EnOcean IoT Kit
プログラム開発
学習テキスト

2015 年 2 月 10 日

株式会社デバイスドライバーズ

目次

1. はじめに

この学習テキストは.NET Gadgeteer と Microsoft Azure Mobile Service を利用して、EnOcean の各種センサー類を Internet 上のクラウドサービスに接続して動作させる IoT (Internet of Things) システムの開発方法を示します。

.NET Gadgeteer の操作に慣れていない方、演習を始める前の準備については、この演習を始める前に別演習テキスト「準備と基本操作」を修了しておくことをお勧めします。

このテキストでは次の印と項目を使用しています。独習する際に参考にしてください。

●演習	今回のセミナー実習する演習問題です。
○演習	今回は実習しない演習問題です。時間に余裕がある人は実習してください。
★注意★	演習を進める上での重要な注意点を示します。
■ヒント■	演習を進める上での有効なヒントを示します。
□解説□	演習を進める上で参考となる解説を示します。

■ヒント：インストール先のフォルダ名■

本文中の説明では、各種ソフトウェアを 64bit 版 Windows 環境にインストールして利用する場合を想定します。32bit 版環境にインストールして使用する場合には、インストール先のフォルダ名を C:¥Program Files (x86) から C:¥Program Files に読み替えてご利用ください。

1.1. 出所

本資料は日本マイクロソフト株式会社のテクニカルエバンジェリスト、太田寛の許可を得て、太田寛の著作による「Internet Of Things ハンズオン（初期版）」を元に、.NET Gadgeteer V4.3 用に修正し、さらに EnOcean 無線センサーに対応させて作成しています。従ってライセンスと免責事項に関しては、この「Internet Of Things ハンズオン」に従います。

「Internet Of Things ハンズオン」に関連するページを次に示します。

① Internet Of Things ハンズオン（初期のバージョン、V4.2 用）
http://aka.ms/IoTHandsOn

② Internet Of Things キット ハンズオントレーニング（最近のバージョン）
http://aka.ms/IoTKitHoL （Internet of Things キット ハンズオントレーニングテキスト）

③ 太田寛氏のブログ「デバイスと IT の架け橋」
http://blogs.msdn.com/b/hirosho/

1.2. 情報の鮮度

① テキストは、2015/2/10 時点の技術情報を基に作成されています

② 技術情報は日々更新されるものであり、本資料の記述が最新状況とは異なる場合があります

③ 演習する時点で本資料の内容との最新技術情報と異なる場合は、最新の方法での実装をお勧めします

1.3. ライセンスと免責事項

① 本資料に基づいて演習、自己学習を実施した結果、付帯して配布するソフトウェアの運用において生じた、いかなる損害について一切責任は負いません。

② 本資料で提供するソフトウェアは、Ms-PL（http://opensource.org/licenses/ms-pl）で提供します。ハンズオンで使用するソフトウェアで、別のライセンスが明示されている場合には、そのライセンスに従って使用してください。

1.4. 演習の概要

① 目的

本テキストでは、センサークラウドや M2M など、各種デバイスがネットワーク連携するシステム開発について.NET Gadgeteer 対応ボード、Windows ストアアプリ、Microsoft Azure Mobile Service を使って実現する IoT システム開発方法の基礎を学びます。

② 必要知識

・ Visual Studio の基本的な使い方を知っていることが望ましい(必須ではない)

・ C#に関する知識があることが望ましい（必須ではない）

③ 開発言語

・ C#

④ 演習終了後に得られる知識

・ Microsoft Azure Mobile Service の利用方法

・ Microsoft Azure Mobile Service でデータ共有する.NET Micro Framework / .NET Gadgeteer アプリ開発方法

1.5. システム概要

① EnOcean センサーノードで計測した温度・湿度・スイッチ情報を受信

② 受信したデータを Microsoft Azure モバイルサービスに接続して蓄積・共有

③ Excel へのデータインポート

1.6. 開発環境

オペレーティング・システム：Windows Vista 以降を搭載した PC

開発環境（Visual Studio）：Visual Studio Express 2013 with Update 4 for Windows Desktop, Visual Studio Community 2013 Update 4, または Visual Studio Professional 2013 with Update 4 以上（上位版も可、他バージョンは利用不可）

NETMF SDK：Microsoft .NET Micro Framework SDK 4.3 (QFE2)（他バージョンは利用不可）

Gadgeteer Core：Microsoft Gadgeteer Core 2.43.1000 以降

GHI SDK：GHI Electronics NETMF SDK 2014.R5 以降

.NET Gadgeteer 修正版ライブラリ：Gadgeteer.Webserver43

（株式会社デバイスドライバーズにて作成・配布、本演習に添付）

必要な DISK 容量： 空きディスク領域 2GB

マシン仕様：Pentium 4 2GHz 以上、メモリ 2GB 以上を推奨。

■ヒント：開発環境のインストール■

開発環境のインストール方法については、別テキスト「準備と基本操作」をご参照ください。

1.7. ターゲット環境

メインボード GHI electronics FEZ Spider（TinyBooter4.3.4.0 以降 TinyCLR 4.3.4.0 以降）

モジュール：USB Client SP Module または USB Client DP Module

Ethernet J11D Module

USB Host Module

Multicolor LED Module（オプション扱い）

SD Card Module（オプション扱い）

★注意：ファームウェアのバージョン★

上記以外のバージョンでも動作する可能性はありますが、本テキスト執筆時点では動作確認していませんのでご注意下さい。

1.8. クラウド環境

Microsoft Azure (Windows Azure) のサブスクリプション契約が必要です。本テキストでは 1 か月間無料評価版（有償サブスクリプションに移行可能）の権利取得手順を示します。

すでに有効な Azure のサブスクリプションをお持ちの場合には、それを利用することが可能です。

1 か月間無料評価版を使用してアカウントを開設する場合には次のものが必要です。

① 有効なクレジットカード

② すぐに通話可能な携帯電話または固定電話

1.9. Windows ストアアプリ開発

本テキストの演習では扱いませんが、Azure に登録したモバイル サービスを Windows ストアアプリから使用し、Push 通知を受け取る様にする場合には、Windows ストアアプリの開発者登録（有償）が必要になります。

■ヒント：開発者アカウントを開く■

http://msdn.microsoft.com/ja-jp/library/windows/apps/hh868184.aspx

2. Microsoft Azure のアカウント作成

Microsoft Azure には Microsoft アカウント（旧 Live ID）を利用してサインイン（ログイン）します。Azure の利用者登録をする際に Microsoft アカウントを持っていない場合には、

http://www.microsoft.com/ja-jp/msaccount/signup/

にブラウザからアクセスして Microsoft アカウントを取得しておく必要があります。

□解説：アカウントとサブスクリプション□

Azure のサブスクリプションとは Azure の各種サービスを利用する際に必要な利用権限（使用権）です。わかり易い言い方をすれば課金単位に相当します。

1 か月間無料評価版を使用する場合には、サインインに使用したアカウントに 1 か月間有効な ¥20,500 相当のサブスクリプションが無償で付いてきます。1 か月間経過後も引き続き利用する場合には、サブスクリプションの更新（有償）または変更（新規購入）をする必要があります。サブスクリプションには定額制や従量制等、様々な種類があります。

一つのアカウントに複数のサブスクリプションを登録可能です。その場合は Azure の各サービスを利用する際に、どのサブスクリプションを使用するかの設定をする必要があります。

2.1. アカウント開設手順

すでに Azure アカウントと有効なサブスクリプションをお持ちの場合は、この章の演習はスキップして下さい。Microsoft アカウントで利用できる無料評価版は 1 回限りです。すでに無料評価版を利用済みの場合は、従量制等の有効なサブスクリプションを購入して下さい。

★注意：Visual Studio と Azure で使用する Microsoft アカウント★

今回の演習では使用しませんが、Visual Studio 2013 では使用時に Microsoft アカウントへのサインインが必須となりました。一方で今回利用する Azure モバイルサービスでは Visual Studio 2013 で開発可能なモバイルサービスにアクセス可能なソースコード・テンプレートを提供する機能があります。この機能を利用するためには **Visual Studio 2013 で使用する Microsoft アカウントと Microsoft Azure の Microsoft アカウントで同じものもの使用する必要があります**。ご注意下さい。

○演習 1）

Microsoft Azure に 1 か月間無償のアカウントを作成します。

手順 1 ：登録サイトへのアクセス

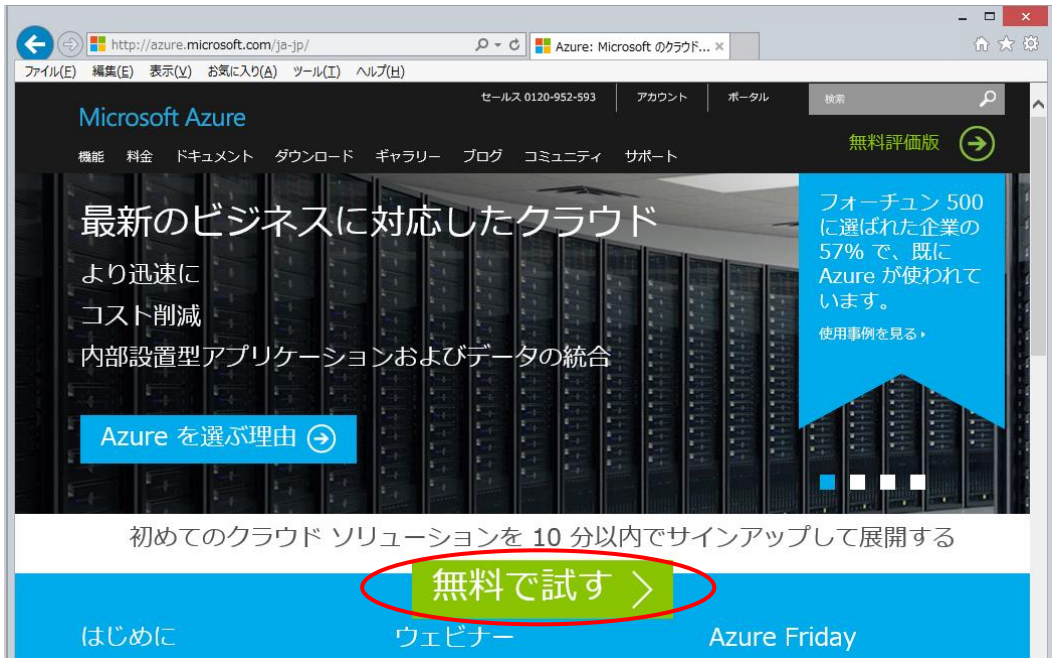
ブラウザから以下のいずれかの手順で登録申請サイトにサインインします。

① http://azure.microsoft.com/ja-jp/

② http://azure.microsoft.com/ja-jp/pricing/free-trial/

③ インターネットで「Azure」で検索する。

以下に各画面でクリックする場所を示します。



手順２：サインイン

「今すぐ試す」をクリック後は Microsoft アカウントのサインイン画面に切り替わるので、所得している Microsoft アカウントとパスワードを入力します。すでにサインイン済の場合にはこの画面は表示されないので、次画面でサインイン済のアカウント名を確認して下さい。

手順３：申請フォームの送信

赤印が付いている項目を入力します。連絡先のメールアドレスはメールを読むことができるアドレスであればアカウントと同じでも構いません。勤務先の電話は、連絡がつけばどこでも構いません。市外局番や携帯番号の先頭の「0」を外して記入します。

携帯電話確認の項は申請者の個人認証確認を行います。確認方法は次の 2 種類から選択できます。従って電話が利用できない場合には、申請を行う事ができません。

① テキストメッセージを受信

携帯電話に SMS（ショートメッセージサービス）のメールが送られて来ます。携帯電話番号入力後、「テキストメッセージを受信」をクリックします。その後フォームは、認証コードを入力可能になります。

通常 1 分程度でメールを受信しますが、メールの受信拒否機能の設定によりメッセージを受け取れないことがあります。1 分程度待ってメールを受信することができない場合には、「電話で確認コードを受け取る」の方法を試すことをお勧めします。

② 電話で確認コードを受け取る

こちらを選択するとフォーム画面が次の様に変わるので、電話番号を入力して「電話で確認コードを受け取る」をクリックします。固定電話の番号も使用可能です。電話番号が確認された場合、フォームには認証コードが入力可能になります。

通常クリック直後に電話がかかって来て確認コードの番号が告げられます。確認コードは 6 桁の数字で、英語で伝えられます。確認コードの数字は英語で機械的に 2 回告げられるだけで、聞き直すことができません。十分注意して下さい。

確認コードを受け取ることができない場合には、次の原因が考えられます。

① 電話番号の入力に誤りがある

② 認証で使用する電話番号をすでに他のアカウントで使用して無償アカウント開設済である

★注意：個人認証確認手順★

Azure のアカウント開設の際、この電話による個人認証が一番面倒な手順です。現在、他の手段はありません。問題が発生しても適切なエラー・メッセージは表示されません。何回か試してもコードが受信できない場合、サポート電話番号に問い合わせすることをお勧めします。

アカウント開設に関する日本マイクロソフトのサポート電話番号：

0120-41-6755（平日 9:00-17:30）

■ヒント：Windows Azure 新規契約時の個人認証■

http://blogs.msdn.com/b/dsazurejp/archive/2013/09/12/new-subscription-personal-indentification.aspx

手順４：コードの確認

受け取った確認コードを入力して「コードの確認」をクリックします。

確認コードが受理されると、支払情報の入力が可能になります。

手順５：支払情報の登録とサインアップ

入力フォームが次の様に変化するので、クレジットカード情報を入力し、契約項目にチェックして「サインアップ」をクリックします。この段階では課金されません。

また１か月経過して評価期間終了後も、何もアクションを起こさなければ一切課金されることはありません。

手順６：Azure へのサインイン

サインアップをクリック後しばらくすると Azure のログイン画面が表示されます。ログインすると次の初期画面が表示されるので、「ポータル」をクリックします。

「ここをクリックして今すぐアップグレードしてください」をクリックすると、課金用の手続きが始まります。

「ポータル」をクリックして Azure の最初の画面（ダッシュボード）に進みます。

■ヒント：Microsoft Azure サブスクリプション申し込み Step by Step■

http://msdn.microsoft.com/ja-jp/windowsazure/ee943806.aspx

■ヒント：サポートにお問い合わせする方法について■

http://blogs.msdn.com/b/dsazurejp/archive/2013/10/31/10462044.aspx

3. Azure のサービス利用設定

3.1. モバイルサービスの作成と準備

●演習２）

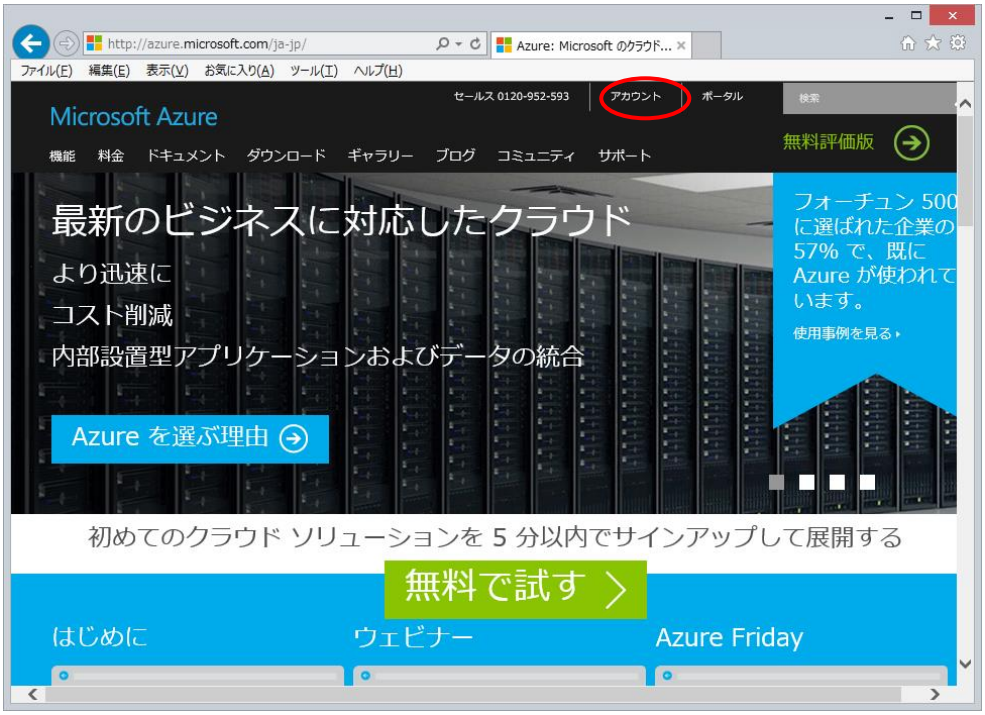
次の手順に従って Azure のモバイルサービスの作成と登録を行います。

手順１：サインイン

ブラウザから次のいずれかの方法で Azure のポータル（ダッシュボード）にアクセスします。

① http://manage.windowsazure.com

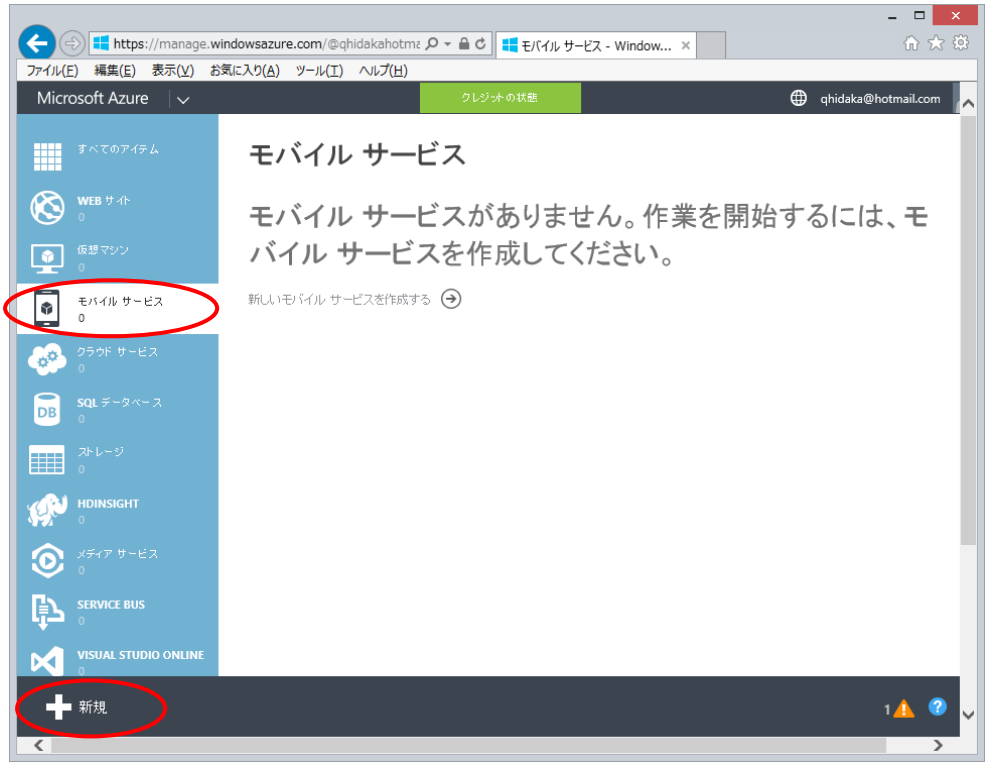
② ブラウザで「Azure」を検索後、右上の「ポータル」をクリック（以下参照）



サインインが求められたらば、アカウント開設時の Microsoft アカウントでサインインします。

手順２：モバイルサービスの作成

左側メニューの上から 3 番目「モバイルサービス」タブをクリックします。その後画面の左下にある「+新規」をクリックします。



モバイルサービスのサブメニューで「作成」をクリックします。

「モバイルサービスの作成」画面で次の項目を設定します。

① URL：外部からデータにアクセスするためのユニークな名前（英数字）です。
この名前により、https://名前.azure-mobile.net/ の URL が与えられるので、自由な名前を設定します。（入力すると名前が利用可能かどうかをリアルタイムにチェックします。）
この例では、名前を「iot-kit」として入力しています。

② データベース：「無料の 20 MB SQL データベースの作成」のままとします。

③ 地域：選択可能であれば、日本（西）または日本（東）、東アジアを選択します。

④ バックエンド：「JavaScript」を選択します。

「プッシュの詳細通知の構成」にはチェックを入れません。入力完了後、右下の→をクリックして次の設定画面に移動します。

手順３：データベースの設定の指定

この画面では、データベースの名前が自動的に割り当てられているので、サーバーログイン名とパスワードを設定します。**ここで設定したログイン名とパスワードは後で使用するので、覚えておきます。**

「データベースの詳細設定を構成します」にはチェックをしません。右下の「チェック」をクリックするとモバイルサービスの作成を開始します。

以下は作成中の画面です。環境にも依存しますが、通常約 1 分程度で終了します。

■ヒント：複数のモバイルサービス■

ここまでの手順を繰り返すことで、アプリケーション毎に複数のモバイルサービスを登録して利用することが可能です。無料アカウントの場合「無料の 20MB SQL データベース」は 1 個しか利用できません。2 回目以降サービス作成では「ログイン名」と「ログイン パスワード」を入力して同じデータベースを利用します。

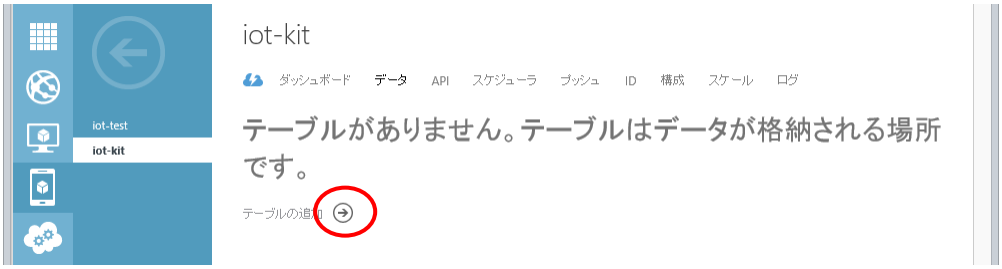
手順４：計測値保存用テーブルの作成

作成したモバイルサービスに対して、次の手順で計測値保存用テーブルを作成します。

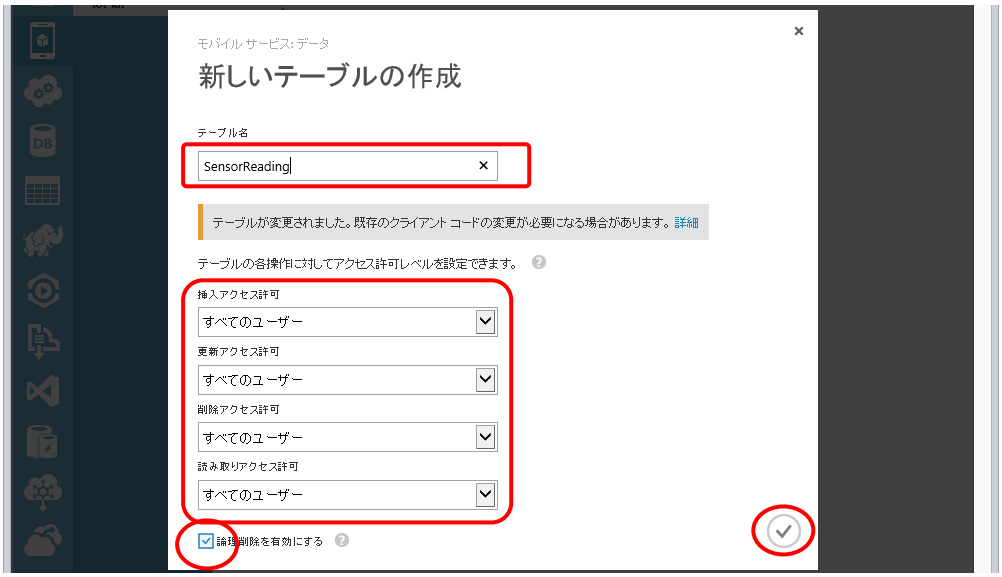
① テーブルを作成するモバイルサービス名をクリックして選択します。この場合は前手順で作成したモバイルサービス「iot-kit」にテーブルを作成します。

② モバイルサービスのメニュー画面が表示されます。ここで上部メニューの「データ」をクリックします。

③ テーブル管理画面です。「テーブルの追加」をクリックします。



④ 新しいテーブルの作成：テーブル名を入力して、アクセス許可（権限）を設定します。以下の例では、テーブル名を「SensorReading」として入力しています。テーブル名は自由な名前に設定可能ですが、後で開発するプログラムと合わせておく必要があります。
アクセス権限は後から変更可能です。業務用途では厳格に設定すべき項目ですが、ここでは実験目的のテーブルであるのとデバッグを容易にするために、全ての権限を「すべてのユーザー」に設定しておきます。「論理削除を有効にする」の項目は、デフォルトのチェックをしたままとしておきます。右下の「チェック」をクリックするとモバイルサービスの作成を開始します。



⑤ テーブル管理画面に戻ります。環境にも依存しますが、テーブルの作成は通常約 1 分程度で終了します。左上の←をクリックすると、モバイルサービス選択画面に戻ります。

4. IoT アプリケーションの開発

この章では.NET Gadgeteer で作成する EnOcean モバイルサービス・アプリケーションの開発手順の演習を行います。

.NET Gadgeteer の開発環境の準備と、簡単なアプリケーション開発方法、及び演習で使用する FEZ Spider のファームウェアのアップデート手順、ネットワーク設定方法については、別テキスト「準備と基本操作」をご参照ください。

4.1. モバイルサービス･アプリケーションの開発準備

プログラム開発に入る前に、開発環境の準備と確認を行います。

□解説：開発パッケージの既知の問題と対応策□

本書作成時点で、今回の開発環境で使用する.NET Gadgeteer SDK パッケージ（Microsoft Gadgeteer Core 2.43.1000 と GHI Electronics NETMF SDK 2014.R5）には次の問題があります。弊社で確認した各問題と対応策を示します。なおこの対応策はいずれも暫定的なものです。今後パッケージが修正されるか、より良い対応策が明らかになる可能性があります。

① DHCP でのネットワーク情報取得ができない場合がある
アプリケーション･プログラムの ProgramStarted() メソッド内で Network Interface を利用する前に Thread.Sleep(1) を呼び出して、タイミング調節をする。
② スタティック IP アドレス利用時にネットワークが有効化できない
アプリケーション・プログラム内で使用するスタティック IP を保存後、一旦 DHCP で Network Interface を起動し、スタティック IP アドレスを有効にする。
③ Builtin ネットワークデバイスのイベントが上がらない場合がある
アプリケーション･プログラム内で Network Interface 起動時にイベント処理を使用せず、V4.2 の頃までと同様に、NetworkInterface.GetAllNetworkInterfaces() を使用して Network Interface を呼び出して逐次処理を行います。
④ バグにより Gadgeteer.Webserver ネームスペースのライブラリに例外が発生する
開発前にライブラリを弊社作成の Gadgeteer.Webserver43 に入れ替えて使用します。

○演習３）

前記問題点④の対策として、別テキスト「準備と基本操作」解説の「演習１１」の手順に従って Gadgeteer.Webserver ライブラリの DLL の入れ替えをしておきます。すでに入れ替え済みの場合には、この作業は不要です。**トラブルを避けるため、入れ替え作業時には必ず Visual Studio 2013 を修了しておくことに注意して下さい。**

4.2. モバイルサービス･アプリケーションの開発

最初に、Microsoft Azure モバイルサービスへの接続を確認するために、簡単な.NET Gadgeteer で動作するモバイルサービス対応アプリケーション・プログラムを開発します。

EnOcean の USB 受信ドングルで受信したデータを実際に解析する部分は複雑なため、後回しにしますが、USB シリアル経由で受信したデータがあったタイミングでモバイルサービスにデータを送ります。

●演習４）

解説の手順に従ってモバイルサービス・アプリケーションを開発します。

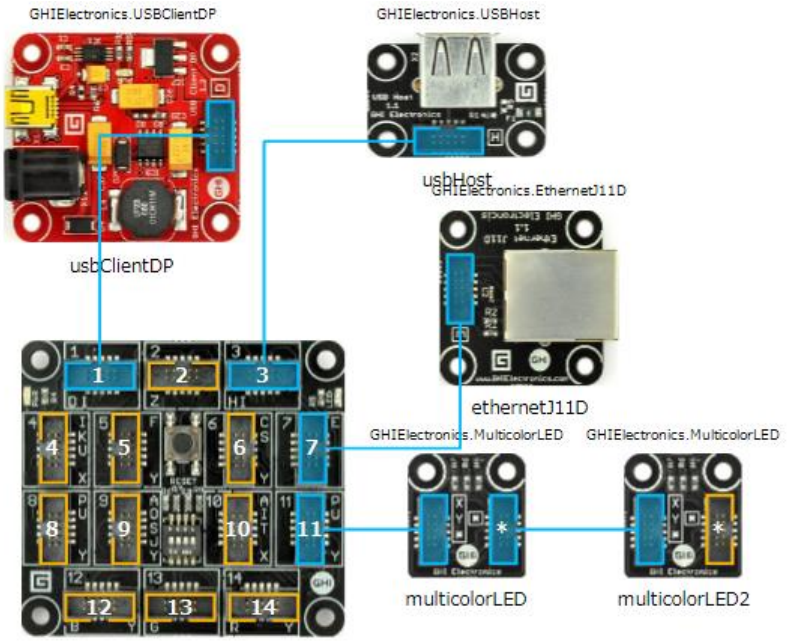
手順１：プロジェクトの作成とメインボード選択

C:¥METMF 以下に「MobileService」という名前の「FEZ Spider」のプロジェクトを作成します。**メインボード選択画面でバージョン番号を「4.3」に設定することに注意して下さい。**このソリューション名は別の名前に変えることも可能です。

「FEZ Spider」メインボードに次のモジュールを組み込み、配線して保存します。

① USB Client SP または Client DP モジュール
② Ethernet J11D
③ USB Host
④ Smart Multicolor LED ２個（オプション扱い）

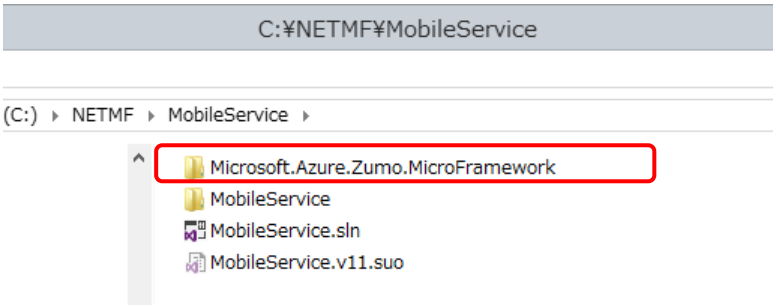
組み込み・配線したキャンバスの例を以下に示します。



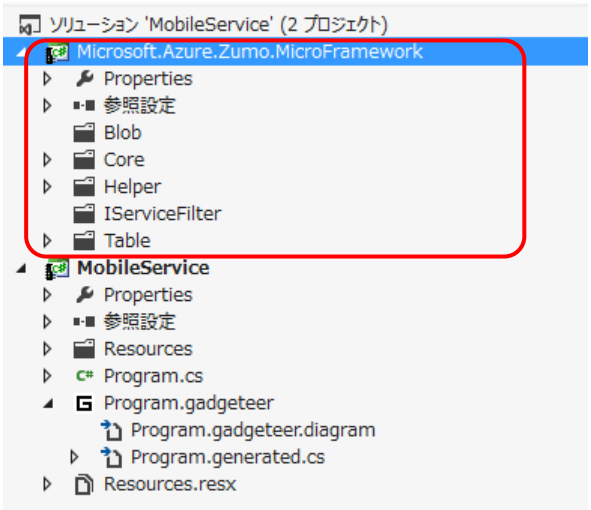
手順２：Mobile Service アクセスライブラリの追加

次の手順で Visual Studio で作成した MobileService ソリューションに、Azure Mobile Service アクセスに必要なライブラリを追加します。

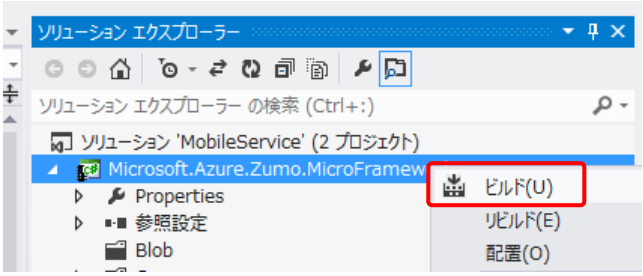
① 配布 CD 内の Microsoft.Azure.Zumo.MicroFramework フォルダをソリューションのフォルダ内にコピーします。（下図）



② Visual Studio のソリューションエクスプローラーで操作して、ソリューションにコピーした既存のプロジェクト「Microsoft.Azure.Zumo.MicroFramework」を追加します。
操作はソリューションを右クリックしてメニュー表示させ「追加」「既存のプロジェクト」で「csproj」ファイルを選択します。（下図）

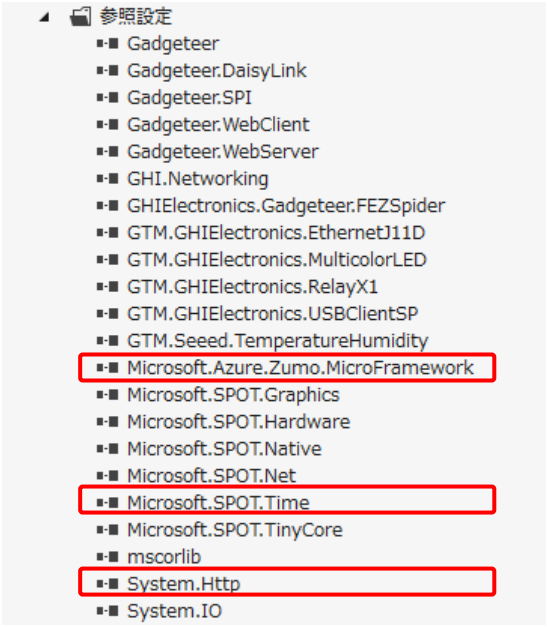


③ 前項で追加した「Microsoft.Azure.Zumo.MicroFramework」を選択後右クリックしてライブラリをビルドしておきます。



④ MobileService プロジェクトに次の参照設定を追加します。

a. Microsoft.Azure.Zumo.MicroFramework
b. Microsoft.SPOT.Time
c. System.Http

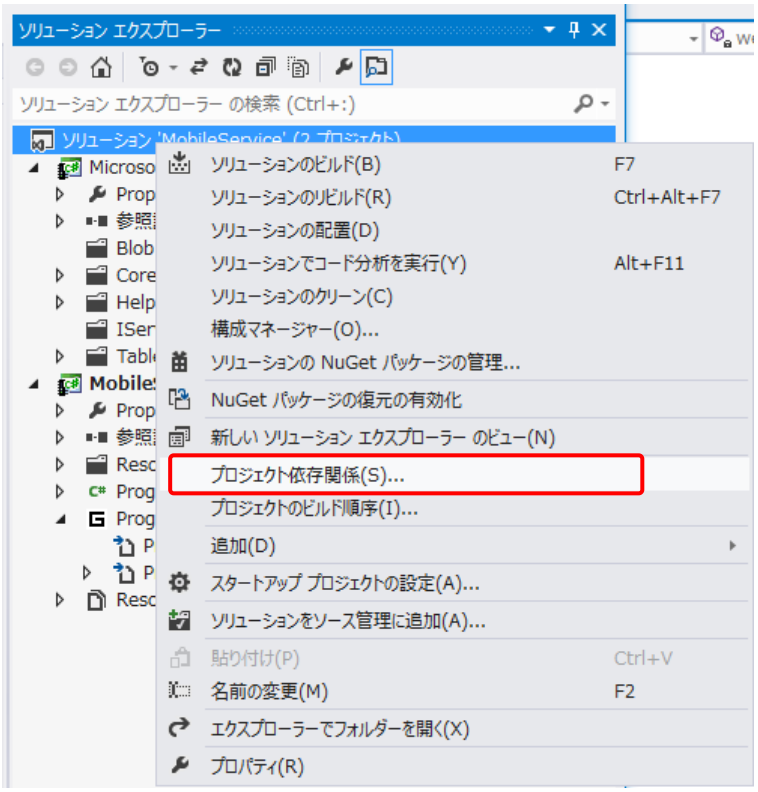


★注意：Microsoft.Azure.Zumo.MicroFramework の参照★

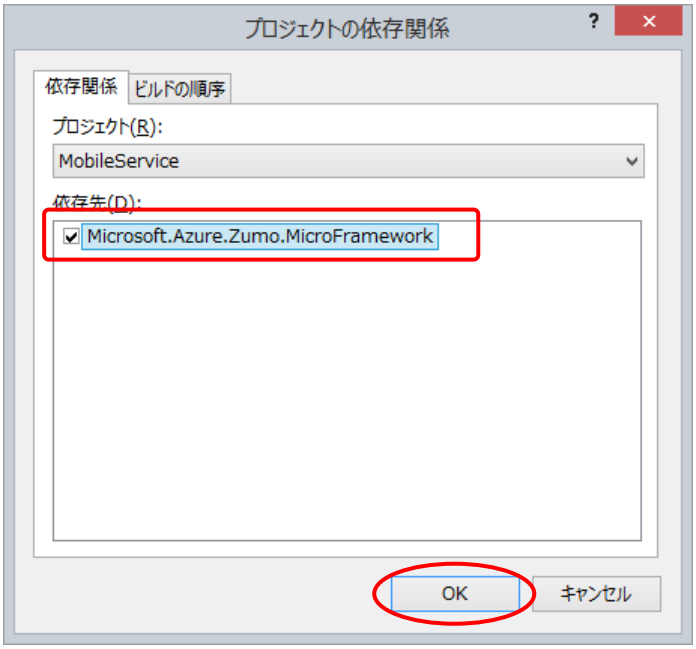
Microsoft.Azure.Zumo.MicroFramework の参照は、フォルダを開いて Microsoft.Azure.Zumo.MicroFramework¥bin¥Debug 以下のフォルダの DLL を選択して設定します。従って DLL のオブジェクトがビルドできていないと参照設定ができません。

⑤ 次の手順でプロジェクトの依存関係を設定します。

a. ソリューションのトップツリーで右クリックしてメニューを表示させ、「プロジェクトの依存関係」を表示させます。



b. MobileService プロジェクトが Microsoft.Azure.Zumo.MicroFramework に依存するチェックを付けた後、「OK」をクリックして修了します。



□解説：Mobile Service アクセスライブラリ□

今回使用する Mobile Service アクセスライブラリ Microsoft.Azure.Zumo.MicroFramework は元々、Nick Harris 氏により次の GitHub で公開されているライブラリです。

https://github.com/nickharris/Microsoft.Azure.Zumo.MicroFramework

これには、いくつか使い難いところがあったため、本演習のもととなったハンズオンを作成した太田寛氏が http://1drv.ms/1rG4JoF において改造したソースコードを公開しています。今回の演習ではこのソースコードを 4.3 用のプロジェクト用に修正したものを使用しています。

■ヒント：ライブラリの出所■

太田寛氏のブログ　Gadgeteer と Azure で作る簡易温度計 - Part 1

http://blogs.msdn.com/b/hirosho/archive/2014/07/25/gadgeteer-azure-part-1.aspx

手順３：ソースコードの修正

ソースコードを次の様に追加修正します。このプログラムでは Flash ROM 領域内の DHCP のチェックに基づいて、DHCP の使用・未使用を決定する様に動作します。

① 11 行目以降に次の using 文を using Microsoft.SPOT.Touch;の後に追加します。

```
using Microsoft.SPOT.Net.NetworkInformation;
using Microsoft.SPOT.Time;

using GHI.Usb.Host;
using Microsoft.WindowsAzure.MobileServices;
using Microsoft.Azure.Zumo.MicroFramework.Core;
```

```
1  using System;
2  using System.Collections;
3  using System.Threading;
4  using Microsoft.SPOT;
5  using Microsoft.SPOT.Presentation;
6  using Microsoft.SPOT.Presentation.Controls;
7  using Microsoft.SPOT.Presentation.Media;
8  using Microsoft.SPOT.Presentation.Shapes;
9  using Microsoft.SPOT.Touch;
10
11 using Microsoft.SPOT.Net.NetworkInformation;
12 using Microsoft.SPOT.Time;
13
14 using Gadgeteer.Networking;
15 using GT = Gadgeteer;
16 using GTM = Gadgeteer.Modules;
17 using Gadgeteer.Modules.GHIElectronics;
18 using GHI.Usb.Host;
19 using Microsoft.WindowsAzure.MobileServices;
20 using Microsoft.Azure.Zumo.MicroFramework.Core;
```

② Program クラス内の次の変数宣言を 26 行目、Public partial class Program 以降に追加します。

```
string ipAddress = "";
string ipAddress = "";
bool isNetworkConnected = false;
//ntp-b2.nict.jp
static byte[] TimeServerIPAddress = new byte[] { 133, 243, 238, 163 };

// Network
private enum Network { Started, Completed, Error };
private Network azureConnection;
private AutoResetEvent azureLock = new AutoResetEvent(true);
```

□解説：TimeServerIPAddress□

TimeServerIPAddress で指定している 133, 243, 238, 163 は、NICT が運営しているタイムサーバー「ntp-b2.nict.jp」の IP アドレスです。NICT ではほかにも次のサーバーを運営しています。必要に応じて、NICT に限らずより最適なタイムサーバーを指定して下さい。

133, 243, 238, 164 // ntp-b3.nict.jp
133, 243, 238, 243 // ntp-a2.nict.jp
133, 243, 238, 244 // ntp-a3.nict.jp

③ スタートメッセージ

Program Started 内の処理を次の様に書き替えます。
BaseDevice.DisconnectedEventHandler は現在正常動作しないので、コメントのままにしておきます。

```
// Subscribe to USBH event.
Controller.UsbSerialConnected += DeviceConnectedEvent;
//BaseDevice.DisconnectedEventHandler += DisconnectedEventHandler;

azureConnection = Network.Started;
SetupNetwork();
if (isNetworkConnected)
{
    SyncTimeService(); // Get time of day
    Thread.Sleep(500); // Wait for time event
    if (timeSynced)
```

```
            {
                Debug.Print("Time synced: " + DateTime.Now);
            }
            else
            {
                Debug.Print("Warning! Time is not synced");
            }
        }
        else
        {
            Debug.Print("No network connected. Running on local mode.");
        }
        if (isNetworkConnected)
        {
            SetupMobileService();
        }

        Debug.Print("Program Started with " + (isNetworkConnected ? "online" :
"offline"));

        Debug.Print("Connect USB!");
```

④ SetupNetwork() メソッドを追加します。

```
//
// Network
//
private void SetupNetwork()
{
    string saveIpAddress = "";
    string saveSubnetMask = "";
    string saveGatewayAddress = "";
    string[] saveDnsAddresses = null;
    GHI.Networking.EthernetBuiltIn ei = ethernetJ11D.NetworkInterface;
    NetworkInterface en = ethernetJ11D.NetworkSettings;

    Thread.Sleep(1);
    if (!ei.Opened)
    {
        Debug.Print("now open J11D");
        ei.Open();
    }
    if (ei.IsDhcpEnabled)
    {
        Debug.Print("now enable DHCP");
        ei.EnableDhcp();
    }
    else
    {
        // Save Flash ROM settings
        char[] caIpAddress = en.IPAddress.ToCharArray();
        saveIpAddress = new string(caIpAddress);

        char[] caSubnetMask = en.SubnetMask.ToCharArray();
        saveSubnetMask = new string(caSubnetMask);

        char[] caGatewayAddress = en.GatewayAddress.ToCharArray();
        saveGatewayAddress = new string(caGatewayAddress);

        int numOfDns = en.DnsAddresses.Length;
```

```
    saveDnsAddresses = new string[numOfDns];
    if (numOfDns > 0)
    {
        char[] dnsAddress0 = en.DnsAddresses[0].ToCharArray();
        saveDnsAddresses[0] = new string(dnsAddress0);
    }
    if (numOfDns > 1)
    {
        char[] dnsAddress1 = en.DnsAddresses[1].ToCharArray();
        saveDnsAddresses[1] = new string(dnsAddress1);
    }
}

PrintNetworkInfo("Start settings");
while (!isNetworkConnected)
{
    int trial = 0;
    foreach (var ni in NetworkInterface.GetAllNetworkInterfaces())
    {
        if (ni.NetworkInterfaceType == NetworkInterfaceType.Ethernet)
        {
            if (ni.IsDhcpEnabled)
            {
                ipAddress = ni.IPAddress;
                if (ipAddress != null && ipAddress != "0.0.0.0")
                {
                    break;
                }
                else
                {
                    ni.RenewDhcpLease();
                    Thread.Sleep(500);
                    ipAddress = ni.IPAddress;
                }
            }
            else
            {
```

```
                        ni.EnableDhcp();
                        Thread.Sleep(500);
                        ni.EnableStaticIP(saveIpAddress, saveSubnetMask,
saveGatewayAddress);
                        ni.EnableStaticDns(saveDnsAddresses);
                        ipAddress = ni.IPAddress;
                    }
                    break;
                }
            }
            isNetworkConnected = ipAddress != null && ipAddress != "0.0.0.0";
            if (!isNetworkConnected)
            {
                trial++;
                if (trial <= 3)
                {
                    Debug.Print(trial + ": Network failed, retry again.");
                    Thread.Sleep(500 * trial);
                }
                else
                {
                    Debug.Print("Too many network failed. Gave up.");
                    break;
                }
            }
        }

        PrintNetworkInfo("Completed");
    }
```

□解説：isNetworkConnected□

isNetworkConnected は、ネットワーク初期化設定が成功した場合にセットされる変数です。動作環境によっては、DHCP アドレスを取得できない場合があるため、アドレスが空の場合には３回試行する様にしています。

⑤ ネットワーク設定関連のメソッドを追加して行きます。

```
void PrintNetworkInfo(string arg)
{
    NetworkInterface en = ethernetJ11D.NetworkSettings;
    const string line = "------------------";
    Debug.Print(line + " " + arg + " " + line);
    Debug.Print("IP Address:   " + en.IPAddress);
    Debug.Print("DHCP Enabled: " + en.IsDhcpEnabled);
    Debug.Print("Subnet Mask:  " + en.SubnetMask);
    Debug.Print("Gateway:      " + en.GatewayAddress);
    Debug.Print(line + line + line);
}

private void SyncTimeService()
{
    TimeService.SystemTimeChanged += TimeService_SystemTimeChanged;
    TimeServiceStatus status = TimeService.UpdateNow(TimeServerIPAddress, 10);
    TimeService.SetTimeZoneOffset(540); // JST time origin 9h
}

bool timeSynced = false;
void TimeService_SystemTimeChanged(object sender, SystemTimeChangedEventArgs e)
{
    lock (this)
    {
        timeSynced = true;
    }
}
```

⑥ モバイルサービスへの接続処理です。この部分で先ほど作成したモバイルサービスへの接続パラメータを記述します。

WAMSUrl はモバイルサービスの URL です。

TableName は、データを保存するテーブル名です。

SensorDeviceId はデータを送信したセンサーデバイスを識別する GUID です。Visual Studio 付属「GUID の作成ツール」等を使用して、固有の GUID を持つ様に付け直すことを推奨します。

このプログラムでは、例としてモバイルサービス名を iot-kit、テーブル名を SensorReading

としています。必要に応じて変更して下さい。

```
private MobileServiceClient mobileService;
private static readonly string WAMSAppKey = "WAMS-APP-KEY";
private static readonly string WAMSUrl = "http://iot-kit.azure-mobile.net/";
private static readonly string TableName = " SensorReading ";
static readonly Guid SensorDeviceId = new Guid(
    0x35ecece1, 0xb391, 0x4879, 0xbe, 0x28, 0xd, 0x28, 0x6f, 0x5a, 0x3f, 0xa0);

private void SetupMobileService()
{
    mobileService = new MobileServiceClient(new Uri(WAMSUrl), WAMSAppKey);
}
```

⑦ モバイルサービスへのデータ追加処理です。最初に EnOcean のセンサーデータの種類を識別するための EEP(EnOCean Equipment Data) の構造体を宣言しています。

```
public struct EEPData
{
    public byte porg;
    public byte func;
    public byte type;
    public ushort manID;
    public EEPData(byte p, byte f, byte t, ushort m)
    {
        porg = p;
        func = f;
        type = t;
        manID = m;
    }
};

object InsertMobileService(object o)
{
    EnOceanTable queueMessage = (EnOceanTable)o;

    azureLock.Reset();

    Debug.Print("Set Lock");
```

```
            azureConnection = Network.Started;
            Debug.Print("Network Insert Start");

            try
            {
                var response = mobileService.GetTable(TableName).Insert(queueMessage);
                azureConnection = response == null ? Network.Error : Network.Completed;
                Debug.Print("Network Insert Completed OK");
            }
            catch (Exception ex)
            {
                Debug.Print(ex.Message);
                azureConnection = Network.Error;
                Debug.Print("Network Error");
            }
            azureLock.Set();
            Debug.Print("Release Lock");

            return null;
        }

        void SendToAzure(uint id, int rorg, ref EEPData eep, ref byte[] data, string
message)
        {
            SendToAzure(id, rorg, ref eep, ref data, message, 0);
        }

        void SendToAzure(uint id, int rorg, ref EEPData eep, ref byte[] data, string
message, int num)
        {
            var reading = new EnOceanTable()
            {
                NodeId = (new Guid((int)id, (short)eep.manID, 0xEEE, eep.porg, eep.func,
eep.type, 0, 0, 1, 0, 0)).ToString(),
                MeasuredDate = DateTime.Now,
                EEP = eep.porg.ToString("X2") + "-" + eep.func.ToString("X2") + "-" +
eep.type.ToString("X2"),
```

```
    AD0 = 0,
    AD1 = 0,
    AD2 = 0,
    Status = 0,
    Message = message,
    Alert = 0
};

if (rorg == 0x20) // RPS
{
    if (data.Length == 0)
    {
        return;
    }
    reading.AD0 = data[0];
}
else if (rorg == 0x22) // 4BS
{
    if (data.Length < 4)
    {
        return;
    }
    reading.AD0 = data[0];
    reading.AD1 = data[1];
    reading.AD2 = data[2];
    reading.Status = data[3];
}
else
{
    return;
}
if (azureConnection != Network.Error)
{
    BeginInvoke(
        new DispatcherOperationCallback(InsertMobileService), reading);

    Debug.Print("GC start");
```

```
            GC.WaitForPendingFinalizers();
            Debug.Print("GC end");
        }
    }
```

⑧ モバイルサービスに渡すデータ構造をプロパティで定義します。ここに記述のデータがJSON形式でモバイルサービスに送られます。モバイルサービス上ではここで記述した形式がそのままデータベースのレコードとなって保存されます。

```
public class EnOceanTable : IMobileServiceEntity
{
    public int Id { get; set; } // Will be changed to GUID
    public string NodeId { get; set; }
    public string Message { get; set; }
    public int Alert { get; set; }
    public int AD0 { get; set; }
    public int AD1 { get; set; }
    public int AD2 { get; set; }
    public int Status { get; set; }
    public string EEP { get; set; }
    public DateTime MeasuredDate { get; set; }
}
```

⑨ USB Host コネクタのコネクタ接続処理と、接続した EnOcean USB 受信器からの USB シリアルデータの受信処理部分です。本来は usbSerial_DataReceived() メソッドの中で受信データの解析をと整合性チェックを行ってデータの振り分け処理を行う必要があるのですが、ここではまず Microsoft Azure モバイルサービスに接続できることの検証を行うため、データ内容は確認していません。データ受信のタイミングで固定データを送信するようにしています。

```
void DisconnectedEventHandler(object sender, EventArgs e)
{
    Debug.Print("Device disconnected");
}

void DeviceConnectedEvent(object sender, UsbSerial usbSerial)
{
    Debug.Print("Device connected");
    Thread.Sleep(1000);
    Debug.Print(isNetworkConnected ? "Azure OK" : "Local Mode");
    usbSerial.BaudRate = 57600;
    usbSerial.DataBits = 8;
    usbSerial.Parity = System.IO.Ports.Parity.None;
    usbSerial.Handshake = System.IO.Ports.Handshake.None;

    usbSerial.DataReceived += usbSerial_DataReceived;

}

private static EEPData eep = new EEPData(0xA5, 0x02, 0x05, 0x0B);
private static byte[] data = { 0, 1, 2, 3 };
private static int testCount = 0;

private void usbSerial_DataReceived(UsbSerial sender,
    UsbSerial.DataReceivedEventArgs e)
{

    Debug.Print("R: " + e.Data[0].ToString("X2") + " " + e.Data.Length);
    SendToAzure(0, 0x22, ref eep, ref data, "test message", testCount++);
}
```

手順４：ビルドとデバッグ

追加修正のコーディングが終わったらばプログラムを保存して、次の手順でビルド・デバッグを行います。

① モジュールを組立・配線します。最初の段階では USB Host コネクタには何も接続しません。

② USB ケーブルで PC に接続して、ビルドしたプログラムを転送して動作させます。

③ デバッグ出力ディスプレイで「Connect USB」のメッセージを確認後、USB ドングルを USB Host ソケットに装着して下さい。データ受信が始まります。

④ Visual Studio の「出力」画面で動作を確認します。次の様に温度と湿度が表示されるはずです。

```
Date:09/272014 12:35:19, 27.1℃, 54.5%, 0
Mobile Service Requested - 201
```

MobileService フォルダ内に動作確認済のソリューションを置いています。参考にして下さい。

4.3. モバイルサービスのデータ検証

次の手順で Azure モバイルサービスにサインインして、転送したデータを確認します

●演習５）

手順１：モバイルサービスへのサインイン

サインインしていない場合は、Azure のトップページ

http://azure.microsoft.com/

から「ポータル」をクリックしてサインインします。

手順２：モバイルサービスへのサインイン

右側メニュータブから「モバイルサービス」選択後、今回作成したモバイルサービス名をクリックします。この例では「iot-kit」が参照するモバイルサービ名です。

手順３：データの表示

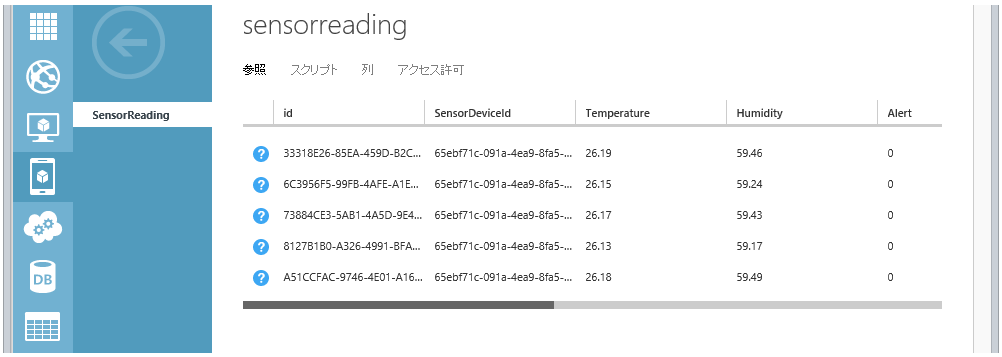
モバイルサービスのトップメニューから上部メニューの「データ」をクリックします。

手順４：テーブルの表示

データ領域内のテーブル選択画面で、今回作成した「SensorReading」という名前のテーブル名をクリックします。

手順５：レコードの表示と操作

テーブル内に蓄積したレコードが表示されます。



下部のボタンをクリックすることで、データ表示を更新したり削除したりできます。

■ヒント：モバイルサービスデータの Excel Power Query/View で可視化■

太田寛氏の以下のブログ「Gadgeteer と Azure で作る簡易温度計 - Part 1」では、モバイルサービスデータの Excel Power Query/View で可視化する手順について記載されています。

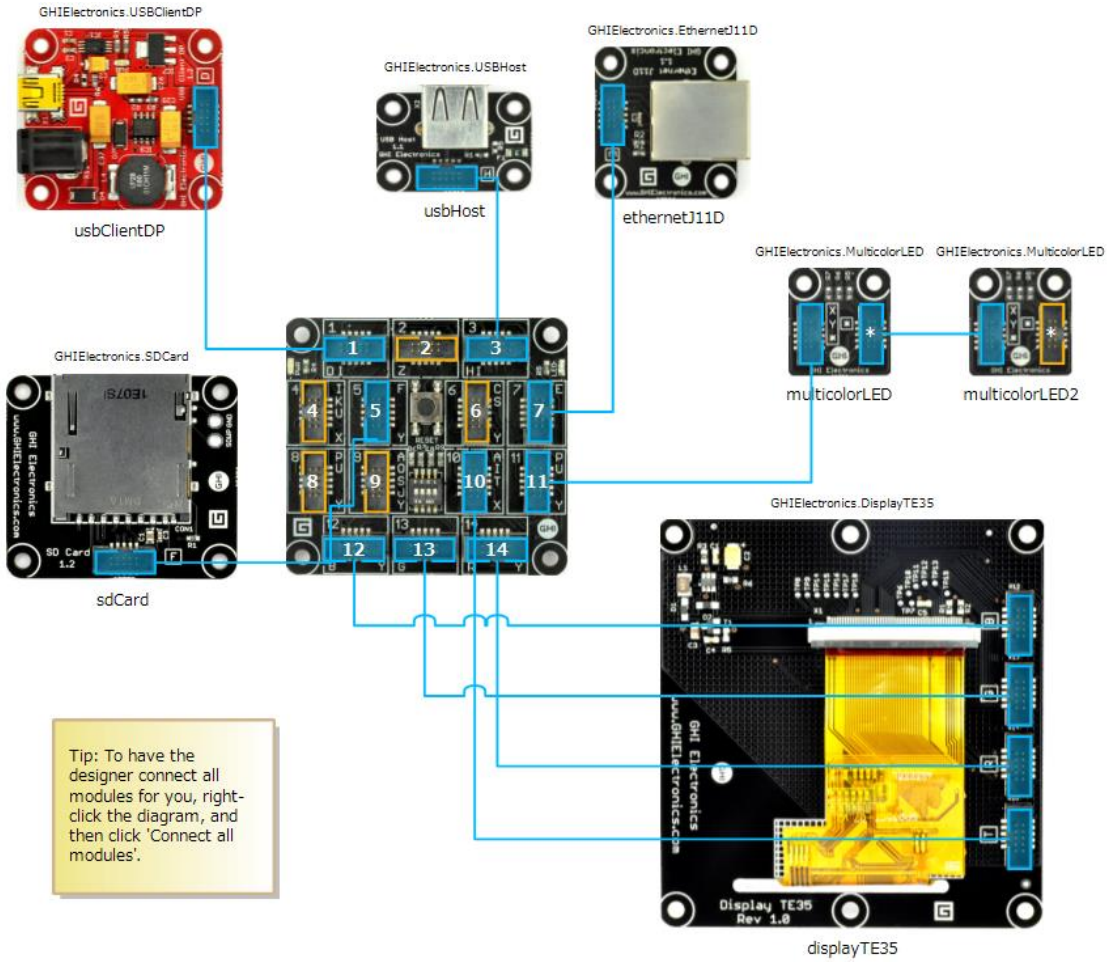
http://blogs.msdn.com/b/hirosho/archive/2014/07/25/gadgeteer-azure-part-1.aspx

Power Query を使用するためには Office 2013 Professional Plus 相当の Microsoft Excel が必要です。お持ちのかたは試してみて下さい。

4.4. EnOcean アプリケーションの開発

前項は.NET Gadgeteer のモジュールをセンサーとして利用していました。FEZ Spider メインボードには USB Host モジュールを接続して利用することが可能なため、EnOcean 無線プロトコルを使用する約 1000 種類ある各種のセンサーやアクチュエータを使用して、同様に Azure モバイルサービスに接続することが可能です。

① 使用する.NET Gadgeteer モジュール



.NET Gadgeteer では以下のモジュールを使用します。

-FEZ Spider メインボード

-USB ケーブル付 Client DP 電源モジュール

-TE35 3.5"タッチパネル付カラーディスプレイ

-Multicolor LED モジュール x2

-Ethernet モジュール

-USB Host モジュール

-SD Card モジュール

EnOceaan 無線通信では以下のモジュールを使用します。
-EnOcean PTM210J バッテリーレス　スイッチ（Double Verve Switch）
-EnOcean USB400J　 受信モジュール
-EnOcean STM431J　 無線温度センサー
-EnOcean HSM100　デジタル湿度センサー（EnOcean STM431J に組み込み済）

② ソリューション・ファイル

添付の「AzureMobileEnOcean」フォルダに動作確認済ソリューションが入っていますので、利用して下さい。そのままビルドして動作するはずです。

③ 使用上の注意点

.NET Gadgeteer の USB Host ソケットの使用制限として、システムの起動時には USB ドングル(EnOceanUSB400J) を起動時に外しておく必要があります。これは接続したまま起動すると USB デバイスの接続を認識しないというバグが原因です。

ディスプレイで「Connect USB」のメッセージを確認後、USB ドングルを装着して下さい。

現在のプログラムでは SD カードモジュールは接続しても書き込みを行いません。次のバージョンでサポートする予定です。それまでは、以下の情報を参考にしてプログラムを改造して利用して下さい。

https://www.ghielectronics.com/docs/51/accessing-folders-and-files

●演習６）

「AzureMobileEnOcean」フォルダ内のソリューションをコピー、Microsoft Azure 関連の接続情報パラメータを修正後ビルドして動作確認して下さい。

○演習７）

SD カードモジュールを追加して、受信したセンサーデータをログとして SD カードに日付時間情報とともにテキスト形式ファイルで保存するように追加修正して下さい。

■ヒント：マニュアルと参考資料■

.NET Micro Framework / .NET Gadgeteer のアプリケーション・プログラムを開発する際に参考となる API の解説やサンプルプログラムの入手先を次に示します。

●MSDN：.NET Micro Framework V4.3 API オンラインリファレンス

http://msdn.microsoft.com/en-us/library/jj610646.aspx

○MSDN：.NET Gadgeteer V4.3 API オンラインリファレンス

現在公開されていません。公開までお待ちください。

●GHI Electronics 社.NET Gadgeteer メインボードとモジュール等の資料

https://www.ghielectronics.com/docs

●GHI Electronics 社開発者向けコミュニティ

https://www.ghielectronics.com/community

●Socket 仕様、デザイン使用等.NET Gadgeteer 関連ドキュメント

http://gadgeteer.codeplex.com/documentation

●EnOcean 技術資料

http://www.enocean.com/en/knowledge-base/

（日本では ERP2 / ESP3 ・DolphinV4 core を採用しています）

●EnOcean 評価用ソフトウェアツール（アカウント作成後ダウンロード可能です）

http://www.enocean.com/en/download/

□解説：EnOcean 無線システムの動作確認□

EnOcean はバッテリーレスでセンサーデータを送信可能なエネルギーハーベスティング低消費電力デジタル無線システムです。Windows PC で動作確認をするためには、PC に DolphinView Advanced または DolphinView Basic を上記ツール配布先から入手してインストール後、USB400J を PC の USB コネクタに接続した後、DolphinView を立ち上げることで確認できます。

5. その他

本テキストについて、誤りや不明な点をみつけた場合には、以下宛にメールでご連絡をお願いします。

メール連絡先）
株式会社デバイスドライバーズ E-Kit 事業部
e-kit@devdrv.co.jp

以上