DEXPFユーザーズガイド

Version 1.1-08

2020 年 6 月 29 日

ぷらっとホーム株式会社

目次

はじめに

DEXPFはREST APIを使用して各種設定を行う。ここではREST APIを利用するコマンドを使用して、データが流通するまでの一連のAPI操作を具体例で示す。

なお、DEXPFでの用語やDEXPFの機能については「DEXPF機能説明書」を、APIの一覧については「DEXPF API仕様書」を参照のこと。

DEXPFでデータが流通するまでの手順

DEXPFで実際にデータが流通するまでには、以下の手順をとる。

DEXPF Admin のAPI操作

1. DEXPFの初期設定
2. Adminのアクセストークンの生成
3. デバイス側のオーナー、アプリケーション側のオーナーの作成

アプリケーション側のオーナーのAPI操作

4. アクセス用トークンの生成
5. アプリケーショングループの作成
6. アプリケーションの登録

デバイス側のオーナーのAPI操作

7. アクセス用トークンの生成
8. デバイスグループの作成
9. デバイスの登録

チャネルの作成

10. アプリケーション側のオーナーがチャネルを作成する
11. デバイス側のオーナーがチャネルを承認する

デバイスの接続

12. デバイスへDEXPFへの接続情報の登録とデータの送信

注意

- デバイス側とアプリケーション側の操作は独立であり、どちらを先に行っても構わない。例ではアプリケーション側の操作を先に記述している。

- この例でチャネルは、アプリケーション側のオーナーがチャネルを作成するが、デバイス側のオーナーがチャネルを作成し、それをアプリケーション側のオーナーが承認する形でも構わない。
- DEXPFの提供形態にはAdminアカウントを提供する場合としない場合がある。
- Adminアカウントが提供される場合は、利用者がオーナー管理、つまりオーナーの登録や削除等を行う(上記の2, 3他)。
- Adminアカウントが提供されない場合は、利用者はオーナーそのものとなる。この場合利用者は、DEXPFのサービス提供者からAPIをアクセスするのに必要なアカウント情報を受け取るか、ユーザー名とパスワードの情報をサービス提供者に渡してオーナーの作成を依頼する。

DEXPF APIでのデータフォーマット

DEXPF APIの操作では、リクエスト、レスポンスともJSONフォーマットを使用する。したがって、APIを操作する場合、Content-Typeヘッダーには「application/json」を指定する。

```
Content-Type: application/json
```

DEXPF APIの利用

DEXPFのAPIは一般的なREST APIに準じている。そのためDEXPFを利用するためには、REST APIを操作できるツールが必要となる。以降の例ではAPIの操作に Linux環境でのシェルのコマンドラインでの curl コマンドを使用し、JSONデータは見やすくするため、整形して表記している。

またAPIにアクセスする場合ホスト名が必要になるが、本書では次のホスト名を使用している。

ホスト名　　dexpf-test.azurewebsites.net

このホスト名は例であり、**実際にDEXPFサービスが提供される場合はサービスに応じた別のホスト名**が提供される。

1. DEXPFの初期設定

DEXPFのインストール直後に実行する作業で、Adminアカウントの登録を行う。登録にはusernameとしてメールアドレス、passwordとしてAdminのパスワードを指定する。DEXPFはアカウントの識別のためにメールアドレスを使い実際にメールを送ることは無いため、架空のメールアドレスであっても問題ない。

▼ DEXPFの初期設定

```
curl --data @- --header "Content-Type: application/json" \
    https://dexpf-test.azurewebsites.net/api/v1/Admin/Setup <<EOT
{
  "username": "dexpf-admin@iot.example.jp"
  "password": "VpQj0GCBJRCpyu9waMRZ"
}
EOT
```

▽ レスポンス

```
{
  "systemId": "0f8c672f",
  "ownerId": "3eef4b14"
}
```

初期設定はDEXPFの提供側が行い1度しか実行できないため、本書の参照者が実行することは無いが、例として示している。

2. Adminのアクセストークンの生成

AdminがAPI操作のために必要なトークンをDEXPFから受領する。1.で設定したものと同じパラメータを使用する。

▼ Adminのアクセストークンの生成

```
curl --request POST --data @- \
    ---header "Content-Type: application/json" \
    https://dexpf-test.azurewebsites.net/api/v1/Authorizations/Token << EOT
{
      "username": "dexpf-cresnect@example.jp",
      "password": "g25izfyy58D0jvPX"
}
EOT
```

▽ レスポンス (トークンの具体的な値は省略する)

```
{
      "token": "XXXXXXXXXXX..."
}
```

ここで得られたトークンの値を以降の説明では**ADMINTOKEN**として表記する。なおトークンの有効時間は 3600秒(1時間)であるため、有効時間を過ぎた場合はトークンを再度取得する必要がある。

3. デバイス側、アプリケーション側それぞれのオーナーの作成

Adminがデバイス側のオーナーとアプリケーション側のオーナーを作成する。DEXPFの初期設定と同様に、ユーザー名とパスワードを指定したJSONデータを用意する。なおDEXPFにはアプリケーション側のオーナーとデバイス側のオーナーに差異は無く、デバイスを扱うのか、アプリケーションを扱うのかで区別している。

▼ アプリケーション側のオーナーの作成例

```
curl --header "Authorization: Bearer ${ADMINTOKEN}" \
    --header "Content-Type: application/json" \
    --data @- \
    https://dexpf-test.azurewebsites.net/api/v1/Accounts << EOT
{
  "username": "appown@iot.example1.jp",
  "password": "iNhg8TKqpPgJkfZKu2D5"
}
EOT
```

▽ レスポンス

```
{
  "ownerId": "f293491a"
}
```

アプリケーション側のオーナーのIDは `f293491a` となる。

▼ デバイス側のオーナーの作成例

```
curl --header "Authorization: Bearer ${ADMINTOKEN}" \
    --header "Content-Type: application/json" \
    --data @- \
    https://dexpf-test.azurewebsites.net/api/v1/Accounts << EOT
{
  "username": "devown@iot.example2.jp",
  "password": "iWijYD7KrbZ1h6XxeB2q"
}
EOT
```

▽ レスポンス

```
{
  "ownerId": "84e943b0"
}
```

デバイス側のオーナーのIDは `84e943b0` となる。

4. アプリケーション側のアクセストークンの生成

アプリケーション側のオーナー、デバイス側のオーナーのいずれであっても、DEXPFのAPIを操作するにはそれぞれのトークンが必要となる。そのためユーザー名とパスワードを使って、トークンを生成する。ここではアプリケーション側のオーナーが実行する。

▼ アプリケーション側のオーナーがトークンを生成する実行例

```
curl --header "Content-Type: application/json" \
    --data @- \
    https://dexpf-test.azurewebsites.net/api/v1/Authorizations/Token << EOT
{
  "username": "appown@iot.example1.jp",
  "password": "iNhg8TKqpPgJkfZKu2D5"
}
EOT
```

▽ レスポンス (具体的なトークン文字列は省略)

```
{
  "token": "XXXXXXXXXXX..."
}
```

以後の説明では、アプリケーション側のオーナーのトークンを ${APPTOKEN} とする。

5. アプリケーショングループの作成

アプリケーションを登録する場合は、事前にアプリケーショングループを用意する必要がある。そのためまずアプリケーショングループを作成する。なおアプリケーショングループのIDはアプリケーション側のオーナーが指定する。

アプリケーション側のオーナーの権限で実行する。

▼ アプリケーショングループの作成例

```
curl --header "Authorization: Bearer ${APPTOKEN}" \
    --header "Content-Type: application/json" \
    --data @- \
    https://dexpf-test.azurewebsites.net/api/v1/AppGroups << EOT
{
  "appGrId": "a0000001"
}
```

```
EOT
```

▽ レスポンス

```
{
  "appGrId": "a0000001"
}
```

6. アプリケーションの登録

アプリケーショングループが登録できたら、アプリケーションの登録を行う。DEXPFからアプリケーションへはHTTPのPOSTを使用するため、POSTが受けられるURLを指定する。また認証等に必要なヘッダーも登録する。

現時点でのDEXPFはデータに利用できるのはJSON形式だけであるため、Content-Type ヘッダーに application/json では無いものを指定すると無視され、 application/json が使用される。

アプリケーション側のオーナーの権限で実行する。

▼ アプリケーションの登録例

```
curl --header "Authorization: Bearer ${APPTOKEN}" \
    --header "Content-Type: application/json" \
    --data @- \
    https://dexpf-test.azurewebsites.net/api/v1/Apps << EOT
{
  "appGrId": "a0000001",
  "appId": "f0000001",
  "url": "http://app.example.jp/app/dexpf-sample-app",
  "headers": [
    "Content-Type: application/json",
    "Authorization: Bearer n38h1bnS34sPGLTyQGs1y1IbmtdZ0WiUR3it2JcB"
  ]
}
EOT
```

▽ レスポンス

```
{
  "appGrId": "a0000001",
  "appId": "f0000001"
}
```

7. デバイス側のアクセストークンの作成

デバイス側のオーナーもアクセストークンを作成する必要がある。

▼ デバイス側のオーナーがトークンを生成する実行例

```
curl --header "Content-Type: application/json" \
    --data @- \
    https://dexpf-test.azurewebsites.net/api/v1/Authorizations/Token << EOT
{
  "username": "devown@iot.example2.jp",
  "password": "iWijYD7KrbZ1h6XxeB2q"
}
EOT
```

▽ レスポンス (具体的なトークン文字列は省略)

```
{
  "token": "XXXXXXXXXXX..."
}
```

以降の説明では、得られたトークンを ${DEVTOKEN} で表す。

8. デバイスグループの作成

デバイスを登録する際も、事前にデバイスグループを作成する必要がある。デバイス側のオーナーがデバイスグループを作成する

▼ デバイスグループの作成例

```
curl --header "Authorization: Bearer ${DEVTOKEN}" \
    --header "Content-Type: application/json" \
    --data @- \
    https://dexpf-test.azurewebsites.net/api/v1/DeviceGroups << EOT
{
  "devGrId": "90000001"
}
EOT
```

▽ レスポンス

```
{
  "devGrId": "90000001"
}
```

---

9. デバイスの登録

次にデバイスを登録する。デバイスの登録はデバイスIDを決め、作成済のデバイスグループIDを指定して登録する。デバイスは同時に複数まとめて登録できる。

▼ デバイスの登録例

```
curl --header "Authorization: Bearer ${DEVTOKEN}" \
    --header "Content-Type: application/json" \
    --data @- \
    https://dexpf-test.azurewebsites.net/api/v1/Devices << EOT
{
  "devGrId": "90000001",
  "devIds": [
    "00000001",
    "00000002"
  ]
}
EOT
```

▽ レスポンス

```
{
  "hostname": "dex-iot-hub-0002.azure-devices.net",
  "devGrId": "90000001",
  "devices": [
    {
      "devId": "00000001",
      "hubId": "dev-f293491a-00000001",
      "key1": "tqEaC08z1BtxxTgE3+m8jNVV24cDs1IFFZJW9qfebwU8+aQK50H0FuGx4sNhEJh6",
      "key2": "n/mqQyOw8eN0kiX8EkorAmLyEE62DrmoJYabvnHrufnkkPuoTG5YiTyKq6I1J9c/"
    },
    {
      "devId": "00000002",
      "hubId": "dev-f293491a-00000002",
      "key1": "5AX2N0KpZvPRQx3vlCug2vEbX0QhTqE4jJUaC8vLfor9YYCJiOQGVxeQLfoiZRDg",
      "key2": "ElnQR30fLmQ9yxfCDTtJkR8t3jUbM0ih02xXsjuJ4poAbkBFexAvdwhRBs52DXBt"
    }
  ]
}
```

レスポンスデータの各意味は次の通りである。

hostname	デバイスに登録するDEXPFへの接続先(Azure IoT Hub)のホスト名 (各デバイスで共通)
hubId	デバイスに登録するDEXPFへの接続Id (IoT Hubがこのidで認識する)
key1、key2	IoT Hubの接続に必要なプライマリキーとセカンダリーキー 通常、どちらか一方のみを利用する

10. チャネルの作成

チャネルの作成には、アプリケーション側のオーナーIDとアプリケーショングループID、デバイス側のオーナーiDとデバイスグループIDを指定する。作成はアプリケーション側のオーナーとデバイス側のオーナーの何れが行っても問題ない。ここではアプリケーション側のオーナーが作成する例とする。

なおDEXPF単体では、デバイス側とアプリケーション側のID等を交換する手段は用意していないため、それぞれのオーナーはDEXPF外の仕組みで相手側のIDを知る必要がある。

▼ チャネルの作成 (アプリケーション側のオーナーが実行)

```
curl --header "Authorization: Bearer ${APPTOKEN}" \
    --header "Content-Type: application/json" \
    --data @- \
    https://dexpf-test.azurewebsites.net/api/v1/Channels << EOT
{
  "appOwnId": "f293491a",
  "appGrId": "a0000001",
  "devOwnId": "84e943b0",
  "devGrId": "90000001"
}
EOT
```

▽ レスポンス

```
{
  "chanId": "c51a463b",
  "devGrId": "90000001",
  "devOwnId": "84e943b0",
  "appGrId": "a0000001",
  "appOwnId": "f293491a",
  "appSide": "enable",
  "devSide": "disable"
}
```

---

chanIdとしてチャネルのIDが得られる。appSide, devSideのパラメータでチャネルの接続状況を示し、アプケーション側は有効となっているが、デバイス側はまだ有効で無いことが分かる。

この状態でデバイス側のオーナーがチャネルの一覧を調べると次のようになる。

▼ チャネルの一覧の取得 (デバイス側のオーナーが実行)

```
curl --header "Authorization: Bearer ${DEVTOKEN}" \
    https://dexpf-test.azurewebsites.net/api/v1/Channels
```

▽ レスポンス

```
{
  "chanIds": [
    "c51a463b"
  ]
}
```

さらにチャネルの詳細を調べると次のようになる

▼ 指定チャネルの詳細の取得 (デバイス側のオーナーが実行)

```
curl --header "Authorization: Bearer ${DEVTOKEN}" \
    https://dexpf-test.azurewebsites.net/api/v1/Channels/c51a463b
```

▽ レスポンス

```
{
  "chanId": "c51a463b",
  "appOwnId": "f293491a",
  "appGrId": "a0000001",
  "devOwnId": "84e943b0",
  "devGrId": "90000001",
  "appSide": "enable",
  "devSide": "disable"
}
```

---

11. チャネルの承認

アプリケーション側でチャネルを作った場合は、デバイス側でチャネルの承認を行う。逆の操作として、デバイス側でチャネルを作った場合は、アプリケーション側でチャネルの承認を行う。

チャネルの承認が行われない限り、データ交換は行われない。チャネルの承認には、チャネル作成時のチャネルIDが必要である。

▼ チャネルの承認（PUTを使用する）

```
curl --header "Authorization: Bearer ${DEVTOKEN}"
       --request PUT --data "" \
        https://dexpf-test.azurewebsites.net/api/v1/Channels/c51a463b
```

▽ 応答メッセージの例

```
{
  "chanId": "c51a463b",
  "appOwnId": "f293491a",
  "appGrId": "a0000001",
  "devOwnId": "84e943b0",
  "devGrId": "90000001",
  "appSide": "enable",
  "devSide": "enable"
}
```

---

12. デバイスの接続

デバイスを登録した時点でその情報をデバイスに設定することで、デバイスからDEXPFへデータを送ることができる。しかしながらチャネル作成前であると、デバイスから送られたデータはDEXPF内部で破棄される。具体的なデバイスの接続の例を付録に掲載する。

付録 DEXPFへ接続するデバイスの設定例

実際にDEXPFにデバイス接続する場合に必要な具体的な設定を、OpenBlocks IoT ファミリとNode-REDの場合を例に説明する。

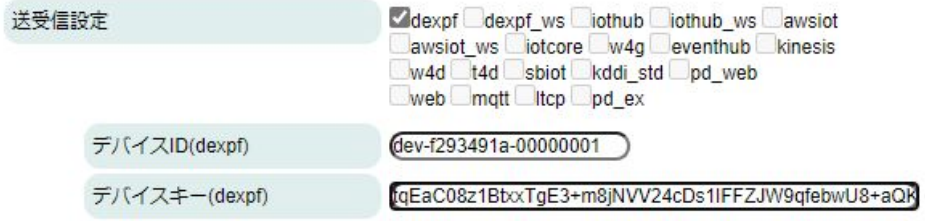
A. OpenBlocks IoTファミリのDEXPFへの接続

OpenBlocks IoT ファミリ(以下OBSIOT)では、FW 4以降DEXPFへの接続が標準でサポートされており、OBSIOTに接続済のデバイスであれば、OBSIOTのWEB UIより簡単にDEXPFへの接続設定ができる。本設定にあたり予めデバイス等をOBSIOTに接続する設定が完了している必要がある。

OBSIOTのWEB UIにログインしたら、「サービス」→「IoTデータ」→「送受信設定」の中に各種IoTサービスへ接続設定の一覧があり、そこに「DEXPF」の設定がある。

「送受信設定」から「DEXPF(dexpf)」の項目の「使用する」を選択するとDEXPFの設定メニューが表示されるので、「ホスト名」に、DEXPFにデバイスを登録した際に返される hostname の値を設定する。

また登録済デバイスの送受信設定で「dexpf」のチェックボックスを選択すると、デバイスIDとデバイスキーの入力項目が表示される。



デバイスIDにはDEXPFにデバイスを登録した際に返される hubIdを設定し、デバイスキーには、key1 または key2 のいずれかを登録する。

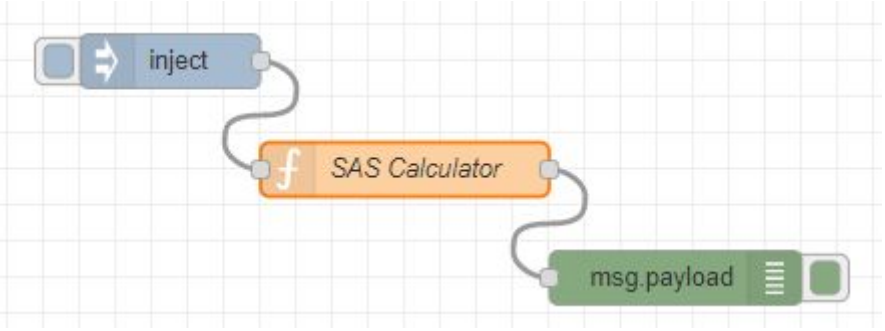
以上でOBSIOTからDEXPFへデバイスデータを送る設定は完了する。

B. Node-REDによるSASトークンの生成

DEXPFではデバイスの接続にAzure IoT Hubのサービスを使用しており、IoT Hubではデバイス接続の認証に、Shared Access Signatures(SAS)トークンを使用する。なおX509証明書を利用する方法もあるが、DEXPFではサポートしていない。

OBSIOTのように専用の接続機能を持っているものを利用する場合は、SASトークンを内部で生成するためユーザーが直接SASトークンに触れる機会は無いが、Node-REDで直接DEXPFに接続するためには、プログラムを使ってSASトークンを生成する必要がある。ここではNode-REDを使ってSASトークンを生成するプログラムを紹介する。

次の図はSASトークンを生成するNode-REDのフローである。



injectノードでは次の形式のJSONデータを指定する(keyの部分は後半省略)。

```
{
  "iotHub": "dex-iot-hub-0002.azure-devices.net",
  "expireDate": "1924959599",
  "devices": [
    { "id": "dev-84e943b0-00000001", "key": "Gf4Gok5K69cysMoJipKmRvR..." },
    { "id": "dev-84e943b0-00000002", "key": "aDnV6b4AaP2SvSzrmNlLycc..." },
    { "id": "dev-84e943b0-00000003", "key": "NDVIw0DgWfuMDAlKoasAinC..." }
  ]
}
```

ここで各パラメータは次の通りである。

iotHub	DEXPFにデバイスを登録した際に返される hostname
expireDate	SASの有効時間をUNIX Timeで指定する。本例の1924959599は 2030年12月31日23時59分59秒(日本時間)を示す。
devices	idとkeyからなるJSON配列
id	DEXPFにデバイスを登録した際に返される hubId
key	DEXPFにデバイスを登録した際に返される key1 または key2 のいずれか

UNIX Timeは協定世界時1970年1月1日午前0時0分0秒からの経過時間を秒数で表現したもの(うるう秒は無視)で、Linuxのdateコマンドを利用して次の方法で変換できる。

日時からUNIX Timeへの変換

```
$ date --date "2030-12-31T23:59:59" +%s
1924959599
```

UNIX Timeから日時への変換

```
$ date --date @1924959599
2030年 12月 31日 火曜日 23:59:59 JST
```

SAS Calculatorの関数ノードで実際にSASトークンを計算している。この関数ノードのプログラムは次の通りである。

```
const crypto = global.get('crypto');

var generateSasToken = function(resourceUri, signingKey, policyName, expireTime) {
    resourceUri = encodeURIComponent(resourceUri);

    // Set expiration in seconds
    var toSign = resourceUri + '\n' + expireTime;

    // Use crypto
    var hmac = crypto.createHmac('sha256', Buffer.from(signingKey, 'base64'));
    hmac.update(toSign);
    var base64UriEncoded = encodeURIComponent(hmac.digest('base64'));

    // Construct authorization string
    var token = "SharedAccessSignature sr=" + resourceUri + "&sig=" +
                    base64UriEncoded + "&se=" + expireTime;
    if (policyName) token += "&skn="+policyName;
    return token;
};


let devices = msg.payload.devices;

for (let i = 0 ; i < devices.length ; i ++) {
    devices[i].sas = generateSasToken(msg.payload.iotHub + '/devices/' +
                devices[i].id, devices[i].key, false, msg.payload.expireDate);
    delete devices[i]['key'];
}

msg.payload = devices;

return msg;
```

このプログラムを動かす前に、Node-RED の settings.js のfunctionGlobalContext で、cryptoライブラリを利用することを定義する必要がある。本変更はNode-REDの再起動で反映される。

```
functionGlobalContext: {
    crypto: require("crypto")
},
```

注意： OBSIOTでNode-REDを利用している場合、settings.jsを変更するには Node-REDの設定で「ユーザー定義コンフィグ」を選び「編集」のタブで settings.jsの編集を行う。

injectノードを操作すると、デバッグタブに以下の例で示すような id とSASトークンのペアのJSON配列が出力される。実際のSASトークンは1行であるが、以下の例では見やすさの為に途中で改行している。

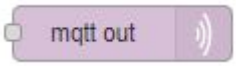
```
[
  {
    "id": "dev-84e943b0-00000001",
    "sas": "SharedAccessSignature sr=dex-iot-hub-0002.azure-devices.net
            %2Fdevices%2Fdev-84e943b0-00000001&sig=Kfw1t99eVf4IlN%2FXSK
            tvjbrY74HWC%2B7WKmsyZhc8dxg%3D&se=1924959599"
  },
  {
    "id": "dev-84e943b0-00000002",
    "sas": "SharedAccessSignature sr=dex-iot-hub-0002.azure-devices.net
            %2Fdevices%2Fdev-84e943b0-00000002&sig=DVW5LoEVcBktpt9JjsdX
            18zHeo74CvAuXFQ9WsgbtGU%3D&se=1924959599"
  },
  {
    "id": "dev-84e943b0-00000003",
    "sas": "SharedAccessSignature sr=dex-iot-hub-0002.azure-devices.net
            %2Fdevices%2Fdev-84e943b0-00000003&sig=6OCj1t5Xhfe%2FianqGm
```

```
            QnpJaWAIYgalmWiMWu4toDR6M%3D&se=1924959599"
  }
]
```

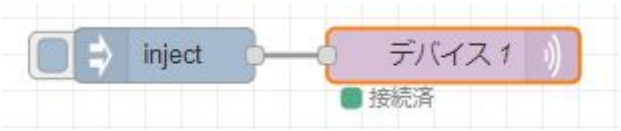
C. Node-REDからDEXPFへの接続設定 (MQTT)

Node-REDからDEXPFに接続するにはMQTTプロトコルを使う方法と、HTTP RESTを利用する方法があり、ここではMQTTプロトコルで設定する方法について説明する。

DEXPFのデバイスに接続するには、mqtt-outノードを使用し、



次のようにデータを送信するノードと接続する。



mqtt-outノードをダブルクリックすると、次のメニューが表示されるので各項目を設定する。

トピック	MQTTのトピックとして「devices/{デバイス名}/messages/events/」を設定する。デバイス名にはDEXPFにデバイスを登録した際の hubIdを指定する。 hubIdが"dev-f293491a-00000001"の場合は、次のようになる。 **devices/dev-f293491a-00000001/messages/events/**
QoS	「1」 を選択する
保持	「する」を選択する

次にmqtt-brokerノードの設定を追加する。「サーバ」の右側にある鉛筆のボタンをクリックすると次の設定メニューが表示される。

サーバ	DEXPFにデバイスを登録した際に得られる hostname を指定する
ポート	8883
SSL/TLS接続を使用	選択をチェックする
クライアント	DEXPFにデバイスを登録した際に得られる hubId を指定する

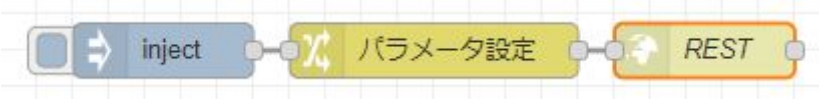
次にセキュリティのタブを選択し、ユーザー名とパスワードを指定する

ユーザー名	DEXPFにデバイスを登録した際に得られる hostnameと hubId を”/”で接続した文字列を指定する。ここの例では次の文字列となる **dex-iot-hub-0002.azure-devices.net/dev-f293491a-00000001**
パスワード	SASトークンそのもの、つまり**SharedAccessSignature** から始まる文字列をそのまま入力する

注意：mqtt out ノードで個別のMQTT brokerの接続を設定できるが、Node-REDでは複数のノードでMQTT brokerを共有できる関係で、新たなmqtt outノードを用意したら、次のように「新規に mqtt broker を追加」を選択して設定する必要がある。

D. Node-REDからDEXPFへの接続設定 (HTTP REST)

Node-REDからDEXPFにHTTP RESTを利用する設定である。次のNode-REDのフローはinjectノードで生成されたデータをDEXPFへRESTインターフェースで接続している。



実際にデータを送るのはhttp requestノードであるが、必要なパラメータは直前のchangeノードで設定している。そのためhttp requestノードではメソッドを次の図のように、「-msg method に定義 -」を選択する

changeノードでは次のように設定する。

msg.url	https://{ホスト名}/devices/{デバイスID}/messages/events/?api-version=2016-11-14 の形式で、DEXPFにデバイスを登録した際に得られる hostnameをホスト名に設定し、hubIdをデバイスIDに設定する。具体的に次のようになる。 **`https://dex-iot-hub-0002.azure-devices.net/devices/dev-f293491a-00000001/messages/events/?api-version=2016-11-14`**
msg.method	**POST** を指定する
msg.headers.Authorization	SASトークンそのもの、つまり**SharedAccessSignature** から始まる文字列をそのまま入力する
msg.headers.Content-Type	JSON形式であるため、**`application/json`** を指定する。

以上

改訂履歴

バージョン	日付	内容
Ver 1.1-01	2019-05-27	初版
Ver 1.1-02	2019-10-11	些細なミスを修正
Ver 1.1-03 から Ver 1.1-05		他のドキュメントとバージョン番号を合わせるため欠番
Ver 1.1-06	2020-01-10	本書の名前を「DEXPF設定マニュアル」から 「DEXPFユーザーズガイド」に変更 「DEXPF機能説明書」に合わせ用語を修正 APIの実行例をより具体的なものに変更 実行例のレスポンスも具体的に記載 チャネルの詳細を得る例を追加 DEXPF提供形態を注意部分に追加 目次を追加 「はじめに」の項を追加
Ver 1.1-07	2020-03-27	正式版としてリリース
Ver 1.1-08	2020-06-29	付録としてにOpenBlocks IoTファミリとNode-REDから DEXPFへ接続する具体例を追加