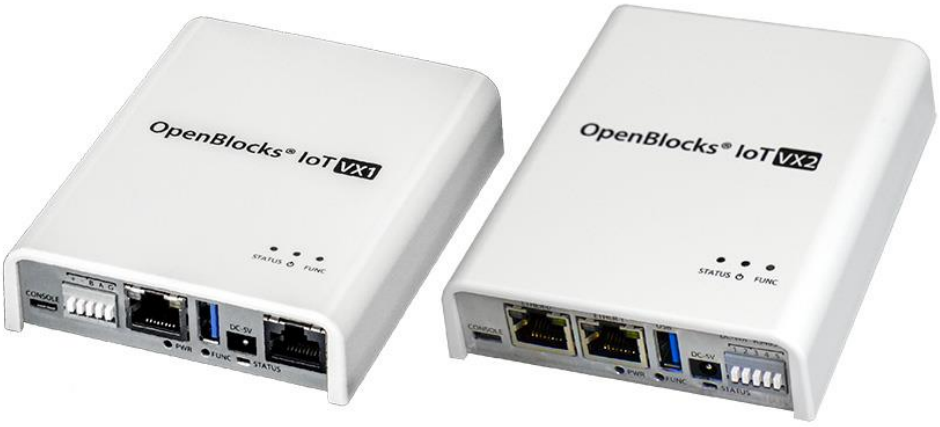
Plat' Home
TECHNOLOGY to serve you.

OpenBlocks IoT Family向け
データハンドリングガイド



Ver.3.0.0

ぷらっとホーム株式会社

■ 商標について

・ 文中の社名、商品名等は各社の商標または登録商標である場合があります。
・ その他記載されている製品名などの固有名詞は、各社の商標または登録商標です。

■ 使用にあたって

・ 本書の内容の一部または全部を、無断で転載することはご遠慮ください。
・ 本書の内容は予告なしに変更することがあります。
・ 本書の内容については正確を期するように努めていますが、記載の誤りなどにご指摘がございましたら弊社サポート窓口へご連絡ください。
  また、弊社公開の WEB サイトにより本書の最新版をダウンロードすることが可能です。
・ 本装置の使用にあたっては、生命に関わる危険性のある分野での利用を前提とされていないことを予めご了承ください。
・ その他、本装置の運用結果における損害や逸失利益の請求につきましては、上記にかかわらずいかなる責任も負いかねますので予めご了承ください。

目次

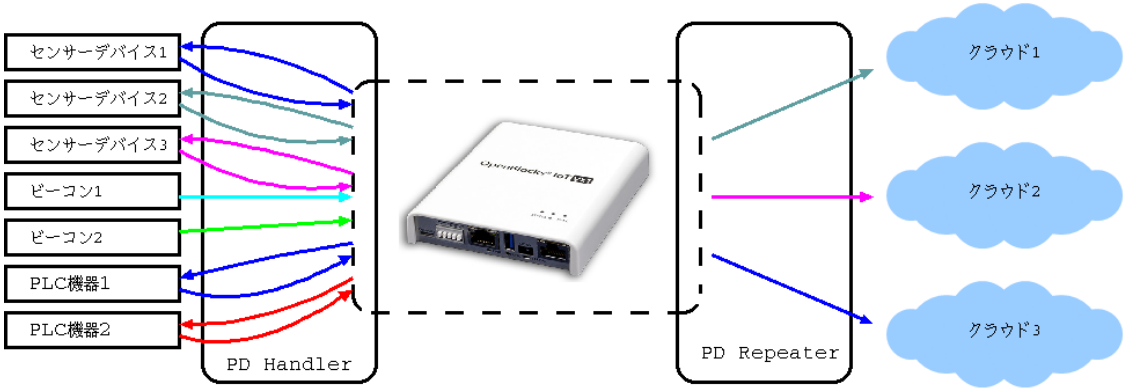
第 1 章 はじめに

本書は、OpenBlocks IoT Family にて用いているデータハンドリング機能について解説しています。本設定には、WEB ブラウザが使用可能なクライアント装置(PC やスマートフォン、タブレット等)が必要になります。また、WEB ユーザーインターフェース(以下、WEB UI)自体については『OpenBlocks IoT Family 向け WEB UI セットアップガイド』を参照してください。

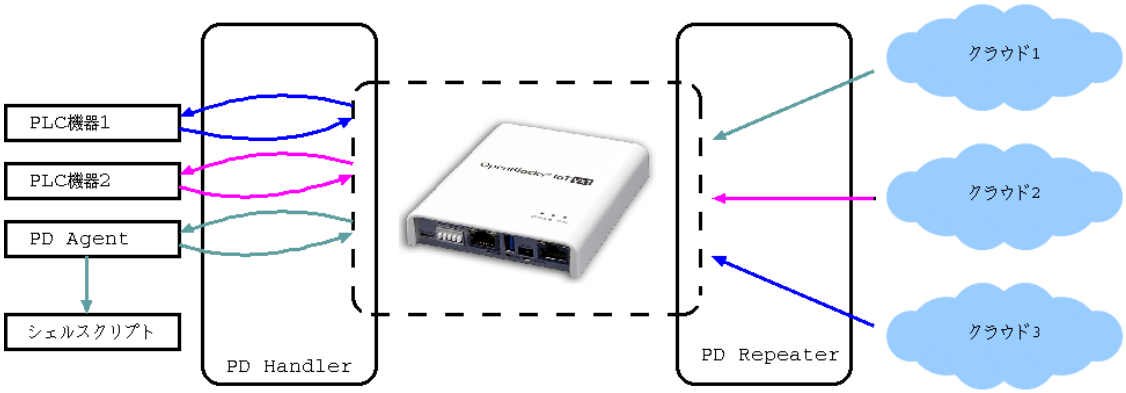
第 2 章 IoT データ制御機能

OpenBlocks IoT Family の IoT データ制御機能は、BLE・UART 等のセンサーデバイス並びに Modbus 対応 PLC 機器からデータを取得しクラウド等へ情報を送信する収集機能と、クラウド等からメッセージを受信し Modbus 対応 PLC 機器等を制御する下流方向の制御機能をサポートしています。センサーデバイス等のサポート状況については、弊社 WEB ページを参照してください。

収集機能は各デバイス等からデータを取得し、各送信先のクラウド等へ情報を送信します。データを一時バッファーとして OpenBlocks IoT Family 内に保存している為、ネットワーク障害等が発生しても、再送信が行える為データを安全に送信することが出来ます。



下流方向制御機能はクラウドから制御メッセージを受け取り、Modbus 対応 PLC 等を制御することが可能です。



IoT データ制御機能を使用するためには、サービス機能の基本設定と IoT データ制御パッケージのインストールが必要となります。

2-1. サービス機能の基本設定

IoT データ制御機能を使用するためには、WEB UI の「サービス」→「基本」タブより、サービス機能の基本設定として BT インターフェースの制御及び各種デバイスの登録等を行う必要があります。
サービス機能の基本設定については、「OpenBlocks IoT Family 向け WEB UI セットアップガイド」の第 5 章を参照して下さい。

2-2. IoT データ制御パッケージのインストール

IoT データ制御機能を使用するためには、WEB UI の「メンテナンス」→「機能拡張」タブより、IoT データ制御パッケージをインストールする必要があります。
※IoT データ制御は Node-RED と連携する機能があります。そのため、Node-RED との連携をする場合には、Node-RED についてもインストールを行ってください。

「メンテナンス」→「機能拡張」タブの「インストール機能」で「IoT データ制御」を選択し、「実行」をクリックします。IoT データ制御パッケージが既にインストールされている場合は、「インストール機能」の選択肢として「IoT データ制御」が表示されません。

2-3. IoT データ制御のアプリ設定

IoT データ制御パッケージがインストールされていると WEB UI の「サービス」→「基本」タブに「IoT データ」が表示されます。

「サービス」→「基本」タブより「IoT データ」リンクをクリックすると、ルートタブが IoT データ制御の設定を行う「ダッシュボード」／「基本」／「IoT データ」に切り替わります。
※「サービス」→「基本」タブに表示されるリンクは、「メンテナンス」→「機能拡張」タブよりインストールされたパッケージにより異なります。

WEB UI の「IoT データ」→「アプリ設定」タブより、使用するアプリケーションの起動制御とコンフィグ設定モードの設定を行います。

※「デフォルト使用アプリモード表示」チック時の表示例

「IoT データ」→「アプリ設定」タブの初期状態を左図に示します。

「デフォルト使用アプリモード表示」をチェックすると、「デフォルトアプリ起動制御」の設定メニューを表示します。各アプリケーションの起動制御を「使用する」に設定するためにはデフォルト側の設定が「使用する」に設定されている必要があります。

「コンフィグ設定モード表示」をチェックすると本 WEB UI により設定されるコンフィグを使用するかユーザー定義のコンフィグを使用するか切り替える「コンフィグ設定」メニューを表示します。

「一括有効」「一括無効」は、全ての設定を一括して「使用する」もしくは「使用しない」に設定します。

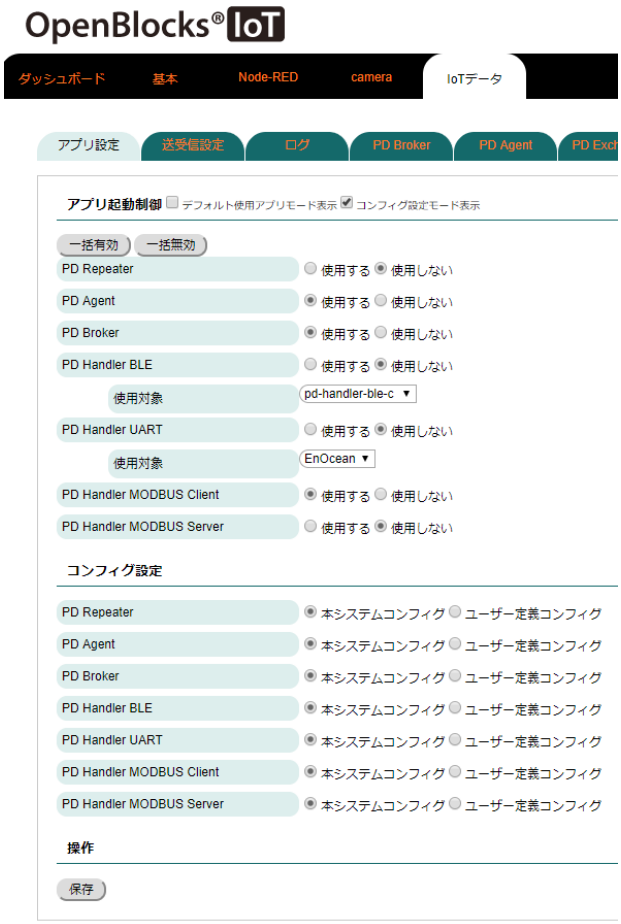
PD Repeater：クラウドに対しデータもしくは制御メッセージを送受するアプリケーションです。

PD Agent：PD Reperter を介しクラウドから制御メッセージを受け設定されたシェルスクリプトもしくは実行オブジェクトを実行するアプリケーションです。

PD Broker：データもしくは制御メッセージを複数のアプリケーションに分配するアプリケーションです。

PD Handler BLE：BT デバイスセンサーもしくはビーコンからデータを受け取るアプリケーションです。「使用対象」として C 言語版と Java スクリプト版のいずれかを選択できます。C 言語版の方がハイパフォーマンスですがビーコン及び非コネクション型 BLE センサーのみの対応となります。

PD Handler UART： URAT 接続のデバイス



※「コンフィグ設定モード表示」チック時の表示例

からデータを受け取るアプリケーションです。「使用対象」は EnOcean をサポートしています。)

PD Handler MODBUS Client：MODBUS プロトコルを用いて PLC 機器のレジスタを読み書きするアプリケーションです。

PD Handler MODBUS Server：PLC 機器から MODBUS プロトコルによる接続を待ち受けるアプリケーションです。

2-4. 送受信設定（PD Repeater）

WEB UI の「IoT データ」→「送受信設定」タブより、データの送信先もしくは制御メッセージの受信先の設定を行います。

送受信設定には、送受信先毎に「送受信設定」メニューより設定される個々のデバイスに依存しない設定と各デバイス設定メニューより設定されるデバイス毎の設定があります。

2-4-1.送受信設定

「IoT データ」→「送受信設定」タブの「送受信設定」メニューより、送受信先の選択と送受信設定の内個々のデバイスに依存しない設定を行います。本体内(lo)を除き最大４つの送受信先を選択することができます。

本体内(lo)：BLE デバイスの JSON テーブルと温度／湿度グラフを「IoT データ」→「データ表示」タブに出力します。

PD Exchange(pd_ex)：送受信先として PD Exchange を選択します。

MS Azure IoT Hub(iothub)：送受信先として Microsoft Azure IoT Hub を選択します。

AWS IoT(awsiot)：送受信先として Amazon AWS IoT を選択します。

Google IoT Core(iotcore)：送受信先として Google IoT Core を選択します。

Watson IoT for Gateway(w4g)：送受信先として IBM Watson IoT for Gateway を選択します。

MS Azure Event hubs(eventhub)：送信先として Microsoft Azure Event hubs を選択します。

Amazon Kinesis(kinesis)：送信先として Amazon Kinesis を選択します。

Watson IoT for Device(w4d)：送受信先として IBM Watson IoT for Device を選択します。

IoT デバイスハブ(nf_dvhub)：送受信先としてニフティクラウド IoT デバイスハブを選択します。

Toami for DOCOMO(t4d)：送信先として NTT

docomo Toami for DOCOMO を選択します。

KDDI IoT クラウド Standard(kddi_std)：送信先として KDDI IoT クラウド Standard を選択します。

PH 社独自仕様 WEB サーバー(pd_web)：送受先として弊社独自仕様の WEB サーバーを選択します。

WEB サーバー(web)：送信先として汎用の WEB サーバーを選択します。

MQTT サーバー(mqtt)：送受信先として汎用の MQTT サーバーを選択します。

TCP(ltcp)：送受信先として汎用の TCP サーバーを選択します。

ドメインソケット(lsocket)：送信先として UNIX ドメインソケットを選択します。Node-RED パッケージがインストールされている場合のみ表示されます。

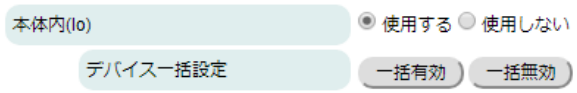
2-4-2.送受信先毎の設定

送受信先毎の「送受信設定」メニューにおける設定と各デバイス設定メニューにおける設定を説明します。

2-4-2-1.本体内(lo)

BLE デバイスの JSON テーブルと温度／湿度グラフを「IoT データ」→「データ表示」タブに出力します。本体内(lo)は、BLE デバイスのみに適用されます。

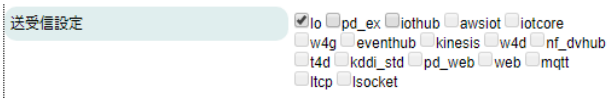
■送受信設定メニューにおける設定



「使用する」／「使用しない」以外の設定はありません。

デバイス一括設定：BLE デバイス情報送信設定において送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

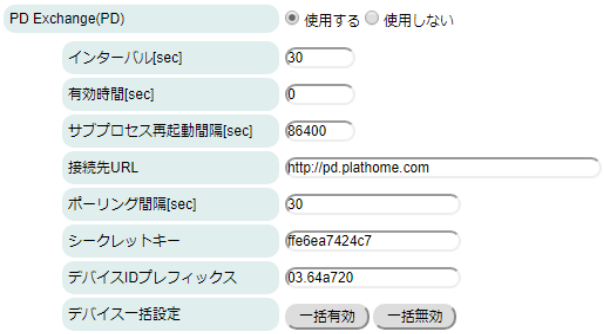
■BLE デバイス情報送信設定メニューにおける設定



「送受信設定」をチェックする以外の設定項目はありません。

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

■送受信設定メニューにおける設定



インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。　0 を指定した場合、データ送信が完了するまで保持し続けます。

サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。 通常デォルト値（86400 秒）から変更する必要はありません。　0 を指定した場合、再起動は行いません。

接続先 URL：接続先の PD Exchange の URL を設定します。

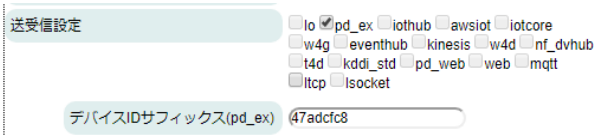
ポーリング間隔[sec]：PD Exchange から制御メッセージを読み出す間隔を設定します。

シークレットキー：PD Exchange のアカウントのシークレットキーを設定します。

デバイス ID プレフィックス：PD Exchange のアカウントに対するデバイス ID プレフィックスを設定します。

デバイス一括設定:各デバイス設定メニューにおいて送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

■各デバイス設定メニューにおける設定



デバイス ID サフィックス(pd_ex)：PD Exchange のデバイス ID のサフィックスを設定します。

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

■送受信設定メニューにおける設定

インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。　0 を指定した場合、データ送信が完了するまで保持し続けます。

サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。 通常デォルト値（86400 秒）から変更する必要はありません。　0 を指定した場合、再起動は行いません。

ドメイン名：接続先の IoT Hub のドメイン名を設定します。

IoT Hub 名：接続先の IoT Hub 名を設定します。

QoS：IoT Hub へデータを送信する際の MQTT プロトコルの QoS を設定します。

受信 QoS：IoT Hub から制御メッセージを受信する際の MQTT プロトコルの QoS を設定します。

詳細設定：MQTT プロトコルの詳細値を設定します。

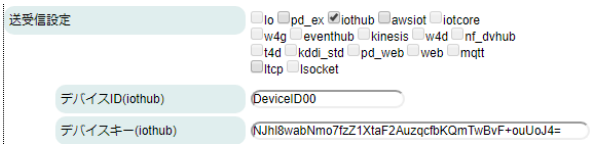
キープアライブ間隔[sec]：MQTT プロトコルのキープアライブインターバルを設定します。

クリーンセッション：MQTT プロトコルのクリーンセッションの使用／不使用を選択します。

Retain 機能：MQTT プロトコルの Retain 機能の使用／不使用を選択します。

デバイス一括設定:各デバイス設定メニューにおいて送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

■各デバイス設定メニューにおける設定



デバイス ID(iothub) : IoT Hub のデバイス ID を設定します。

デバイスキー(iothub) : IoT Hub のデバイスキーを設定します。

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

■送受信設定メニューにおける設定

インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。　0 を指定した場合、データ送信が完了するまで保持し続けます。

サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。　通常デォルト値（86400 秒）から変更する必要はありません。　0 を指定した場合、再起動は行いません。

送信先ホスト：接続先の AWS IoT のホスト名を設定します。

ポート番号：接続先の AWS IoT のポート番号を設定します。

QoS：AWS IoT へデータを送信する際の MQTT プロトコルの QoS を設定します。

受信 QoS：AWS IoT から制御メッセージを受信する際の MQTT プロトコルの QoS を設定します。

ルート証明書：AWS IoT へ接続する際のルート証明書ファイルのパス名を設定します。

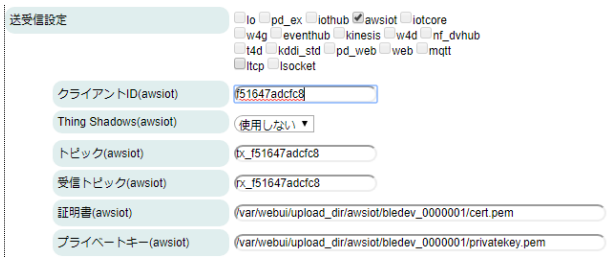
詳細設定：MQTT プロトコルの詳細値を設定します。

キープアライブ間隔[sec]：MQTT プロトコルのキープアライブインターバルを設定します。

Retain 機能：MQTT プロトコルの Retain 機能の使用／不使用を選択します。

デバイス一括設定:各デバイス設定メニューにおいて送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

■各デバイス設定メニューにおける設定



クライアント ID(awsiot)：AWS IoT のクライアント ID を設定します。

デバイスキー(awsiot)：IoT Hub のデバイスキーを設定します。

Thing Shadows(awsiot)：AWS IoT の Thing Shadows 機能の使用／不使用を選択します。

トピック(awsiot)：AWS IoT へデータを送る際に用いる MQTT プロトコルのトピックを設定します。

受信トピック(awsiot)：AWS IoT から制御メッセージを待ち受けるために用いる MQTT プロトコルのトピックを設定します。

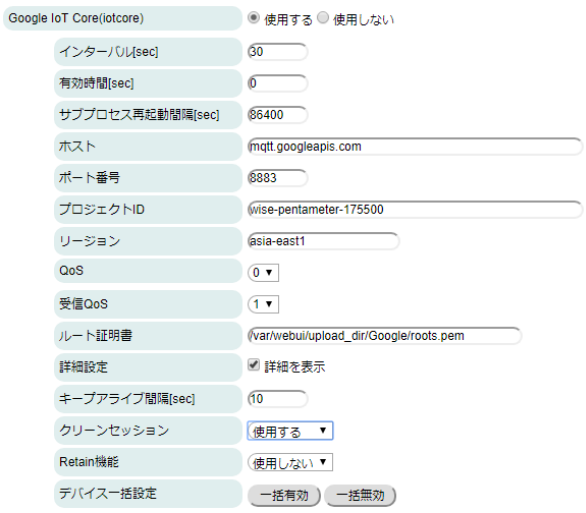
証明書(awsiot)：AWS IoT へ接続する際の証明書ファイルのパス名を設定します。

プライベートキー(awsiot)：AWS IoT へ接続する際のプライベートキーファイルのパス名を設定します。

※ルート証明書・証明書・プライベートキーは WEB UI の「システム」→「ファイル管理」タブにてアップロードしてください。

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

■送受信設定メニューにおける設定



インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。　0 を指定した場合、データ送信が完了するまで保持し続けます。

サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。　通常デォルト値（86400 秒）から変更する必要はありません。　0 を指定した場合、再起動は行いません。

ホスト：接続先の Google IoT Core のホスト名を設定します。

ポート番号：接続先の Google IoT Core のポート番号を設定します。

プロジェクト ID：Google IoT Core のプロジェクト ID を設定します。

リージョン：Google IoT Core のリージョンコードを設定します。

QoS：Google IoT Core へデータを送信する際の MQTT プロトコルの QoS を設定します。

受信 QoS：Google IoT Core から制御メッセージを受信する際の MQTT プロトコルの QoS を設定します。

ルート証明書：Google IoT Core へ接続する際のルート証明書ファイルのパス名を設定します。

詳細設定：MQTT プロトコルの詳細値を設定します。

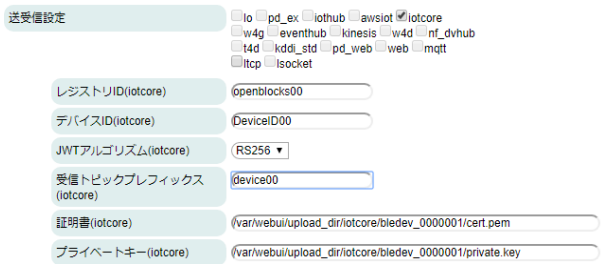
キープアライブ間隔[sec]：MQTT プロトコルのキープアライブインターバルを設定します。

クリーンセッション：MQTT プロトコルのクリーンセッションの使用／不使用を選択します。

Retain 機能：MQTT プロトコルの Retain 機能の使用／不使用を選択します。

デバイス一括設定:各デバイス設定メニューにおいて送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

■各デバイス設定メニューにおける設定



レジストリ ID(iotcore):Google IoT Core のクライアント ID を設定します。

デバイス ID(iotcore):Google IoT Core のデバイス ID を設定します。

JWT アルゴリズム(iotcore)：Google IoT Core への接続に用いる JWT アルゴリズムを選択します。

受信トピック(iotcore)：Google IoT Core から制御メッセージを待ち受けるために用いる MQTT プロトコルのトピックを設定します。

証明書(iotcore)：Google IoT Core へ接続する際の証明書ファイルのパス名を設定します。

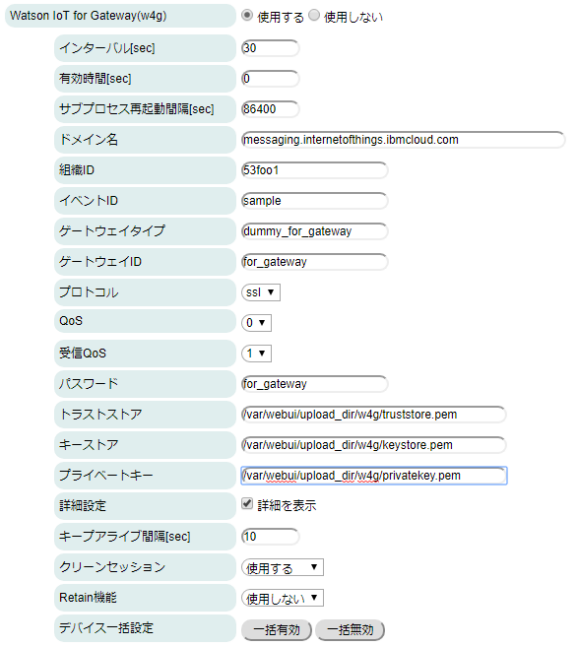
プライベートキー(iotcore)：Google IoT Core へ接続する際のプライベートキーファイルのパス名を設定します。

※ルート証明書・証明書・プライベートキーは WEB UI の「システム」→「ファイル管理」タブにてアップロードしてください。

※Google IoT Core 自体はβサービスとなっております。そのため、正式リリース時には仕様が変更となる恐れがあります。

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

■送受信設定メニューにおける設定



インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。　0 を指定した場合、データ送信が完了するまで保持し続けます。

サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。 通常デォルト値（86400 秒）から変更する必要はありません。　0 を指定した場合、再起動は行いません。

ドメイン名：接続先の Watson IoT for Gateway のドメイン名を設定します。

組織 ID：Watson IoT for Gateway の組織 ID を設定します。

イベント ID：Watson IoT for Gateway のイベント ID を設定します。

ゲートウェイタイプ：Watson IoT for Gateway のゲートウェイタイプを設定します。

ゲートウェイ ID：Watson IoT for Gateway のゲートウェイ ID を設定します。

プロトコル：接続に用いるプロトコル(TCP/SSL)を選択します。

QoS：Watson IoT for Gateway へデータを送信する際の MQTT プロトコルの QoS を設定します。

受信 QoS：Watson IoT for Gateway から制御メッセージを受信する際の MQTT プロトコルの QoS を設定します。

パスワード：Watson IoT for Gateway への接続に用いるパスワードを設定します。

トラストストア：Watson IoT for Gateway へ接続する際のルート証明書ファイルのパス名

を設定します。

キーストア：Watson IoT for Gateway へ接続する際のクライアント証明書ファイルのパス名を設定します。

プライベートキー：Watson IoT for Gateway へ接続する際のプライベートキーファイルのパス名を設定します。

詳細設定：MQTT プロトコルの詳細値を設定します。

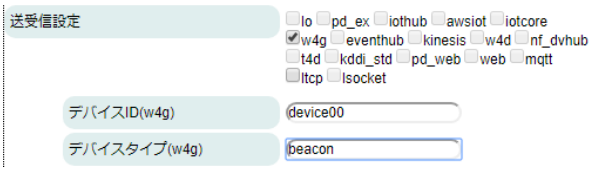
キープアライブ間隔[sec]：MQTT プロトコルのキープアライブインターバルを設定します。

クリーンセッション：MQTT プロトコルのクリーンセッションの使用／不使用を選択します。

Retain 機能：MQTT プロトコルの Retain 機能の使用／不使用を選択します。

デバイス一括設定:各デバイス設定メニューにおいて送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

■各デバイス設定メニューにおける設定



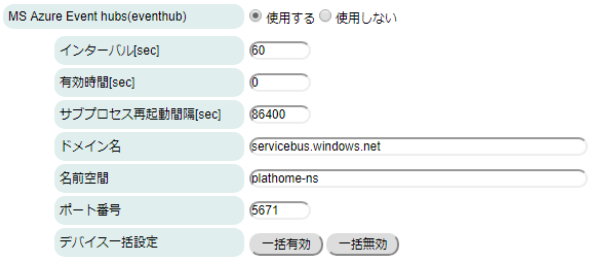
デバイス ID(w4g)：Watson IoT for Gateway のデバイス ID を設定します。

デバイスタイプ(w4g)：Watson IoT for Gateway のデバイスタイプを設定します。

※トラストストア・キーストア・プライベートキーは WEB UI の「システム」→「ファイル管理」タブにてアップロードしてください。

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

■送受信設定メニューにおける設定



インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。　0 を指定した場合、データ送信が完了するまで保持し続けます。

サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。　通常デォルト値（86400 秒）から変更する必要はありません。　0 を指定した場合、再起動は行いません。

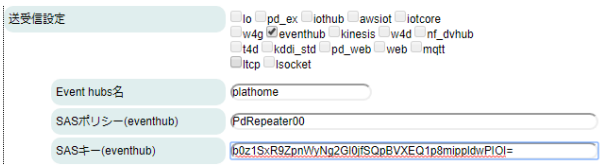
ドメイン名：接続先の Event hubs のドメイン名を設定します。

名前空間：Event hubs の名前空間を設定します。

ポート番号：送信先の Event hubs のポート番号を設定します。

デバイス一括設定:各デバイス設定メニューにおいて送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

■各デバイス設定メニューにおける設定



Event hubs 名(eventhub): Event hubs の hub 名を設定します。

SAS ポリシー(eventhub): Event hubs の SAS ポリシーを設定します。

SAS キー(eventhub)：Event hubs の SAS キーを設定します。

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

■送受信設定メニューにおける設定

インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。　0 を指定した場合、データ送信が完了するまで保持し続けます。

サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。　通常デォルト値（86400 秒）から変更する必要はありません。　0 を指定した場合、再起動は行いません。

ドメイン名：送信先の Amazon Kinesis のドメイン名を設定します。

リージョン：送信先の Amazon Kinesis のリージョンコードを設定します。

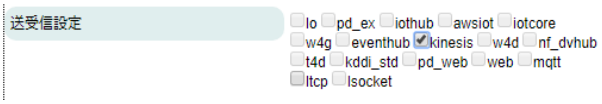
アクセス ID：Amazon Kinesis のアクセス ID を設定します。

アクセスキー：Amazon Kinesis のアクセスキーを設定します。

ストリーム名：Amazon Kinesis のストリーム名を設定します。

デバイス一括設定：各デバイス設定メニューにおいて送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

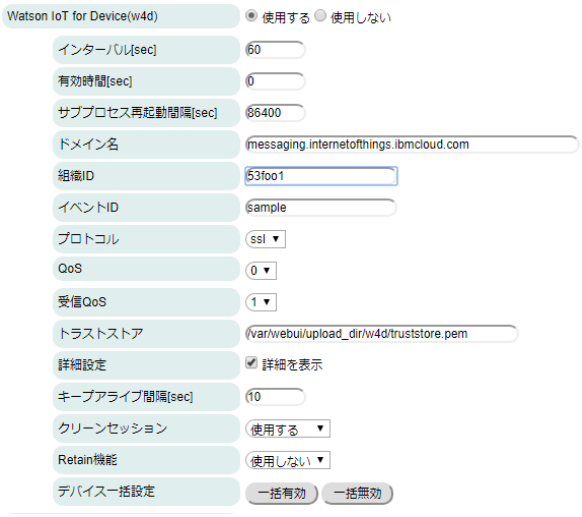
■各デバイス設定メニューにおける設定



「送受信設定」をチェックする以外の設定項目はありません。

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

■送受信設定メニューにおける設定



インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。　0 を指定した場合、データ送信が完了するまで保持し続けます。

サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。　通常デォルト値（86400 秒）から変更する必要はありません。　0 を指定した場合、再起動は行いません。

ドメイン名：接続先の Watson IoT for Device のドメイン名を設定します。

組織 ID：Watson IoT for Device の組織 ID を設定します。　Quickstart を使用する場合には、“quickstart”を設定してください。

イベント ID：Watson IoT for Device のイベント ID を設定します。

プロトコル：接続に用いるプロトコル(TCP/SSL)を選択します。

QoS：Watson IoT for Device へデータを送信する際の MQTT プロトコルの QoS を設定します。　Quickstart を使用する場合には、0 に設定する必要があります。

受信 QoS：Watson IoT for Device から制御メッセージを受信する際の MQTT プロトコルの QoS を設定します。

詳細設定：MQTT プロトコルの詳細値を設定します。

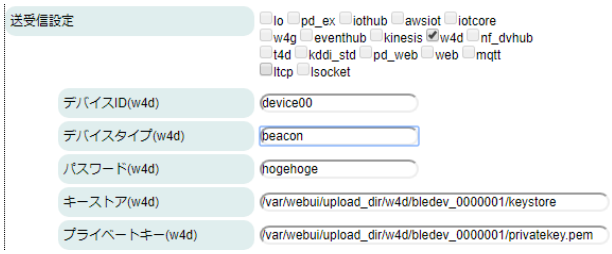
キープアライブ間隔[sec]：MQTT プロトコルのキープアライブインターバルを設定します。

クリーンセッション：MQTT プロトコルのクリーンセッションの使用／不使用を選択します。

Retain 機能：MQTT プロトコルの Retain 機能の使用／不使用を選択します。

デバイス一括設定:各デバイス設定メニューにおいて送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

■各デバイス設定メニューにおける設定



デバイス ID(w4d)：Watson IoT for Device のデバイス ID を設定します。

デバイスタイプ(w4d)：Watson IoT for Device のデバイスタイプを設定します。

パスワード：Watson IoT for Device への接続に用いるパスワードを設定します。

トラストストア：Watson IoT for Device へ接続する際のルート証明書ファイルのパス名を設定します。

キーストア：Watson IoT for Device へ接続する際のクライアント証明書ファイルのパス名を設定します。

プライベートキー：Watson IoT for Device へ接続する際のプライベートキーファイルのパス名を設定します。

※トラストストア・キーストア・プライベートキーは WEB UI の「システム」→「ファイル管理」タブにてアップロードしてください。

2-4-2-10. IoT デバイスハブ(nf_dvhub)

■送受信設定メニューにおける設定

インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。　0 を指定した場合、データ送信が完了するまで保持し続けます。

サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。 通常デォルト値（86400 秒）から変更する必要はありません。　0 を指定した場合、再起動は行いません。

送信先ホスト：接続先の IoT デバイスハブのホスト名を設定します。

プロトコル：接続に用いるプロトコル(TCP/SSL)を選択します。

QoS：IoT デバイスハブへデータを送信する際の MQTT プロトコルの QoS を設定します。

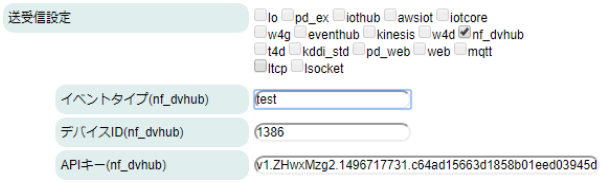
ルート証明書：IoT デバイスハブへ接続する際のルート証明書ファイルのパス名を設定します。

詳細設定：MQTT プロトコルの詳細値を設定します。

キープアライブ間隔[sec]：MQTT プロトコルのキープアライブインターバルを設定します。

デバイス一括設定:各デバイス設定メニューにおいて送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

■各デバイス設定メニューにおける設定



イベントタイプ(nf_dvhub)：IoT デバイスハブのイベントタイプを設定します。

デバイス ID(nf_dvhub)：IoT デバイスハブのデバイス ID を設定します。

API キー(nf_dvhub)：IoT デバイスハブの API キーを設定します。

※ルート証明書は WEB UI の「システム」→「ファイル管理」タブにてアップロードしてください。

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

■送受信設定メニューにおける設定

インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

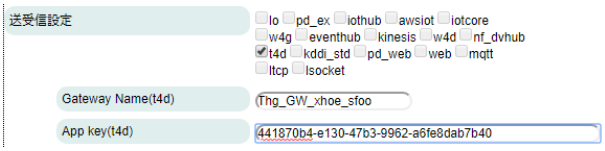
有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。　0 を指定した場合、データ送信が完了するまで保持し続けます。

サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。　通常デォルト値（86400 秒）から変更する必要はありません。　0 を指定した場合、再起動は行いません。

接続先 URL：接続先の Toami for DOCOMO の URL を設定します。

デバイス一括設定:各デバイス設定メニューにおいて送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

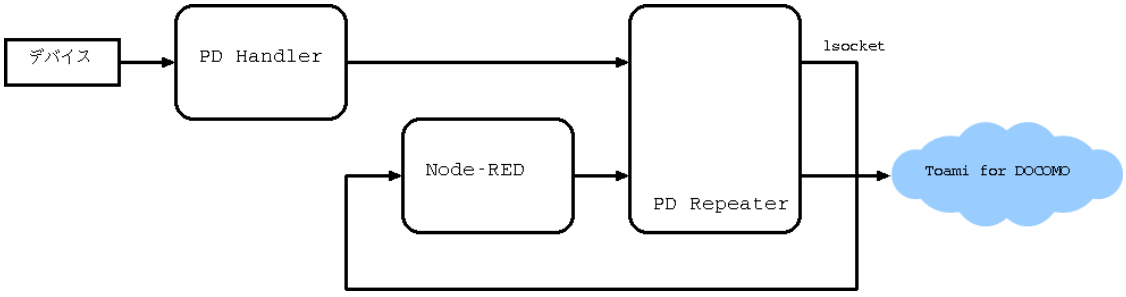
■各デバイス設定メニューにおける設定



Gateway Name(t4d)：Toami for DOCOMO の Gateway Name を設定します。

App Key(t4d)：Toami for DOCOMO の Gateway Name を設定します。

※Toami for DOCOMO では扱えるデータ（JSON 文字列）のキー情報が固定されているため、下図の様に PD Handler の出力を PD Reperter で Node-RED に渡し、Node-RED でキー情報の変換を行い、PD Reperter を介して Toami for DOCOMO へ送る構成とする必要があります。



Beacon デバイス　（PD Handler BLE）を例に設定方法を示します。

1. PD Handler BLE　（デバイス番号 device_beacon）の送り先をドメインソケット(lsocket)に設定します。
   ドメインソケット(lsocket)の設定については、第 2-4-2-17 章を参照して下さい。
   ここで「ソケットパスプレフィックス」は、デフォルト値”@/node-red/”のまま設定されるものとします。
2. Node-RED をユーザー定義のデバイスとして登録します。
   ユーザー定義デバイスの登録については、第 2-4-8 章を参照して下さい。
   ここでユーザー定義のデバイスとして”userdev_0000001”が割り振られたものとします。
3. Node-RED において、ipc-in ノード・ipc-out ノード・t4d-keymapper ノードを配置し結線します。
   Node-RED の利用方法については「OpenBlocks IoT Family 向け Node-RED スターターガイド」を参照して下さい。

ipc-in ノード･ipc-out ノード･t4d-keymapper ノードを配置し結線します。

4. ipc-in ノードのプロパティを設定します。

プロパティの「Path」を”/node-red/device_beacon.sock”に設定し、「Abstract Flag」をチェックします。

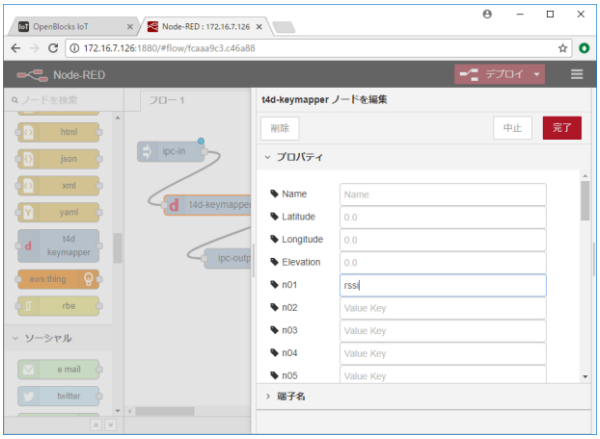
5. ipc-out ノードのプロパティを設定します。

プロパティの「Path」を”/pd_repeater/userdev_0000001.sock”に設定し、「Abstract Flag」をチェックします。

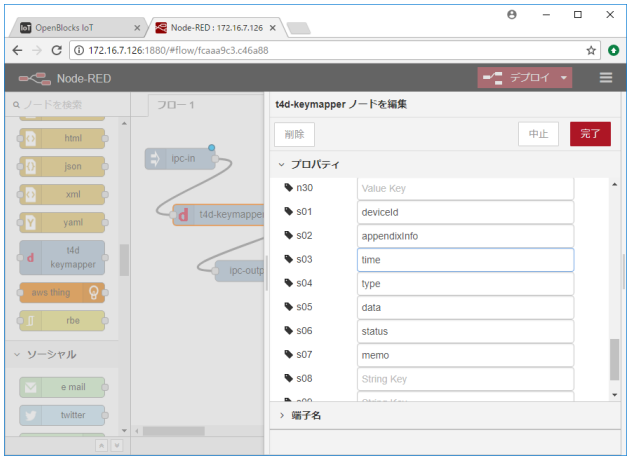
6. PD Handler BLE のビーコン出力の JSON キーと本設定例で設定する変換後のキーを示します。

JSON キー	データ形式	内容	変換後のキー
deviceId	文字列	デバイス ID	s01
appendixInfo	文字列	付随情報	s02
time	文字列	データ取得時刻	s03
rssi	整数値	受信信号強度	n01
type	文字列	ビーコン種別	s04
data	文字列	ペイロードデータ（HEX 表記）	s05
status	文字列	ビーコンステータス	s06
memo	文字列	WEB UI から設定された文字列	s07

t4d-mapper ノードのプロパティにおいて上表の設定を行います。



1. 「n01」を「rssi」に設定します。



2. 「s01」を「deviceId」に設定します。
3. 「s02」を「appendixInfo」に設定します。
4. 「s03」を「time」に設定します。
5. 「s04」を「type」に設定します。
6. 「s05」を「data」に設定します。
7. 「s06」を「status」に設定します。
8. 「s07」を「memo」に設定します。

※ここで、「s03」に設定される「データ取得時刻」は、Toami for DOCOMO 上では単なる文字列として扱われます。　Toami for DOCOMO の時刻情報（「timestamp」キーの値）は t4d-keymapper ノード内で自動的に生成され付加されます。

※t4d-keymapper ノードは、PD Handler Modbus Client/Server 等、値を配列や JSON オブジェクトで出力するデバイスには対応できません。　値を配列や JSON オブジェクトで出力するデバイスについては、その変換条件に応じた独自の機能（ファンクション）ノードを作成して下さい。

2-4-2-12. KDDI IoT クラウド Standard(kddi_std)

■送受信設定メニューにおける設定

インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。　0 を指定した場合、データ送信が完了するまで保持し続けます。

サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。 通常デォルト値（86400 秒）から変更する必要はありません。　0 を指定した場合、再起動は行いません。

ドメイン名：接続先の KDDI IoT クラウド Standard のドメイン名を設定します。

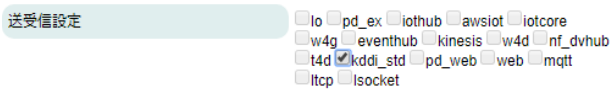
端末 ID：KDDI IoT クラウド Standard の端末 ID を設定します。

ユーザー名：KDDI IoT クラウド Standard の BASIC 認証に用いるユーザー名を設定します。

パスワード：KDDI IoT クラウド Standard の BASIC 認証に用いるパスワードを設定します。

デバイス一括設定:各デバイス設定メニューにおいて送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

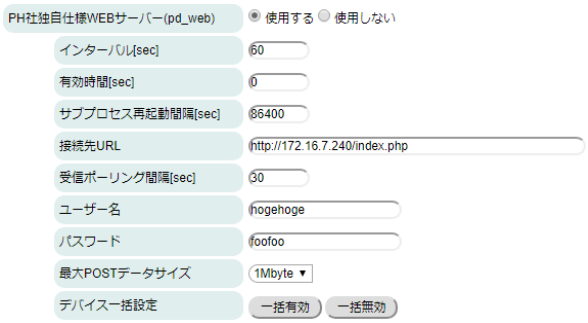
■各デバイス設定メニューにおける設定



「送受信設定」をチェックする以外の設定項目はありません。

2-4-2-13. PH 社独自仕様 WEB サーバー(pd_web)

■送受信設定メニューにおける設定



インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。　0 を指定した場合、データ送信が完了するまで保持し続けます。

サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。 通常デォルト値（86400 秒）から変更する必要はありません。　0 を指定した場合、再起動は行いません。

接続先 URL：接続先の PH 社独自仕様 WEB サーバーの URL を設定します。

受信ポーリング間隔[sec]：WEB サーバーから制御メッセージを読み出す間隔を設定します。

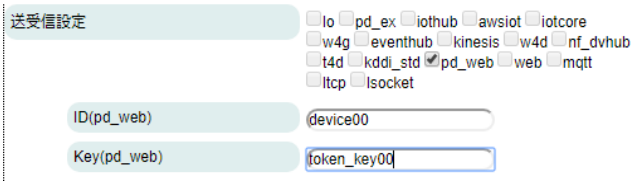
ユーザー名：WEB サーバーの BASIC 認証に用いるユーザー名を設定します。

パスワード：WEB サーバーの BASIC 認証に用いるユーザー名を設定します。

最大 POST データサーズ：1 回の POST メソッドで送信する最大データサイズを選択します。1~4Mbyte の中で選択します。

デバイス一括設定:各デバイス設定メニューにおいて送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

■各デバイス設定メニューにおける設定



ID(pd_web)：WEB サーバーのトークン認証に用いる ID を設定します。

Key(pd_web)：WEB サーバーのトークン認証に用いる Key を設定します。

PH 社独自仕様 WEB サーバーでのデータ送信方法は Content-Type を "application/json" として POST しています。別途 HTTP ヘッダー等を付与して送信していますので、詳細仕様については第 4-11 章を参照して下さい。

2-4-2-14. WEB サーバー(web)

■送受信設定メニューにおける設定

インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。　0 を指定した場合、データ送信が完了するまで保持し続けます。

サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。 通常デォルト値（86400 秒）から変更する必要はありません。　0 を指定した場合、再起動は行いません。

接続先 URL：接続先の汎用 WEB サーバーの URL を設定します。

最大 POST データサーズ：1 回の POST メソッドで送信する最大データサイズを選択します。1~4Mbyte の中で選択します。

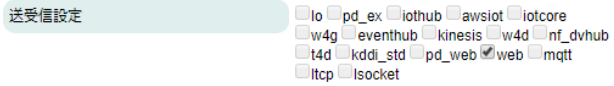
コンテンツタイプ：WEB サーバーのコンテンツタイプ（WEB フォーム形式／プレーンテキスト形式／JSON 形式）を選択します。

ユーザー名：WEB サーバーの BASIC 認証に用いるユーザー名を設定します。

パスワード：WEB サーバーの BASIC 認証に用いるユーザー名を設定します。

デバイス一括設定:各デバイス設定メニューにおいて送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

■各デバイス設定メニューにおける設定



「送受信設定」をチェックする以外の設定項目はありません。

WEB サーバに対しては、データを POST メソッドにて送信します。送信形式設定が”WEB フォーム形式”の場合、Content-Type は "application/x-www-form-urlencoded"となります。ペイロード（送信データ本体）は、”Records” という x-www-form-urlencoded 変数に複数データをまとめて送信します。

送信形式設定が”プレーンテキスト形式”の場合、Content-Type は "text/plain"となります。ペイロード（送信データ本体）の URL セーフエンコードは行われず、複数データ(JSON 前提)をまとめて送信します。

送信形式設定が”JSON 形式”の場合、Content-Type は "application/json"となります。ペイロード（送信データ本体）の URL セーフエンコードは行われず、複数データ(JSON 前提)をまとめて送信します。

■送信形式設定 ：WEB フォーム形式

●データ書式

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

●送信サンプル

```
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.
91259765625,"temperature":25.32928466796875,"pressure":1016.37,"time":"2016-11-24T16:50:23.431+0900"},{"dev
iceId":"b0b448b81105","memo":"cc2650-1","objectTemp":19.75,"ambientTemp":24.875,"humidity":47.91259765625,"t
emperature":25.3594970703125,"pressure":1016.31,"time":"2016-11-24T16:50:26.459+0900","lux":427.68}]
```

■送信形式設定 ： プレーンテキスト形式

●データ書式

[{DATA1},{DATA2},{DATA3},…{DATAn}]

●送信サンプル

```
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"},{"devic
eId":"e135535f61e4","appendixInfo":"G8H00010","rssi":-74,"time":"2017-12-04T14:07:18.762+09:00"}]
```

■送信形式設定 ：JSON 形式

●データ書式

[{DATA1},{DATA2},{DATA3},…{DATAn}]

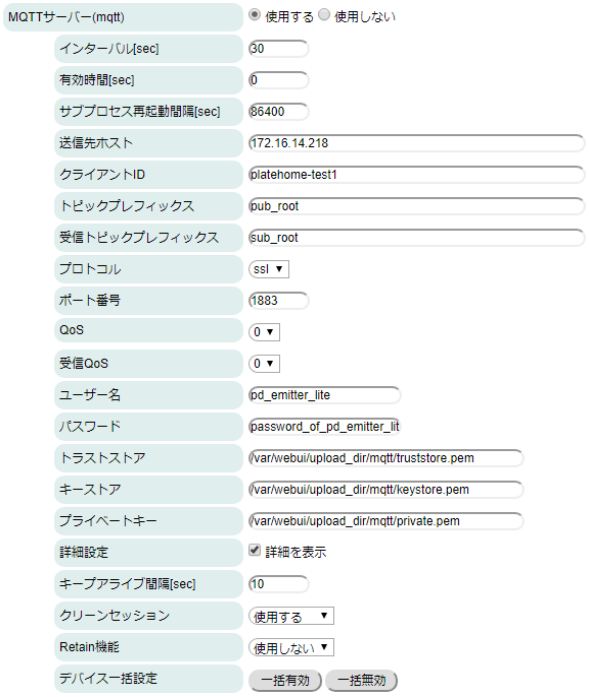
●送信サンプル

```
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"},{"devic
eId":"e135535f61e4","appendixInfo":"G8H00010","rssi":-74,"time":"2017-12-04T14:07:18.762+09:00"}]
```

2-4-2-15. MQTT サーバー(mqtt)

■送受信設定メニューにおける設定



インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。　0 を指定した場合、データ送信が完了するまで保持し続けます。

サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。　通常デォルト値（86400 秒）から変更する必要はありません。　0 を指定した場合、再起動は行いません。

送信先ホスト：接続先の汎用 MQTT サーバーのホスト名を設定します。

クライアント ID:汎用 MQTT サーバーに接続するためのクライアント ID を設定します。

トピックプレフィックス：汎用 MQTT サーバーへデータを送る際に用いる MQTT プロトコルのトピックのプレフィックスを設定します。　サフィックスは'/'に続きユニーク ID が設定されます。

受信トピックプレフィックス：汎用 MQTT サーバーから制御メッセージを待ち受けるために用いる MQTT プロトコルのトピックのプレフィックスを設定します。　サフィックスは'/'に続きユニーク ID が設定されます。

プロトコル：接続に用いるプロトコル(TCP/SSL)を選択します。

QoS：汎用 MQTT サーバーへデータを送信する際の MQTT プロトコルの QoS を設定します。

受信 QoS：汎用 MQTT サーバーから制御メッセージを受信する際の MQTT プロトコルの QoS を設定します。

ユーザー名：汎用 MQTT サーバーへの接続に

用いるユーザー名を設定します。

パスワード：汎用 MQTT サーバーへの接続に用いるパスワードを設定します。

トラストストア：汎用 MQTT サーバーへ接続する際のルート証明書ファイルのパス名を設定します。

キーストア：汎用 MQTT サーバーへ接続する際のクライアント証明書ファイルのパス名を設定します。

プライベートキー：汎用 MQTT サーバーへ接続する際のプライベートキーファイルのパス名を設定します。

詳細設定：MQTT プロトコルの詳細値を設定します。

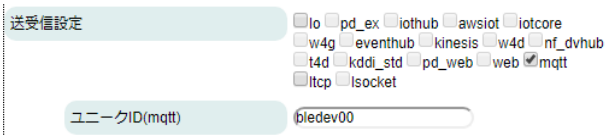
キープアライブ間隔[sec]：MQTT プロトコルのキープアライブインターバルを設定します。

クリーンセッション：MQTT プロトコルのクリーンセッションの使用／不使用を選択します。

Retain 機能：MQTT プロトコルの Retain 機能の使用／不使用を選択します。

デバイス一括設定:各デバイス設定メニューにおいて送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

■各デバイス設定メニューにおける設定



ユニーク ID(mqtt):汎用 MQTT サーバーへデータを送る際、もしくは制御メッセージを受け取る際の MQTT プロトコルのトピックのサフィックスに用いる各デバイス毎にユニークな文字列を設定します。 ’.’ （ASCII コード 0x2e）は ’/’ (ASCII コード 0x2f)に変換して扱われます。

※トラストストア・キーストア・プライベートキーは WEB UI の「システム」→「ファイル管理」タブにてアップロードしてください。

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

■送受信設定メニューにおける設定

インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。 0 を指定した場合、データ送信が完了するまで保持し続けます。

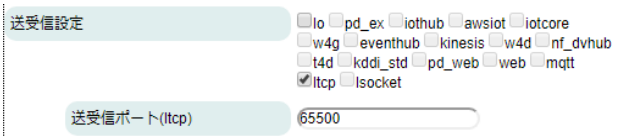
サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。 通常デォルト値（86400 秒）から変更する必要はありません。 0 を指定した場合、再起動は行いません。

IP アドレス：接続先の汎用 TCP サーバーの IP アドレスを設定します。

デリミタ:データーの境界となる 1 文字もしくは 2 文字の ASCII コードを設定します。 例えば CR(復帰)をデリミタとする場合は 0x0d、CRLF（復帰・改行）をデリミタとする場合は 0x0d0a となります。

デバイス一括設定:各デバイス設定メニューにおいて送信対象設定が”送信する”となっている各対象の送信先設定を一括で有効/無効を選択できます。

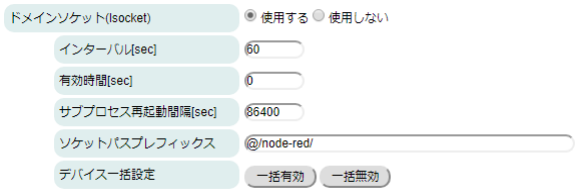
■各デバイス設定メニューにおける設定



送受ポート番号(ltcp)：接続先の汎用 TCP サーバーのポート番号を 49152～65535 の範囲で設定します。

2-4-2-17. ドメインソケット(lsocket)

■送受信設定メニューにおける設定



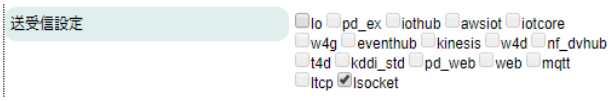
インターバル[sec]：送信完了後～送信開始までの時間間隔を秒単位で設定します。

有効時間[sec]：PD Reperter がデータ送信できない場合において、保持する時間を設定します。　0 を指定した場合、データ送信が完了するまで保持し続けます。

サブプロセス再起動間隔[sec]：サブプロセスを再起動する間隔を設定します。　通常デォルト値（86400 秒）から変更する必要はありません。　0 を指定した場合、再起動は行いません。

ソケットパスプレフィックス：データを出力する UNIX ドメインソケットのパス名のプレフィックスを設定します。　先頭が'@'の場合は Abstract Domain Socket として扱われます。　パス名のサフィックスにはデバイス番号が設定されます。

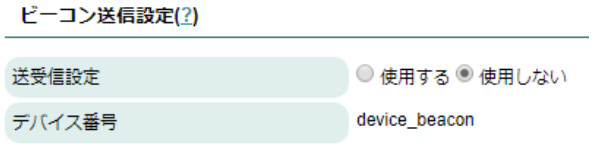
■各デバイス設定メニューにおける設定



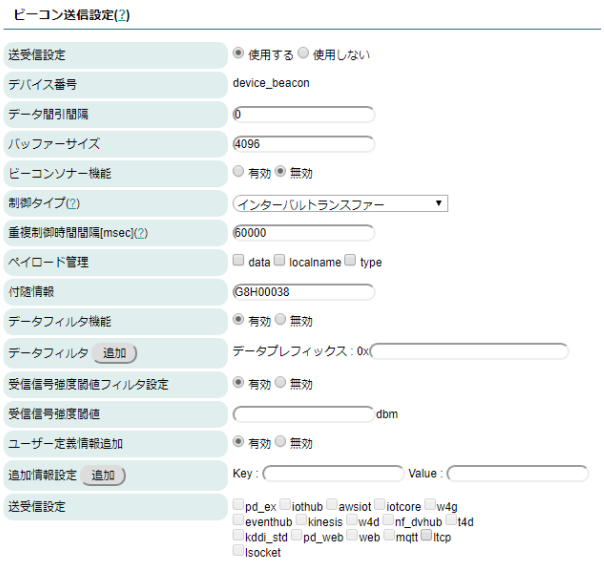
「送受信設定」をチェックする以外の設定項目はありません。

2-4-3. ビーコン送信設定

WEB UI の「IoT データ」→「送受信設定」タブの「ビーコン送信設定」メニューより、ビーコンの送信設定を行うことが出来ます。
ビーコンの送信を行うためには、WEB UI の「サービス」→「基本」→「BT I/F」タブより、サービス機能の基本設定とし BT I/F (hci0)が「使用する」に設定されている必要があります。
サービス機能の基本設定については、「OpenBlocks IoT Family 向け WEB UI セットアップガイド」の第 5 章を参照して下さい。
「ビーコン送信設定」メニューが表示されていない場合は、「第 2-3 章 IoT データ制御のアプリ設定」を参照し、PD Handler BLE を「使用する」に設定して下さい。



初期状態の送信先設定は左図のようになっています。
ここで、ビーコンデータをクラウド等への送信する場合には、”送信する”を選択します。
注意）後述のデバイス情報送信設定で送信対象としているビーコンには、本項は適用されません。



「送信する」を選択すると各設定項目が表示されます。
デバイス番号：OpenBlocks IoT Family の WEB UI 内で管理している番号です。変更はできません。
データ間引間隔:データを間引くための入力データを受け取らない時間を msec 単位で設定します。　0 の場合、間引きは行われません。
バッファーサイズ:データの最大サイズを設定します。単位はバイトです。
ビーコンソナー機能：受信対象となっているビーコンデータを受信した際にビーコンソナーを有効にするか無効を設定します。
制御タイプ:ビーコンデータを管理する方式を以下から選択します。各方式については後述の”ビーコン重複制御アルゴリズム”を参照してください。

・インターバルトランスファー
・エントリーポイントトランスファー
・インアウトステータストランスファー

複制御時間間隔[ms]：各制御タイプにて用いる制御時間を msec 単位で設定します。

ペイロード管理：ビーコンデータを PD Repeater へ渡す際に、ビーコンの各情報を付随させるかを選択します。

data：アドバタイズデータ(16 進数)

localname：デバイス名

type：データ種別

付随情報:ビーコンデータを各クラウドへ送信する際に、どこの OpenBlocks IoT Family から送信されたか等の付随させる情報を設定します。

※デフォルトは本体シリアル番号です。

データフィルタ機能:(データプレフィックス)送信対象のビーコンを選別するフィルタを設定します。データプレフィックスに 16 進文字列でフィルタ条件を入力すると、ビーコンのアドバタイズ情報を前方一致で比較し一致したもののみを送信先へ送信します。

※「追加」ボタンにて、複数登録できます。

※データフィルタを設定する場合には、本装置内(local)内のログの data を参照しデバイスをフィルタリングしてください。本装置内のログは(local)内のログについてもフィルタは適用されます。

受信信号強度閾値フィルタ設定：受信対象とするビーコンの信号強度閾値フィルタを使用するか設定します。

受信信号強度閾値:受信対象とするビーコンの信号強度を設定します。

ユーザー定義情報追加：(追加情報設定) PD Repeater へ渡す際のデータにキー名/値の組合せで追加できます。

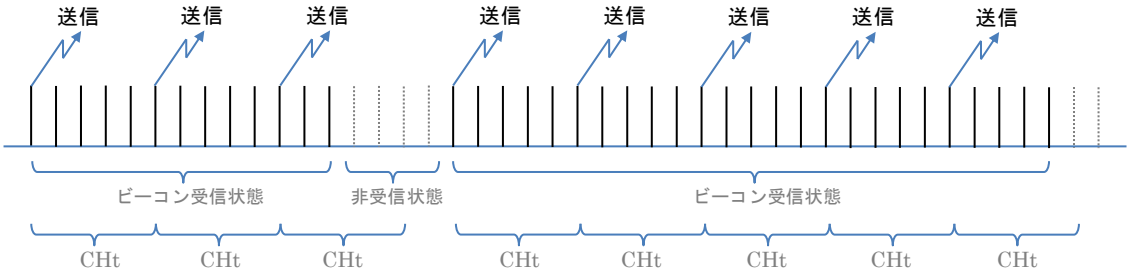
※「追加」ボタンにて、最大 5 個まで登録できます。
※「位置情報設定」ボタンにて、既に登録している位置情報をフォームに設定します。
送信先設定:“使用する”を選択した送信先に対してチェックボックスが選択できるようになります。 チェックを付けた送信先に対して、送信を行います。 チェックをつけると送信先固有の設定項目が表示されます。 送信先固有の設定については、「2-4-2.送受信先毎の設定」を参照して下さい。

※ビーコンソナー機能を送信対象にした状態において USB スピーカー(型番：MM-SPU8BK)を接続した状態にて受信対象(データフィルタ及び受信信号強度閾値フィルタについても考慮)となっているビーコンデータを受信した場合には、スピーカーから検出音が鳴ります。

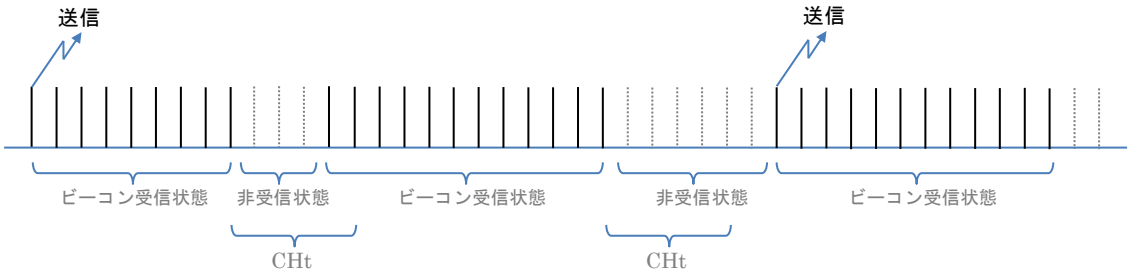
ビーコン重複制御アルゴリズム

この説明における前提条件となる設定
ビーコンの送信間隔 = 1 秒
重複制御時間間隔(CHt) = 5 秒

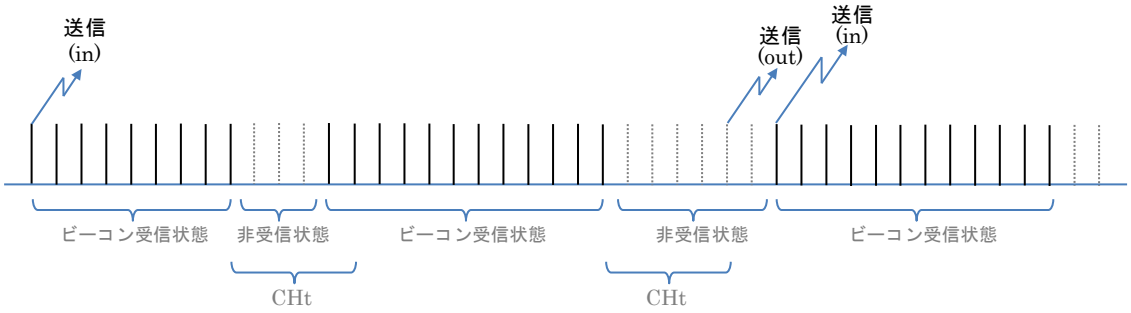
① インターバルトランスファー
ビーコンを受信している間は指定された一定間隔で送信プログラムへ。



② エントリーポイントトランスファー
ビーコンが受信されたタイミングで 1 回送信プログラムへ。
（CHt 時間内の一時非受信は退場扱いしない）

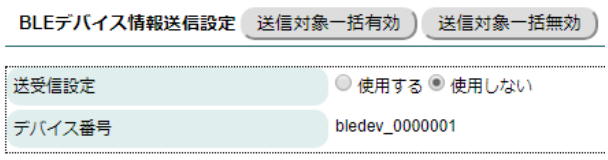


③ インアウトステータストランスファー
ビーコンが入場・退場のタイミングで IN/OUT フラグ付きで送信プログラムへ。
（CHt 時間内の一時非受信は退場扱いしない）



2-4-4. BLE デバイス情報送信設定

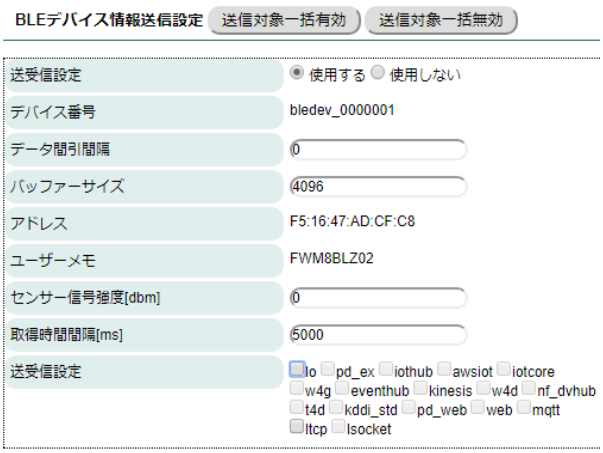
WEB UI の「IoT データ」→「送受信設定」タブの「BLE デバイス情報送信設定」メニューより、BLE デバイス情報の送信設定を行うことが出来ます。
BLE デバイス情報の送信を行うためには、WEB UI の「サービス」→「基本」→「BT I/F」タブより、サービス機能の基本設定として BT I/F (hci0)が「使用する」に設定されている必要があります。
また、WEB UI の「サービス」→「基本」→「BLE 登録」タブより、サービス機能の基本設定として BLE デバイスが登録されている必要があります。
サービス機能の基本設定については、「OpenBlocks IoT Family 向け WEB UI セットアップガイド」の第 5 章を参照して下さい。
「BLE デバイス情報送信設定」メニューが表示されていない場合は、「第 2-3 章 IoT データ制御の基本設定」を参照し、PD Handler BLE を「使用する」に設定して下さい。



登録済の BLE デバイスが存在している場合、初期状態では左図のようになっています。
※デバイスが 1 個登録されている場合です。

デバイス毎に送信対象項目にて「送信する」を選択すると各設定項目が表示されます。

※送信対象一括有効、送信対象一括無効ボタンにて全ての登録済のデバイスの送信対象を制御できます。



「送信する」を選択すると各設定項目が表示されます。

デバイス番号：OpenBlocks IoT Family の WEB UI 内で管理している番号です。変更はできません。

データ間引間隔:データを間引くための入力データを受け取らない時間を msec 単位で設定します。　0 の場合、間引きは行われません。

バッファーサイズ:データの最大サイズを設定します。単位はバイトです。

アドレス:登録されたデバイスの BT のアドレスを表示します。

ユーザーメモ:登録されたデバイスにて設定されたメモ情報を表示します。

センサー信号強度[dbm]：センサーに信号強度を設定できる機種の場合、設定したい信号強度を入力します。設定した信号強度が無い場合、近似値またはデフォルト値が設定されます。

取得時間間隔[ms]：センサーからデータを取得する時間間隔を数字で設定します。単位はmsec です。

※”富士通コンポーネント製 BT Smart センサービーコン”と” Texas Instruments 製 SimpleLink SensorTag”のみサポートしています。

送信先設定：“使用する”を選択した送信先に対してチェックボックスが選択できるようになります。 チェックを付けた送信先に対して、送信を行います。 チェックをつけると送信先固有の設定項目が表示されます。 送信先固有の設定については、「2-4-2.送受信先毎の設定」を参照して下さい。

※既存のデバイス不良等の差し替え時に以前のものと同様に扱う為に設定を同一にすることを推奨します。（不良となったデバイスは送信対象設定を「送信しない」へ変更してください。）

2-4-5. EnOcean デバイス設定

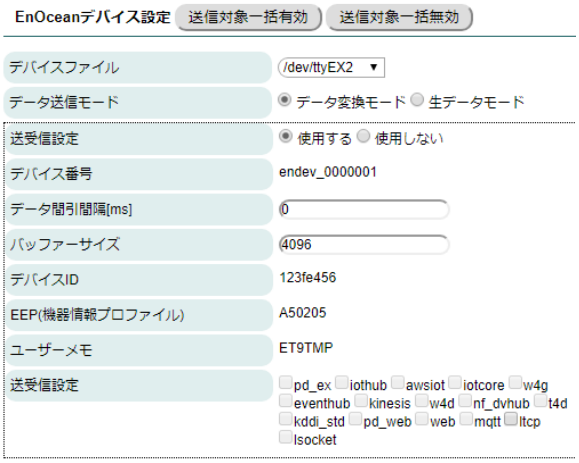
OpenBlocks IoT Family に EnOcean モジュール（拡張追加モジュール）を搭載する、もしくは EnOcean 受信用 USB ドングルを装着することで、PD Handler UART を用いて EnOcean デバイスから情報を取得することができます。
WEB UI の「IoT データ」→「送受信設定」タブの「EnOcean デバイス設定」メニューより、EnOcean デバイスから情報を取得し送信する設定を行うことが出来ます。
EnOcean デバイスから情報を取得するためには、WEB UI の「サービス」→「基本」→「EnOcean 登録」タブより、サービス機能の基本設定として EnOcean デバイスが登録されている必要があります。
サービス機能の基本設定については、「OpenBlocks IoT Family 向け WEB UI セットアップガイド」の第 5 章を参照して下さい。
「EnOcean デバイス設定」メニューが表示されていない場合は、「第 2-3 章 IoT データ制御の基本設定」を参照し、PD Handler UART を「使用する」に設定して下さい。

登録済の EnOcean デバイスが存在している場合、初期状態では左図のようになっています。
※デバイスが 1 個登録されている場合です。

デバイスファイル：拡張追加モジュールを使用する場合は「/dev/ttyEX2」、USB ドングルを使用する場合は対応するデバイスファイルを選択して下さい。
※USB ドングルの場合、USB デバイスの認識順によってデバイスファイル名が異なります。
データ送信モード：PD Repeate へ送信するデータのモードを設定します。データ変換モードは対応している EEP の場合は解析したデータを PD Repeater へ送信します。対応していない EEP の場合は、受信データを 16 進数文字列へ変換したデータを PD Repeater へ送信します。また、生データモードは対応 EEP を問わず、受信データを 16 進数文字列へ変換したデータを PD Repeater へ送信します。

デバイス毎に送信対象項目にて「送信する」を選択すると各設定項目が表示されます。

※送信対象一括有効、送信対象一括無効ボタンにて全ての登録済のデバイスの送信対象を制御できます。



「送信する」を選択すると各設定項目が表示されます。

デバイス番号：OpenBlocks IoT Family の WEB UI 内で管理している番号です。変更はできません。

データ間引間隔:データを間引くための入力データを受け取らない時間を msec 単位で設定します。　0 の場合、間引きは行われません。

バッファーサイズ:データの最大サイズを設定します。単位はバイトです。

デバイス ID：登録された EnOcean デバイスの ID を表示します。

EEP（機器情報プロファイル）：登録された EnOcean デバイスの機器情報プロファイルを表示します。

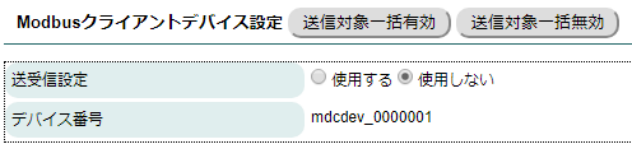
ユーザーメモ:登録されたデバイスにて設定されたメモ情報を表示します。

送信先設定:“使用する”を選択した送信先に対してチェックボックスが選択できるようになります。 チェックを付けた送信先に対して、送信を行います。 チェックをつけると送信先固有の設定項目が表示されます。 送信先固有の設定については、「2-4-2.送受信先毎の設定」を参照して下さい。

※既存のデバイス不良等の差し替え時に以前のものと同様に扱う為に設定を同一にすることを推奨します。（不良となったデバイスは送信対象設定を「送信しない」へ変更してください。）

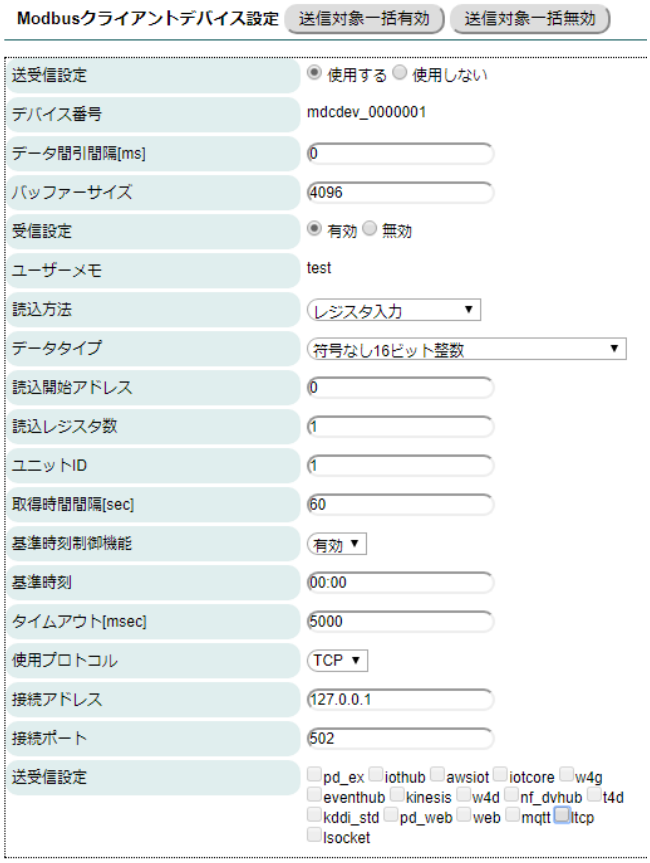
2-4-6. Modbus クライアントデバイス設定

Modbus クライアントは、Modbus プロトコルを用いて PLC 機器から定期的にデータを読み込み（ポーリングを行い）、PD Repeater を介してクラウドに送ります。
また、PD Repeater を介してクラウドから送られる制御メッセージ(JSON 文字列)に基づき PLC 機器に Modbus プロトコルを用いて接続しデータを読み書きすることもできます。
WEB UI の「IoT データ」→「送受信設定」タブの「Modbus クライアントデバイス設定」メニューより、Modbus クライアントデバイスの設定を行うことが出来ます。
Modbus クライアントデバイスを利用するためには、WEB UI の「サービス」→「基本」→「Modbus(C)登録」タブより、サービス機能の基本設定として Modbus クライアントデバイスが登録されている必要があります。
サービス機能の基本設定については、「OpenBlocks IoT Family 向け WEB UI セットアップガイド」の第 5 章を参照して下さい。
「Modbus クライアントデバイス設定」メニューが表示されていない場合は、「第 2-3 章 IoT データ制御のアプリ設定」を参照し、PD Handler MODBUS Client を「使用する」に設定して下さい。



登録済の Modbus クライアントデバイスが存在している場合、初期状態では左図のようになっています。
※デバイスが 1 個登録されている場合です。
※Modbus クライアントデバイスとは、PLC 機器そのものではなく、対象となる PLC 機器への接続方法の他、データを取得するための「読込方法」や「読込開始アドレス」、「読込レジスタ数」等の設定の組み合わせを意図します。

デバイス毎に送信対象項目にて「送信する」を選択すると各設定項目が表示されます。
※送信対象一括有効、送信対象一括無効ボタンにて全ての登録済のデバイスの送信対象を制御できます。



※「使用プロトコル」として「TCP」（ネットワーク）を選択した場合の表示

「送信する」を選択すると各設定項目が表示されます。

デバイス番号：OpenBlocks IoT Family の WEB UI 内で管理している番号です。変更はできません。

データ間引間隔：データを間引くための入力データを受け取らない時間を msec 単位で設定します。　0 の場合、間引きは行われません。

バッファーサイズ：データの最大サイズを設定します。単位はバイトです。

受信設定：PD Repeater を介してクラウドからの制御メッセージを受ける下流方向制御を行うか否かを選択します。

ユーザーメモ：登録されたデバイスにて設定されたメモ情報を表示します。

読込方法：デジタル出力（ビット）、デジタル入力（ビット）、レジスタ出力、レジスタ入力から選択します。

「デジタル出力」または「デジタル入力」を選んだ場合は、0 または 1 の並びが出力されます。

データタイプ：読込方法を「レジスタ出力」、「入力レジスタ」を選択した際に、出力のデータタイプを以下から選択します。

・符号なし 16 ビット整数

・符号付き 16 ビット整数

・符号なし 32 ビット整数/リトルエンディアン

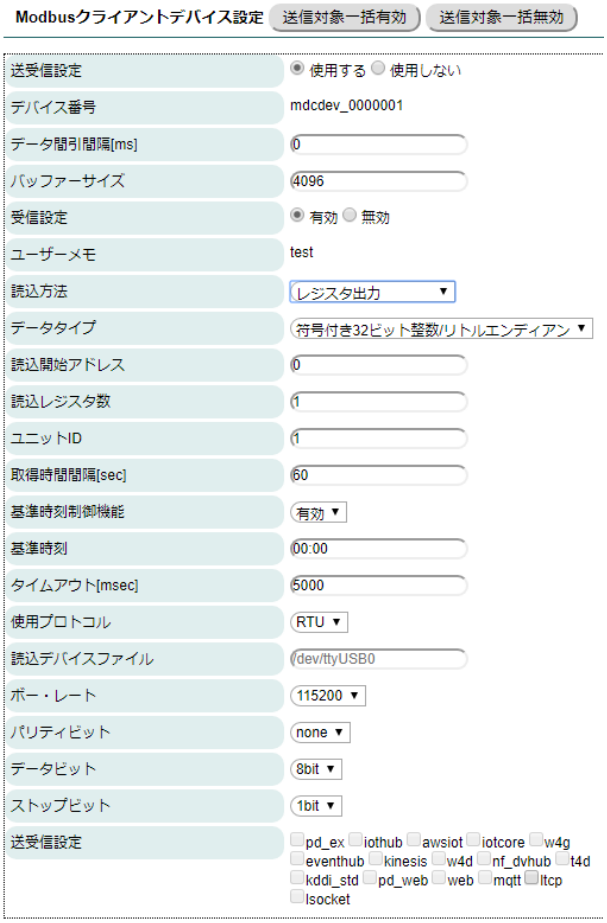
・符号付き 32 ビット整数/リトルエンディアン

・符号なし 32 ビット整数/ビッグエンディアン

・符号付き 32 ビット整数/ビッグエンディアン

読込開始アドレス：読み込みたいデータが格納されている PLC 機器上の開始アドレスを設定します。

読込レジスタ数：「読込方法」として「デジタル出力」または「デジタル入力」が選



※「使用プロトコル」として「RTU」（シリアル）を選択した場合の表示

択されている場合は、読み込まれるビット数と解釈されます。

「開始アドレス」に設定されるアドレスから読み込むレジスタ数もしくビット数を設定します。

ユニット ID：PLC 機器の Modbus ユニット ID を設定します。ユニット ID は、1～247 または 255 の数値です。

取得時間間隔[sec]：PLC デバイスからデータを取得する時間間隔を数字で設定します。単位は秒です。後述の基準時刻制御を使用する場合、時間間隔は以下の値へと内部的に変更されます。

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

基準時刻制御：特定の時刻にデータを取得する場合、本機能を有効とし基準時刻を設定しください。

基準時刻：特定の時刻にデータを取得する際の基準時刻を設定します。HH:MM 形式となります。

タイムアウト[msec]：PLC 機器からデータを取得する際のタイムアウトを設定します。単位はミリ秒です。

使用プロトコル：「TCP」、「RTU」のいずれかを選択します。「TCP」はネットワーク、「 RTU」はシリアルです。

PLC 接続アドレス(TCP)：接続する PLC 機器の IP アドレスを設定します。

PLC 接続ポート(TCP)：接続する PLC 機器の TCP ポート番号を設定します。　デフォルト値は、502 です。

読込デバイスファイル(RTU)：PLC 機器を接続するシリアルポートのデバイスファイル名を設定します。

ボー・レート(RTU)：シリアルポートのボー・レートを選択します。

パリティビット(RTU)：シリアルポートのパリティビットを選択します。

データビット(RTU)：シリアルポートのデータビット数を選択します。

ストップビット(RTU)：シリアルポートのストップビット数を選択します。

送信先設定：“使用する”を選択した送信先に対してチェックボックスが選択できるようになります。 チェックを付けた送信先に対して、送信を行います。　チェックをつけると送信先固有の設定項目が表示されます。　送信先固有の設定については、「2-4-2.送受信先毎の設定」を参照して下さい。

■CSV ファイルを用いた取得 Modbus クライアントデバイスの拡張

WEB UI の「システム」→「ファイル管理」タブより、/var/webui/upload_dir にファイル名 **pd_handler_modbus_client.csv** の CSV ファイルを置くことで、WebUI で設定された 1 つのデバイス番号に対し複数の取得 Modbus クライアントデバイスを設定することができます。
CSV ファイルの書式は、次の通りです。

デバイス番号，ユニット ID，読込方式，データタイプ，読込開始アドレス，読込レジスタ数

パラメータ	データの型式	説明
デバイス番号	半角英数字	WebUI により割り振れたデバイス番号を記載します。 WebUI に設定されていないデバイス番号は無視されます。 先頭が'#'または"/"の場合は、コメント行として扱われます。
ユニット ID	半角数字	PLC 機器の Modbus ユニット ID を設定します。ユニット ID は、1～247 または 255 を記載します。
読込方式	半角英数字	読込方式として次の何れかを記載します。 デジタル出力：'read_coils' 又は '0x01' 又は '1' デジタル入力：'read_discrete_input' 又は '0x02' 又は '2' レジスタ出力：'read_holding_registers' 又は '0x03' 又は '3' レジスタ入力：'read_input_registers' 又は'0x04' 又は '4'
データタイプ	半角英数字	データタイプとして次の何れかを記載します。 符号なし 16 ビット整数： 'uint16_t' 又は '0' 符号付き 16 ビット整数： 'int16_t' 又は '1' 符号なし 32 ビット整数/リトルエンディアン：'uint32lsb_t' 又は '2' 符号付き 32 ビット整数/リトルエンディアン：'int32lsb_t' 又は '3' 符号なし 32 ビット整数/ビッグエンディアン：'uint32msb_t' 又は '4' 符号付き 32 ビット整数/ビッグエンディアン：'int32msb_t' 又は '5'
読込開始アドレス	半角英数字	読み込みたいデータが格納されている PLC 機器上の開始アドレスを設定します。 先頭が'0x'の場合は 16 進数と解釈されます。
読込レジスタ数	半角数字	読み込みたいレジスタ数を記載します。

PD Handler Modbus Client の CSV ファイルの書式

各パラメータの区切りはカンマ、先頭がシャープ'#'もしくはスラッシュ'/'の行はコメント行と見なされます。

記載例）

```
#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
```

CSV ファイルが読み込まれると WebUI で設定された Modbus クライアントデバイスは上書きされます。そのため、CSV ファイルには WebUI で設定した Modbus クライアントデバイスを含む取得したい全ての Modbus クライアントデバイスを記載して下さい。

■基準時刻制御

基準時刻制御は、特定の時刻にデータを取得する機能です。

「Modbus クライアントデバイス設定」メニューの「基準時刻制御」を「有効」にし、「取得時間間隔[sec]」と「基準時刻」で取得間隔と取得時刻を設定します。

基準時刻制御における「取得時間間隔」は 300, 600, 900, 1800, 3600, 7200, 10800, 14400, 21600, 28800, 43200 と 86400 の倍数に限られます。「取得時間間隔」としてこれら以外の値が設定されると PD Handler Modbus Client 内で次のように扱われます。

取得時間間隔の設定値	実動作値
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～	86400 の倍数

基準時刻制御における取得時間間隔の設定と実動作値

「基準時刻」とは動作の起点となる時刻で、例えば「取得時間間隔」を 300 とし、「基準時刻」を”00:01”とした場合、データの取得は 00:01, 00:06, 00:11 ... 00:56, 01:01 ... 23:56, 00:01 の定刻に行われます。

データの取得開始時刻は「基準時刻」に設定さた時刻そのものではなく、「基準時刻」と「取得時間間隔」から算定される直近の時刻となります。

例えば 08:30 に「基準時刻」”01:05”、 「取得時間間隔」10800 の設定が行われた場合、最初のデータ取得は 10:05 に行われ、以降 13:05, 16:05, 19:05, 22:05, 01:05 の順におこなわれます。

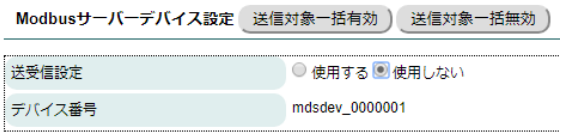
2-4-7. Modbus サーバーデバイス設定

Modbus サーバーは次表に示すレジスタマップを保持し、PLC 機器からの Modbus プロトコルによる接続を待ち受け、PLC 機器の書き込み操作によりレジスタの値を更新すると共に更新されたレジスタとその値を PD Repeater を介してクラウドに送ります。
また、PD Repeater を介してクラウドから送られる制御メッセージ(JSON 文字列)に基づきレジスタマップを読み書きすることもできます。

レジスタ	開始アドレス	サイズ
デジタル出力(Coils)	0x000	uint8 _t × 2048
デジタル入力(Discrete Input)	0x000	uint8_t × 2048
レジスタ出力(Holdig Registers)	0x000	uint16_t × 2048
レジスタ入力(Input Registers)	0x000	uint16_t × 2048

PD Handler Modbus Server のレジスタマップ

WEB UI の「IoT データ」→「送受信設定」タブの「Modbus サーバーデバイス設定」メニューより、Modbus サーバーデバイスの設定を行うことが出来ます。
Modbus サーバーデバイスを利用するためには、WEB UI の「サービス」→「基本」→「Modbus(S)登録」タブより、サービス機能の基本設定として Modbus サーバーデバイスが登録されている必要があります。
サービス機能の基本設定については、「OpenBlocks IoT Family 向け WEB UI セットアップガイド」の第 5 章を参照して下さい。
「Modbus サーバーデバイス設定」メニューが表示されていない場合は、「第 2-3 章 IoT データ制御のアプリ設定」を参照し、PD Handler MODBUS Server を「使用する」に設定して下さい。

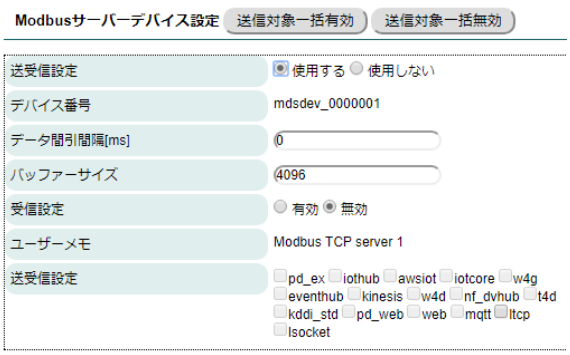


登録済の Modbus サーバーデバイスが存在している場合、初期状態では左図のようになっています。
※デバイスが 1 個登録されている場合です。

デバイス毎に送信対象項目にて「送信する」を選択すると各設定項目が表示されます。

※送信対象一括有効、送信対象一括無効ボタンにて全ての登録済のデバイスの送信対象を制御できます。

■Modbus サーバーデバイスの Device Type（待ち受けタイプ）が「TCP」で登録されている場合



「送信する」を選択すると各設定項目が表示されます。

デバイス番号：OpenBlocks IoT Family の WEB UI 内で管理している番号です。変更はできません。

データ間引間隔：データを間引くための入力データを受け取らない時間を msec 単位で設定します。　0 の場合、間引きは行われません。

バッファーサイズ：データの最大サイズを設定します。単位はバイトです。

受信設定：PD Repeater を介してクラウドからの制御メッセージを受けるか否かを選択します。

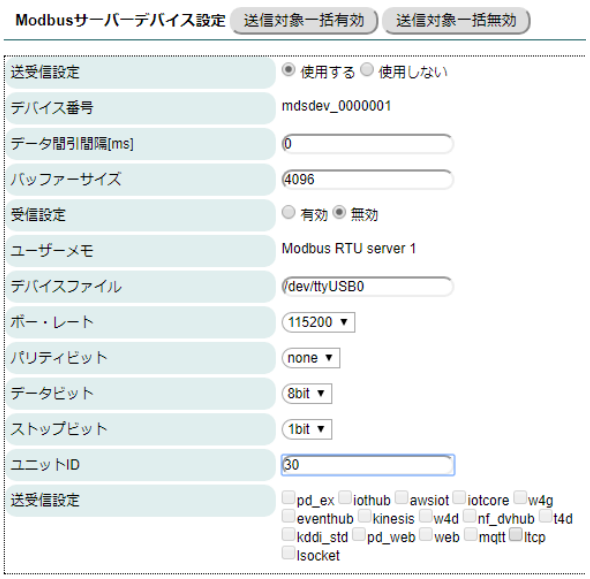
ユーザーメモ：登録されたデバイスにて設定されたメモ情報を表示します。

送信先設定："使用する"を選択した送信先に対してチェックボックスが選択できるようになります。 チェックを付けた送信先に対して、送信を行います。 チェックをつけると送信先固有の設定項目が表示されます。 送信先固有の設定については、「2-4-2.送受信先毎の設定」を参照して下さい。

※TCP で待ち受ける場合、Modbus プロトコルの TCP ポート（502）を開放しておく必要があります。　WEB UI の「システム」→「フィルター」タブの「フィルター開放設定」メニューにおいて「Modbus」を「有効」に設定して下さい。

WEB UI から登録できる Device Type（待ち受けタイプ）が「TCP」の Modbus サーバーデバイスは、TCP ポート（502）の開放操作の兼ね合いから 1 つに制限されています。

■Modbus サーバーデバイスの Device Type（待ち受けタイプ）が「RTU」で登録されている場合



「送信する」を選択すると各設定項目が表示されます。

デバイス番号：OpenBlocks IoT Family の WEB UI 内で管理している番号です。変更はできません。

データ間引間隔:データを間引くための入力データを受け取らない時間を msec 単位で設定します。　0 の場合、間引きは行われません。

バッファーサイズ:データの最大サイズを設定します。単位はバイトです。

受信設定：PD Repeater を介してクラウドからの制御メッセージを受ける下流方向制御を行うか否かを選択します。

ユーザーメモ:登録されたデバイスにて設定されたメモ情報を表示します。

デバイスファイル：PLC 機器からの接続を待ち受けるシリアルポートのデバイスファイル名を設定します。

ボー・レート：PLC 機器からの接続を待ち受けるシリアルポートのボー・レートを選択します。

パリティビット:シリアルポートのパリティビットを選択します。

データビット:シリアルポートのデータビット数を選択します。

ストップビット:シリアルポートのストップビット数を選択します。

ユニット ID:PLC 機器からの接続を待ち受ける Openbloks IoT Family 側の Modbus ユニット ID を設定します。ユニット ID は、1～247 の数値です。

送信先設定:“使用する”を選択した送信先に対してチェックボックスが選択できるようになります。　チェックを付けた送信先に対して、

送信を行います。 チェックをつけると送信先固有の設定項目が表示されます。 送信先固有の設定については、「2-4-2.送受信先毎の設定」を参照して下さい。

※Device Type（待ち受けタイプ）が「RTU」の Modbus サーバーデバイスは、複数登録することが可能ですが、「デバイスファイル」の設定が重複した場合の動作は保障されません。

※Modbus サーバーデバイスが複数登録されている場合、レジスタマップは複数の Modbus サーバーデバイスで共有されることになります。 Modbus サーバーデバイスとは Modbus サーバーの入出力デバイスと解釈して下さい。

2-4-8. デバイス設定(ユーザー定義)

OpenBlocks IoT Family の標準データ収集アプリケーション(PD Handler)を用いず、ユーザーが独自に用意するアプリケーションによりデータをハンドリングし、PD Repeater と連携させることができます。

WEB UI の「IoT データ」→「送受信設定」タブの「デバイス設定（ユーザー定義)」メニューより、ユーザー定義のデバイスの設定を行うことが出来ます。

ユーザー定義のデバイスを利用するためには、WEB UI の「サービス」→「基本」→「ユーザーデバイス登録」タブより、サービス機能の基本設定としてユーザー定義のデバイスが登録されている必要があります。

サービス機能の基本設定については、「OpenBlocks IoT Family 向け WEB UI セットアップガイド」の第５章を参照して下さい。

ユーザー定義のデバイスの登録とは、デバイス番号を割り振りメモ情報を付与する作業です。

登録済のユーザー定義のデバイスが存在している場合、初期状態では左図のようになっています。

※デバイスが１個登録されている場合です。

デバイス毎に送信対象項目にて「送信する」を選択すると各設定項目が表示されます。

※送信対象一括有効、送信対象一括無効ボタンにて全ての登録済のデバイスの送信対象を制御できます。

「送信する」を選択すると各設定項目が表示されます。

デバイス番号：OpenBlocks IoT Family の WEB UI 内で管理している番号です。変更はできません。

データ間引間隔:データを間引くための入力データを受け取らない時間を msec 単位で設定します。　0 の場合、間引きは行われません。

バッファーサイズ:データの最大サイズを設定します。単位はバイトです。

受信設定：PD Repeater を介してクラウドからの制御メッセージを受けるか否かを選択します。

ユーザーメモ:登録されたデバイスにて設定されたメモ情報を表示します。

送信先設定:“使用する”を選択した送信先に対してチェックボックスが選択できるようになります。 チェックを付けた送信先に対して、送信を行います。 チェックをつけると送信先固有の設定項目が表示されます。 送信先固有の設定については、「2-4-2.送受信先毎の設定」を参照して下さい。

2-5. 下流方向制御

クラウド等からメッセージを受け取り Modbus 対応 PLC 機器等を制御する下流方向の制御機能について説明します。

2-5-1. 下流方向制御の概要

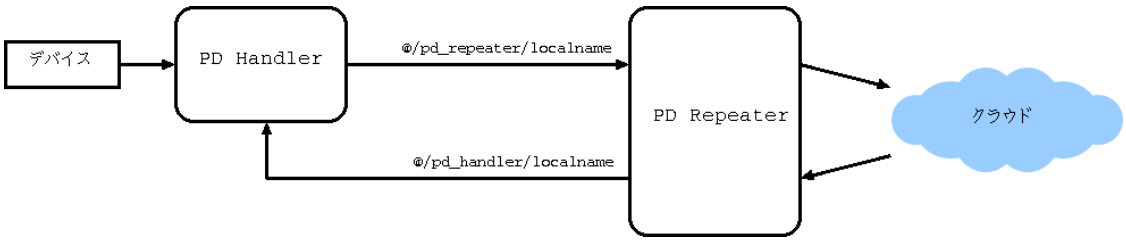
IoT データ制御機能を提供するソフトウェアモジュールにおいては、そのプロセス間通信において UNIX ドメインソケットを使用しています。下流方向制御も UNIX ドメインソケットを用いて動作します。

下流方向制御は PD Repeater が対応する送信先の内、MQTT プロトコルに対応したクラウドサービスと PD Exchange(pd_ex)、PH 社独自仕様 WEB サーバー(pd_web)並び TCP(ltcp)接続に対応します。

MQTT プロトコルと TCP 接続においては、送受(パブ／サブ)兼用の常時接続状態で動作し、PD Exchange については定期的に制御メッセージが無いかポーリングを行います。

PH 社独自仕様 WEB サーバーにおいては、データ送信時の応答メッセージを下流方向への制御メッセージとして扱います。

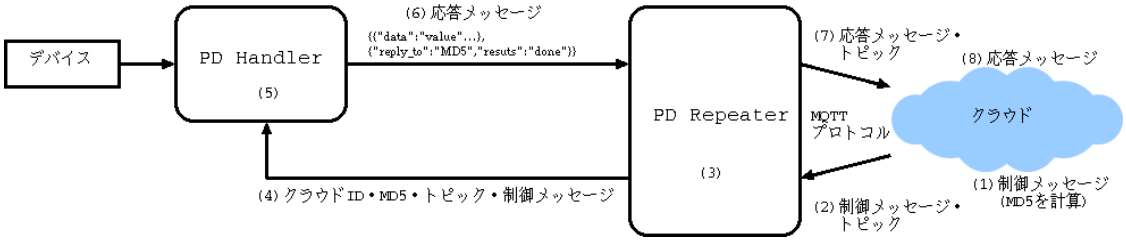
上流（クラウド）からの制御メッセージは PD Repeater を介し UNIX ドメインソケットにより PD Handler Modbus Client 等の下流のソフトウェアモジュールへ転送されます。下流のソフトウェアモジュールはデータの送信に用いる UNIX ドメインソケットにより PD Repeater を介しクラウドへ制御結果を返します。



制御メッセージは JSON 文字列を想定していますが、クラウド側の制御要件としてペイロードに載せられない情報、MQTT においてはトピック、PD Exchange においてはコマンド ID、PH 社独自仕様 WEB サーバーにおいては応答ヘッダー、TCP 接続においてはポート番号が必要となる場合が考えられるため、PD Repeater から下流のソフトウェアモジュールへの制御メッセージの転送においては、これらの情報と制御メッセージのハッシュ値(MD5)を付加しています。

制御メッセージのハッシュ値は、下流のソフトウェアモジュールが制御結果を返す際に、どの制御メッセージに対する応答なのか明確にすることを想定したものです。

MQTT プロトコルで接続されるクラウドを例に下流方向制御の流れを説明します。



(1) 制御メッセージのハッシュ値(MD5)を計算しておきます。
(2) MQTT プロトコルにより制御メッセージをパブリシュします。
(3) 制御メッセージを受け、制御メッセージのハッシュ値(MD5)を計算します。
(4) クラウド ID（PD Repeater 上の便宜上の番号)、制御メッセージのハッシュ値(MD5)、トピックと共に制御メッセーを下流のソフトウェアモジュールに転送します。
(5) 制御メッセージに従って処理を行います。
(6) 処理結果と共に制御メッセージのハッシュ値(MD5)を応答メッセージに載せて返します。
(7) MQTT プロトコルにより制御メッセージをパブリシュします。
(8) 応答メッセージ上のハッシュ値(MD5)と（１）で計算したハッシュ値(MD5)を比較することで、どの制御メッセージに対する応答であるかを明確にすることができます。

2-5-2. PD Repeater の下流方向メッセージ

Openblocks IoT Family に搭載される IoT データ制御機能をそのまま利用する上で本項目は特に意識する必要はありません。

■PD Repeater が流のソフトウェアモジュールに転送するメッセージのフォーマットを示します。

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

Cloud ID (1Byte)：PD Repeater 上の便宜上割り当てている送受信先プロトコルの番号
Sub ID(1Byte)：PD Repeater 上の便宜上割り当てている送受信先のサブ番号(0x00 又は 0x01)
Header Size(2Bytes)：PD Reperater で付加された Header のサイズ
Header：PD Reperater で付加された MQTT のトピックや HTTP の応答ヘッダ
Payload：クラウドから送られて来た制御メッセージ

■Cloud ID とヘッダーの内容を示します。

クラウド	Cloud ID	下流方向制御	ヘッダーの内容
PD Exchange	0x00	対応	Command ID, ApplicationID
MQTT サーバー	0x01	対応	MQTT 受信トピック
Watson IoT for Device	0x02	対応	MQTT 受信トピック
Amazon Kinesis	0x03		
<予約>	0x04		
<予約>	0x05		
WEB サーバー	0x06		
AWS IoT	0x07	対応	MQTT 受信トピック
MS Azure Event hubs	0x08		
Watson IoT for Gateway	0x09	対応	MQTT 受信トピック
MS Azure IoT Hub	0x0a	対応	MQTT 受信トピック
Toami for DOCOMO	0x0b		
ドメインソケット	0x0c		
KDDI IoT クラウド Standard	0x0d		
IoT デバイスハブ	0x0e	対応	MQTT 受信トピック
PH 社独自仕様 WEB サーバー	0x0f	対応	HTTP 応答ヘッダー
Google IoT Core	0x10	対応	MQTT 受信トピック
TCP	0x11	対応	接続先 IP アドレスとポート番号

クラウド ID とヘッダの内容

Watson IoT については、ペイロードを”d”キーの JSON オブジェクトとしているため、ハッシュ値（MD5）を得た後、デコードし”d”キーの JSON オブジェクトのみを制御メッセージとして出力しています。

IoT デバイスハブについては、ペイロードを”parameters”キーの JSON オブジェクトとしているため、ハッシュ値（MD5）を得た後、デコードし”parameters”キーの JSON オブジェクトのみを制御メッセージとして出力しています。

TCP の接続先 IP アドレスとポート番号は、次の JSON 文字列で付加されます。

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

2-5-3. Modbus クライアント／サーバーの下流方向制御

PD Handler Modbus の下流方向制御について説明します。
PD Handler Modbus の下流方向制御を利用するためには、Modbus に用いられるファンクションコードを理解しておく必要があります。
PD Handler Modbus に用いられているファンクションコードを示します。

コード	名称	機能説明
0x01	Read Coils	デジタル出力に設定されているビット値を読み込みます。
0x02	Read Discrete Input	デジタル入力のビット値を読み込みます。
0x03	Read Holding Registers	レジスタ出力に設定されている値を読み込みます。
0x04	Read Input Registers	レジスタ入力の値を読み込みます。
0x05	Write Single Coil	デジタル出力１ビットのビット値を設定します。
0x06	Write Single Register	レジスタ出力１レジスタの値を設定します。
0x07	Read Exception Status	クライアント／サーバー間でエラーステータスを通知します。
0x09	Write Single Discrete Input	デジタル入力１ビットのビット値を設定します。
0x0a	Write Single Input Register	レジスタ入力１レジスタの値を設定します。
0x0f	Write Multiple Coils	連続する複数のデジタル出力のビット値を設定します。
0x10	Write Multiple Registers	連続する複数のレジスタ出力の値を設定します。
0x11	Report Slave ID	接続可能なスレーブ（サーバー）機器の ID を通知します。
0x13	Write Multiple Discrete Input	連続する複数のデジタル入力のビット値を設定します。
0x14	Write Multiple Input Registers	連続する複数のレジスタ入力の値を設定します。
0x16	Mark Write Registers	レジスタ出力をマスクします。
0x17	Write and Read Registers	連続する複数のレジスタ出力の値を設定し、その値を読み込みます。

PD Handler Modbus に用いられているファンクションコード

ここで、0x09, 0x0a, 0x13, 0x14 物理的な入力を持たない PD Handler Modbus Server のデジタル入力もしくはレジスタ入力をクラウド側から設定できるよう用意された、本来の Modbus プロトコルには存在しない独自拡張のファンクションです。

Modbus ファンクションコードは、PD Handler Modbus のコンフィグファイル（pd_handler_modbus_client.conf, pd_handler_modbus_server.conf）もしくは下流方向制御において JSON 文字列の”function”キーに設定されます。
また、CVS ファイル（pd_handler_modbus_client.csv）においては「読込方式」として第３カラムに設定されます。

“0x”から始まる 16 進表記のファンクションコードを文字列と”function”キーに設定することも可能ですが、16 進表記とは別に整数表記と文字列表記を用いることも可能です。
“function”キーに用いるファンクションコードの別称を示します。

コード	名称	整数表記	文字列表記
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

“function”キーに用いるファンクションコードの別称

2-5-3-1. Modbus クライアント

PD Handler Modbus Client の下流方向制御について説明します。
PD Handler Modbus Client の下流方向制御に用いる JSON 文字列のオブジェクトを示します。

キー	データ型	必須	説明
protocol	文字列	○	TCP 接続機器の場合は'tcp'、RTU 接続機器の場合は'rtu'
node	文字列	△	TCP 接続 PLC 機器の IP アドレス
port	文字列	△	TCP 接続 PLC 機器のポート番号
device	文字列	△	RTU 接続 PLC 機器のデバイスファイル
unit	整数	○	PLC 機器の ModbusID、1～247 又は 255(TCP 接続のみ
function	文字列又は整数	○	第 2-5-3 章に説明するファンクションコードの内、次表「PD Handler Modbus Client で利用可能なファンクションコード」に記載されるコード
data_type	文字列又は整数	-	次の何れか 符号なし 16 ビット整数： 'uint16_t' 又は '0' 符号付き 16 ビット整数： 'int16_t' 又は '1' 符号なし 32 ビット整数/リトルエンディアン：'uint32lsb_t' 又は '2' 符号付き 32 ビット整数/リトルエンディアン：'int32lsb_t' 又は '3' 符号なし 32 ビット整数/ビッグエンディアン：'uint32msb_t' 又は '4' 符号付き 32 ビット整数/ビッグエンディアン：'int32msb_t' 又は '5' 指定されない場合は、符号なし 16 ビット整数
address	文字列又は整数	-	データが格納されている、あるいは格納する PLC 機器上の開始アドレスを設定します。 先頭が'0x'の場合は 16 進数と解釈されます。 指定されない場合は、0x00
number	文字列又は整数	-	読み書きするレジスタ数を記載します。指定されない場合は、1
values	整数配列	△	書き込むデータ（整数値）の並び

PD Handler Modbus Client の下流方向制御に用いる JSON 文字列のオブジェクト

PD Handler Modbus Client で利用可能なファンクションコードを示します。

コード	名称	利用可能
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	○

PD Handler Modbus Client で利用可能なファンクションコード

例えば、TCP 接続されている入力レジスタを読み込むのであれば、クラウドより次のような制御メッセージを送ります。

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

対する応答メッセージは、次のようになります。

```
{
    "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]
}
```

ここで”reply_to”は、制御メッセージのハッシュ値（MD5）です。

シリアル接続の PLC 機器の出力レジスタに値を書き込むのであれば、クラウドより次のような制御メッセージを送ります。

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

ここで”data_type”が 32bits の場合、”values”は上位／下位の 16bits に分割されて処理されるため、”number”が 1 であっても、”function”は Write Multiple Registers を使用しなくてはなりません。

対する応答メッセージは、次のようになります。

```
{
    "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]
}
```

なお、”protocol”,”node”,”port”,”device”で指定される PLC 機器（への接続方法）は、Modbus クライアントデバイスとして登録され使用設定が有効になっているものに限られます。

2-5-3-2. Modbus サーバー

PD Handler Modbus Server の下流方向制御について説明します。
PD Handler Modbus Server の下流方向制御に用いる JSON 文字列のオブジェクトを示します。

キー	データ型	必須	説明
function	文字列又は整数	○	第 2-5-3 章に説明するファンクションコードの内、次表「PD Handler Modbus Server で利用可能なファンクションコード」に記載されるコード
data_type	文字列又は整数	-	次の何れか 符号なし 16 ビット整数： 'uint16_t' 又は '0' 符号付き 16 ビット整数： 'int16_t' 又は '1' 符号なし 32 ビット整数/リトルエンディアン：'uint32lsb_t' 又は '2' 符号付き 32 ビット整数/リトルエンディアン：'int32lsb_t' 又は '3' 符号なし 32 ビット整数/ビッグエンディアン：'uint32msb_t' 又は '4' 符号付き 32 ビット整数/ビッグエンディアン：'int32msb_t' 又は '5' 指定されない場合は、符号なし 16 ビット整数
address	文字列又は整数	-	データが格納されている、あるいは格納する PLC 機器上の開始アドレスを設定します。 先頭が'0x'の場合は 16 進数と解釈されます。 指定されない場合は、0x00
number	文字列又は整数	-	読み書きするレジスタ数を記載します。指定されない場合は、1
values	整数配列	△	書き込むデータ（整数値）の並び

PD Handler Modbus Server の下流方向制御に用いる JSON 文字列のオブジェクト

PD Handler Modbus Client とは異なり PD Handler Modbus Server 自身のレジスタマップに対する操作を行うものであるため、"node"や"device"キーの設定は必要としません。

PD Handler Modbus Server で利用可能なファンクションコードを示します。

コード	名称	利用可能
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	○

PD Handler Modbus Server で利用可能なファンクションコード

例えば、入力レジスタを読み込むのであれば、クラウドより次のような制御メッセージを送ります。

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

対する応答メッセージは、次のようになります。

```
{
    "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]
}
```

ここで”reply_to”は、制御メッセージのハッシュ値（MD5）です。

レジスタマップに”node”, “port”もしくは”device”の区別はありませんが、PD Repeater からは異なるデバイス番号を持つ個別の Modbus サーバーデバイスと位置付けられるため、

応答メッセージには、PD Repeater から制御メッセージを受けた UNIX ドメインソケットと対をなす”node”,”port”もしくは”device”が付加されます。

入力レジスタに値を書き込むのであれば、クラウドより次のような制御メッセージを送ります。
入力レジスタは本来物理的に AD コンバータ等を介してセットされるレジスタですが、PD Handler Modbus Server では下流方向制御によりレジスタマップを書き換えることができます。

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

ここで”data_type”が 32bits の場合、”values”は上位／下位の 16bits に分割されて処理されるため、”number”が 1 であっても、”function”は Write Multiple Input Registers を使用しなくてはなりません。

対する応答メッセージは、次のようになります。

```
{
    "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 は、PD Repeater から制御メッセージ（JSON 文字列）を受け取り、予め設定されたキーと値の一致を持ってユーザーが用意する実行オブジェクトもしくはシェルスクリプトを実行します。(PD Agent での比較条件は文字列型での完全一致となります。)
PD Agent はユーザー定義のコンフィグ設定を用いることで下流方向の制御機能を持たない PD Handler BLE 等と連携することを前提に用意されているアプリケーションですが、本章では PD Handler とは連携させず、PD Agent をユーザー定義デバイスとして登録し単独で動作させる方法を例にその使用方法を説明します。

2-5-4-1. ユーザー定義デバイスの登録と送受信先の設定

1．WEB UI の「サービス」→「基本」→「ユーザーデバイス登録」タブより、PD Agent をユーザー定義デバイスとして登録します。
送受信先が複数ある場合等、複数のデバイスとして運用する場合は、複数登録します。
サービス機能の基本設定については、「OpenBlocks IoT Family 向け WEB UI セットアップガイド」の第 5 章を参照して下さい。
ユーザー定義のデバイスの登録とは、デバイス番号を割り振りメモ情報を付与する作業です。

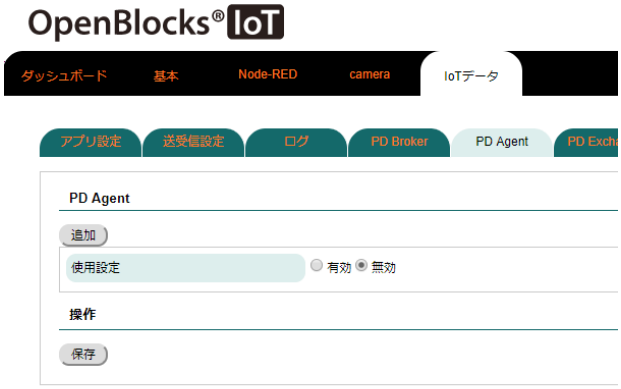
2．WEB UI の「IoT データ」→「アプリ設定」タブのアプリケーション制御メニューより、PD Agent が起動されるよう設定します。
アプリケーションの起動設定については、第 2-3 章を参照して下さい。

3．WEB UI の「IoT データ」→「送受信設定」タブの「送受信設定」メニューより、送受信先の選択と送受信設定の内個々のデバイスに依存しない設定を行います。
送受信設定については第 2-4-2 章を参照して下さい。

4．WEB UI の「IoT データ」→「送受信設定」タブの「デバイス設定（ユーザー定義)」メニューよりユーザー定義デバイスの送受信設定を行います。
ユーザー定義デバイスの送受信設定については第 2-4-8 章を参照して下さい。

2-5-4-1. PD Agent の設定

WEB UI の「IoT データ」→「PD Agent」タブより、PD Agent の設定を行います。

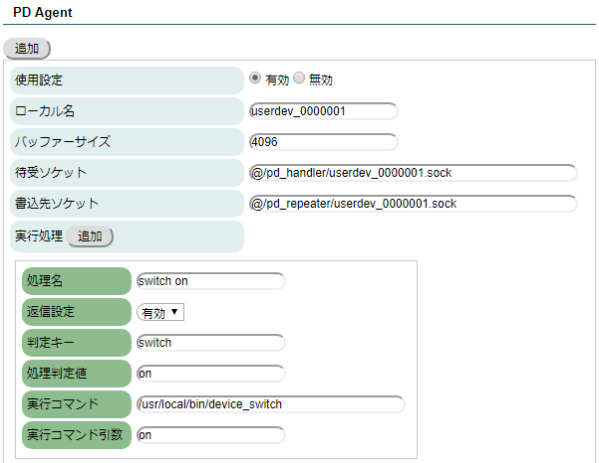


初期状態では左図のようになっています。

「使用設定」を「有効」にすると設定項目が表示されます。

複数のデバイスに対する設定行う場合は「追加」をクリックし設定フォームを追加します。

「使用設定」を「有効」にすると設定項目が表示されます。



使用設定: この設定を有効にするか無効にするか選択します。

ローカル名: 通常はデバイスに付与されたデバイス番号とします。

バッファーサイズ:データの最大サイズを設定します。デバイス側の設定と一致させて下さい。単位はバイトです。

待ち受けソケット: 制御メッセージを受け取る UNIX ドメインソケットのパス名です。

本例では、

@/pd_handler/userdev_0000001.sock

に設定します。

詳細は欄外に記載します。

書込先ソケット: 応答メッセージを送る UNIX ドメインソケットのパス名です。

本例では、

@/pd_repeater/userdev_0000001.sock

詳細は欄外に記載します。

実行処理（追加）: 以下に記載する判定キー・処理判断値・実行コマンド・実行コマンド引数を複数設定したい場合に設定フォームを追加します。

処理名： 処理の名称を記載します。

送信設定：「有効」とすることで応答メッセージを送ります。実行されるコマンドに応答メッセージを返す機能が無い場合等に用います。

判定キー：「実行コマンド」に指定されるオブジェクトの実行条件として評価される制御メッセージに含まれる JSON 文字列のキーを設定します。

処理判定値：「実行コマンド」に指定されるオブジェクトの実行条件として評価される制御メッセージに含まれる JSON 文字列の値を設定します。

実行コマンド：「判定キー」に指定されるキーと「処理判断値」に指定される値が制御メッセージに含まれる場合に実行される実行オブジェクトもしくはシェルスクリプトのパス名を設定します。

実行コマンド引数:実行コマンドの引数を設定します。

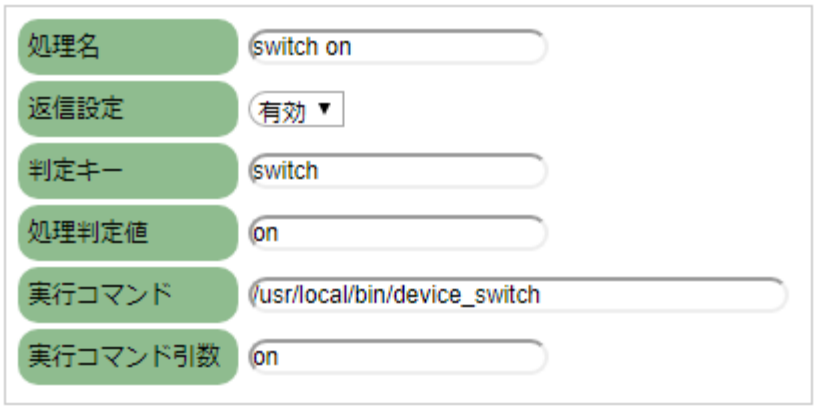
※待ち受けソケット
制御メッセージを受け取る UNIX ドメインソケットのパス名です。
デバイスとしてユーザー定義デバイスを用い、PD Repeater から直接制御メッセージを受け取る本例では、@/pd_handler/userdev_0000001.sock に設定します。
先頭の@は、パス名が Abstract UNIX ドメインソケット名であることを意図し、Abstract UNIX ドメインソケット名をサポートしない実行オブジェクトから待ち受ける場合は、/tmp/pd_agent_0000001.sock 等のファイルシステムに実在するパス名とします。

※書込ソケットのパス名
応答メッセージを送る UNIX ドメインソケットのパス名です。
デバイスとしてユーザー定義デバイスを用い、PD Repeater へ直接応答メッセージを送る本例では、@/pd_repeater/userdev_0000001.sock に設定します。
先頭の@は、パス名が Abstract UNIX ドメインソケット名であることを意図し、Abstract UNIX ドメインソケット名をサポートしない実行オブジェクトへ送る場合は、/tmp/anyobject_0000001.sock 等のファイルシステムに実在するパス名とします。

2-5-4-2. PD Agent の設定と制御メッセージ

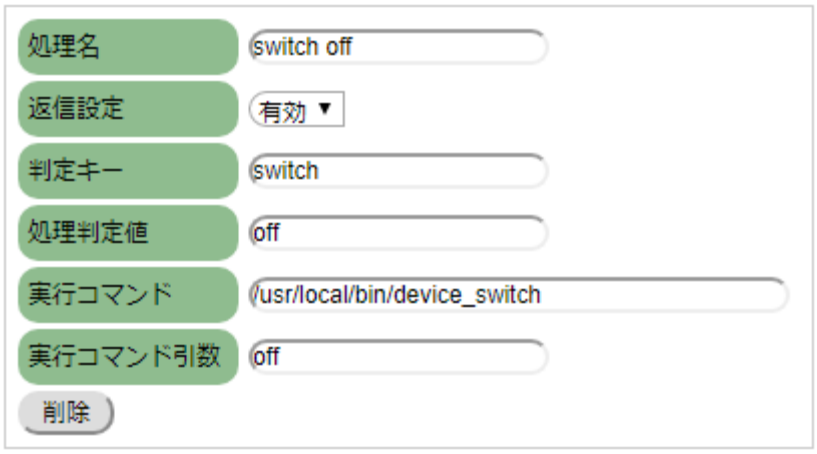
PD Agent の設定と制御メッセージの例を幾つか示します。

１．次の設定に対する制御メッセージは　{“switch”:”on”}　となります。



一致する制御メッセージを受け取ると　‘/usr/local/bin/device_switch on’ が実行されます。

２．次の設定に対する制御メッセージは　{“switch”:”off”}　となります。



一致する制御メッセージを受け取ると　‘/usr/local/bin/device_switch off’ が実行されます。

３．次の設定に対する制御メッセージは　{“report”:”do”}　です。

一致する制御メッセージを受け取ると　‘/usr/local/bin/device_report’ が実行されます。

４．上記の例において制御メッセージとして{“switch”:”on”, “report”:”do”}場合は、‘/usr/local/bin/device_switch on’と‘/usr/local/bin/device_report’ が実行されます。

複数の実行条件が適用されるメッセージを受信した場合、連続で処理が実行されます。
実行処理順については担保されませんので、ご注意ください。

2-5-4-3. 環境変数への継承

制御メッセージ（JSON 文字列）の内、実行オブジェクトもしくはシェルスクリプトの実行条件として用いられるキーと値を含め、その値が文字列か数値であれば、それらは環境変数に設定され実行オブジェクトもしくはシェルスクリプトへ継承されます。

例えば、以上の例において {“switch”:”on”, “report”:”do”,”page”:40} の制御メッセージを受け取った場合、コマンドの実行においては次の環境変数が継承されます。

switch=”on”
report=”do”
page=40

また、制御メッセージの内容に関わらず PD Agent は、実行オブジェクトもしくはシェルスクリプトの実行において次の値を環境変数として実行オブジェクトもしくはシェルスクリプトへ継承します。

環境変数	データ型	説明
request_cloud_id	整数	第 2-5-2 章に記載される Cloud ID
request_sub_id	整数	第 2-5-2 章に記載される Sub ID
request_header	文字列	第 2-5-2 章に記載されるヘッダー
request_md5	文字列	第 2-5-2 章に記載される制御メッセージのハッシュ値
request_payload	文字列	クラウドから送られてきたペイロード
agent_localname	文字列	第 2-5-4-1 章の「ローカル名」に設定されるローカル名
agent_bind	文字列	第 2-5-4-1 章の「待ち受けソケット」に設定される待ち受けソケット名
agent_push_to	文字列	第 2-5-4-1 章の「書込ソケット」に設定される書込ソケット名
agent_buffer_size	整数	第 2-5-4-1 章の「バッファサイズ」に設定されるバッファサイズ

実行オブジェクトもしくはシェルスクリプトに環境変数として継承されるパラメータ

2-5-4-4. 応答メッセージ

第 2-5-4-1 章の「送信設定」が「有効」に設定されている場合は、実行ステータスとして次の応答メッセージを返します。

実行コマンドを起動した：

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

実行コマンドの実行に失敗した：

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

実行コマンドの実行を終了した：

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

一致するキーと値が存在しない：

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

制御メッセージが JSON 文字列でない：

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

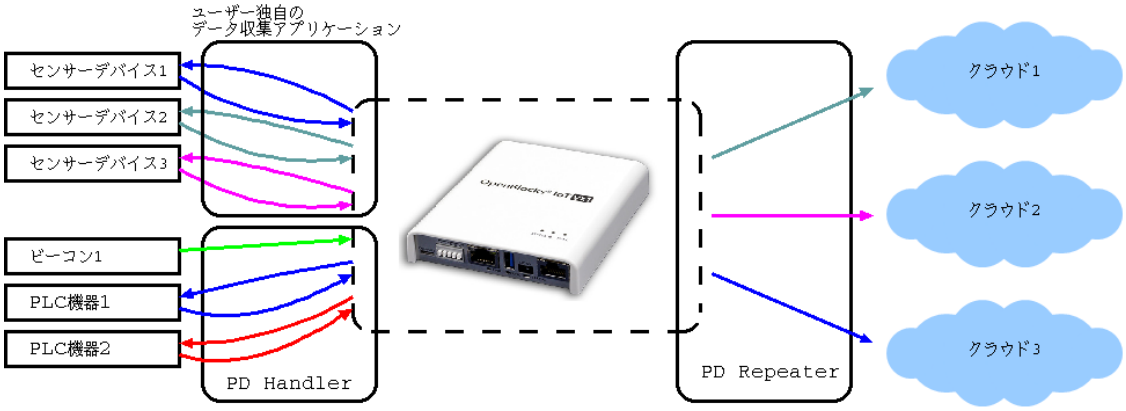
ここで、*md5* は、第 2-5-2 章に記載される制御メッセージのハッシュ値、{“*key*”:”*value*”} は、一致した「判断キー」と「処理判断値」です。
実行ステータスは、実行オブジェクトもしくはシェルスクリプトを呼び出す execv()の戻り値であり、実行オブジェクトもしくはシェルスクリプトの処理結果を示すものではありません。　処理の完全性を求めるのであれば応答メッセージは実行オブジェクトもしくはシェルスクリプトから agent_push_to 環境変数で継承される UNIX ドメインソケットに返すようにして下さい。

第 3 章 カスタマイズ

OpenBlocks IoT Family の IoT データ制御機能とユーザーが独自に用意するアプリケーションとの連携方法について説明します。

3-1. 独自開発データ収集アプリケーション

データの収集において弊社用意するアプリケーション(PD Handler)を用いず、各デバイス等からデータを取得するアプリケーションをユーザーが独自に開発し、PD Repeater と連携しクラウドにデータを送信する方法を説明します。
構成イメージは次の通りです。



3-1-1. ユーザー定義デバイスの登録と送受信先の設定

1．WEB UI の「サービス」→「基本」→「ユーザーデバイス登録」タブより、独自に開発したアプリケーションをユーザー定義デバイスとして登録します。
サービス機能の基本設定については、「OpenBlocks IoT Family 向け WEB UI セットアップガイド」の第 5 章を参照して下さい。
ユーザー定義のデバイスの登録とは、デバイス番号を割り振りメモ情報を付与する作業です。

2．独自に開発したアプリケーションが利用するデバイスファイルやハードウェアが弊社用意するアプリケーション(PD Handler)のそれと競合する場合は、WEB UI の「IoT データ」→「アプリ設定」タブのアプリケーション制御メニューより、該当するアプリケーションがが起動しないように設定します。

3．独自開発のアプリケーションの起動／停止制御については第 3-3 章を参照して下さい。

4．WEB UI の「IoT データ」→「送受信設定」タブの「送受信設定」メニューより、送受信先の選択と送受信設定の内個々のデバイスに依存しない設定を行います。

送受信設定については第 2-4-2 章を参照して下さい。

５．WEB UI の「IoT データ」→「送受信設定」タブの「デバイス設定（ユーザー定義)」メニューよりユーザー定義デバイスの送受信設定を行います。
ユーザー定義デバイスの送受信設定については第 2-4-8 章を参照して下さい。

3-1-2. PD Repeater へのデータ書き込み

PD Repeater は WEB UI にて設定したデバイス番号を元に、抽象名前空間(abstract)の Unix ドメインソケットを作成します。(作成する対象は送信対象を”送信する”とし、送信先が有効でかつ local 以外が設定されているデバイスです)
この Unix ドメインソケットに対して書き込みを行った場合、書き込んだデータをクラウドへデータを送信します。
対象の Unix ドメインソケットのパス規則は以下となります。
（第 2-3 章に示すユーザー定義のコンフィグを使用する場合、このパス名を任意のパス名に変更することが出来ます。）

●PD Repeater の Unix ドメインソケット
¥0/pd_repeater/<*デバイス番号*>.sock

以下は、’{“x”:1}’を各々で PD Repeater の Unix ドメインソケットに書き込みを行ったサンプルです。

コマンドラインでの書き込みサンプルは以下となります。

※<デバイス番号>：userdev_0000001 として PD Repeater へ書き込んだ場合[※1]

```
# echo -n '{"x":1}' | socat stdin abstract-connect:/pd_repeater/userdev_0000001.sock
```

PHP でのスクリプトサンプルは以下となります。

※<デバイス番号>：userdev_0000001 として書き込んだ場合

```
<?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);
?>
```

[※1] socat コマンドはインストールされていません。そのため、”apt-get install socat”にてインストールしてください。

Node.js でのスクリプトサンプルは以下となります。

※<デバイス番号>：userdev_0000001 として PD Repeater へ書き込んだ場合※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();
```

※2 abstract-socket はインストールされていません。そのため、" npm install abstract-socket"にてインストールしてください。

3-2. 独自開発下流方向制御アプリケーション

クラウドから PD Repeater を介し制御メッセージを受け取り、何らかの処理をおこなうユーザーが独自に開発したアプリケーションを用いる場合の要点を説明します。

3-2-1. ユーザー定義デバイスの登録と送受信先の設定

ユーザー定義デバイスの登録と送受信先（クラウド）の設定については、第 3-1-1 章に示すデータ収集用アプリケーションにおける設定と同じですが、
WEB UI の「IoT データ」→「送受信設定」タブの「デバイス設定（ユーザー定義)」メニューの「受信設定」を「有効」とする必要があります。

3-2-2. PD Repeater からのデータ書き込み

PD Repeater は、クラウドから制御メッセージを受け取ると WEB UI にて設定したデバイス番号を元に定められる抽象名前空間(abstract)の Unix ドメインソケットに第 2-5-2 章に示すフォーマットで書き込みます。
Unix ドメインソケットのパス規則は以下となります。
（第 2-3 章に示すユーザー定義のコンフィグを使用する場合、このパス名を任意のパスに変更することが出来ます。）

●PD Repeater が制御メッセージを送る Unix ドメインソケット
¥0/pd_handler/<*デバイス番号*>.sock

3-3. 独自開発アプリケーションの起動／停止制御

幾つかのファイルを用意することで、WEB UI による PD Repeater の起動／停止制御に独自開発アプリケーションを連動さることができます。

3-3-1. アプリケーションの登録

連動させるアプリケーションの名称を次のファイルに登録(記載)します。

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

このファイルには、1 行に 1 個のアプリケーション名を記載します。空行及びアプリケーション名の重複は許容されません

記載例）

```
testhandler
myhandler
```

3-3-2. アプリケーションが用いるスクリプトの指定

連動させるアプリケーションが用いるスクリプトを次のファイルに設定します。

/etc/default/<アプリケーションの名称>

ファイルにおいては次のように起動・停止・状態確認の各スクリプトを指定します。

```
bootcmd_<アプリケーション名>=”<起動スクリプト>”
haltcmd_<アプリケーション名>=”<停止スクリプト>”
statuscmd_<アプリケーション名>=”<状態確認スクリプト>”
```

状態確認スクリプトの結果、” is running”または” RUNNING” が出力される場合、または Status が” inactive”,” failed”,” dead”ではない場合、WEB UI のダッシュボードでは稼働中として認識します。

設定例）/etc/default/ testhandler

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

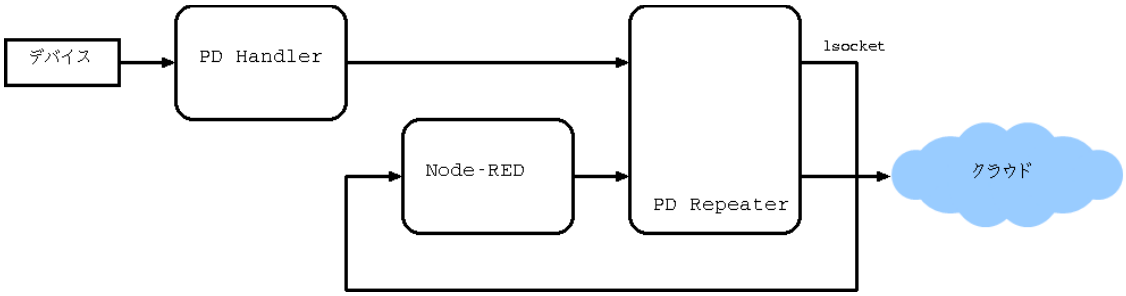
3-3-3. deb パッケージ

第 3-3-1 章に示すアプリケーション登録処理や第 3-3-2 章に示すファイルは、アプリケーションとその起動／停止スクリプトを含めて deb パッケージ（Debian GNU Linux のソフトウェアパッケージ）としてまとめておくことを推奨します。
deb パッケージの作成方法については Debian 公式ページをご確認ください。
deb パッケージの作成においては次の点に注意して下さい。

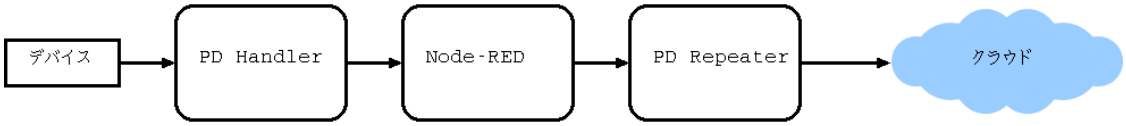
- 第 3-3-1 章に示すアプリケーション登録ファイル(/etc/default/obsiot-webui-ext-handler)は、複数の deb パッケージの postinstall または postrm スクリプトによって編集されるものです。
- ログファイルを syslog 経由で吐きだすアプリケーションの場合には、postinstall にて rsyslog をリスタートしてください。(OpenBlocks IoT Family で用いている syslog サービスは rsyslog です。)
- WEB UI で起動制御等が実施されますが、アプリケーションに用いるコンフィグファイルは生成されません。そのため、deb パッケージにひな形となるコンフィグファイルを入れておくことを推奨します。
- syslog 経由にてログを出力する設定等のコンフィグファイルが必要となります。また、出力先については通常の実ストレージ領域ではなく tmpfs 領域を推奨します。WEB UI にて” /var/webui/pd-logs”に tmpfs 領域を用意していますので、こちらに書き込んでください。尚、この領域に拡張子を”.log”として用意したファイルはログ確認タブから閲覧できます。
  ログを大量に吐き続けた場合、ファイルサイズが大きくなり tmpfs 領域を圧迫します。そのため、ログのローテーション設定を追加してください。tmpfs 溢れの観点からローテーション設定はファイルサイズでのトリガーを推奨とします。

3-4. 複雑な構成の実現

PD Handler からのデータを Node-RED フィルタリングしたい場合、WEB UI の設定だけでも Node-RED をユーザー定義デバイスとすれば、PD Repeater から lsocket を用いてデータを Node-RED に渡し、Node-RED の出力を PD Handler へ返す構成が可能ですが、



第 2-3 章に示すユーザー定義のコンフィグを使用し UNIX ドメインソケット名を変更することで、下図のようなよりスループットを意識した構成とすることが可能です。



設定ファイル(コンフィグ)は JSON 形式で記載されており、UNIX ドメインソケット名は次のキーで表記されます。

キー	形式	
bind	文字列	データを待ち受ける UNIX ドメインソケット名。　空白の場合は各アプリケーションのデフォルト値をプリフックスとし、デバイス番号を差フックスとするソケット名がもちいられる。　先頭が@の場合は抽象的ドメイン名(Abstract UNIX doamin name)であることを意図します。
push_to	文字列	データの送り先となる UNIX ドメインソケット名。　空白の場合は各アプリケーションのデフォルト値をプリフックスとし、デバイス番号を差フックスとするソケット名がもちいられる。　先頭が@の場合は抽象的ドメイン名(Abstract UNIX doamin name)であることを意図します。

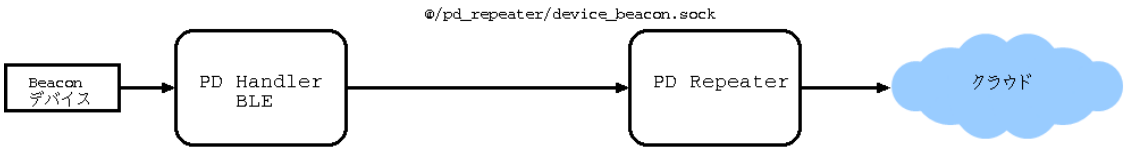
:

Beacon デバイス　（PD Handler BLE）を例に設定方法を示します。

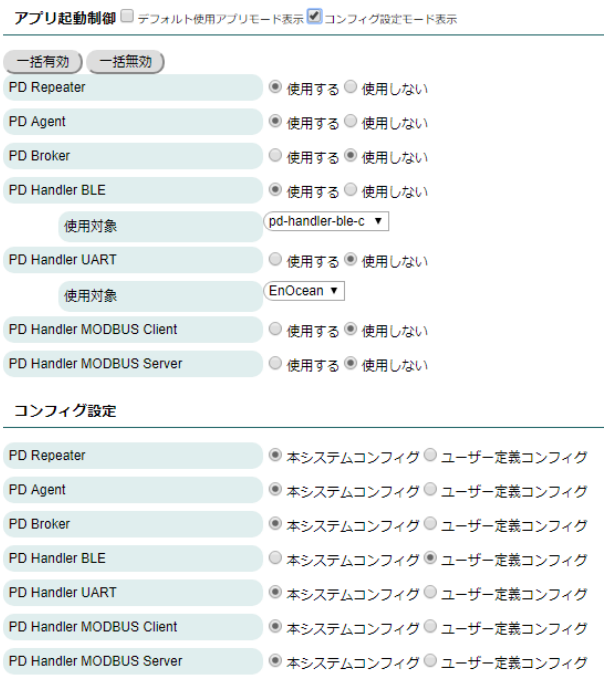
1. 第 2-3 章に示す「コンフィグ設定モード」が「本システムコンフィグ」であることを確認します。

WEB UI の「IoT データ」→「アプリ設定」タブより、「コンフィグ設定モード表示」をチェックして PD Handler BLE が「本システムコンフィグ」となっていることを確認します。

2. WEB UI の「IoT データ」→「送受信設定」タブの「送受信設定」メニューより、送受信先の設定を行います。　設定方法は第 2-4-2 章を参照して下さい。
3. WEB UI の「IoT データ」→「送受信設定」タブの「ビーコン送信設定」メニューより、ビーコンを受信し送信する設定を行います。設定方法は第 2-4-3 章を参照して下さい。
4. ここまでの設定で下図のように PD Repeater は、UNIX ドメインソケット @/pd_repeater/device_beacon.sock でデータを持ち受け、PD Handler BLE は @/pd_repeater/device_beacon.sock にデータ書き込むよう設定されます。
ここで UNIX ドメインソケット名の'@'はソケットが抽象的ドメイン名(Abstract UNIX doamin name)であることを意図するものです。

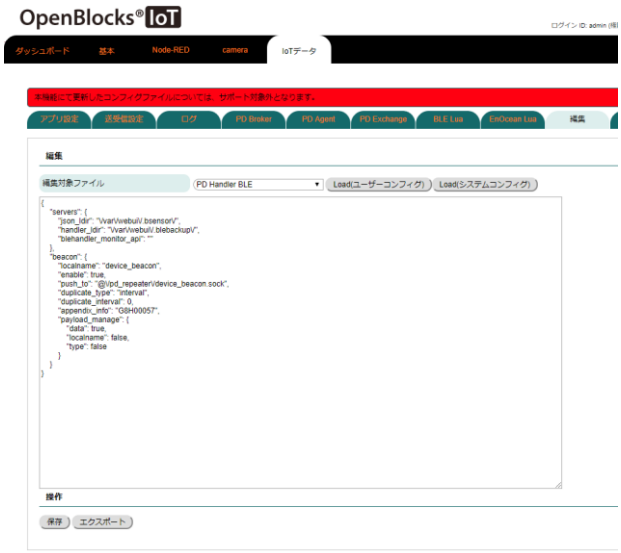


5. 「コンフィグ設定モード」を「ユーザー設定コンフィグ」に切り替えます。



WEB UI の「IoT データ」→「アプリ設定」タブより、「コンフィグ設定モード表示」をチェックして PD Handler BLE を「ユーザー設定コンフィグ」に切り替えます。

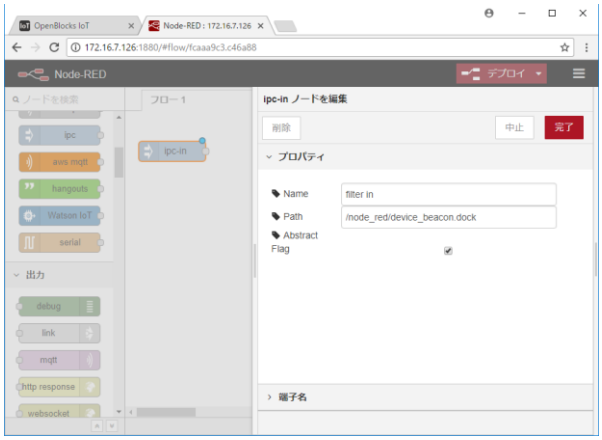
6. WEB UI により作成されたシステムコンフィグをユーザーコンフィグとしてコピーし、UNIX ドメインソケット名を@/pd_repeater/device_beacon.sock から @/node-red/device_beacon.sock に変更します。



1. WEB UI の「IoT データ」→「編集」タブより、「編集対象ファイル」としてを「PD Handler BLE」を選択します。
2. 「Load(システムコンフィグ)」をクリックします。テキストエリアに表示された設定情報（JSON 文字列）をコピーします。
3. 「Load(ユーザーコンフィグ)」をクリックします。２項で取得した設定情報（JSON 文字列）をテキストエリアにペーストします。
4. “push_to”キーの値を
5. ”@¥/node-red¥/device_beacon.sock”に変更します。
6. 「保存」をクリックします。

7. Node-RED の UNIX ドメインソケット入力 ipc-in ノードのプロパティを設定します。

Node-RED の利用方法については「OpenBlocks IoT Family 向け Node-RED スターターガイド」を参照して下さい。

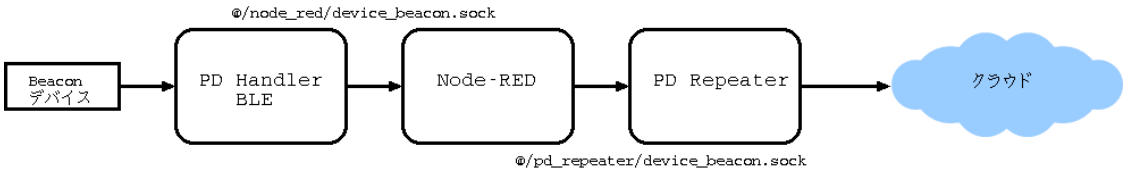


ipc-in ノードを配置し、プロパティの「Path」を”/node-red/device_beacon.sock”に設定し、「Abstract Flag」をチェツクします。

8. Node-RED の UNIX ドメインソケット出力 ipc-out ノードのプロパティを設定します。

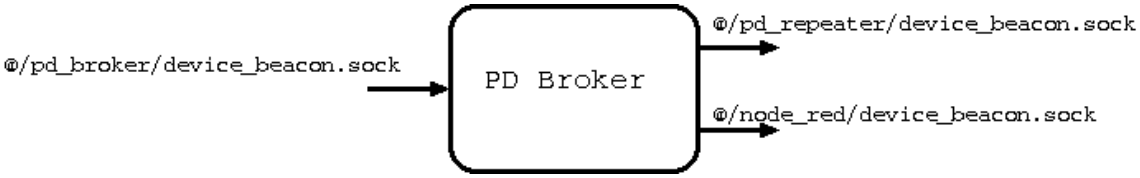
ipc-out ノードを配置し、プロパティの「Path」を”/pd_repeater/device_beacon.sock”に設定し、「Abstract Flag」をチェツクします。

以上の設定で PD Handler BLE は@/node-red/device_beacon.sock にデータ書き込み、Node-RED は@/node-red/device_beacon.sock でデータを待ち受け @/pd_repeater/device_beacon.sock に書き込むように設定されます。

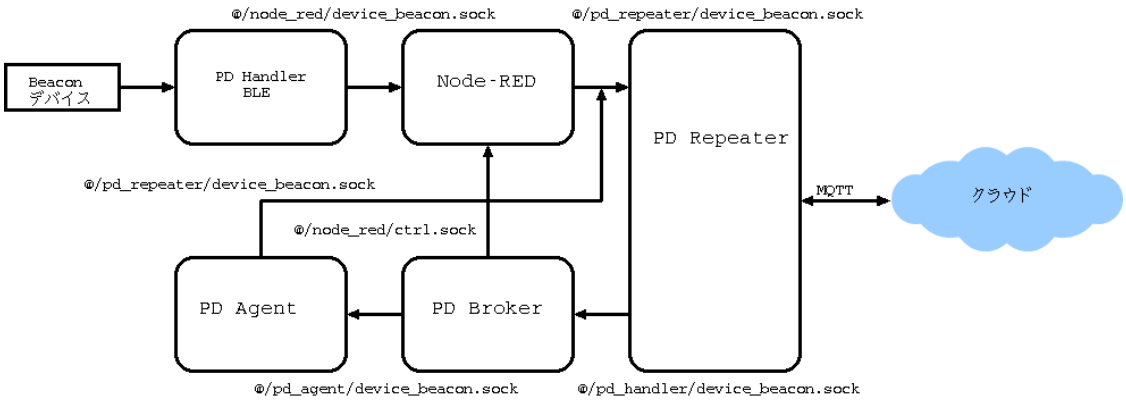


3-4-1. PD Broker

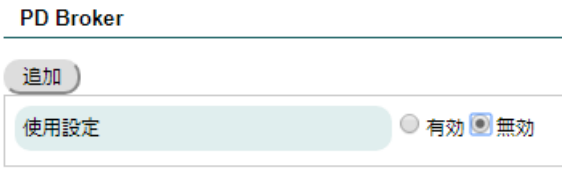
PD Broker は単一の UNIX ドメインソケット受けたデータを複数の UNIX ドメインソケットに送る、接続構成をカスタマイズするためのアプリケーションです。



例えば、下図のように PD Handler BLE でビーコンを受け、Node-RED でフィルタリングし、PD Repeater を介し MQTT 接続でクラウドへデータ送る構成で、同じ MQTT 接続を用いて制御メッセージを受け取り Node-RED のフィルタリングパラメータと PD Agent を制御しようとする場合等に用いられます。



WEB UI の「IoT データ」→「PD Broker」タブから PD Broker の設定を行うことができます。「PD Broker」タブが表示されていない場合は、「第 2-3 章 IoT データ制御のアプリ設定」を参照し、PD Broker を「使用する」に設定して下さい。



初期状態では左図のようになっています。

「有効」を選択すると設定項目が表示されます。

「有効」を選択すると下図のように表示されます。

追加:複数の待受けソケットを設定する場合にクリックします。2個目以降の入力フォームが表示されます。

バッファーサイズ:データの最大サイズを設定します。単位はバイトです

待受けソケット:データを待受る UNIX ドメインソケットのパス名を設定します。先頭が'@'の場合は抽象的ドメイン名となります。

書込みソケット:データの送り先となる UNIX ドメインソケットのパス名を設定します。先頭が'@'の場合は抽象的ドメイン名となります。

「追加」をクリックすると2個目以降の入力フォームが表示されます。

3-5. Lua 拡張

3-5-1. BLE Lua

PD Handler BLE の C 言語版(pd-handler-ble-c)では、対応していない非コネクション型の BLE センサーの対応が可能となっています。
WEB UI の「IoT データ」→「BLE Lua」タブから、pd-handler-ble-c にて対応したいセンサー用の Lua ファイルをアップロードすることにより対応可能です。
※アップロード後に、プロセスの再起動が必要となります。

更新：Lua ファイルのアップロード先のファイル一覧表示を更新します。
削除：ファイルを選択し、ボタンを押すことで対象ファイルを削除します。
ダウンロード:選択したファイルをダウンロードできます。
実行権付与:選択したファイルに実行権を付与します。通常では不要です。
アップロード:アップロードしたいファイルをダイアログにて選択後、押すことによりアップロードを行います。

アップロード先に Lua ファイル以外が存在している場合には正常に動作しません。
そのため、ファイル内容については skelton.lua を参考に作成してください。

3-5-2. EnOcean Lua

PD Handler UART の EnOcean 機能では、対応していない EnOcean デバイスへの対応が可能となっています。
WEB UI の「IoT データ」→「EnOcean Lua」タブから、対応したいセンサー用の Lua ファイルをアップロードすることにより対応可能です。
※アップロード後に、プロセスの再起動が必要となります。

更新：Lua ファイルのアップロード先のファイル一覧表示を更新します。

削除：ファイルを選択し、ボタンを押すことで対象ファイルを削除します。

ダウンロード:選択したファイルをダウンロードできます。

実行権付与:選択したファイルに実行権を付与します。通常では不要です。

アップロード:アップロードしたいファイルをダイアログにて選択後、押すことによりアップロードを行います。

アップロード先に Lua ファイル以外が存在している場合には正常に動作しません。
そのため、ファイル内容については skelton.lua を参考に作成してください。

第 4 章 補足事項

4-1.データ送信量及び回線速度について

ビーコンやデバイスからの情報取得量に対して、データ送信が遅い場合には、OpenBlocks IoT Family 内のバッファーに情報が溜まっていきます。この場合、データ送信部の改善を行わない場合には溜まり続けてしまう為、バッファーデータを確認しインターバルや取得時間間隔等を調整してください。
※バッファーデータは「サービス」→「状態」タブにてバッファーファイルのサイズを確認できます。

4-2.PD Repeater への書き込みデータフォーマット

PD Repeater は JSON 形式のデータを扱うことを前提としていますが、接続先によっては、JSON 形式でない文字列を扱うことができます。

クラウド	非 JSON 形式対応
PD Exchange	
MQTT サーバー	○
Watson IoT for Device	
Amazon Kinesis	○
WEB サーバー	
AWS IoT	
MS Azure Event hubs	○
Watson IoT for Gateway	
MS Azure IoT Hub	○
Toami for DOCOMO	
ドメインソケット	○
KDDI IoT クラウド Standard	
IoT デバイスハブ	
PH 社独自仕様 WEB サーバー	
Google IoT Core	○
TCP	○

非 JSON 形式の文字列に対応可能な接続先

4-3.PD Repeater へのデータの書き込みサイズ

PD Repeater へのデータの書き込みサイズはデバイス毎に変更することが可能ですが、大きくすればするほどメモリを消費しパフォーマンスも低下するため、データのサイズに応じ可能な限り小さくしておくことを推奨します。
また、クラウドによって規定されている最大値も異なるため、使用するクラウドの仕様をご確認の上、設定して下さい。

4-4.PD Repeater のバッファーサイズ

PD Repeater は送信用のバッファーとして一時溜めこみを行う為、DB にバッファーとして書き込みます。DB のサイズ上限のデフォルトは 16Mbyte です。このサイズを超えた場合、新しいデータは廃棄され、DB のサイズが 8Mbyte 以下になるまでデータの受信は行われません。

4-5.PD Repeater のエラー時の再送信

ネットワークの通信状況によって、PD Repester からクラウドに対しての送信が失敗することがあります。この時、連続 4 回失敗した場合や想定外のエラー状態が発生した場合には、1 分後に再送信処理を開始します。

4-6. 独自開発アプリケーションの設定ファイルについて

ユーザー側にて作成した独自開発アプリケーションの設定ファイル（Config）作成機能は存在していません。ユーザー様側にて各筐体に保存する必要がありますので、ファイルアップロード機能等をご使用ください。

4-7. Node-RED へのデータ経由方法について

PD Repeater から Node-RED へのデータは Unix ドメインソケット経由となります。
PD Repeater が各データにて書き込む Unix ドメインソケットのパスは以下のものとなります。

<ソケットパスプレフィックス>/*<デバイス番号>*.sock

PD Repeater においては、パス名の先頭を'@'とすることで Abstract Domain Socket として扱います。　（第 2-4-2-17 章を参照して下さい。）

Node-RED 側では、対象デバイスの ipc-in ノードを入力として Flow に用意してください。
OpenBlocks IoT Family にインストールされてる ipc-in ノードは、Abstract Domain Socket が扱えるよう拡張されています。
パス名は ipc-in ノードのプロパティの「Path」に設定して下さい。パス名を Abstract Domain Socket として扱いたい場合は、同プロパティの「Abstract Flag」をチェックして下さい。PD Repeater とは異なり、パス名の先頭に'@'を記載する必要はありません。

4-8. BLE デバイスとして追加したビーコンについて

WEB UI に BLE デバイスとして登録し送信対象として設定したセンサーやビーコンは個別に扱われます。
この場合において、特にビーコンデバイスはビーコンの送信設定との依存が無くなります。そのため、ビーコン送信設定の制御タイプ、データフィルタ等は適用されません。
また、対応しているセンサー付きビーコンの PD Repeater へ渡すデータは、解析されたセンサーデータとなります。しかし、通常のビーコンの場合は時刻/デバイス ID/メモ情報を PD Repeater へ渡します。

4-9. WEB サーバーへのデータ送信について

PD Repeater(OpenBlocks IoT のファームウェア)側からは、指定した URL の Endpoint に対して HTTP POST メソッドで送信します。
そのため､HTTP サーバ側では HTTP 200 番台のステータスコードを返す必要があります。
HTTP 200 番台のステータスコードを返却する際、HTTP ヘッダやペイロードで必要なものはありません。
尚、HTTP 200 番台以外のステータスが返された場合、PD Repeater(OpenBlocks IoT のファームウェア)側ではエラーとして扱います。

4-10. Handler コンフィグユーザー設定

「IoT データ」→「アプリ設定」タブより、Handler コンフィグ設定を「ユーザー定義コンフィグ」を選択し保存することで使用するものがユーザー定義のものへと切り替わります。
ファイル編集自体については､「IoT データ」→「編集」タブから編集を適用してください。
尚、本機能を用いた場合、サポート対象外となります。

4-11. PH 社独自仕様 Web サーバー（PD Web）

HTTP を用いた双方向通信を実現するために弊社が独自に仕様を定義した Web サーバーです。

4-11-1. PD Web の概要

- PD Web は HTTP において双方向通信を提供します。
  - 下流方向へのペイロード転送は、上流方向への POST に対する応答メッセージとして提供されます。
  - 上流方向へ送るべきペイロードが存在しない場合、PD Repeater は、「受信ポーリング間隔」に指定される間隔で空接続を行います。
  - 下流方向へ送るべきペイロードが存在しない場合、Web サーバーは応答ヘッダーのみを返します。PD Repeater は応答メッセージが空でない場合、push_to キーに指定される下流モジュールの UNIX ドメインソケットにペイロードを転送します。
- PD Web の認証方式は双方向のトークン認証です。
  - PD Repeater と Web サーバーには､デバイス毎に指定された ID と鍵(Key)の情報を持ちます。
  - PD Repeater は、リクエストヘッダに記載するバージョン番号（PD Web 仕様の

バージョン番号）・ID・タイムスタンプ・ペイロードのハッシュ値(MD5)から構成される署名対象文字列（リクエスト用）を鍵(Key)で著名したハッシュ値(Shignature)をリクエスト用のトークンとします。

- Web サーバーは、バージョン番号・ID・タイムスタンプ・ペイロードのハッシュ値(MD5)にリクエストヘッダのトークンを加えた署名対象文字列（応答用）を鍵(Key)で著名したハッシュ値(Shignature)を応答用のトークンとします。
- 応答ヘッダーの Shignature もしくはペイロードの MD5 が期待される値と一致しない場合、以降、PD Repeater は応答ヘッダーに期待される Shignature が返されるまでペイロードを空とします。
- Web サーバー一括の BASIC 認証を併用することも可能です。

- エラーハンドリング
  - PD Repeater と web サーバー間のエラーハンドリングは HTTP ステータスコードのみで行われます。ステータスが 200～299 以外の場合は、応答ヘッダーの内容に関わらず下流モジュールへのペイロードの転送は行われません。
- ペイロードのフォーマットとトランザクション管理
  - 上流方向のペイロードのフォーマットは JSON 文字列であり、複数のメッセージをまとめて転送できるようメッセージの数に限らず最上位階層は配列となります。
  - PD Repeater の下流方向のメッセージフォーマットは第 2-5-2 章に示す通りですが、ペイロードのフォーマットは規定されていません。下流方向のペイロードのフォーマットは PD Repeater からペイロードを受け取る下流のモジュールの仕様に依存します。
  - ペイロードのトランザクション（到達や再送処理）管理は、Web サーバーと下流のモジュール間で行われることを前提とし、PD Repeater はトランザクションの管理を行いません。
  - 弊社が提供する PD Handler Modbus と PD Agent はトランザクション管理用途に下流方向のペイロードのハッシュ値(MD5)を JSON 文字列”reply_to”キーの値として返す仕様となっています。

4-11-2. PD Web の HTTP ヘッダー

PD Web で規定されている HTTP ヘッダーは次の通りです。

HTTP ヘッダー	内容
X-Pd-Web-Version	PD Web の仕様のバージョン番号
X-Pd-Web-Id	クライアントの ID
X-Pd-Web-Time	RFC3339 準拠のタイムスタンプ
X-Pd-Web-Md5	ハッシュ値
X-Pd-Web-Signature	ヘッダ情報と鍵(Key)から作成されたハッシュ値
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
```

PD Repeater への応答ヘッダーの例を示します。

```
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. PD Web のトークン

PD Repeater で作成されるリクエストヘッダのトークン(X-Pd-Web-Sinature)は、Web サーバーにおいて次の PHP スクリプトにより再生できます。

```
$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);
```

ここで$keyは、は$_SERVER['HTTP X PD WEB ID'] と対をなす予めWebサーバーに保存された鍵(Key)となります。再生した$signature と$_SERVER['HTTP X PD WEB SIGNATURE'] を比較することで認証します。

応答ヘッダーのトークン(X-Pd-Web-Sinature)は、Web サーバーにおいて次の PHP スクリプトにより作成します。

```
$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);
```

ここで、$payload は、ペイロードの文字列、送信すべき文字列（下流方向の制御メッセージ）が無い場合は、$payload="";とします。
リクエストヘッダのトークンとは異なり、被署名文字列に PD Repeater から送られて来たリクエストヘッダのトークン($_SERVER['HTTP X PD WEB SIGNATURE'])が、含まれる点に注意して下さい。

4-11-4. Web サーバー(PHP スクリプト)の実装例

Web サーバー(PHP スクリプト)の実装例を示します。

```
<?php
/*
 * 本 PHP スクリプトは PD Repeater の PD Web を利用するための、
 * サーバ側 PHP スクリプトのサンプルです.
 *
 * 本スクリプトを動作させるためには、次の SQL 構文で作成された SQLite3 データベース
 * (スクリプト中の $db_file)が必要となります.
 *
 *      CREATE TABLE client (id TEXT, key TEXT, flags INTEGER, payload BLOB, md5 TEXT);
 *      CREATE INDEX index_client ON client (id);
 *
 * ここで、id は、各センサーデバイス毎に設定する PD Web の ID, key はトークンを
 * 作成するための鍵、contens は PD Repeater を介して各センサーデバイスへ送信する
 * JSON 文字列(ペイロード)、
 * md5 は JSON 文字列の MD5 ハッシュ値、flags はペイロードの送信状態を示すコードで
 * 0:送信済、1:処理中、2:未送信 を意味します.
 *
 * 以下に初期値の設定例を示します.
 *
 *      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');"
 *
 * ここで、d29e8a13452e5bc5218d9df7e6ea991 は、'{"any_key":"any_value"}' の MD5 値です.
 *  Linux OS では、次のコマンドで取得することができます.
 *
 *      echo -n '{"any_key":"any_value"}' | md5sum
 *
 * ペイロードを送信するには SQL の UPDATE を用いて flags を 2:未送信 に変更します.
 *
 *      UPDATE client SET flags = 2 WHERE id = 'id00';
 *
 * 勿論ペイロードと MD5 値を合わせてセットすることも可能です.
 *
 *      UPDATE client
 *                      SET flags = 2,
 *                          payload = '{"any_key":"new_value"}',
 *                          md5 = '94f030ebd7bed4a5ee08fc6fa75ae64e'
 *                      WHERE id = 'id00';
 *
 * 送信を終えるを PHP スクリプトは flag を 1 に更新し、'reply_to' キーを含む応答ペイロードを
 * 待ちます.
 *
 * 応答ペイロードの例
 *
 *       {"reply_to":"94f030ebd7bed4a5ee08fc6fa75ae64e","result":"done"}
 *
 * 応答ペイロードは PD Repeater を介してペイロードを受け取る各センサーデバイス(のハンドラ)が
 * PD Repeater 介して返すものです.
```

```
* PD Agent や PD Handler Modbus は 'reply_to' キーで payload の MD5 値をハンドリングしますが、
* 独自のハンドラを用いる場合は、独自の応答ペイロードとすることも可能です.
*
* 応答ペイロード受け取ると PHP スクリプトは、データベース上の md5 と reply_to の値を比較し、
* flag を 0 に更新します.
* 受信ペイロードに応答ペイロードが含まれていない場合、PHP スクリプトは flag を 2 に戻し、
* 再送します.
*
*/
    $dump_file = '/tmp/dump.txt';
    $db_file = '/tmp/pd_web.db';

    /* HTTP ヘッダーの確認 */
    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']))) {
        /* HTTP ヘッダーが不正な場合 Bad Request 400 を返す. */
        http_response_code (400);
        exit;
    }

    /* SQLite3 データーベースの読み込み */

    /* 本コードはサンプルであるため、SQL インジェクション対策を省略しています.
       実運用に利用するためには $_SERVER['HTTP_X_PD_WEB_ID'] のバリデーション
       を十分行って下さい. */

    $db = new SQLite3($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) {
        /* HTTP_X_PD_WEB_ID が存在しない場合は Unauthorized 401 を返す. */
        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'];
    }

    /* HTTP リクエストヘッダ内の所定の文字列とデータベース上の key を用い
       signature を作成する */

    $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);
```

```
/* HTTP 応答ヘッダ共通部の設定 */
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');

/* リクエストヘッダの被署名文字列が VERSION.ID.TIME.MD5 で構成されているのに対し、
        応答ヘッダの被署名文字列は、VERSION.ID.TIME.MD5.SIGNATURE
   (SIGNATURE は、リクエストヘッダに含まれる文字列) で構成されている点に注意 */

if ($signature != $_SERVER['HTTP_X_PD_WEB_SIGNATURE']) {
      /* HTTP_X_PD_WEB_SIGNATURE と signature が一致しない場合は、
          401 Unauthorized を返す. */
      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']) {
      /* payload の MD5 値 と HTTP_X_PD_WEB_MD5 が一致しない場合は、
          406 Not Acceptable を返す. */
      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;
}

/* 受信ペイロードを HTTP リクエストヘッダと共に dump_file に格納する */
$fp = fopen($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 );

/* flags が 1 (処理中ペイロードあり) の場合、受信ペイロードから、
   'reply_to' キーの値を取得 */
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) {
        /* 受信ペイロードは、JSON オブジェクトの配列です. */
        $payload_count = count($array);
        for($i=0;$i<$payload_count;$i++) {
            if (isset($array[$i]['reply_to'])) {
                /* 'reply_to' キーの値と md5_tx を比較 */
                if($array[$i]['reply_to'] === $md5_tx) {
                    /* 一致している場合は flags を 0 にする */
                    $flags = 0;
                }
                else {
                    /* 一致していない場合は flags を 2 にする */
                    $flags = 2;
                }
            }
            break;
        }
        if($i == $payload_count) {
            /* 'reply_to' キーが存在しない場合は flags を 2 にする */
            $flags = 2;
        }
    }
    else {
        /* JSON 文字列で無い場合、flags を 2 にする */
        $flags = 2;
    }

    /* データベースの flags を更新する. */
    $query = sprintf("UPDATE client SET flags = %d WHERE id = '%s';" ,
        $flags, $_SERVER['HTTP_X_PD_WEB_ID']);
    $results = $db->query($query);
}

if($flags == 2) {
    /* flags が 2 (送信ペイロードあり) の場合は、payload_tx を送り、200 OK を返す. */
    header('Pd_Web_Md5: ' . md5($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 を 1 (処理中ペイロードあり) に変更し、データベースの flags を更新する. */
    $flags = 1;
    $query = sprintf("UPDATE client SET flags = %d WHERE id = '%s';" ,
        $flags, $_SERVER['HTTP_X_PD_WEB_ID']);
    $results = $db->query($query);
}
else {
    /* 200 OK を返す. */
    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;
?>
```

OpenBlocks IoT Family 向けデーターハンドリングガイド
（2018/01/25 第 1 版）

ぷらっとホーム株式会社
〒102-0073　東京都千代田区九段北 4-1-3　日本ビルディング九段別館 3F