the UNIX domain socket to output data. If the name starts with "@", it is treated as an abstract domain socket. As a suffix to the path name, the device number is given.

■Settings for individual device setting menu



Tx/Rx setting: lo pd_ex iothub awsiot iotcore w4g eventhub kinesis w4d nf_dvhub t4d kddi_std pd_web web mqtt ltcp [✓ lsocket]

No setting item is available other than checking check boxes in Tx/Rx setting.

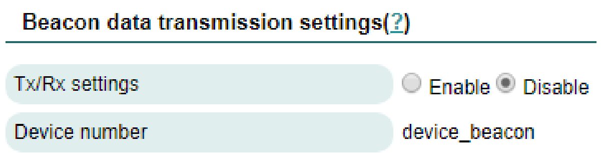
2-4-3. Beacon data transmission settings

From the **Beacon data transmission settings** menu in the **[IoT Data]-[Tx/Rx setting]** tab, it is possible to set up beacon data transmission.

To enable beacon data transmission, BT I/F (hci0) must be set to “Enable” in the **[Basic]-[BT I/F]** tab as the basic setup of the service function.

For basic setup of the service function, refer to Chapter 5 of the OpenBlocks IoT Family WEB UI Set-up Guide.

If the **Beacon data transmission settings** menu is not displayed, refer to Chapter 2-3 "Application setup for IoT data control" and set “Enable” to the PD Handler BLE.



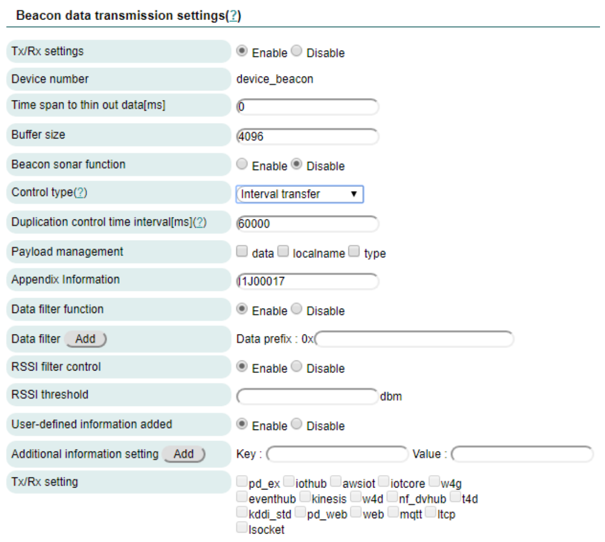
Beacon data transmission settings(?)

Tx/Rx settings	Enable [✓ Disable]
Device number	device_beacon

Default destination setup is as shown in the figure on the left.

To transmit beacon data to cloud, etc., choose “Enable”

Note: This item does not apply to beacons subject to transmissions by a device information transmission setting, which will be discussed later.



Beacon data transmission settings(?)

Tx/Rx settings	[✓ Enable] Disable
Device number	device_beacon
Time span to thin out data[ms]	0
Buffer size	4096
Beacon sonar function	Enable [✓ Disable]
Control type(?)	Interval transfer ▼
Duplication control time interval[ms](?)	60000
Payload management	☐ data ☐ localname ☐ type
Appendix Information	I1J00017
Data filter function	[✓ Enable] Disable
Data filter Add	Data prefix : 0x
RSSI filter control	[✓ Enable] Disable
RSSI threshold	dbm
User-defined information added	[✓ Enable] Disable
Additional information setting Add	Key : Value :
Tx/Rx setting	☐ pd_ex ☐ iothub ☐ awsiot ☐ iotcore ☐ w4g ☐ eventhub ☐ kinesis ☐ w4d ☐ nf_dvhub ☐ t4d ☐ kddi_std ☐ pd_web ☐ web ☐ mqtt ☐ ltcp ☐ lsocket

Choosing “Enable” will display individual setup items.

Device number: This is controlled by the OpenBlocks IoT Family WEB UI and cannot be modified.

Time span to thin out data[ms]: Sets the time input data is not received in order to thin out time in milliseconds. If set to "0", data will not be thinned out.

Buffer size: Sets maximum size of data in bytes.

Beacon sonar function: Sets if beacon sonar is enabled or disabled when beacon data subject to reception is received.

Control type: Selects method to control beacon data. For a description of each system, refer to "Duplicated beacon data control algorithm".

・Interval transfer

・Entry point transfer

・In/out status transfer

Duplication control time interval[msec]: Sets control time for each control type in milliseconds.

Payload management: Selects if beacons will accompany information when beacon data is transferred to PD Repeater.

data: advertise data (hexadecimal)

localname: device name

type: data type

Appendix information: Sets information to accompany beacon data such as from which OpenBlocks IoT Family data is sent, when being sent to individual clouds.

*Default is the unit serial number.

Data filter: (Data prefix) Sets a filter to select a beacon to be transmitted. By entering a filtering condition in the Data prefix field in a hexadecimal string, the system compares the advertised information of beacons with a prefix search and transmits only those that match the search to the destination.

*Use the Add button to make more than one selection.

*To set a data filter, refer to the log data in the unit (local) and filter the device. Filtering will be applied to local logs in this unit.

RSSI threshold: Sets if an RSSI threshold filter is applied to beacons to receive or not.

RSSI threshold: Sets signal strength of beacons to receive in dBm.

User-defined information added: Enables user to add a combination of key names and values to data to be sent to the PD Repeater.

*Use the Add button to register up to five pieces of information.

*Use the Location Information Setting button to enter already registered location information to the form.

Tx/Rx setting: Check boxes for destinations with “Enable” selected will become available for selection. Transmission will be made to destinations with a checked check box. If checking the box, setting items unique to the destination will be displayed. For settings unique to destinations, refer to "2-4-2. Settings for individual Tx/Rx destinations."

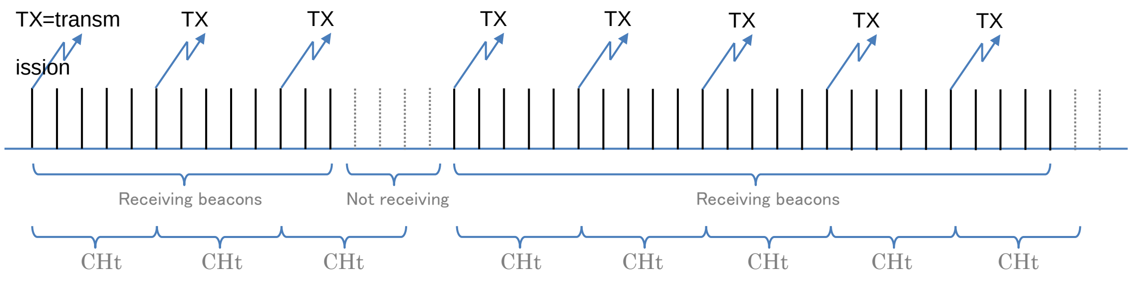
*With beacon sonar function set to the destination and when beacon data set to receive (in consideration of data filter and RSSI threshold) are received with USB speaker connected, detection alarms will be outputted from the speaker.

Duplicated beacon data control algorithm

Presumed settings for this explanation:
Beacon transmission interval: 1 sec.
Duplication control time interval (CHt) = 5 sec.

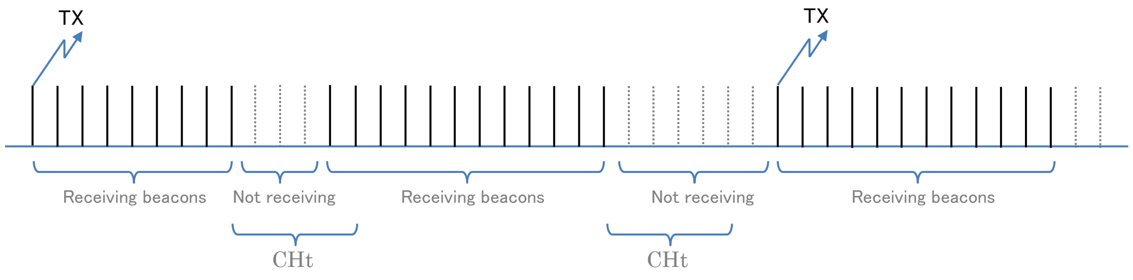
① Interval transfer

Data transmtted to the TX program at designated intervals while beacons are received



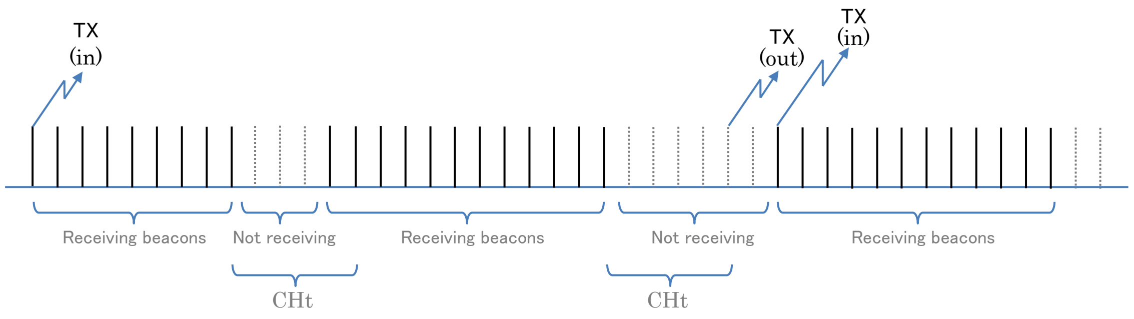
② Entry point transfer

Data transmitted to the TX program once when a beacon is received（Temprary non-reception within CHt time is not regarded as exiting.）



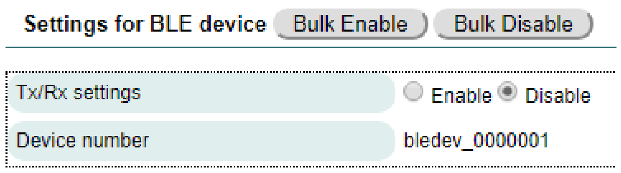
③ In-out status transfer

Data with IN/OUT flag transmitted to TX program at the timing of entering/exiting
（Temprary non-reception within CHt time is not regarded as exiting.）



2-4-4 Setting for BLE device

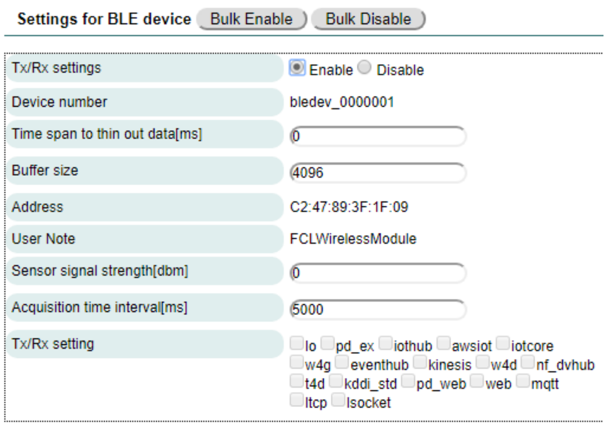
Use the **Setting for BLE device** menu in the **[IoT Data]-[Tx/Rx setting]** tab, it is possible to make transmission setups for BLE device information.
To enable BLE device information transmission, BF I/F (hci0) must be set to “Enable” in the **[Basic]-[BT I/F]** tab as the basic setup of the service function.
In addition, the BLE device must be registered in the **[Basic]-[BLE regist.]** tab as the basic setup of the service function.
For basic setup of the service function, refer to Chapter 5 of the OpenBlocks IoT Family WEB UI Set-up Guide.
If the **Setting for BLE device** menu is not displayed, refer to Chapter 2-3 "Application setup for IoT data control”, and set PD Handler BLE to “Enable”


Settings for BLE device

Tx/Rx settings	Enable [✓ Disable]
Device number	bledev_0000001

If any BLE device is already registered, default settings are as in the figure on the left.
*Here, only one device is registered.
Choosing “Enable” for Tx/Rx setting for each device will display setup items.

*Use the Bulk Enable and Bulk Disable buttons to control all Tx/Rx settings for all registered devices.


Settings for BLE device

Tx/Rx settings	[✓ Enable] Disable
Device number	bledev_0000001
Time span to thin out data[ms]	0
Buffer size	4096
Address	C2:47:89:3F:1F:09
User Note	FCLWirelessModule
Sensor signal strength[dbm]	0
Acquisition time interval[ms]	5000
Tx/Rx setting	lo, pd_ex, iothub, awsiot, iotcore, w4g, eventhub, kinesis, w4d, nf_dvhub, t4d, kddi_std, pd_web, web, mqtt, ltcp, lsocket

Choosing “Enable” will display individual setup items.
Device number: This is controlled by the OpenBlocks IoT Family WEB UI and cannot be modified.
Time span to thin out data[ms]: Sets the time input data is not received in order to thin out time in milliseconds. If set to "0", data will not be thinned out.
Buffer size: Sets maximum size of data in bytes.
Address: Shows BT address of registered device.
User Note: Shows notes created for registered device.
Sensor signal strength[dbm]: For a model

that can set a signal strength to sensors, it is possible to enter a desired signal strength. If a desired signal strength is not available, an approximation or default value will be set.

Acquisition time interval[ms]: Sets time interval to obtain data from sensors in numerical value in milliseconds.

” *Fujitsu Component BT Smart Sensor Beacon and Texas Instruments SimpleLink Sensor Tag only are supported.

Tx/Rx setting: Check boxes for destinations with “Enable” selected will become available for selection. Transmission will be made to destinations with a checked check box. If checking the box, setting items unique to the destination will be displayed. For settings unique to destinations, refer to "2-4-2. Settings for individual Tx/Rx destinations."

*It is recommended to use the same settings so that when existing faulty devices are replaced, they can be handled in the same manner. (Change destination setting of the faulty device to "Disable").

2-4-5. Setting for EnOcean device (For domestic use in Japan only)

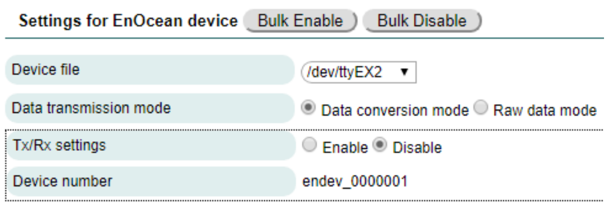
By mounting an EnOcean module (additional expansion module) or USB dongle for EnOcean reception to an OpenBlocks IoT Family product, it is possible to obtain information form an EnOcean device using PD Handler UART.

Use the **Setting for EnOcean device** menu in the **[IoT data]-[Tx/Rx setting]** tab, it is possible to perform settings to obtain information from an EnOcean device and send it.

To collect information from an EnOcean device, an EnOcean device must be registered in the **[Basic]-[EnOcean regist.]** tab as a basic setup of service functions.

For basic setup of the service function, refer to Chapter 5 of the OpenBlocks IoT Family WEB UI Set-up Guide.

If the **Setting for EnOcean device** menu is not displayed, refer to "Chapter 2-3. Application setup for IoT data control," and set PD Handler UART to “Enable”



Device file	/dev/ttyEX2 ▼
Data transmission mode	[✓ Data conversion mode] Raw data mode
Tx/Rx settings	Enable [✓ Disable]
Device number	endev_0000001

If any EnOcean device is already registered, default settings are as in the figure on the left.

*Here, only one device is registered.

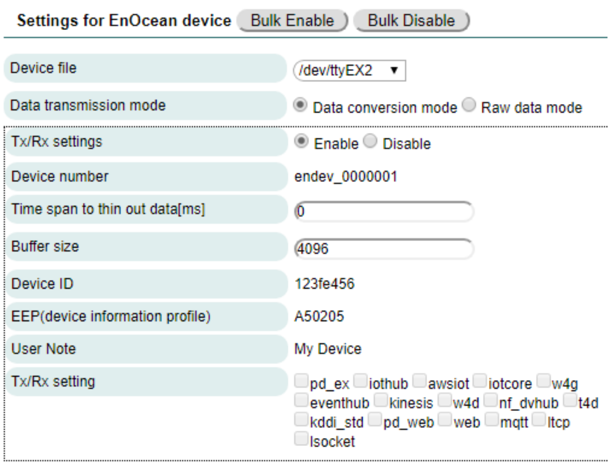
Device file: To use an additional expansion module, choose "/dev/ttyEX2". To use a USB dongle, choose a corresponding device file.

*In the case of a USB dongle, its device file name may differ, depending on the order of recognition of USB devices.

Data transmission mode: Sets data mode to be transmitted to PD Repeater. For data conversion mode, analyzed data will be sent to the PD Repeater for compatible EEP and received data converted to a hexadecimal string will be sent to the PD Repeater for an incompatible EEP. Furthermore, in raw data mode, regardless of compatibility of EEP, received data is converted to a hexadecimal string and sent to the PD Repeater.

Choosing “Enable” for Tx/Rx setting for each device will display setup items.

*The "Bulk Enable" and "Bulk Disable" buttons will set all settings to “Enable” or "Disable."


Settings for EnOcean device

Device file	/dev/ttyEX2
Data transmission mode	[✓ Data conversion mode] Raw data mode
Tx/Rx settings	[✓ Enable] Disable
Device number	endev_0000001
Time span to thin out data[ms]	0
Buffer size	4096
Device ID	123fe456
EEP(device information profile)	A50205
User Note	My Device
Tx/Rx setting	☐ pd_ex ☐ iothub ☐ awsiot ☐ iotcore ☐ w4g ☐ eventhub ☐ kinesis ☐ w4d ☐ nf_dvhub ☐ t4d ☐ kddi_std ☐ pd_web ☐ web ☐ mqtt ☐ ltcp ☐ lsocket

Choosing “Enable” will display individual setup items.

Device number: This is controlled by the OpenBlocks IoT Family WEB UI and cannot be modified.

Time span to thin out data[ms]: sets the time input data is not received in order to thin out time in milliseconds. If set to "0", data will not be thinned out.

Buffer size: Sets maximum size of data in bytes.

Device ID: Shows ID of registered EnOcean device.

EEP(device information profile): Shows device information profile of registered EnOcean device. User Note: Shows notes created for registered device.

Tx/Rx setting: Check boxes for destinations with “Enable” selected will become available for selection. Transmission will be made to destinations with a checked check box. If checking the box, setting items unique to the destination will be displayed. For settings unique to destinations, refer to "2-4-2. Settings for individual Tx/Rx destinations."

*It is recommended to use the same settings so that when existing faulty devices are replaced, they can be handled in the same manner. (Change destination setting of the faulty device to "Disable").

2-4-6. Setting for Modbus client device

A Modbus client can use Modbus protocol to periodically read data from PLC devices (polling) and send data to cloud via PD Repeater.

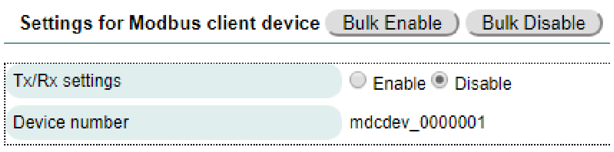
In addition, based on control messages (JSON strings) sent from cloud via the PD Repeater, it can connect to a PLC device using Modbus protocol to read/write data.

Use the **Setting for Modbus client device** menu in the **[IoT data]-[Tx/Rx setting]** tab, it is possible to perform settings of the Modbus client device.

To use a Modbus client device, a Modbus client device must be registered in the **[Basic]-[Modbus(C) regis.]** tab as the basic setup of the service function.

For basic setup of the service function, refer to Chapter 5 of the OpenBlocks IoT Family WEB UI Set-up Guide.

If the **Setting for Modbus client device** menu is not displayed, refer to Chapter 2-3 "Application setup for IoT data control" and set the PD Handler MODBUS Client to “Enable”



Tx/Rx settings	Enable [✓ Disable]
Device number	mdcdev_0000001

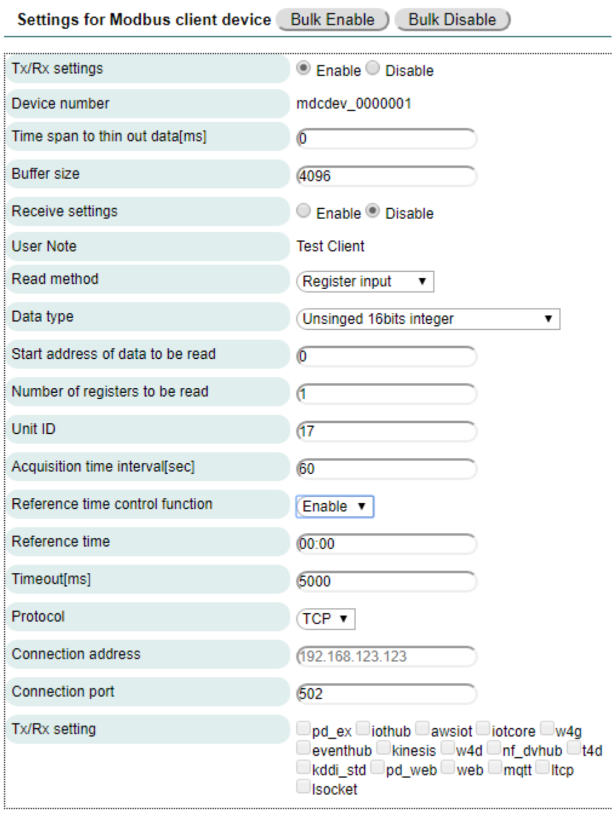
If a Modbus client device is already registered, default settings are as in the figure on the left.

*Here, only one device is registered.

*Modbus client devices do not refer to PLC devices themselves but are conceived as a combination of settings such as "read method," "read start address" and "read register number" to collect data, in addition to connection methods to subject PLC devices.

Choosing “Enable” for Tx/Rx setting for each device will display setup items.

∗The "Bulk Enable" and "Bulk Disable" buttons will set all settings to “Enable” or "Disable."


Settings for Modbus client device | Bulk Enable | Bulk Disable

Tx/Rx settings	[✓ Enable] Disable
Device number	mdcdev_0000001
Time span to thin out data[ms]	0
Buffer size	4096
Receive settings	Enable [✓ Disable]
User Note	Test Client
Read method	Register input
Data type	Unsinged 16bits integer
Start address of data to be read	0
Number of registers to be read	1
Unit ID	17
Acquisition time interval[sec]	60
Reference time control function	Enable
Reference time	00:00
Timeout[ms]	5000
Protocol	TCP
Connection address	192.168.123.123
Connection port	502
Tx/Rx setting	☐ pd_ex ☐ iothub ☐ awsiot ☐ iotcore ☐ w4g ☐ eventhub ☐ kinesis ☐ w4d ☐ nf_dvhub ☐ t4d ☐ kddi_std ☐ pd_web ☐ web ☐ mqtt ☐ ltcp ☐ lsocket

*When "TCP" is selected as Protocol

Choosing “Enable” will display individual setup items.

Device number: This is controlled by the OpenBlocks IoT Family WEB UI and cannot be modified.

Time span to thin out data[ms]: Sets the time input data is not received in order to thin out time in milliseconds. If set to "0", data will not be thinned out.

Buffer size: Sets maximum size of data in bytes.

Receive setting: Selects if downstream control to receive control messages from cloud via PD Repeater or not.

User Note: Shows notes created for registered device.

Read method: Selects "Digital output (bit)", "digital input (bit)", "Register output" or "Register input".

If "Digital output" or "Digital input" is selected, a row of "0" or "1" will be output.

Data type: Selects one of the following data types when "Register output" or "Register input" is selected for Read method.

・Unsigned 16 bits integer

・Signed 16 bits integer

・Unsigned 32 bits integer/little endianness

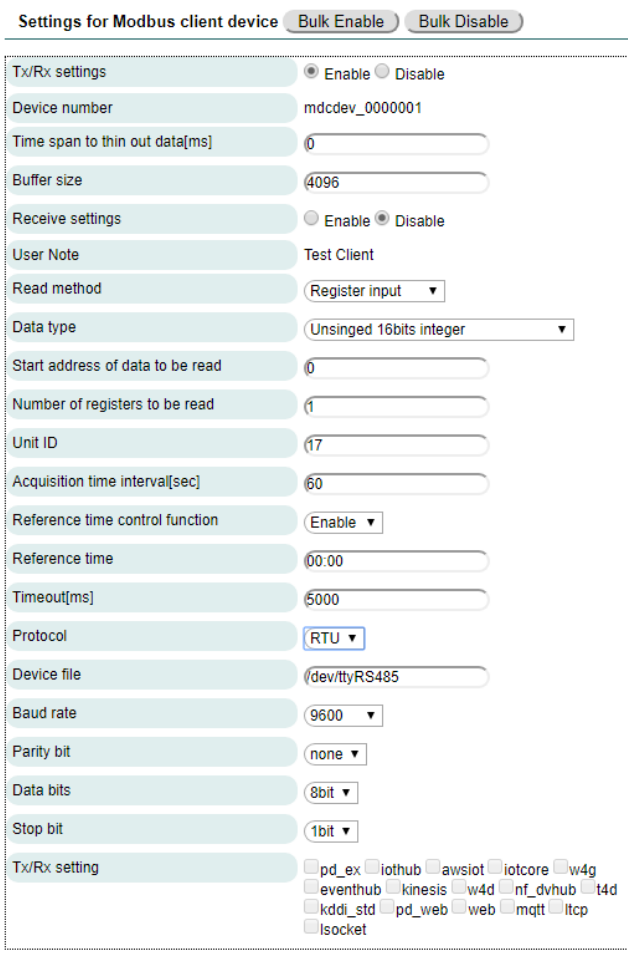
・Signed 32 bits integer/little endianness

・Unsigned 32 bits integer/big endianness

・Signed 32 bits integer/big endianness

Start address of data to be read: Sets read start address on PLC device storing data to be read.

Number of registers to be read: Interpreted as the number of bits to be read if "Digital output" or "Digital input" is selected



Settings for Modbus client device Bulk Enable Bulk Disable

Tx/Rx settings	[✓ Enable] Disable
Device number	mdcdev_0000001
Time span to thin out data[ms]	0
Buffer size	4096
Receive settings	Enable [✓ Disable]
User Note	Test Client
Read method	Register input
Data type	Unsinged 16bits integer
Start address of data to be read	0
Number of registers to be read	1
Unit ID	17
Acquisition time interval[sec]	60
Reference time control function	Enable
Reference time	00:00
Timeout[ms]	5000
Protocol	RTU
Device file	/dev/ttyRS485
Baud rate	9600
Parity bit	none
Data bits	8bit
Stop bit	1bit
Tx/Rx setting	☐pd_ex ☐iothub ☐awsiot ☐iotcore ☐w4g ☐eventhub ☐kinesis ☐w4d ☐nf_dvhub ☐t4d ☐kddi_std ☐pd_web ☐web ☐mqtt ☐ltcp ☐lsocket

*When "RTU" is selected as Protocol

for Read method.

Sets number of registers or bits to be read from the address set for "Start address of data to be read".

Unit ID: Sets Modbus unit ID of PLC device. Unit idea is number between 1 and 247 or 255.

Acquisition time interval[sec]: Sets interval of time to acquire data from PLC device in seconds. If the reference time control function described below is used, time interval is internally converted to the values below:

・86400[sec] × integral multiple
・43200[sec]
・28800[sec]
・21600[sec]
・14400[sec]
・10800[sec]
・7200[sec]
・3600[sec]
・1800[sec]
・900[sec]
・60[sec]

Reference time control function: To acquire data at a specific hour, enable this function and set the reference time.

Reference time: Sets reference time to acquire data at a specific hour. Format is HH:MM.

Timeout[msec]: Sets timeout when data is acquired from PLC device in milliseconds.

Protocol: Selects "TCP" or "RTU". The former is network, while the latter is serial.

Connection address (TCP): Sets IP address of PLC device to be connected.

Connection port: Sets TCP port number of PLC device to be connected. Default value is "502".

Device file (RTU): Sets device filename of serial port to connect PLC device.

Baud rate: Selects baud rate for the serial port.

Parity bit (RTU): Selects parity bit for the serial port.

Data bit (RTU): Selects data bit for the serial port.

Stop bit (RTU): Selects stop bit for the serial port.

Tx/Rx setting: Check boxes for destinations with “Enable” selected will become available for selection. Transmission will be made to destinations with a checked check box. If checking the box, setting items unique to the destination will be displayed. For settings unique to destinations, refer to "2-4-2. Settings for individual Tx/Rx destinations."

■ Expansion of acquired Modbus client devices using CSV file

By placing a CSV file whose filename is **pd_handler_modbus_client.csv** in /var/webui/upload_dir using the **[System]-[File Management]** tab, it is possible to assign multiple acquired Modbus client devices for a single device number set up by WEB UI.
Format of this CSV is as follows:

Device number, Unit ID, Read method, Data type, Start address of data to be read, Number of registers to be read

Parameter	Data format	Description
Device number	Alphanumerical	Gives device number assigned by WEB UI. Device numbers not set up in WEB UI will be ignored. If the first character of a line is "#" or "/", the line will be treated as a comment line.
Unit ID	Numerical	Sets Modbus unit ID of PLC device. Unit ID is between 1 and 247 or 255.
Read method	Alphanumerical	One of the following to be given as a read method: Digital output: "read_coils" or "0x01" or "1" Digital input: "read_discrete_input" or "0x02" or "2" Register output: "read_holding_registers" or "0x03" or "3" Register input: "read_input_registers" or "0x04" or "4"
Data type	Alphanumerical	One of the following to be given as a data type: Unsigned 16 bit integer: "uint16_t" or "0" Signed 16 bit integer: "int16_t" or "1" Unsigned 32 bit integer/little endianness: "uint32lsb_t" or "2" Signed 32 bit integer/little endianness: "int32lsb_t" or "3" Unsigned 32 bit integer/big endianness: "uint32msb_t" or "4" Signed 32 bit integer/big endianness: "int32msb_t" or "5"
Start address of data to be read	Alphanumerical	Sets start address on the PLC device storing data to read. If the address starts with "0x", it will be interpreted as hexadecimal.
Number of registers to be read	Numerical	Enter the register number to read.

CSV file format for PD Handler Modbus Client

Parameters are divided by a comma. If the parameter starts with a sharp (#) or slash (/), it is considered a comment line.

Examples:

```
#localname,unit_id,read_function,data_type,read_addr,read_registers
mdcdev_0000001,15,read_coils,uint16_t,0x130,37
mdcdev_0000001,15,read_discrete_input,uint16_t,0x1c4,22
mdcdev_0000001,15,read_holding_registers,uint16_,0x160,3
mdcdev_0000001,16,read_input_registerss,uint32lsb_t,0x108,1
mdcdev_0000002,17,read_coils,uint16_t,0x130,37
mdcdev_0000002,18, read_discrete_input,u_int16,0x1c4,22
mdcdev_0000002,19,read_holding_registers,int16,0x160,3
mdcdev_0000002,20,read_input_registers,int32lsb,0x108,1
mdcdev_0000003,30,read_coils,u_int16,0x130,37
mdcdev_0000003,30,read_discrete_input,u_int16,0x1c4,22
mdcdev_0000003,31,read_holding_registers,u_int16,0x160,3
mdcdev_0000003,31read_,input_registers,u_int32msb,0x108,1
mdcdev_0000004,32,read_coils,u_int16,0x130,37
mdcdev_0000004,32,read_discrete_input,u_int16,0x1c4,22
mdcdev_0000004,33,read_holding_registers,int16,0x160,3
mdcdev_0000004,33,read_input_registers,int32msb,0x108,1
```

When a CSV file is read, the Modbus client device set by WEB UI will be overwritten. For this reason, in the CSV file, give all the Modbus client devices to be acquired, including Modbus client devices set by WEB UI.

■**Reference time control**

Reference time control is a function to acquire data at a specific hour.

Set the Reference time control item in the **Setting for Modbus client device** menu to "Enable" and set the acquisition time interval and acquisition hour in the **Acquisition time interval[sec]** and the **Reference time**, respectively.

For reference time control, the **Acquisition time interval** is limited to 300, 600, 900, 1800, 3600, 7200, 10800, 14400, 21600, 28800, 43200 and multiples of 86400. If any other time is entered for the **Acquisition time interval**, it will be treated in the PD Handler Modbus client as per the following:

Setting values of acquisition time interval	Actual working values
0 ～ 599	300
600 ～ 899	600
900 ～ 1799	900
1800 ～ 3599	1800
3600 ～ 7199	3600
7200 ～ 10799	7200
10800 ～ 14399	10800
14400 ～ 21599	14400
21600 ～ 28799	21600
28800 ～ 43199	28800
43200 ～ 86399	43200
86400～	Multiple of 86400

Settings and actual working values of **Acquisition time intervals** in reference time control

The **Reference time** is the time serving as the starting point of action. For example, suppose the **Acquisition time interval** is set to "300" and the **Reference time** to "00:01." In this case, data is acquired at 00: 01, 00: 06, 00: 11 ... 00: 56, 01: 01 ... 23: 56, 00: 01 sharp.

The starting time of data acquisition will not be the time set in the **Reference time** but the time just coming, calculated by the **Reference time** and the **Acquisition time interval**.

For example, if settings are made at "08:30" for the **Reference time** of "01:05" and the **Acquisition time interval** of "10800," the first data acquisition will take place at "10:05", followed by 13: 05, 16: 05, 19: 05, 22: 05, 01: 05.

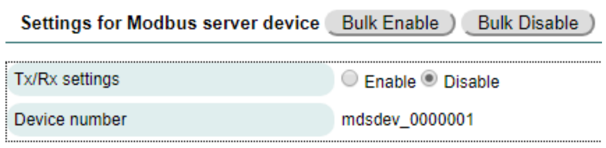
2-4-7. Setting for Modbus server device

The Modbus server maintains a register map, as shown in the table below, a wait for connections from PLC device via Modbus protocol, updates the register value written by the PLC device and sends the updated register and its values to cloud via the PD Repeater. Based on control messages sent from cloud (JSON string) via the PD Repeater, it can also read and write a register map.

Register	Start address	Size
Digital output (Coils)	0x000	uint8 _t × 2048
Digital input (Discrete Input)	0x000	uint8_t × 2048
Register output (Holding Registers)	0x000	uint16_t × 2048
Register input (Input Registers)	0x000	uint16_t × 2048

Register map of PD Handler Modbus Server

Use the **Setting for Modbus server device** menu in the **[IoT Data]-[Tx/Rx setting]** tab, it is possible to setup a Modbus server device.
To use a Modbus server device, a Modbus server device must be registered in the **[Basic]-[Modbus(S) regis.]** tab as the basic setup of the service function.
For basic setup of the service function, refer to Chapter 5 of the OpenBlocks IoT Family WEB UI Set-up Guide.
If the Setting for Modbus server device menu is not displayed, refer to Chapter 2-3 "Application setup for IoT data control" and set PD Handler MODBUS Server to “Enable”



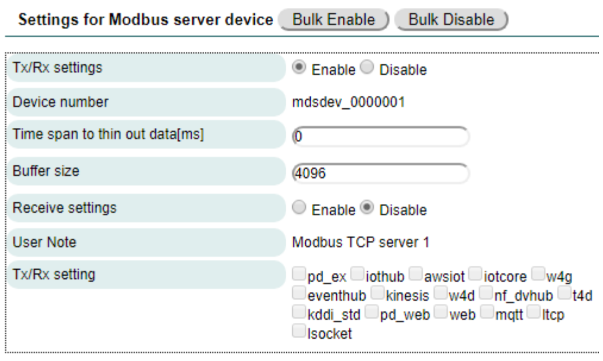
Tx/Rx settings	Enable [✓ Disable]
Device number	mdsdev_0000001

If a Modbus server device is already registered, default settings are as in the figure on the left.
*Here, only one device is registered.

Choosing “Enable” for Tx/Rx setting for each device will display setup items.

∗The "Bulk Enable" and "Bulk Disable" buttons will set all settings to “Enable” or "Disable."

■ When the Device type (waiting type) of the Modbus server device is set to "TCP":



Settings for Modbus server device | Bulk Enable | Bulk Disable

Tx/Rx settings	[✓ Enable] Disable
Device number	mdsdev_0000001
Time span to thin out data[ms]	0
Buffer size	4096
Receive settings	Enable [✓ Disable]
User Note	Modbus TCP server 1
Tx/Rx setting	☐pd_ex ☐iothub ☐awsiot ☐iotcore ☐w4g ☐eventhub ☐kinesis ☐w4d ☐nf_dvhub ☐t4d ☐kddi_std ☐pd_web ☐web ☐mqtt ☐ltcp ☐lsocket

Choosing “Enable” will display individual setup items.

Device number: This is controlled by the OpenBlocks IoT Family WEB UI and cannot be modified.

Time span to thin out data[ms]: Sets the time input data is not received in order to thin out time in milliseconds. If set to "0", data will not be thinned out.

Buffer size: Sets maximum size of data in bytes.

Receive setting: Sets if control messages from cloud are received or not via PD Repeater.

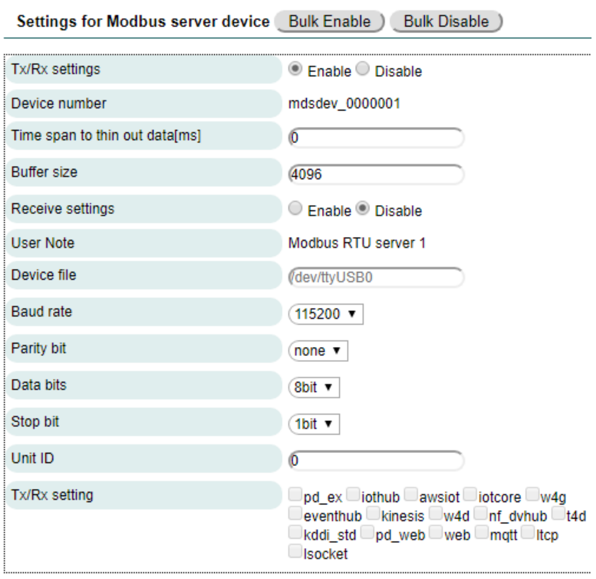
User Note: Shows notes created for registered device.

Tx/Rx setting: Check boxes for destinations with “Enable” selected will become available for selection. Transmission will be made to destinations with a checked check box. If checking the box, setting items unique to the destination will be displayed. For settings unique to destinations, refer to "2-4-2. Settings for individual Tx/Rx destinations."

*If waiting with TCP, it is necessary to keep the TCP port (502) of Modbus protocol open. In the **Filter open Settings** menu in the **[System]-[Filter]** tab, set Modbus to "Enable."

The number of Modbus server devices whose Device type (waiting type) that can be registered from WEB UI for TCP is limited to one due to the relationship with opening operation of the TCP port (502).

■ When Device type (waiting type) of the Modbus server device is set to "RTU":



Settings for Modbus server device | Bulk Enable | Bulk Disable

Tx/Rx settings	[✓ Enable] Disable
Device number	mdsdev_0000001
Time span to thin out data[ms]	0
Buffer size	4096
Receive settings	Enable [✓ Disable]
User Note	Modbus RTU server 1
Device file	/dev/ttyUSB0
Baud rate	115200
Parity bit	none
Data bits	8bit
Stop bit	1bit
Unit ID	0
Tx/Rx setting	☐pd_ex ☐iothub ☐awsiot ☐iotcore ☐w4g ☐eventhub ☐kinesis ☐w4d ☐nf_dvhub ☐t4d ☐kddi_std ☐pd_web ☐web ☐mqtt ☐ltcp ☐lsocket

Choosing “Enable” will display individual setup items.

Device number: This is controlled by the OpenBlocks IoT Family WEB UI and cannot be modified.

Time span to thin out data[ms]: Sets the time input data is not received in order to thin out time in milliseconds. If set to "0", data will not be thinned out.

Buffer size: Sets maximum size of data in bytes.

Receive setting: Selects if downstream control to receive control messages from cloud via PD Repeater or not.

User Note: Shows notes created for registered device.

Device file: Sets device filename for the serial port to wait for connections from PLC devices.

Baud rate: Selects baud rate for the serial port to wait for connections from PLC devices.

Parity bit (RTU): Selects parity bit for the serial port.

Data bit (RTU): Selects data bit for the serial port.

Stop bit (RTU): Selects stop bit for the serial port.

Unit ID: Sets the Modbus unit ID on the OpenBlocks IoT Family side waiting for connections from PLC device. Unit ID is a numerical value between 1 and 247.

Tx/Rx setting: Check boxes for destinations with “Enable” selected will become available for selection. Transmission will be made to destinations with a checked check box. If checking the box, setting items unique to the destination will be displayed. For settings unique to destinations, refer to "2-4-2. Settings for individual Tx/Rx destinations."

*Multiple Modbus server devices whose Device Type (waiting type) is set to "RTU" can be registered, but if the **Device file** settings overlap, performance cannot be guaranteed.

*If multiple Modbus service devices are registered, the register map will be shared by multiple Modbus server devices. As a result, these Modbus server devices are the input/output devices of the Modbus server.

2-4-8. Setting for User defined device

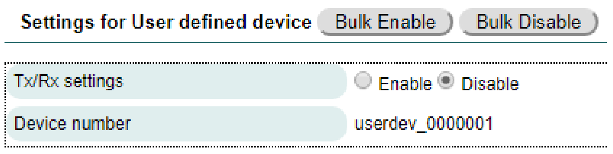
Instead of using standard OpenBlocks IoT Family data collection application (PD Handler), users can independently prepare an application to handle data and cooperate with the PD Repeater.

Use the **Setting for User defined device** menu in the **[IoT Data]-[Tx/Rx setting]** tab, it is possible to make settings for a user-defined device.

To use a user-defined device, a user device must be registered in the **[Basic]-[User Dev. regis.]** tab as the basic setup of the service function.

For basic setup of the service function, refer to Chapter 5 of the OpenBlocks IoT Family WEB UI Set-up Guide.

The registration of a user-defined device refers to the operation of allocating a device number and adding note information.



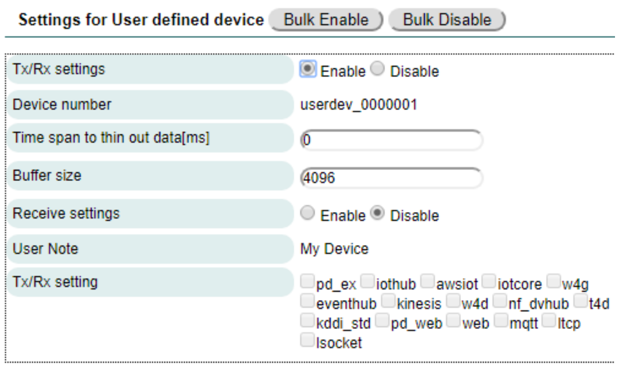
Tx/Rx settings	Enable [✓ Disable]
Device number	userdev_0000001

If a user-defined device is already registered, default settings are as in the figure on the left.

*Here, only one device is registered.

Choosing “Enable” for Tx/Rx setting for each device will display setup items.

∗The "Bulk Enable" and "Bulk Disable" buttons will set all settings to “Enable” or "Disable."



Tx/Rx settings	[✓ Enable] Disable
Device number	userdev_0000001
Time span to thin out data[ms]	0
Buffer size	4096
Receive settings	Enable [✓ Disable]
User Note	My Device
Tx/Rx setting	☐pd_ex ☐iothub ☐awsiot ☐iotcore ☐w4g ☐eventhub ☐kinesis ☐w4d ☐nf_dvhub ☐t4d ☐kddi_std ☐pd_web ☐web ☐mqtt ☐ltcp ☐lsocket

Choosing “Enable” will display individual setup items.

Device number: This is controlled by the OpenBlocks IoT Family WEB UI and cannot be modified.

Time span to thin out data[ms]: Sets the time input data is not received in order to thin out time in milliseconds. If set to "0", data will not be thinned out.

Buffer size: Sets maximum size of data in bytes.

Receive setting: Selects if control messages from cloud are received via PD Repeater.

User Note: Shows notes created for registered device.

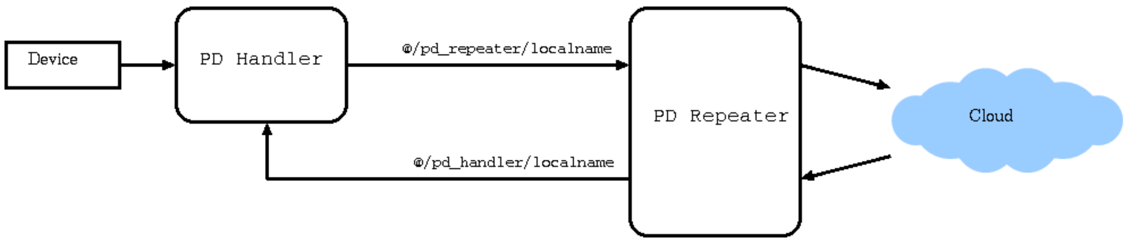
Tx/Rx setting: Check boxes for destinations with “Enable” selected will become available for selection. Transmission will be made to destinations with a checked check box. If checking the box, setting items unique to the destination will be displayed. For settings unique to destinations, refer to "2-4-2. Settings for individual Tx/Rx destinations."

2-5. Downstream control

This section explains the downstream control function to receive messages from cloud, etc. and how to control PLC devices supporting Modbus.

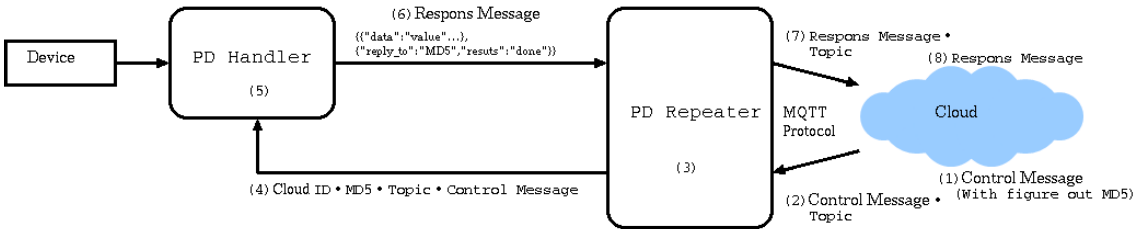
2-5-1. Overview of downstream control

Software modules that provide an IoT data control function use a UNIX domain socket for inter-process communication. Downstream control also operates using a UNIX domain socket.
Of destinations supported by the PD Repeater, downstream control supports MQTT protocol-based cloud service, PD Exchange(pd_ex), the web server of Plat'Home's original specifications(pd_web) and TCP(ltcp) connections.
With MQTT protocol-based and TCP connections, downstream control operates with an always-on connection for both transmission/reception (Pub/Sub) and for PD Exchange, while periodically performing polling to determine if there are any control messages.
The web server of Plat'Home's original specifications deals with response messages at the time of data transmission as downstream control messages.
Control messages from upstream (cloud) are transferred to downstream software modules such as PD Handler Modbus Client from a UNIX domain socket via the PD Repeater. Downstream software modules return control results to cloud via the PD Repeater using a UNIX domain socket for data transmission.



Control messages assume a JSON character string, yet as information that cannot be included in payload may be required as control requirement on the cloud side, and while topics may be required for MQTT, command IDs for PD Exchange, response headers for web server of Plat'Home's original specifications and port numbers for TCP connection, the said information as well as a hash value (MD5) is added when control messages are transferred from the PD Repeater to downward software modules.
The hash value of a control message is used to clarify to which control message control results from downstream software modules respond to.

Taking the example of cloud connected with MQTT protocol, the flow of downstream control is explained below.


Device
PD Handler
(5)
(6) Respons Message
{{"data":"value"...},
{"reply_to":"MD5","resuts":"done"}}
PD Repeater
(3)
(7) Respons Message • Topic
(8) Respons Message
Cloud
MQTT Protocol
(1) Control Message (With figure out MD5)
(2) Control Message • Topic
(4) Cloud ID • MD5 • Topic • Control Message

(1) Calculates the hash value (MD5) for control messages.
(2) Publishes a control message using MQTT protocol.
(3) Based on the control message, calculates its hash value (MD5).
(4) Transfers the control message to a downstream software module, together with cloud ID (number for convenience on PD Repeater), hash value of control message (MD5) and topic.
(5) Based on the control message, performs the processing.
(6) Returns a response message with process results, in addition to the hash value of the control message (MD5).
(7) Publishes the control message via MQTT protocol.
(8) Comparing the hash value on the response message (MD5) and that calculated in (1) (MD5) will clarify to which control message the control message addresses.

2-5-2. Downstream messages of PD Repeater

If using the IoT data control function incorporated in the OpenBlocks IoT Family as-is, no additional attention is needed for this item.

■Shown below is the format of a message to be transferred to downstream software module by the PD Repeater.

Cloud ID (1Byte)	Sub ID (1Byte)	Header Size (2Bytes)	MD5 (16Bytes)	Header	Payload

Cloud ID (1 Byte): Number of the sender and destination protocol on PD Repeater assigned for convenience.

Sub ID (1 Byte): Sub-number ("0x00" or "0x01") of the sender and destination protocol on PD Repeater assigned for convenience.

Header Size (2 Bytes): Size of header added by PD Repeater.

Header: MQTT topic or HTTP response header added by PD Repeater.

Payload: Control message sent from cloud.

■Table below shows the details of Cloud ID and header

Cloud	Cloud ID	Downstream control	Header description
PD Exchange	0x00	Supported	Command ID, ApplicationID
MQTT server	0x01	Supported	MQTT reception topic
Watson IoT for Device	0x02	Supported	MQTT reception topic
Amazon Kinesis	0x03		
<Reserved>	0x04		
<Reserved>	0x05		
WEB server	0x06		
AWS IoT	0x07	Supported	MQTT reception topic
MS Azure Event hubs	0x08		
Watson IoT for Gateway	0x09	Supported	MQTT reception topic
MS Azure IoT Hub	0x0a	Supported	MQTT reception topic
Toami for DOCOMO	0x0b		
Domain socket	0x0c		
KDDI IoT Cloud Standard	0x0d		
IoT device hub	0x0e	Supported	MQTT reception topic
Web server of Plat'Home's original specifications	0x0f	Supported	HTTP response header
Google IoT Core	0x10	Supported	MQTT reception topic
TCP	0x11	Supported	Destination IP address and port number

Cloud ID and contents of header

For Watson IoT, as payload is set as a JSON object of "d" key, after obtaining hash value (MD5), decodes and outputs only JSON object of "d" key as control message.

For IoT Device Hub, as payload is a JSON object of a "parameters" key, after obtaining the hash value (MD5), decodes and outputs only JSON object of "parameters" key as control message.

Destination IP address and port number of TCP socket are added with the following JSON character string.。

{“ip_addr”:”<*IP address*>”,”port”:<*port*>}

2-5-3. Downstream control of Modbus client/server

This section explains the downstream control of PD Handler Modbus.
In order to use the downstream control of PD Handler Modbus, it is necessary to understand the function codes used by Modbus.
Shown below is a list of function codes used by PD Handler Modbus.

Code	Name	Description
0x01	Read Coils	Reads bit value set for digital output.
0x02	Read Discrete Input	Reads bit value of digital input.
0x03	Read Holding Registers	Reads value set in register output.
0x04	Read Input Registers	Reads register input value.
0x05	Write Single Coil	Writes bit value to digital output 1 bit.
0x06	Write Single Register	Writes value to register output 1 register.
0x07	Read Exception Status	Notifies error status between Modbus server/client.
0x09	Write Single Discrete Input	Writes bit value to digital input 1 bit.
0x0a	Write Single Input Register	Writes value to register input 1 register.
0x0f	Write Multiple Coils	Writes bit values to multiple consecutive digital outputs.
0x10	Write Multiple Registers	Writes values to multiple consecutive register outputs.
0x11	Report Slave ID	Notifies IDs of slave (server) devices that can be connected.
0x13	Write Multiple Discrete Input	Writes bit values to multiple consecutive digital inputs.
0x14	Write Multiple Input Registers	Writes values to multiple consecutive register inputs.
0x16	Mark Write Registers	Masks register output.
0x17	Write and Read Registers	Writes values to multiple consecutive register outputs and reads their values.

Modbus Function Code for PD Handler

00x09, 0x0a, 0x13, 0x14 are proprietary expansion functions that are different from original Modbus protocol prepared to be able to set digital input or register input of PD Handler Modbus Server without physical input from cloud side.

Modbus function codes are set to "function" keys in the JSON character string in the configuration file of PD Handler Modbus (pd_handler_modbus_client.conf, pd_handler_modbus_server.conf) or downstream control.
In CSV file (pd_handler_modbus_client.csv), it is set to a third column as "reading method".
It is possible to set function code in a hexadecimal notation starting from "0x" to the value of function keys and strings. It is also possible to use integer notation and string notation in

addition to hexadecimal notation.

Shown below is a list of function codes that can be used for function keys.

Code	Name	Integer notation	String notation
0x01	Read Coils	1	read_coils
0x02	Read Discrete Input	2	read_discrete_input
0x03	Read Holding Registers	3	read_holding_registers
0x04	Read Input Registers	4	read_input_registers
0x05	Write Single Coil	5	write_single_coil
0x06	Write Single Register	6	write_single_register
0x07	Read Exception Status	7	read_exception_status
0x09	Write Single Discrete Input	9	write_single_input_register
0x0a	Write Single Input Register	10	write_single_input_register
0x0f	Write Multiple Coils	15	write_multiple_coils
0x10	Write Multiple Registers	16	write_multiple_register
0x11	Report Slave ID	17	report_slave_id
0x13	Write Multiple Discrete Input	19	write_muliple_discrete_input
0x14	Write Multiple Input Registers	20	write_multiple_input_registers
0x16	Mark Write Registers	22	mark_write_registers
0x17	Write and Read Registers	23	write_and_read_registers

List of function codes that can be used for function keys

2-5-3-1. Modbus client

This section explains the downstream control of PD Handler Modbus Client.
Shown below is a list of objects for JSON character strings used in downstream control of PD Handler Modbus Client.

Key	Data type	Required	Description
protocol	String	○	"tcp" for TCP-connected device and "rtu" for RTU-connected device
node	String	△	IP address of TCP-connected PLC device
port	String	△	Port number of TCP-connected PLC device
device	String	△	Device file for RTU-connected PLC device
unit	Integer	○	Modbus ID of PLC device: 1 ~ 247 or 255 (for TCP connection only)
function	String or integer	○	Codes described in next table, "Function codes that can be used for PD Handler Modbus Client" among function codes are explained in Chapter 2-5-3.
data_type	String or integer	-	One of the following: Unsigned 16 bit integer: "uint16_t" or "0" Signed 16 bit integer: "int16_t" or "1" Unsigned 32 bit integer/little endianness: "uint32lsb_t" or "2" Signed 32 bit integer/little endianness: "int32lsb_t" or "3" Unsigned 32 bit integer/big endianness: "uint32msb_t" or "4" Signed 32 bit integer/big endianness: "int32msb_t" or "5" If not designated, unsigned 16 bit integer
address	String or integer	-	Sets start address on the PLC device to which data is or will be stored. If '0x' is appended to the beginning of a character string, will be interpreted as a hexadecimal notation.
number	String or integer	-	Sets register number to be read/written. If not designated, "1".
values	Integer array	△	An array of data to be written (integer)

JSON character string objects used for downstream control by PD Handler Modbus Client

Shown in table below are function codes that can be used for PD Handler Modbus Client

Code	Name	Can be used or not
0x01	Read Coils	○
0x02	Read Discrete Input	○
0x03	Read Holding Registers	○
0x04	Read Input Registers	○
0x05	Write Single Coil	○
0x06	Write Single Register	○
0x07	Read Exception Status	
0x09	Write Single Discrete Input	
0x0a	Write Single Input Register	
0x0f	Write Multiple Coils	○
0x10	Write Multiple Registers	○
0x11	Report Slave ID	○
0x13	Write Multiple Discrete Input	
0x14	Write Multiple Input Registers	
0x16	Mark Write Registers	
0x17	Write and Read Registers	○

Function codes that can be used for PD Handler Modbus Client

For example, to read from a TCP-connected input resistor, the following control message will be sent from cloud.

```
{
    “protocol”:”tcp”,”node”:”192.168.1.8”,”port”:”502”,“unit”:255,
    ”function”:”0x04”,”data_type”:”uinit16_t”,”address”:”0x160”,”number”:5
}
```

Response message will be as follows:

```
{
    "time":"2017-09-05T15:30:05.758+09:00",
    "reply_to":"452556d8daf1f7eb483d00ee03718e8e","result":"done",
    "memo":"Modbus Client 00",
    "protocol":"tcp","node":"192.168.1.8","port":502,
    "unit":255,"address":352,"function":4,"data_type":"uint16_t",
    "values":[65535,0,1,2,3]
}
```

Here, "reply_to" is the hash value (MD5) of control message.

To write a value to the output resistor on a PLC device serially connected, a message like the following will be sent from cloud.

```
{
        "protocol":"rtu","device":"/dev/ttyEX1",
        "unit":16,"function":"0x10","data_type":"uint32lsb_t","address":"0x120",
        "number":2,"values":[4567,891011]
}
```

Here, if "data_type" is 32 bits, "values" will be divided into upper and lower 16 bits and processed. Therefore, even if "number" is "1", function must use "Write Multiple Registers".

Response message will be as follows:

```
{
        "time":"2017-09-05T16:35:13.653+09:00",
        "reply_to":"fe9b49ff045f435a371303a82281f3f9","result":"done",
        "memo":"Modbus Client 01",
        "protocol":"rtu","device":"/dev/ttyMDF1",
        "unit":16,"address":288,"function":16,"data_type":"uint32_t",
        "values":[4567,891011]
}
```

Note that (connection methods to) PLC devices designated by "protocol", "node", "port" and "device" must be registered as Modbus client devices and use settings enabled.

2-5-3-2. Modbus server

This section explains the downstream control of PD Handler Modbus Server.
Shown below is a list of objects for JSON character strings used in downstream control of PD Handler Modbus Server.

Key	Data type	Required	Description
function	String or integer	○	Codes described in next table, "Function codes that can be used for PD Handler Modbus Client" among function codes are explained in Chapter 2-5-3.
data_type	String or integer	-	One of the following: Unsigned 16 bit integer: "uint16_t" or "0" Signed 16 bit integer: "int16_t" or "1" Unsigned 32 bit integer/little endianness: "uint32lsb_t" or "2" Signed 32 bit integer/little endianness: "int32lsb_t" or "3" Unsigned 32 bit integer/big endianness: "uint32msb_t" or "4" Signed 32 bit integer/big endianness: "int32msb_t" or "5" If not designated, unsigned 16 bit integer
address	String or integer	-	Sets start address on the PLC device to which data is or will be stored. If '0x' is appended to the beginning of a character string, will be interpreted as a hexadecimal notation. If not designated, "0x00"
number	String or integer	-	Sets register number to be read/written. If not designated, "1".
values	Integer array	△	An array of data to be written (integer)

JSON character string objects used for downstream control by PD Handler Modbus Server

Different from PD Handler Modbus Client, the register map of PD Handler Modbus Server itself will be operated, therefore, "node" or "device" key settings are not required.

Shown in table below are function codes that can be used for PD Handler Modbus Server.

Code	Name	Can be used or not
0x01	Read Coils	○
0x02	Read Discrete Input	○
0x03	Read Holding Registers	○
0x04	Read Input Registers	○
0x05	Write Single Coil	○
0x06	Write Single Register	○
0x07	Read Exception Status	
0x09	Write Single Discrete Input	○
0x0a	Write Single Input Register	○
0x0f	Write Multiple Coils	○
0x10	Write Multiple Registers	○
0x11	Report Slave ID	
0x13	Write Multiple Discrete Input	○
0x14	Write Multiple Input Registers	○
0x16	Mark Write Registers	
0x17	Write and Read Registers	○

Function codes that can be used for PD Handler Modbus Server

For example, to read from an input resistor, the following control message will be sent from cloud.

```
    {
        "function":"0x04","data_type":"uint16_t","address":"0x160","number":5
    }
```

Response message will be as follows:

```
    {
        "time":"2017-09-05T15:30:05.758+09:00",
        "reply_to":"452556d8daf1f7eb483d00ee03718e8e","result":"done",
        "memo":"Modbus Server 00",
        "protocol":"tcp","node":"192.168.1.8","port":502,
        "unit":255,"address":352,"function":4,"data_type":"uint16_t",
        "values":[65535,0,1,2,3]
    }
```

Here, "reply_to" is the hash value (MD5) of control message.

Though the register map does not differentiate "node", "port" or "device", as they are positioned as individual Modbus server devices with a different device number from PD

Repeater, response message will be appended with "node", "port" or "device" to form a counterpart to UNIX domain socket that receives a control message from the PD Repeater.

To write a value to the input register, a message like the following will be sent from cloud.

The input resistor is essentially a resistor physically set via an AD converter, etc., but with PD Handler Modbus Server, the register map can be written over by downstream control.

```
    {
        "function":"0x14","data_type":"uint32_t","address":"0x140","number":4,
        "values":[4567,8910,561,435]
    }
```

Here, if "data_type" is 32 bits, "values" will be divided into upper and lower 16 bits and processed. Therefore, even if "number" is "1", function must use "Write Multiple Registers".

Response message will be as follows:

```
    {
        "time":"2017-09-05T16:35:13.653+09:00",
        "reply_to":"d5bcc347ad36fa6e7090d0f763108882","result":"done",
        "memo":"Modbus Server 01",
        "protocol":"rtu","device":"/dev/ttyEX1",
        "unit":17,"address":320,"function":20,"data_type":"uint32_t",
        "values":[4567,8910,561,435]
    }
```

2-5-4. PD Agent

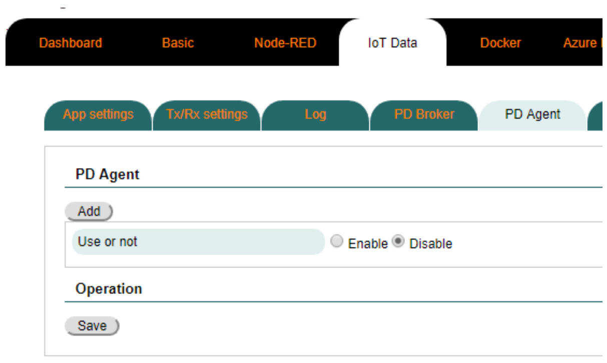
PD Agent receives control messages (JSON character strings) from the PD Repeater and with preset key and value matched, carries out execution objects, as prepared by the user or shell script. (Comparison condition in the PD Agent is perfect match in string type).
PD Agent is an application prepared on the assumption that by using user-defined config. setups, it will cooperate with PD Handler BLE, etc. that does not have any downstream control function. In this chapter, however, instead of cooperation with the PD Handler, we will take the example of a method to enable the application to be registered as a user-defined device and work independently to explain how to use it.

2-5-4.1. Registration of user-defined device and setup of sender and destination

1. Register the PD Agent as a user-defined device using **[Basic]-[User Dev. regist.]** tab.
   If operated as multiple devices as in the case where there are multiple senders and destinations, register multiple devices.
   For basic setup of the service function, refer to Chapter 5 of the OpenBlocks IoT Family WEB UI Set-up Guide.
   Registration of a user-defined device is a process that involves assigning a device number and adding note information.
2. Use the **Application control** menu in the **[IoT Data]-[App setting]** tab, set the PD Agent to be started up.
   For startup settings of applications, refer to Chapters 2 and 3.
3. Use the **Tx/Rx setting** menu in the **[IoT Data]-[Tx/Rx setting]** tab, choose a sender and destination and setup a setting not depending on individual devices in the **Tx/Rx setting** menu.
   For transmission/reception settings, refer to Chapter 2-4-2.
4. Use the **Setting for User defined device** menu in the **[IoT Data]-[Tx/Rx setting]** tab, set transmission and reception setup for a user-defined device.
   For transmission/reception settings of a user-defined device, refer to Chapter 2-4-8.

2-5-4-1. PD Agent setup

Can set the PD Agent using the **[IoT Data]-[PD Agent]** tab.



PD Agent

Add

Use or not	Enable [✓ Disable]

Operation

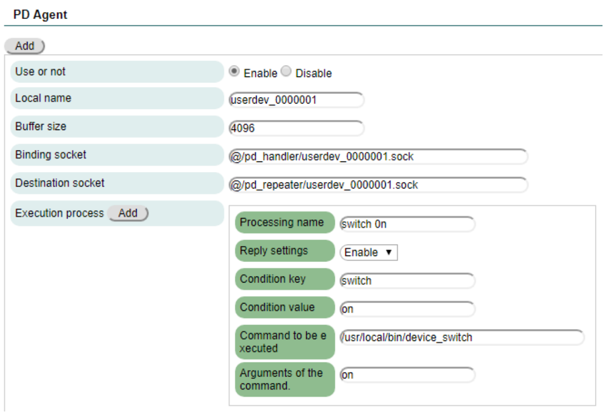
Save

The figure on the left shows the default screen.

Set To be use to "Enable" the showing of setting items.

To set up multiple devices, press the Add button to add the setup form.

Set To be use to "Enable" the showing of setting items.



PD Agent

Add

Use or not	[✓ Enable] Disable
Local name	userdev_0000001
Buffer size	4096
Binding socket	@/pd_handler/userdev_0000001.sock
Destination socket	@/pd_repeater/userdev_0000001.sock
Execution process Add	

Processing name	Switch On
Reply settings	Enable
Condition key	Switch
Condition value	on
Command to be executed	/usr/local/bin/device_switch
Arguments of the command.	on

To be use: Selects if this setup is enabled or disabled.

Local name: Usually, it is the device number assigned to device.

Buffer size: Sets maximum size of data in bytes, match this value with that on device side.

Binding socket: Pathname of the UNIX domain socket to receive control messages.

In this example, set to:

@/pd_handler/userdev_0000001.sock

Details will be given in the margin.

Destination socket: Pathname of the UNIX domain socket to send response messages.

In this example:

@/pd_repeater/userdev_0000001.sock

Details will be given in the margin.

Execution process (Add): To set more than one condition key, condition value, command to execute and argument of the command, use the Add key.

Processing name: Gives name of the process.

Reply settings: Sends response message when this item is set to "Enable." This is used when the command to execute does not have a function to return a message.

Condition key: Sets a JSON character string key included in the control message to be evaluated as an execution condition of an object designated in Command to execute.

Condition value: Sets value of JSON character string included in the control message to be evaluated as an execution condition of an object designated in Command to execute.

Execution command: Sets pathname of an execution object or shell script to be executed when the key designated by Condition key and value designated in Condition value are included in control message.

Arguments of command: Sets argument of execution command.

*Binding socket

This is the pathname of UNIX domain socket to receive control messages.
As a device, it uses a user-defined device. In this example where it directly receives control messages from the PD Repeater, set to "@/pd_handler/userdev_0000001.sock.
The prefix, "@", means that the pathname is an abstract UNIX domain socket, and if wait for an execution object that does not support abstract UNIX domain socket names, it will be a pathname that exists in a file system like, "/tmp/pd_agent_0000001.sock".

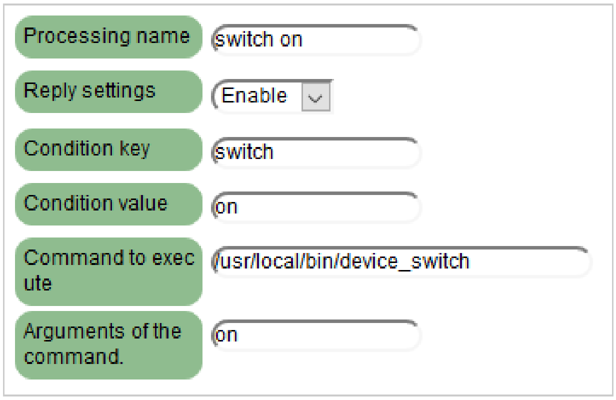
*Pathname of destination socket

This is the pathname of UNIX domain socket to send response messages.
As a device, it uses a user-defined device. In this example where it directly receives control messages from the PD Repeater, set to "@/pd_repeater/userdev_0000001.sock".
The prefix, "@", means that the pathname is an abstract UNIX domain socket, and if wait for an execution object that does not support abstract UNIX domain socket names, it will be a pathname that exists in a file system like, /tmp/ anyobject_0000001.sock".

2-5-4-2. PD Agent setup and control messages

Shown below are examples of the PD Agent settings and control messages.

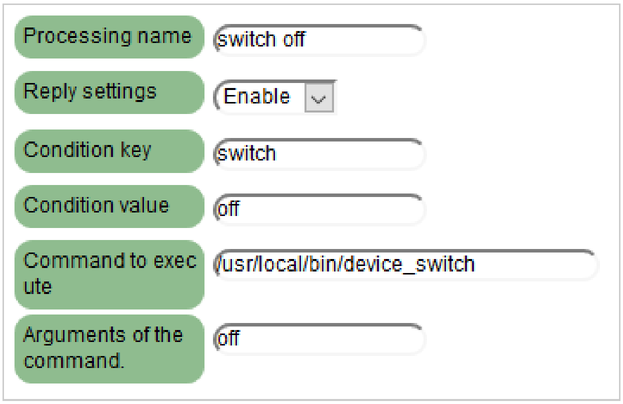
１． Control message for the settings below will be " {“switch”: ”on”}".



Processing name	switch on
Reply settings	Enable
Condition key	switch
Condition value	on
Command to execute	/usr/local/bin/device_switch
Arguments of the command.	on

If a matching control message is received, "/usr/local/bin/device_switch on" will be executed.

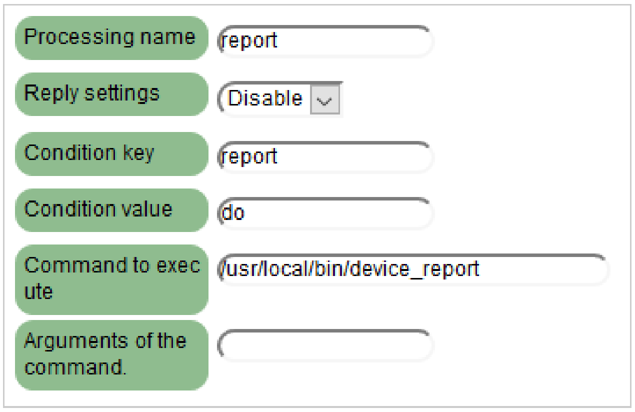
２． Control message for the settings below will be "{“switch”: ”off”}.



Processing name	switch off
Reply settings	Enable
Condition key	switch
Condition value	off
Command to execute	/usr/local/bin/device_switch
Arguments of the command.	off

If a matching control message is received, "/usr/local/bin/device_switch off" will be executed.

3. Control message for the settings below will be " {“report”: ”do”}.



Processing name	report
Reply settings	Disable
Condition key	report
Condition value	do
Command to exec ute	/usr/local/bin/device_report
Arguments of the command.	

If a matching control message is received, "/usr/local/bin/device_report" will be executed.

4. In the above example, if the control message is "{“switch”: ”on”, “report”: ”do”}", "/usr/local/bin/device_switch on" and "usr/local/bin/device_report" will be executed.

If a message to be applied with multiple execution conditions is received, processes will be executed in series. However, the order of execution will not be fixed.

2-5-4-3 Succession to environmental variables

When control messages (as JSON character strings) include the key and value to be used as execution conditions of an execution object or shell scrip and the value is either a string or numerical value, they will be set up as environmental variables and succeeded by the execution object or shell script.

For example, in the example below, if a control message, " {“switch”: ”on”, “report”: ”do”,”page”: 40} " is received, the following environmental variables will be succeeded in command execution.

switch=”on”
report=”do”
page=40

In addition, regardless of the content of control messages, the PD agent will succeed the following values as environmental variables to execution objects or shell scripts in executing the execution object or shell script.

Environmental variable	Data type	Description
request_cloud_id	Integer	Cloud ID described in Chapter 2-5-2
request_sub_id	Integer	Sub ID described in Chapter 2-5-2
request_header	String	Header described in Chapter 2-5-2
request_md5	String	Hash value of control message described in Chapter 2-5-2
request_payload	String	Payload sent from cloud
agent_localname	String	Local name set in Local name in Chapter 2-5-4-1
agent_bind	String	Binding socket name set in Binding socket in Chapter 2-5-4-1
agent_push_to	String	Destination socket name set in Destination socket in Chapter 2-5-4-1
agent_buffer_size	Integer	Buffer size set in Buffer size in Chapter 2-5-4-1

Parameters succeeded to execution object or shell script as environmental variables

2-5-4-4 Response messages

When Reply settings in Chapter 2-5-4-1 are set to "Enable", the following response message will be returned as an execution status.

Execution command started up:

{“time”:”*timestamp*”,”reply_to”:”*md5*”,”result”:”queuing”,
”reason”:”matched”,”matched”:{“*key*”:”*value*”}}

Execution of execution command failed:

{“time”:”*timestamp*”,”reply_to”:”*md5*”,”result”:”faild”,
”reason”:”matched”,”matched”:{“*key*”:”*value*”}}

Execution of execution command completed:

{“time”:”*timestamp*”,”reply_to”:”*md5*”,”result”:”done”,
”reason”:”matched”,”matched”:{“*key*”:”*value*”}}

No matching key and value:

{“time”:”*timestamp*”,”reply_to”:”*md5*”,”result”:”not queuing”,
”reason”:”key and value not matched”}

Control message not a JSON character string:

{“time”:”*timestamp*”,”reply_to”:”*md5*”,”result”:”not queuing”,
”reason”:”not JSON form”}

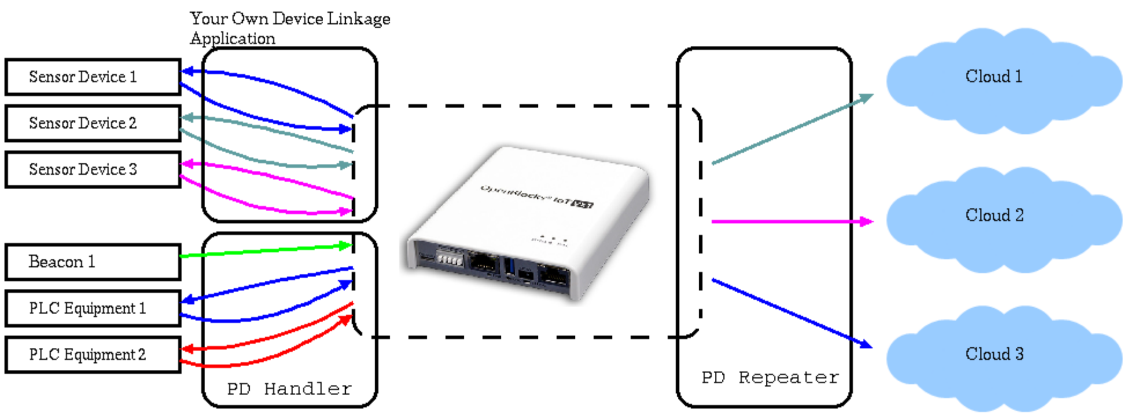
Here, md5 is the hash value of control message described in Chapter 2-5-2, and
{“key”: ”value”} is matched Condition key and Condition value.
The execution status is a return value from execv() to call execution object or shell script and does not show the processing results of such an execution object or shell script. To achieve perfection of process, set response messages to return from the execution object or shell script to the UNIX domain socket that is succeeded by "agent_push_to" an environmental variable.

Chapter 3 Customization

This section explains how to cooperate the IoT data control function of the OpenBlocks IoT Family and applications independently prepared by user.

3-1 Independently developed data collection application

This section describes how the user independently develops an application to collect data from individual devices instead of using our application (PD Handler) and have it cooperate with the PD Repeater to send data to cloud.
Shown below is a conceptual image:



3-1-1 Registration of user-defined device and set up of sender and destination

1. Use the **[Basic]-[User Dev. regist.]** tab, it is possible to register an independently developed application as a user-defined device.
   For basic setup of the service function, refer to Chapter 5 of the OpenBlocks IoT Family WEB UI Set-up Guide.
   Registration of a user-defined device is a process that involves assigning a device number and adding note information.
2. If a device file used by an independently developed application conflicts with that of our application (PD Handler), disable the application using the **Application control** menu in the **[IoT Data]-[App setting]** tab to ensure it does not start up.
3. For the start/stop control of an independently developed application, refer to Chapter 3-3.

4. Use the **Tx/Rx setting** menu in the **[IoT Data]-[Tx/Rx setting]** tab, choose a sender and destination and setup a setting not depending on individual devices in the **Tx/Rx setting** menu.
   For transmission/reception settings, refer to Chapter 2-4-2.
5. Use the **Setting for User defined device** menu in the **[IoT Data]-[Tx/Rx setting]** tab, set transmission and reception setup for a user-defined device.
   For transmission/reception settings of a user-defined device, refer to Chapter 2-4-8.

3-1-2 Writing data to PD Repeater

PD Repeater creates an abstract UNIX domain socket based on the device number set up by WEB UI. (This socket is created for a device whose Tx destination is set to “Enable” with an effective destination, excluding local).
When this UNIX domain socket is written, written data will be sent to cloud.
Path rules of subject UNIX domain socket are as follows:
(If using a user-defined config file described in Chapter 2-3, this pathname can be changed to a desired pathname).

●UNIX domain socket of PD Repeater
    ¥0/pd_repeater/<device number>.sock

Given below is a sample of writing {“x”: 1}" to the PD Repeater UNIX domain socket.

Sample of writing in command line is as follows:

*Written with <device number> as userdev_0000001 and written in PD Repeater[*1]:

```
# echo -n ‘{“x”:1}’ | socat stdin abstract-connect:/pd_repeater/userdev_0000001.sock
```

Script sample with PHP is as follows:
* Written with <device number> as userdev_0000001:

```
<?php
        $value = '{ "x" : 1 }' ;
        $sock = socket_create(AF_UNIX , SOCK_STREAM , 0) ;
        socket_set_nonblock($sock);
        socket_set_option( $sock, SOL_SOCKET, SO_SNDTIMEO, array( "sec"=>1, "usec"=>0 ) ) ;
        socket_connect($sock , "¥0/pd_repeater/userdev_0000001.sock" , 0 );
        socket_write($sock, $value);
        socket_close($sock);
?>
```

Below is a script sample in Node.js:

*Written to PD Repeater with <device number> as "userdev_0000001"[*2]:

```
var absocket = require('abstract-socket');
try {
		var absclient = absocket.connect('¥0/pd_repeater/userdev_0000001.sock', function() {
				console.log('connect ok');
		});
		absclient.write('{"x":1}');
		absclient.end();
} catch(e) {
		console.log('fail');
}
process.exit();
```

3-2. Independently developed downstream control application

Here are key points on using an application independently developed by user to receive control messages from cloud via the PD Repeater and to perform certain processing.

3-2-1. Registration of user-defined device and setup of sender and destination

The registration of user-defined devices, in addition to setup of sender and destination (cloud) are the same as those of data collection applications described in Chapter 3-1-1.
Use the **Setting for User defined device** menu in the **[IoT Data]-[Tx/Rx setting]** tab, choose "Enable" for Receive setting.

3-2-2. Writing data from PD Repeater

Upon receiving a control message from cloud, the PD Repeater will write data in abstract UNIX domain socket to be determined on the basis of device number setup by WEB UI in the format described in Chapter 2-5-2.
Path rules of subject UNIX domain socket are as follows:
(If using a user-defined config file described in Chapter 2-3, this pathname can be changed to a desired pathname).

●UNIX domain socket to which PD Repeater sends control messages:
¥0/pd_handler/<device number>.sock

3-3 Start/stop control of independently developed applications

By preparing a few files, can link the start/stop control of the PD Repeater by WEB UI with independently developed applications.

3-3-1. Registration of applications

Register (write) the name of application to link to the following file:

/etc/default/obsiot-webui-ext-handler

In this file, write one application name per line. A blank line or a duplicated application name is not permitted.

Example:

```
testhandler
myhandler
```

3-3-2. Designating script to be used by the application

Indicate script to be used by the application to link with in the following file:

/etc/default/<application name>

In the file, designate start, stop and status check scripts as follows:

```
bootcmd_<application name>=”<startup script>”
haltcmd_<application name>=”<stop script>”
statuscmd_<application name>=”<status check script>”
```

After running status check script, if "is running" or "RUNNING" is outputted or status is not "inactive," "failed" or "dead", the WEB UI dashboard recognizes the status as being in operation.

Example: /etc/default/ testhandler

```
bootcmd_ testhandler=”/etc/init.d/testhandler start”
haltcmd_ testhandler=”/etc/init.d/testhandler stop”
statuscmd_ testhandler=”/etc/init.d/testhandler status”
```

3-3-3. deb package

It is recommended to organize the application registration processing in Chapter 3-3-1, together with application and its startup/stop script into a deb package (Debian GNU Linux software package).

For detail on how to create a deb package, please visit the official Debian website.

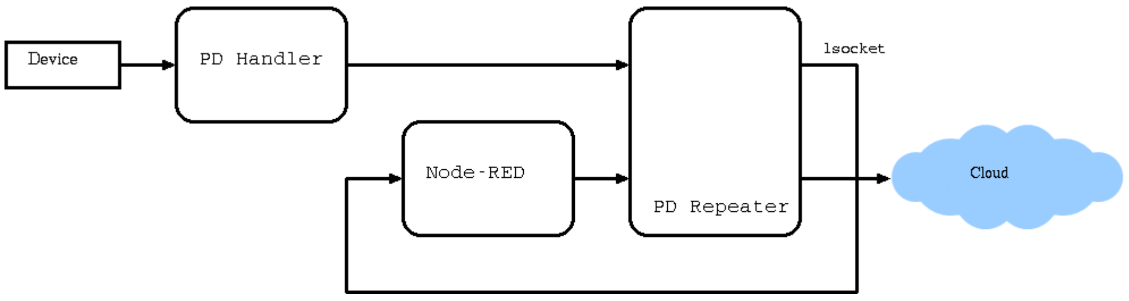
Due caution should be paid to the following:

- ☐ The application registration file (/etc/default/obsiot-webui-ext-handler) described in Chapter 3-3-1 is edited by postinstall or postrm script for multiple deb packages.
- ☐ For an application that outputs log files via syslog, restart rsyslog using postinstall. (The syslog service used by the OpenBlocks IoT Family is rsyslog).
- ☐ WEB UI implements startup control, etc. but a config. file to be used for the application will not be generated. Therefore, it is recommended to put a template config. file in the deb package.
- ☐ A config. file is needed for setup to output logs via syslog. In addition, for the output destination, a timpfs domain is recommended, not regular real storage. As a tmpfs domain is prepared in "/var/webui/pd-logs" by WEB UI, please write here. Also, note that files prepared with the extension, ".log" in this domain can be browsed from the log check tab.

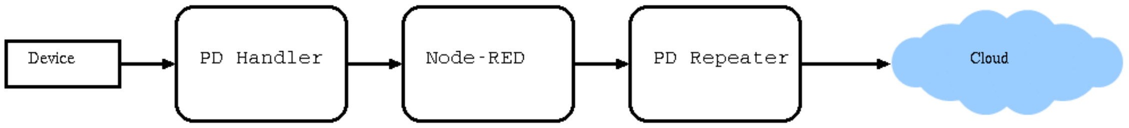
  If a large amount of logs are continuously created, the file size will become large, straining the tmpfs domain. Therefore, add a log rotation setup. From the viewpoint of an overflowing tmpfs, a rotation setting is recommended to be triggered based on file size.

3-4. Realizing complex configurations

In order to filter data from the PD Handler with Node-RED, by defining Node-RED as a user-defined device at WEB UI settings, it is possible to obtain a configuration to send data to Node-RED from the PD Repeater by using Isocket and return output from Node-RED to the PD handler.



However, by using the user-defined config. file described in Chapter 2-3 to change the UNIX domain socket name, it is also possible to achieve a configuration with a more throughput-oriented configuration.



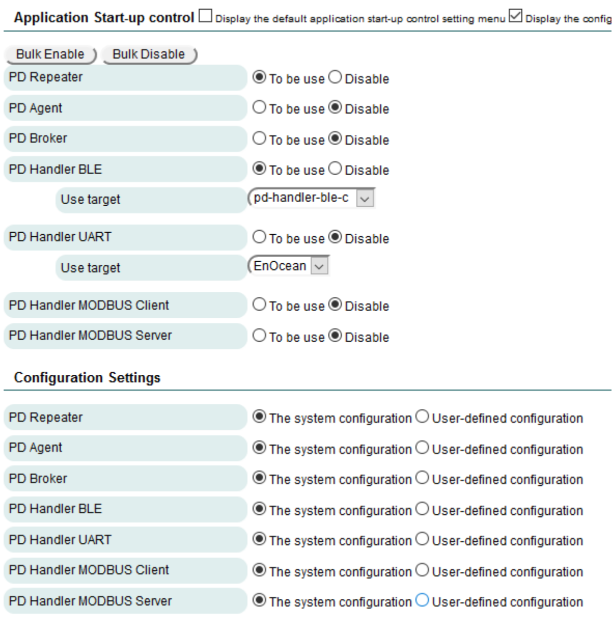
The setup file (config.) is written in a JSON format. The UNIX domain socket is expressed as the following keys:

Key	Format	
bind	String	The name of the UNIX domain socket waiting for data. If this is left blank, a socket name will have a default value of each application appended as a prefix and a device number as a suffix. If "@" is appended as a prefix, it means there is an abstract UNIX domain name.
push_to	String	The name of the UNIX domain socket to send data. If this is left blank, a socket name will have a default value of each application appended as a prefix and a device number as a suffix. If "@" is appended as a prefix, it means there is an abstract UNIX domain name.

:

Here, the setting method is explained by taking a Beacon device (PD Handler BLE) as an example.

1. Check that the **Configuration Settings** mode is set to "The system configuration" as shown in Chapter 2-3.



Application Start-up control ☐ Display the default application start-up control setting menu ☑ Display the config

Bulk Enable | Bulk Disable

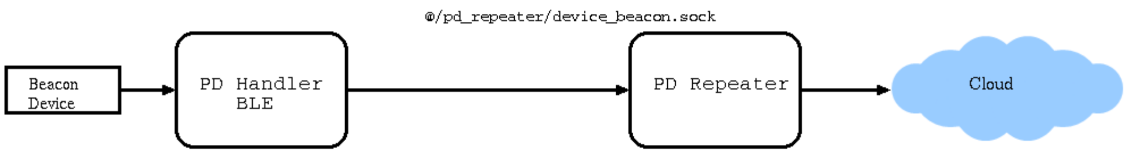
PD Repeater	[✓ To be use] Disable
PD Agent	To be use [✓ Disable]
PD Broker	To be use [✓ Disable]
PD Handler BLE	[✓ To be use] Disable
Use target	pd-handler-ble-c
PD Handler UART	To be use [✓ Disable]
Use target	EnOcean
PD Handler MODBUS Client	To be use [✓ Disable]
PD Handler MODBUS Server	To be use [✓ Disable]

Configuration Settings

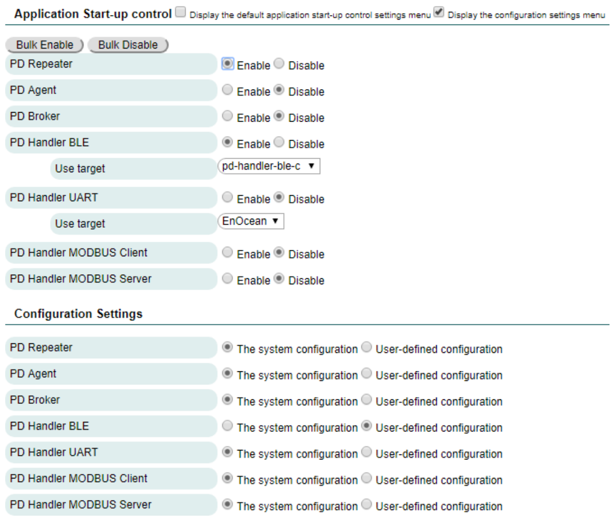
PD Repeater	[✓ The system configuration] User-defined configuration
PD Agent	[✓ The system configuration] User-defined configuration
PD Broker	[✓ The system configuration] User-defined configuration
PD Handler BLE	[✓ The system configuration] User-defined configuration
PD Handler UART	[✓ The system configuration] User-defined configuration
PD Handler MODBUS Client	[✓ The system configuration] User-defined configuration
PD Handler MODBUS Server	[✓ The system configuration] User-defined configuration

Use the **[IoT Data]-[App settings]** tab, check the "Display the config" check box and confirm that the PD Handler BLE is set to "The system configuration."

2. Use the **Tx/Rx setting** menu in the **[IoT Data]-[Tx/Rx setting]** tab, make sender and destination settings. For the setting method, refer to Chapter 2-4-2.
3. Use the **Beacon data transmission setting** menu in the **[IoT Data]-[Tx/Rx setting]** tab, make settings to receive and send beacons. For the setting method, refer to Chapter 2-4-3.
4. With the above settings, the PD Repeater will wait for data with a UNIX domain socket, @/pd_repeater/device_beacon.sock, and the PD Handler BLE will write data in @/pd_repeater/device_beacon.sock.
   Here, "@" in the UNIX domain socket names means that they are abstract UNIX domain names.



5. Switch **Configuration Settings** to "User-defined configuration."


Application Start-up control ☐ Display the default application start-up control settings menu ☑ Display the configuration settings menu

Bulk Enable | Bulk Disable

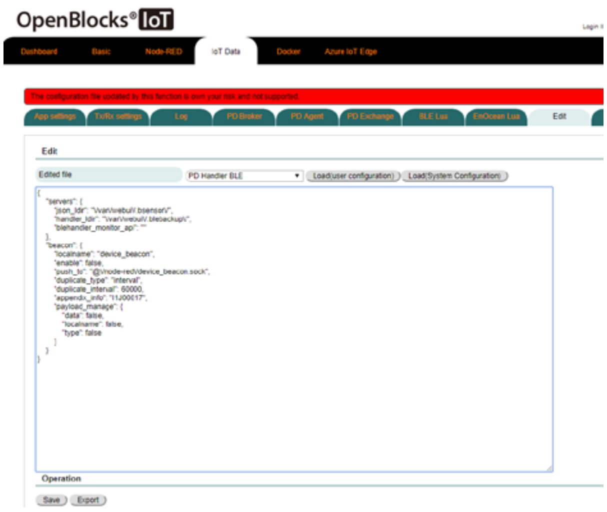
PD Repeater	[✓ Enable] Disable
PD Agent	Enable [✓ Disable]
PD Broker	Enable [✓ Disable]
PD Handler BLE	[✓ Enable] Disable
Use target	pd-handler-ble-c
PD Handler UART	Enable [✓ Disable]
Use target	EnOcean
PD Handler MODBUS Client	Enable [✓ Disable]
PD Handler MODBUS Server	Enable [✓ Disable]

Configuration Settings

PD Repeater	[✓ The system configuration] User-defined configuration
PD Agent	[✓ The system configuration] User-defined configuration
PD Broker	[✓ The system configuration] User-defined configuration
PD Handler BLE	The system configuration [✓ User-defined configuration]
PD Handler UART	[✓ The system configuration] User-defined configuration
PD Handler MODBUS Client	[✓ The system configuration] User-defined configuration
PD Handler MODBUS Server	[✓ The system configuration] User-defined configuration

Use the **[IoT Data]-[App settings]** tab, check the "Display the config" check box and confirm that the PD Handler BLE is set to "User-defined configuration."

6. Copy the system configuration created by WEB UI as a user configuration, and change the UNIX domain socket name from "@/pd_repeater/device_beacon.sock" to "@/node-red/device_beacon.sock.



Edit

Edited file: PD Handler BLE | Load(user configuration) | Load(System Configuration)

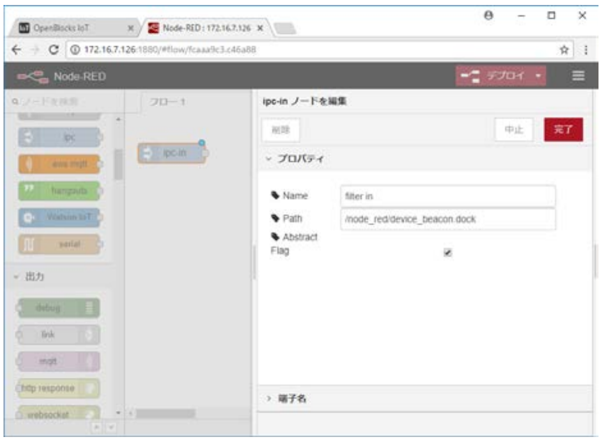
```
{
  "servers": {
    "json_ldir": "\/var\/webui\/.bsensor\/",
    "handler_ldir": "\/var\/webui\/.blebackup\/",
    "blehandler_monitor_ap": ""
  },
  "beacon": {
    "localname": "device_beacon",
    "enable": false,
    "push_to": "@\/node-red\/device_beacon.sock",
    "duplicate_type": "interval",
    "duplicate_interval": 60000,
    "appendix_info": "11J00017",
    "payload_manage": {
      "data": false,
      "localname": false,
      "type": false
    }
  }
}
```

Operation

Save | Export

1. Use the **[IoT Data]-[Edit]** tab, choose "PD Handler BLE" for Edited File.
2. Click the "Load (System Configuration)" button, then copy the setting information shown in the text area (JSON character strings).
3. Click the "Load (user configuration) " button, then paste the setting information (JSON character strings) copied in Item 2 to text area.
4. Change the value of "push_to" key to: @¥/node-red¥/device_beacon.sock".
5. Press the Save button.

7. Set the property of the UNIX domain socket input ipc-in node of Node-RED.
   On how to use Node-RED, refer to the OpenBlocks IoT Family Node-RED Start Guide."


ipc-in ノードを編集

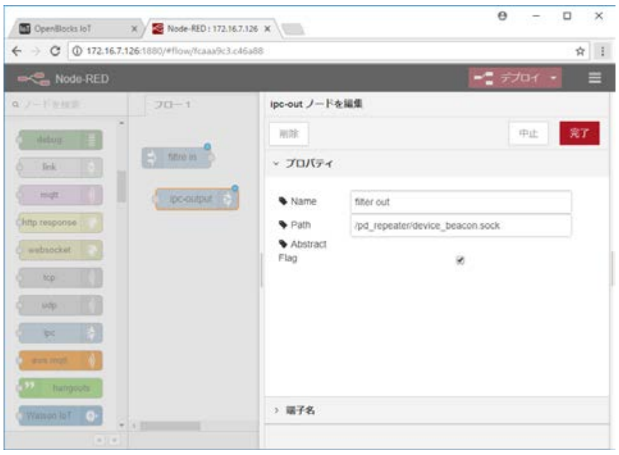
Name: filter in

Path: /node_red/device_beacon.dock

Abstract Flag: ✓

Put the ipc-in node in place, set the Path in Property to "/node-red/device_beacon.sock" and check the Abstract Flag check box.

8. Set the property of the UNIX domain socket output, ipc-out node of Node-RED.


ipc-out ノードを編集

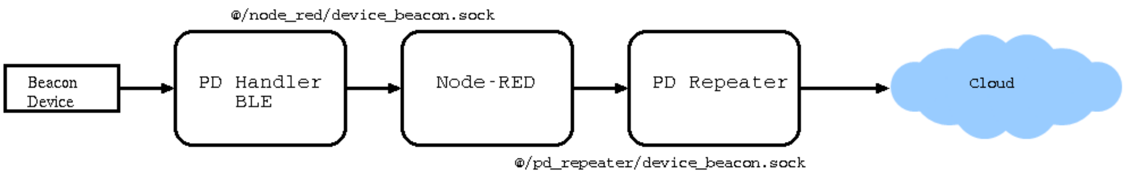
Name: filter out

Path: /pd_repeater/device_beacon.sock

Abstract Flag: ✓

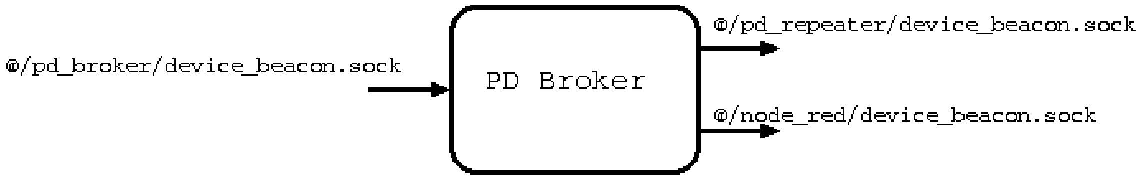
Put the ipc-out node in place, set the Path in Property to "/pd_repeater/device_beacon.sock" and check the Abstract Flag check box.

With the above settings, the PD Handler BLE will write data to "@/node-red/device_beacon.sock" while Node-RED will wait for data at "@/node-red/device_beacon.sock" and write them to "@/pd_repeater/device_beacon.sock".

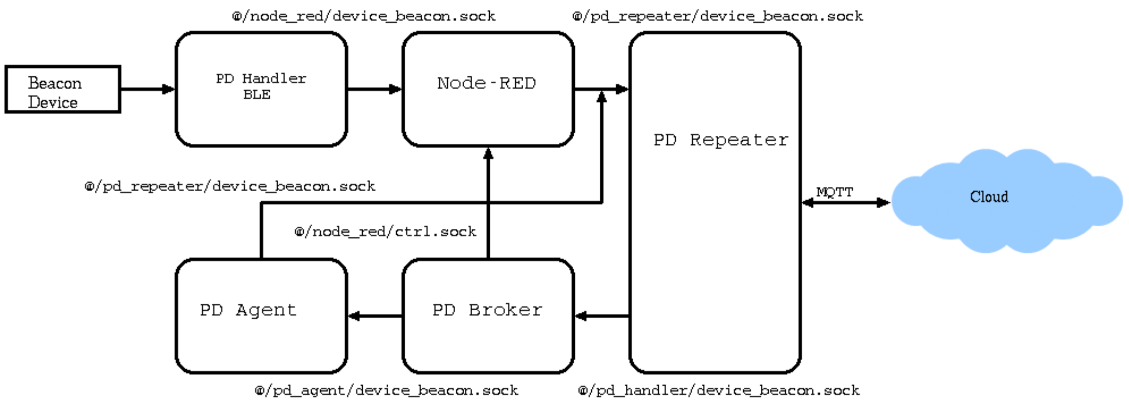

@/node_red/device_beacon.sock
Beacon Device
PD Handler BLE
Node-RED
PD Repeater
Cloud
@/pd_repeater/device_beacon.sock

3-4-1. PD Broker

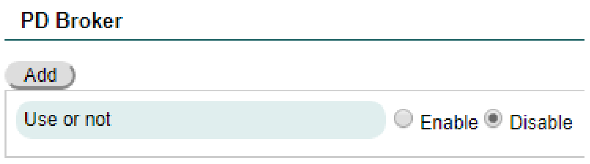
PD Broker is an application to customize connection configurations to send data received from a single UNIX domain socket to multiple UNIX domain sockets.


@/pd_broker/device_beacon.sock
PD Broker
@/pd_repeater/device_beacon.sock
@/node_red/device_beacon.sock

For example, in the configuration shown below, the PD Handler BLE receives beacon signals, which are then filtered by Node-RED and sent to cloud in MQTT connections via the PD Repeater. PD Broker is used to receive control messages using the same MQTT connection and control the filtering parameter of Node-RED and PD Agent.


@/node_red/device_beacon.sock
@/pd_repeater/device_beacon.sock
Beacon Device
PD Handler BLE
Node-RED
PD Repeater
MQTT
Cloud
@/pd_repeater/device_beacon.sock
@/node_red/ctrl.sock
PD Agent
PD Broker
@/pd_agent/device_beacon.sock
@/pd_handler/device_beacon.sock

It is possible to set the PD Broker from the **[IoT Data]-[PD Broker]** tab. If no the **[PD Broker]** tab is displayed, refer to "Chapter 2-3 Application setup for IoT data control," and set To be use to "Enable."


PD Broker

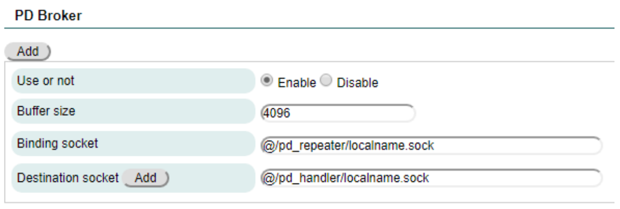
Add

Use or not: Enable [✓ Disable]

The figure on the left shows the default screen.

When "Enable" is selected, setting items will be displayed.

Selecting "Enable" will show the screen as below:


PD Broker

Add

Use or not	[✓ Enable] Disable
Buffer size	4096
Binding socket	@/pd_repeater/localname.sock
Destination socket Add	@/pd_handler/localname.sock

Add: Click this button to set multiple waiting sockets. Input forms for a second socket and beyond will be displayed.

Buffer size: Sets maximum size of data in bytes.

Binding socket: Sets pathname of the UNIX domain socket to wait for data. If "@" is appended as a prefix, it will indicate an abstract domain name.

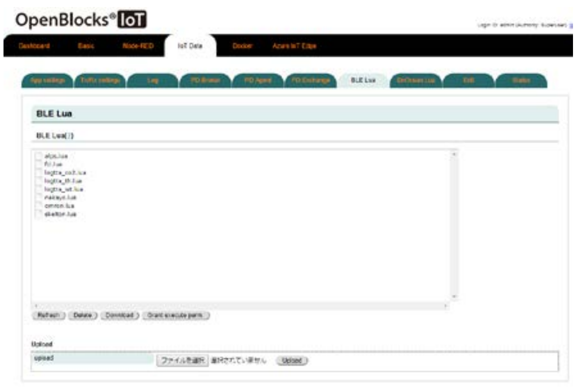
Destination socket: Sets pathname of the UNIX domain socket to send data. If "@" is appended as a prefix, it will indicate an abstract domain name.

Click the Add button to display input forms for a second socket and beyond.

3-5. Lua expansion

3-5-1. BLE Lua

The C-language version of the PD Handler BLE (pd-handler-ble-c) can support unsupported non-connection type BLE sensors.
This is implemented by uploading a Lua file for the sensor to be supported by pd-handler-ble-c from the **[IoT Data]-[BLE Lua]** tab.
*Rebooting the process is required following upload.



Refresh: Refresh a list of files to upload the Lua file.

Delete: Select a file and press this button to delete file.

Download: Enables user to download a selected file.

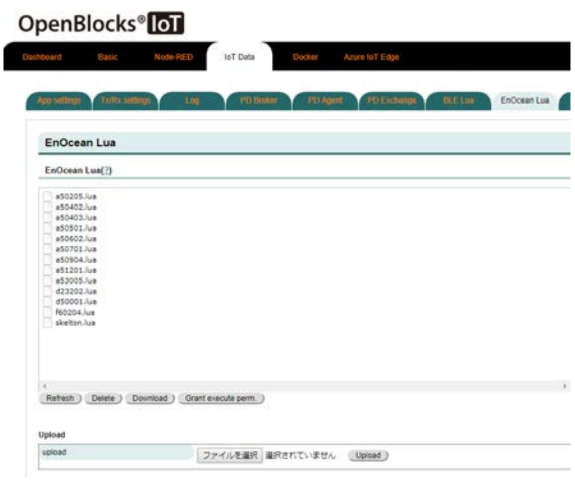
Grant execute perm.: Gives a right of execution to selected file. Not required in normal operation.

Upload: Press this button after choosing a file to be uploaded in the dialog to upload file.

If any file apart from a Lua file is present at the upload destination, this process will not operate normally. For this reason, refer to skelton.lua when creating the content of a file.

3-5-2. EnOcean Lua

The EnOcean function of PD Handler UART can support unsupported EnOcean devices. This is implemented by uploading a Lua file for the sensor to be supported from the **[IoT Data]-[EnOcean Lua]** tab.
*Rebooting the process is required following upload.



a50205.lua
a50402.lua
a50403.lua
a50501.lua
a50602.lua
a50701.lua
a50904.lua
a51201.lua
a53005.lua
d23202.lua
d50001.lua
f60204.lua
skelton.lua

Refresh: Refresh a list of files to upload the Lua file.

Delete: Select a file and press this button to delete file.

Download: Enables user to download a selected file.

Grant execute perm.: Gives a right of execution to selected file. Not required in normal operation.

Upload: Press this button after choosing a file to be uploaded in the dialog to upload file.

If any file apart from a Lua file is present at the upload destination, this process will not operate normally. For this reason, refer to skelton.lua when creating the content of a file.

Chapter 4 Supplementary information

4-1. Amount of transmitted data and line speed

If data transmission is slow against the information acquired from beacons and devices, the amount of information in the buffer within the OpenBlocks IoT Family will increase. In this case, unless an improvement is made to the data transmission section, data will continue being accumulated. Therefore, check the buffer data and adjust intervals, data acquisition time intervals, etc.
*For buffer data, check the buffer file size from the Status tab in Service tab.

4-2. Write data format to PD Repeater

PD Repeater is assumed to handle data in a JSON format, but depending on a connection destination, it may handle character strings not in a JSON format.

Cloud	Non-JSON format support
PD Exchange	
MQTT server	○
Watson IoT for Device	
Amazon Kinesis	○
WEB server	
AWS IoT	
MS Azure Event hubs	○
Watson IoT for Gateway	
MS Azure IoT Hub	○
Toami for DOCOMO	
Domain socket	○
KDDI IoT Cloud Standard	
IoT device hub	
Web server of Plat'Home's original specifications	
Google IoT Core	○
TCP	○

Destination that can support non-JSON format strings

4-3. Data write size to PD Repeater

Though it is possible to change the data write size to the PD Repeater for individual devices, the larger this size is, the more memory will be consumed, thereby deteriorating performance. It is recommended to keep the size as small as possible, depending on data size.
As each cloud service has different maximum values specified, check the specifications of cloud used and make this setting.

4-4. Buffer size of PD Repeater

The PD Repeater temporarily accumulates data as a transmission buffer and writes data in DB as a buffer. The default upper limit of DB size is 16 Mbytes. When this is exceeded, new data will be dumped, and data will not be received until the DB size is reduced to 8 Mbytes or less.

4-5. Resending at the time of PD Repeater error

Depending on communication conditions of a network, transmission from the PD Repeater to cloud may fail. In this case, if four failures in a row have occurred or an unexpected error condition has taken place, a resending process will start in one minute.

4-6. About the configuration file of an independently developed application

A function to create a configuration file (Config) of an independently developed application by the user is not available. Therefore, the user must save this file for each unit. Please use the file upload function, etc.

4-7. Method to send data from PD Repeater to Node-RED

Data is sent from the PD Repeater to Node-RED by way of a UNIX domain socket.
The path of a UNIX domain socket to which the PD Repeater writes individual data is as follows:

```
<Socket path prefix>/<device number>.sock
```

By appending "@" as a prefix for a path, it will be handled as an abstract domain socket. (Refer to Chapter 2-4-2-17)

On the Node-RED side, prepare an ipc-in node of the subject device for inputting in the flow. The ipc-in node installed in the OpenBlocks IoT Family is expanded to deal with an abstract domain socket.
For pathname, set it to the "Path" of the property of the ipc-in node. To deal with the pathname as an abstract domain socket, check the Abstract Flag check box in the same property. Different from the PD Repeater, there is no need to appendix "@" as a prefix for the path.

4-8. Beacon added as a BLE device

Sensors and beacons registered as BLE devices and setup as destinations by WEB UI are handled individually.
In this case, beacon devices in particular have no dependence on the transmission settings of beacons. Therefore, control type, data filter, etc. of beacon transmission settings will not be applied.
In addition, data to be sent from beacons with supported sensors to the PD Repeater will be analyzed sensor data. Normal beacons, however, will send time, device ID and note information to the PD Repeater.

4-9. Data transmission to web server

From the PD Repeater (firmware of OpenBlocks IoT), transmission is made using a HTTP POST method to the endpoint of a designated URL.
For this reason, the HTTP server must return a status code in HTTP 2xx.
No HTTP header or payload requirement exists when a status code in HTTP 2xx is returned.
If any status other than HTTP 2xx is returned, the PD Repeater (OpenBlocks IoT firmware) will handle it as an error.

4-10. Handler config user settings

Use the [IoT data]-[App settings] tab, set the **Handler config settings** to "User-defined configuration" and save it to switch to user-defined configuration.
To edit the file, choose the Edit tab in the IoT Data tab.
However, using this function is not in the scope of our support.

4-11. Web server of Plat'Home's original specifications (PD Web)

This is a web server whose specifications are independently defined by Plat'Home to realize bidirectional communication using HTTP.

4-11-1. Overview of PD Web

- PD Web offers bidirectional communication with HTTP.
  - Downstream payload transfer is provided as a response message against upstream POST.
  - If no payload to be sent upstream exists, the PD Repeater will make empty connections at an interval, as specified in the Polling interval setting.
  - If no payload to be sent downstream exists, the web server only returns the response header. If a response message is not empty, the PD Repeater transfers the payload to the UNIX domain socket of the downstream module designated by the push_to key.
- The authentication method of the PD Web is a bidirectional token authentication.
  - PD Repeater and web server has ID and key information assigned to each device.
  - As token for request, the PD Repeater uses a hash value (signature) that is a

    character string (for request) subject to signature, constituted by version number (version number of the PD Web specification), ID, time stamp, payload's hash value (MD5) and signed by the key.
  - ➢ As token for response, the web server uses a hash value (signature) that is a character string (for response), constituted by version number, ID, time stamp, payload's hash value (MD5) plus the token of the request header and signed by the key.
  - ➢ If the signature of the response header or MD5 of the payload does not match the expected value, the PD Repeater will make the payload empty until the expected signature is returned to the response header.
  - ➢ It is also possible to use BASIC authentication in bulk for web server at the same time.
- Error handling
  - ➢ Error handling between the PD Repeater and web sever is performed with HTTP status codes only. If the status is not in a range of 200 and 299, a payload transfer to downstream modules will not be made, regardless of the content of response header.
- Payload format and transaction control
  - ➢ The format of payload in the upstream direction is a JSON character string. To transfer multiple messages together, regardless of the number of messages, the top layer will be an array.
  - ➢ The message format of the PD Repeater in the downstream direction is shown in Chapter 2-5-2, but a payload format is not defined. The payload format in the downstream direction depends on specifications of the downstream module to receive payload from the PD Repeater.
  - ➢ Transaction (arrival and resend process) control of payload is assumed to be performed between web server and downstream modules. PD Repeater does not control transactions.
  - ➢ For the purpose of transaction control applications, PD Handler Modbus and PD Agent offered by Plat'Home has a specification to return the hash value (MD5) of the payload in the downstream direction, as the value of the reply_to key in a JSON character string.

4-11-2. HTTP header of PD Web

Below is a list of HTTP headers specified by PD Web.

HTTP header	Description
X-Pd-Web-Version	Version number of PD Web specifications
X-Pd-Web-Id	Client ID
X-Pd-Web-Time	RFC3339-compliant time stamp
X-Pd-Web-Md5	Hash value
X-Pd-Web-Signature	Hash value created by header information and key
Content-Type	application/json;charset=UTF-8

```
Host: 127.0.01
Accept: */*
X-Pd-Web-Version: 1.0
X-Pd-Web-Id: pd_web_02
X-Pd-Web-Time: 2017-09-01T18:11:01.101+09:00
X-Pd-Web-Md5: 26b32b7bc6b4587c2ded48128e809b08
X-Pd-Web-Signature: c91279bc0d9896745e4e12e73917aafd56c017966a89375273ba4b9be8b02096
Content-Length: 190
Content-Type: application/json;charset=UTF-8
```

Below is an example of response header to PD Repeater.

```
HTTP/1.1 200 OK
Date: Fri, 01 Sep 2017 08:34:35 GMT
Server Apache/2.4.27 (Unix)
X-Powered-By: PHP/5.6.31
X-Pd-Web-Version: 1.0
X-Pd-Web-Id: pd_web_03
X-Pd-Web-Time: 2017-09-01T17:34:35.000+09:00
X-Pd-Web-Md5: d41d8cd98f00b204e9800998ecf8427e
X-Pd-Web-Signature: b4e23876e866e9225e2f83cbd1df8b6387fe5da9e160b222953464b1ae87a9d3
Content-Length: 0
Content-Type: application/json;charset=UTF-8
```

4-11-3. Token of PD Web

The token of request header created by the PD Repeater (X-Pd-Web-Sinature) can be reproduced at the web server with the following PHP script:

```
$hash_hmac_data =
$_SERVER['HTTP_X_PD_WEB_VERSION'] .
$_SERVER['HTTP_X_PD_WEB_ID'] .
$_SERVER['HTTP_X_PD_WEB_TIME'] .
$_SERVER['HTTP_X_PD_WEB_MD5'];
$signature = hash_hmac ('sha256', $hash_hmac_data, $key, false);
```

Here, "$key" is a key stored in web server in advance to be a pair with "$_SERVER['HTTP X PD WEB ID'] ". It is authenticated by comparing reproduced "$signature" with "$_SERVER['HTTP X PD WEB SIGNATURE'] ".

The token of response header (X-Pd-Web-Sinature) is created at the web server with the following PHP script:

```
$tm = localtime();
$timestamp = sprintf("%04d-%02d-%02dT%02d:%02d:%02d.000+09:00",
$tm[5]+1900,$tm[4]+1,$tm[3],$tm[2],$tm[1],$tm[0]);
$hash_hmac_data = '1.0' . $_SERVER['HTTP_X_PD_WEB_ID'] . $timestamp .
md5($payload) . $_SERVER['HTTP_X_PD_WEB_SIGNATURE'];
$signature = hash_hmac ('sha256', $hash_hmac_data, $key, false);
```

Here, if no payload character string or character string to be sent (control message in the downstream direction) is present, $payload will be set as $payload="";"
Note that different from token of request header, the character string to be signed contains the request header token sent from the PD Repeater ($_SERVER['HTTP X PD WEB SIGNATURE']).

4-11.4. Installation example of web server (PHP script)

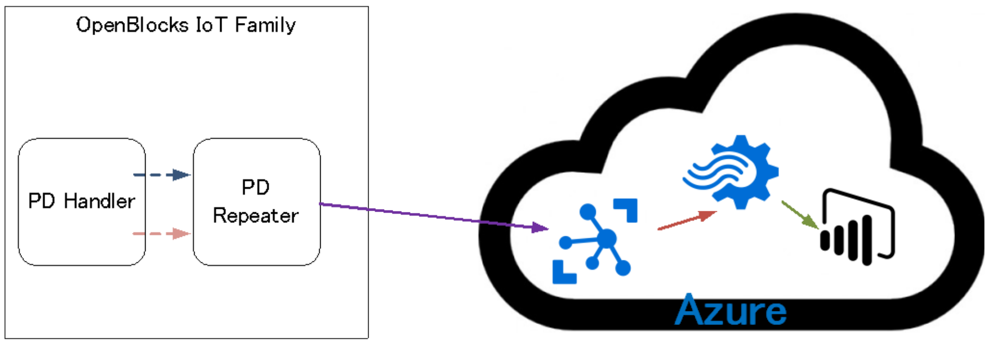
Shown below is an example of an installation of web server (PHP script)

```
<?php
/*
  * This PHP script is a sample of PHP script on the server side.
  * to use PD Web of PD Repeater.
  *
  * To enable this script to function,
  * SQLite3 database ($db_file in the script) created by the following SQL syntax is required.
  *
  *       CREATE TABLE client (id TEXT, key TEXT, flags INTEGER, payload BLOB, md5 TEXT);
  *       CREATE INDEX index_client ON client (id);
  *
  * Here id is the PD Web ID set for each sensor device,
  * key is the key to create a token, and content is a JSON character string (payload)
  * to be sent to individual sensor devices via the PD Repeater,
  * md5 is the MD5 hash value of the JSON character string.
  * Flags is a code to indicate the payload's transmission status:
  * 0: Transmitted, 1: In process and 2: Not transmitted.
  *
  * Shown below is an example of settings of initial values.
  *
  *       INSERT INTO client(id, key, flags, payload, md5)
  *                       VALUES('id00', 'key00', 0, '{"any_key":"any_value"}',
'd29e8a13452e5bc5218d9df7e6ea991f');"
  *       INSERT INTO client(id, key, flags, payload, md5)
  *                       VALUES('id01', 'key10', 0, '{"any_key":"any_value"}',
'd29e8a13452e5bc5218d9df7e6ea991f');"
  *
  * Here d29e8a13452e5bc5218d9df7e6ea991 is MD5 value of '{"any_key": "any_value"}'.
  *   With Linux OS, this can be acquired with the following command.
  *
  *       echo -n '{"any_key":"any_value"}' | md5sum
  *
  * To send a payload, use UPDATE of SQL to change flags to 2: Not transmitted.
  *
  *       UPDATE client SET flags = 2 WHERE id = 'id00';
  *
* It is possible to naturally set a payload and MD5 value together.
  *
  *       UPDATE client
  *                       SET flags = 2,
  *                           payload = '{"any_key":"new_value"}',
  *                           md5 = '94f030ebd7bed4a5ee08fc6fa75ae64e'
  *                       WHERE id = 'id00';
  *
  * When transmission is completed, PHP script will update flag to 1,
  * and wait for a response payload containing "reply_to" key.
  *
  * Example of a response payload
  *
  *        {"reply_to":"94f030ebd7bed4a5ee08fc6fa75ae64e","result":"done"}
  *
  * A response payload is returned from (the handler of) each sensor device to receive a payload via the PD
Repeater.
  * PD Agent and PD Handler Modbus handles the MD5 value of payload using "reply_to" key.
  * To use a unique handler, can make it a unique response payload.
  *
```

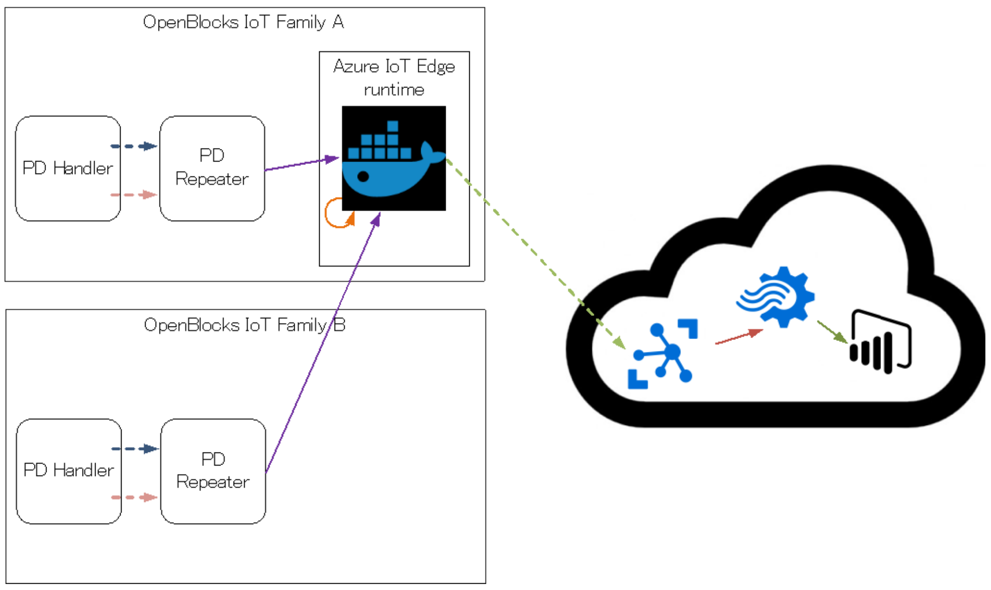
```
* *When a response payload is received, the PHP script will compare the md5 on the database
* with the value of "reply_to" and update flag to "0".
* *If no response payload is included in a received payload, the PHP script will change flag to "2"
* and resend it.
*
*/
    $dump_file = '/tmp/dump.txt';
    $db_file = '/tmp/pd_web.db';

    / * HTTP header confirmation * /
    if (!(isset($_SERVER['HTTP_X_PD_WEB_VERSION']) &&
            isset($_SERVER['HTTP_X_PD_WEB_ID']) &&
            isset($_SERVER['HTTP_X_PD_WEB_TIME']) &&
            isset($_SERVER['HTTP_X_PD_WEB_MD5']) &&
            isset($_SERVER['HTTP_X_PD_WEB_SIGNATURE']))) {
        /* If HTTP header is invalid, returns Bad Request 400. */
        http_response_code (400);
        exit;
    }

    /* SQLite3 database reading */

    /* As this code is a sample, countermeasures against SQL injection are omitted.
       To use this in actual operations, sufficiently conduct validation
       of $_SERVER['HTTP_X_PD_WEB_ID']. */

    $db = new SQLite3($db_file);
    $query = sprintf("SELECT key, flags, payload, md5 FROM client WHERE id = '%s';",
        $_SERVER['HTTP_X_PD_WEB_ID']);
    $results = $db->query($query);

    if(! $results) {
        /* If no HTTP_X_PD_WEB_ID exists, returns Unauthorized 401. */
        http_response_code (401);
        $db->close();
        exit;
    }
    else {
        $row = $results->fetchArray();
        $key = $row['key'];
        $flags = $row['flags'];
        $payload_tx = $row['payload'];
        $md5_tx = $row['md5'];
    }

    /* Create a signature by using a given character string in HTTP request header
         and key on the database. */

    $hash_hmac_data =
        $_SERVER['HTTP_X_PD_WEB_VERSION'] .
        $_SERVER['HTTP_X_PD_WEB_ID'] .
        $_SERVER['HTTP_X_PD_WEB_TIME'] .
        $_SERVER['HTTP_X_PD_WEB_MD5'];

    $signature = hash_hmac ('sha256', $hash_hmac_data, $key, false);

    /* Settings of common part of HTTP response header */
    date_default_timezone_set('Asia/Tokyo');
    $tm = localtime();
    $timestamp = sprintf("%04d-%02d-%02dT%02d:%02d:%02d.000+09:00",
        $tm[5]+1900,$tm[4]+1,$tm[3],$tm[2],$tm[1],$tm[0]);

    header('Pd_Web_Version: 1.0');
    header('Pd_Web_Id: ' . $_SERVER['HTTP_X_PD_WEB_ID']);
    header('Pd_Web_Time: ' . $timestamp);
    header('Content-Type: application/json;charset=UTF-8');

    /* It should be noted that while the string of a request header to be signed is constituted by
VERSION.ID.TIME.MD5,
            the string of response header to be signed is constituted by
VERSION.ID.TIME.MD5.SIGNATURE
      (SIGNATURE is a string contained in request header). */

    if ($signature != $_SERVER['HTTP_X_PD_WEB_SIGNATURE']) {
        /* If HTTP_X_PD_WEB_SIGNATURE and signature do not match,
           401 Unauthorized will be returned. */
        header('Pd_Web_Md5: ' . md5(''));
        $hash_hmac_data = '1.0' . $_SERVER['HTTP_X_PD_WEB_ID'] . $timestamp .
            md5('') . $_SERVER['HTTP_X_PD_WEB_SIGNATURE'];
        $signature = hash_hmac ('sha256', $hash_hmac_data, $key, false);
        header('Pd_Web_Signature: ' . $signature);
        http_response_code (401);
        $db->close();
        exit;
    }

    $payload = file_get_contents("php://input");

    if(md5($payload) != $_SERVER['HTTP_X_PD_WEB_MD5']) {
        /* If the MD5 value of payload does not match HTTP_X_PD_WEB_MD5,
           406 Not Acceptable will be returned. */
        header('Pd_Web_Md5: ' . md5(''));
        $hash_hmac_data = '1.0' . $_SERVER['HTTP_X_PD_WEB_ID'] . $timestamp .
            md5('') . $_SERVER['HTTP_X_PD_WEB_SIGNATURE'];
        $signature = hash_hmac ('sha256', $hash_hmac_data, $key, false);
        header('Pd_Web_Signature: ' . $signature);
        http_response_code (406);
        $db->close();
        exit;
    }

    /* The reception payload and the HTTP request header is stored in dump_file. */
    $fp = fopen($dump_file, 'a+');
    fputs($fp, $_SERVER['HTTP_X_PD_WEB_VERSION']);
    fputs($fp, "¥n");
    fputs($fp, $_SERVER['HTTP_X_PD_WEB_ID']);
    fputs($fp, "¥n");
    fputs($fp, $_SERVER['HTTP_X_PD_WEB_TIME']);
    fputs($fp, "¥n");
    fputs($fp, $_SERVER['HTTP_X_PD_WEB_MD5']);
    fputs($fp, "¥n");
    fputs($fp, $_SERVER['HTTP_X_PD_WEB_SIGNATURE']);
    fputs($fp, "¥n");
    fputs($fp, $_SERVER['CONTENT_LENGTH']);
    fputs($fp, "¥n");
    fputs($fp, $payload);
    fputs($fp, "¥n");
    fputs($fp, "¥n");
    fclose( $fp );

    /* If flags are "1" (payload in process present),
         the value of "reply_to" key will be acquired from reception payload. */
    if($flags === 1) {
        $json = mb_convert_encoding($payload, 'UTF8', 'ASCII,JIS,UTF-8,EUC-JP,SJIS-WIN');
        $array = json_decode($json,true);
        if ($array !== NULL) {
            /* Reception payload is an array of JSON objects. */
            $payload_count = count($array);
            for($i=0;$i<$payload_count;$i++) {
                if (isset($array[$i]['reply_to'])) {
                    /* Compare the value of "reply_to" key and md5_tx */
                    if($array[$i]['reply_to'] === $md5_tx) {
                            /* If matched, flags will be set to "0". */
                        $flags = 0;
                    }
                    else {
                            /* If not matched, flags will be "2". */
                        $flags = 2;
                    }
                }
                break;
            }
            if($i == $payload_count) {
                /* If "reply_to" key does not exit, flags will be set to " 2". */
                $flags = 2;
            }
        }
        else {
              /* If not a JSON character string, flags will be set to " 2". */
            $flags = 2;
        }

        /* Update flags of database. */
        $query = sprintf("UPDATE client SET flags = %d WHERE id = '%s';" ,
            $flags, $_SERVER['HTTP_X_PD_WEB_ID']);
        $results = $db->query($query);
    }

    if($flags == 2) {
        /* If flags is "2" (with transmission payload), payload_tx will be sent, and "200 OK" will be returned.
*/
        header('Pd_Web_Md5: ' . md5($payload_tx));
        $hash_hmac_data = '1.0' . $_SERVER['HTTP_X_PD_WEB_ID'] . $timestamp .
            md5($payload_tx) . $_SERVER['HTTP_X_PD_WEB_SIGNATURE'];
        $signature = hash_hmac ('sha256', $hash_hmac_data, $key, false);
        header('Pd_Web_Signature: ' . $signature);
        echo $payload_tx;
        http_response_code (200);

        /* flags will be changed to "1" (payload in process present), and flags in the database will be
updated. */
        $flags = 1;
        $query = sprintf("UPDATE client SET flags = %d WHERE id = '%s';" ,
            $flags, $_SERVER['HTTP_X_PD_WEB_ID']);
        $results = $db->query($query);
    }
    else {
        /* Returns "200 OK". */
        header('Pd_Web_Md5: ' . md5(''));
        $hash_hmac_data = '1.0' . $_SERVER['HTTP_X_PD_WEB_ID'] . $timestamp .
            md5('') . $_SERVER['HTTP_X_PD_WEB_SIGNATURE'];
        $signature = hash_hmac ('sha256', $hash_hmac_data, $key, false);
        header('Pd_Web_Signature: ' . $signature);
        http_response_code (200);
    }

    $db->close();
    exit;
?>
```

4-12. Data transmission to IoT Hub/IoT Edge

When transmission from the PD Repeater to Azure IoT Hub is conducted, it is made from the host machine (OpenBlocks) via the Internet. For this reason, if filtering is made on cloud side only to obtain data to be used, data to be transmitted must be narrowed down in the host machine (in OpenBlocks) or cooperate with a different service in Azure in order to perform processing.



To transmit from the PD Repeater to Azure IoT Edge, data is transmitted from the host machine (OpenBlocks) to a container within the host or another machine in which Azure IoT is run. Azure IoT Edge can process data by running a container to filter data. In addition, in normal operations, it supports operations in an offline environment and data transmission to IoT Hub is not always required. (To download container or change configuration, communication with IoT Hub is required).



OpenBlocks IoT Family Data Handling Guide

Version 3.1.0 (Jun 4, 2018)

Plat'Home Co., Ltd.

NIHON BUILDING KUDANBEKKAN, 3F

4-2-3, Kudankita, Chiyoda-ku, TOKYO 102-0073, JAPAN