

 [](http://www.netmf.com/)



IoT Kit Hands-on  
トレーニング

Step 2：計測編

改定: 2015年 ５月 27日

初版: 2014年 11月 27日

日本マイクロソフト株式会社

デベロッパー エクスペリエンス ＆ エバンジェリズム 統括本部

太田　寛

Twitter：@embedded\_george

Blog : http://blogs.msdn.com/hirosho

この文章に含まれる情報は、公表の日付の時点でのMicrosoft Corporationの考え方を表しています。市場の変化に応える必要があるため、Microsoftは記載されている内容を約束しているわけではありません。この文書の内容は印刷後も正しいとは保障できません。この文章は情報の提供のみを目的としています。

Microsoft、SQL Server、Visual Studio、Windows、Windows XP、Windows Server、Microsoft Azureは Microsoft Corporation の米国およびその他の国における登録商標です。

その他、記載されている会社名および製品名は、各社の商標または登録商標です。

© Copyright 2014 Microsoft Corporation. All rights reserved.

目次

[0. はじめに 4](#_Toc420497372)

[0.1 はじめに 5](#_Toc420497373)

[0.2 学習内容 6](#_Toc420497374)

[1. センサーで物理情報を計測する 7](#_Toc420497375)

[1.1 温度センサー 8](#_Toc420497376)

[1.2 加速度センサー 12](#_Toc420497377)

[2. 計測データを送信する 14](#_Toc420497378)

[2.1 Webサービスに送信 15](#_Toc420497379)

[2.2 Event Hub に送信 17](#_Toc420497380)

[3. クラウドからデータを受信する 19](#_Toc420497381)

[3.1 IoT Kit に Event Hub からデータを受信する 20](#_Toc420497382)

[3.2 Event Hub にメッセージを送信するAPIを追加 23](#_Toc420497383)

[4. 試してみよう 26](#_Toc420497384)

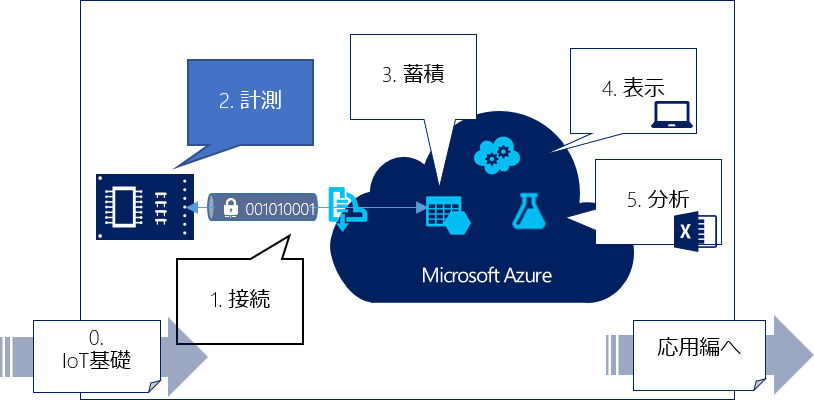
|  |
| --- |
| 1. はじめに |
|  |

## はじめに

本手順書は、一連のIoT Kit ハンズオントレーニングの二番目のステップで行う実習の具体的な手順を解説します。このステップの学習を開始する前に、前のステップの学習をおこなってください。

## 学習内容

このステップでは、組込み機器でのセンサーによる計測と、クラウドへの計測データ送信方法を学びます。



実習は以下の順番に従って行います。

1. センサーで物理情報を計測する
   1. 温度センサーで計測
   2. 加速度センサーで計測
2. 計測データを送信する
   1. Webサービスに送信
   2. Event Hubに送信
3. クラウドからデータを受信する
   1. Webサービスからデータを受信
   2. Event Hubを介してデータを受信
4. 試してみよう

Visual Studioの使い方や、Gadgeteerのハードウェア設計等、前のステップまでで学習した項目は、本手順書では詳しい説明を省きます。やり方が分らない場合は、前のステップに戻って再学習してください。

実習には、.NET Micro Framework / Gadgeteer 対応の小型ハードウェア（IoT Kitと記載）を使用します。本手順書で使用しているハードウェア以外でも、実習は可能です。使用するハードウェアに合わせて、適宜読み替えて実習を行ってください。

2014年11月末日現在、標準のIoT Kit ハードウェアの準備が整っておりません。このバージョンでは、GHI Electoronics 社製のFEZ Spider、及び、各種センサーを使った場合の手順を解説しています。IoT Kitハードウェアの準備が整った段階で、海底の予定です。

学習される皆様にはご迷惑をおかけしますが、ご了承ください。

|  |
| --- |
| 1. センサーで物理情報を計測する |
| このステップでは、.NET Micro Frameworkデバイスでセンサー計測する方法を実習します。   * **温度センサー** * **加速度センサー** |

## 温度センサー

温度センサーを使い、定期的なポーリングで温度を計測する方法を実習します。

※以下の説明は、FEZ SpiderをベースにしたIoT Kitでの実習例です。

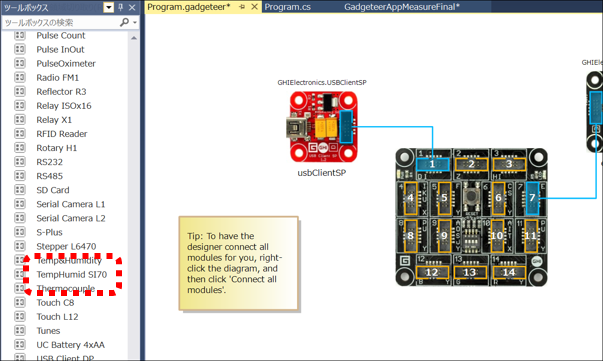
※ハードウェアが無い場合は、<http://blogs.msdn.com/b/hirosho/archive/2015/04/06/howtolearniotkitholwithemulator.aspx> で説明している方法で、IoT Kit Emulatorを使って実習を行ってください。

### 温度センサーをプロジェクトに組み込む

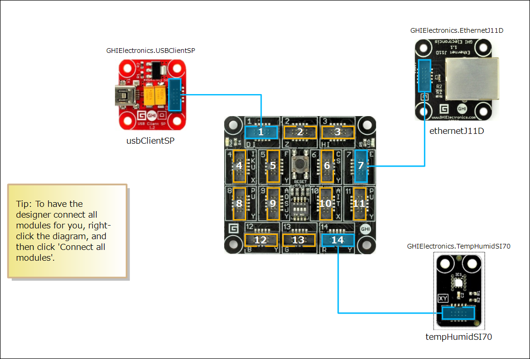
Visual Studio 2013を起動し、新しく、.NET Gadgeteerプロジェクトを作成します。

メインボードは、“FEZ Spider”を選択してください。USB Client SPと、Ethernet J11Dを追加して結線してください。

Program.gadgeteerにツールボックスから、“TempHumid S170”をドラッグ＆ドロップで追加します。



エディタ画面上で右クリックし、“Connect All Module”を選択して結線します。



表示に合わせて、温度センサーをメインボードに接続します



ハードウェアの準備はこれで完了です。

次に定期的にポーリングして、同期的に温度を計測するコードを追加します。定期的なポーリングを行うためにTimerクラスを、温度はTempHumidS170クラスを利用します。

Program.csを開きます。

ProgramStarted()メソッドの最後の“｝”の次の行に、Timerクラスのインスタンスと、温度センサーで測ったデータを格納しておく変数を定義します。

GT.Timer measureTimer;

double lastTemperature;

ProgramStarted（）メソッド内のDebug.Print（“Program Started”）の行のコメント行の直前にポーリング用のタイマー初期化コードを追加します。

**measureTimer = new GT.Timer(2000);**

**measureTimer.Tick += measureTimer\_Tick;**

**measureTimer.Start();**

// Use Debug.Print to show messages in Visual Studio's "Output" window during debugging.

Debug.Print("Program Started");

最初の行は、2000msec単位でのタイマー起動を指示しています。三行目のStart（）メソッドコールでタイマーが動き始め、2秒単位で、二行目でTickイベントに“＋＝”で登録している、measureTimer\_Tickメソッドがコールされます。

measureTimer\_Tickメソッドの本体は、ProgramStarted()メソッドの次に、以下のようなコードとして追加します。

measureTimer = new GT.Timer(2000);

measureTimer.Tick += measureTimer\_Tick;

measureTimer.Start();

// Use Debug.Print to show messages in Visual Studio's "Output" window during debugging.

Debug.Print("Program Started");

}

**void measureTimer\_Tick(GT.Timer timer)**

**{**

**measureTimer.Stop();**

**var th = tempHumidSI70.TakeMeasurement();**

**lock (this)**

**{**

**lastTemperature = th.Temperature;**

**}**

**Debug.Print("Temperature=" + lastTemperature);**

**measureTimer.Start();**

**}**

GT.Timer measureTimer;

double lastTemperature;

}

最初の行でmeasureTimerのStop()をコールすることにより、タイマーの実行を一旦止めています。これは、ポーリングでの実行が終わらないうちに、次のポーリングタイムが来て再入されないようにするためです。最後の行でmeasureTimerのStart()メソッドをコールして、タイマーの実行を再開しています。tempHumidSI70は、Program.gadgeteer上でツールボックスからドラッグ&ドロップした温度センサーにアクセスするための変数です。ドラッグ＆ドロップした際に名前が自動的に付与されています。この変数のTakeMeasurement（）メソッドをコールすると、その時点での温度を含むデータが戻り値として返されます。この中のTemperatureプロパティが計測された温度の値です。

この値をlastTemperature変数に代入して、最後に測った温度センサーとして保持します。この代入ブロックをlock(this){…}でくくっていて、マルチスレッド実行時の変数への同時アクセスを排他します。タイマーのTickイベントに登録されたメソッドのコールは、タイマーのポーリングスレッドからコールされます。複数のタイマーを起動したケースなど、複数のスレッドがlastTemperature変数を同時に読み書きする可能性があり、それを排除しています。良く判らないという方、とりあえず、正しく動くためのお呪いとして、クラスのプロパティに読み書きを行うコードはlockで囲うようにしましょう。

後は、USBケーブルでPCとIoT Kitのボード（FEZ Spiderの場合は赤いボードのUSBソケット）を接続し、Visual Studioの“デバッグ”→“デバッグ開始”を選択、またはF5キーを押して、実行を開始します。

以上で、同期的なセンサー計測は完了です。

使用するセンサーによって、メソッド構成や計測値のデータ構造は異なります。メソッド名や型、プロパティ名はそれなりに判りやすい名前になっているので、インテリセンスを駆使して使い方を類推してみてください

センサーデバイス制御に関するTips

センサーの計測制御は、ローレベルでは、アナログによる電圧計測で取得した値の変換や、UARTやI2C、SPIなど、ハードウェアの標準インターフェイスによるデジタルセンサーとの通信制御によるデジタルセンサーからのデータ取得処理が必要です。.NET Micro Frameworkには、このようなローレベルのハードウェアインターフェイスを制御するAPIも用意されており、これらのAPIを使ったセンサー制御も可能です。しかし、このような制御アプリを書くにはそれぞれのセンサーデバイスに関する詳細な知識が必要であり、読書のタイミング制御など複雑な処理をコーディングしなければなりません。

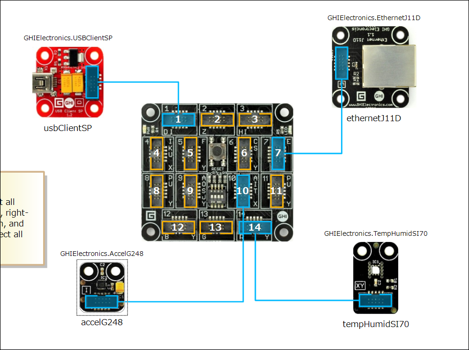
この実習では、これらローレベルな複雑な制御を隠蔽し、分りやすく使いやすいAPIを提供する.NET Gadgeteerを使うことにより、労せず温度や加速度をそのまま取得できるように配慮しています。

一連の実習を終えた後、別のセンサーを使ってみたい方は、是非、.NET Micro Frameworkで提供されているハードウェアインターフェイス向けのAPIにトライしてみてください。

## 加速度センサー

次に加速度センサーの計測値を非同期で取得する方法に取り組みます。

まず、1．1で作成したプロジェクトのProgram.gadgeteerに、Accel G248を加えます。



このセンサーも、TempHumidSI70と同様、同期的に計測値を取り出すセンサーです。1.1で学んだとおりに、計測値を変数に保持するコードを書いてみましょう。

追加したコードは以下の通りです。

void measureTimer\_Tick(GT.Timer timer)

{

measureTimer.Stop();

var th = tempHumidSI70.TakeMeasurement();

**var accel = accelG248.GetAcceleration();**

lock (this)

{

lastTemperature = th.Temperature;

**lastAccelX = accel.X;**

**lastAccelY = accel.Y;**

**lastAccelZ = accel.Z;**

}

Debug.Print("Temperature=" + lastTemperature);

measureTimer.Start();

}

GT.Timer measureTimer;

double lastTemperature;

**double lastAccelX;**

**double lastAccelY;**

**double lastAccelZ;**

}

太字で示した部分が追加したコードです。これで加速度センサーによる計測コードが出来上がりました。

加速度センサーTips

加速度センサーは一般的にX方向、Y方向、Z方向の3次元方向の加速度の値を計測可能です。これを3軸の加速度センサーと呼びます。それぞれの方向ごとの加速度を検出できるわけですが、地球上では下向きに重力が働いているため、静止状態では地球に向かって1重力加速度が検出されます。センサーを動かしたとき、動き始めと動きが止まるときに速度の変化が生じる、つまり加速度が生じるので、その加速度を計測できます。動き出して定速移動状態になったら、加速度は静止している時と同様、地球に向かって働いている重力のみ検出できます。

センサーの種類によっては、非同期で計測値を取得するタイプのものもあります。Program.gadgeteerを開いた状態でツールボックスを覗いてみてください。試しにTemp＆Humidityをハードウェアに追加してみましょう。このセンサーの場合、計測コードは、

measureTimer = new GT.Timer(2000);

measureTimer.Tick += measureTimer\_Tick;

measureTimer.Start();

**tempHumidity.MeasurementComplete += tempHumidity\_MeasurementComplete;**

**tempHumidity.StartTakingMeasurements();**

// Use Debug.Print to show messages in Visual Studio's "Output" window during debugging.

Debug.Print("Program Started");

}

**void tempHumidity\_MeasurementComplete(TempHumidity sender, TempHumidity.MeasurementCompleteEventArgs e)**

**{**

**lock (this)**

**{**

**lastTemperature = e.Temperature;**

**}**

**}**

double lastTemperature;

のようになります。この形式のセンサーの場合は、センサークラスのイベントにコールバック用のメソッドを登録して、計測開始を指示するメソッド（この場合は、StartTakingMeasurements（）メソッド）をコールすると、センサーデバイスで計測が完了するたびに、登録したコールバック用メソッドがコールされます。このメソッドコールもタイマーとは異なるスレッド上でコールされるので、lock(this){…}で囲って排他しています。

以上で、センサー計測の基礎学習が終了しました。

|  |
| --- |
| 1. 計測データを送信する |
| このSTEPでは、計測したデータをAzureに送信する方法を実習します。   * **Webサービスに送信** * **Event Hubに送信** |

## Webサービスに送信

出来上がったセンサー計測コードで計測した値をWebサービスに送信する方法を学習します。

### Webサービス側の変更

IoT Webサービスは、Step 1. 接続編の1-1で作成したASP.NETプロジェクトを拡張します。

“*IoTCloud*”プロジェクトをVisual Studioで開き、“Controllers”フォルダーに“SensorController”という名前で新たにコントローラクラスを追加します。センサーの値は、Step 1-1.でやったように、HeaderのメタデータとしてIoT Kitから送られてくることにします。SensorController.csのSensorControllerクラスに以下のようなGetメソッドを追加します。

public class SensorController : ApiController

{

public string Get()

{

var header = this.Request.Headers;

var deviceId = header.GetValues("device-id").ElementAt(0);

**var accelX = header.GetValues("sensor-accel-x").ElementAt(0);**

**var accelY = header.GetValues("sensor-accel-y").ElementAt(0);**

**var accelZ = header.GetValues("sensor-accel-z").ElementAt(0);**

**var temp = header.GetValues("sensor-temperature").ElementAt(0);**

return "Hello from Azure to " + deviceId;

}

}

コーディングが終わったら、AzureのWeb Sites上に発行することを忘れないでください。

### IoT Kit側の変更

次にIoT Kit側の変更を行います。前節のAccelG248による計測を追加した状態で、Step 1接続の1-3のステップに従って、Webサービスへの接続コードを追加してください。

Webサービスへの送信は、定期的にアップロードするタイマーを一つ追加して、そのタイマーが発火するたびに送信することとします。

諸々追加した後のProgramStarted（）及び関連するコードは、

**if (!ethernetJ11D.NetworkInterface.Opened)**

**{**

**ethernetJ11D.NetworkInterface.Open();**

**}**

**var ipAddr = EGIoTKit.Utility.NetworkUtility.Current.SetupNetwork();**

measureTimer = new GT.Timer(2000);

measureTimer.Tick += measureTimer\_Tick;

measureTimer.Start();

**uploadTimer = new GT.Timer(5000);**

**uploadTimer.Tick += uploadTimer\_Tick;**

**uploadTimer.Start();**

// Use Debug.Print to show messages in Visual Studio's "Output" window during debugging.

Debug.Print("Program Started");

}

Guid deviceId = new Guid(0xDC6C0152, 0xE84B, 0x44BB, 0x92, 0xE6, 0x5D, 0xC4, 0x70, 0x82, 0xE9, 0x77);

**void uploadTimer\_Tick(GT.Timer timer)**

**{**

**uploadTimer.Stop();**

**var request = HttpWebRequest.Create("http://[*IoTCloud*].azurewebsites.net/api/Sensor") as HttpWebRequest;**

**request.Headers.Add("device-id", deviceId.ToString());**

**request.Headers.Add("device-message", "hello from iot-kit");**

**lock (this)**

**{**

**request.Headers.Add("sensor-accel-x", lastAccelX.ToString());**

**request.Headers.Add("sensor-accel-y", lastAccelY.ToString());**

**request.Headers.Add("sensor-accel-z", lastAccelZ.ToString());**

**request.Headers.Add("sensor-temperature", lastTemperature.ToString());**

**}**

**using (var response = request.GetResponse() as HttpWebResponse)**

**{**

**if (response.StatusCode == HttpStatusCode.OK)**

**{**

**var reader = new StreamReader(response.GetResponseStream());**

**string message = reader.ReadToEnd();**

**Debug.Print(message);**

**}**

**}**

**uploadTimer.Start();**

**}**

**GT.Timer uploadTimer;**

以上で修正は完了です。HttpWebRequestのCreateメソッドの引数の最後の文字列が“Sensor”に変更されていることに注意してください。IoT Kitハードウェアで修正したアプリを動かし、Step 1接続で学習した方法で、Webサービス側でデータが受信されているか確認してください。

このステップで開発したプロジェクトは、以降の学習ステップでも利用します。このステップの実装が終わった状態のプロジェクトを“**GadgeteerAppSensorWebAPI**”という名前で呼ぶことにします。

## Event Hub に送信

次にIoT Kitで計測したセンサーデータをEvent Hubに送信する方法を学習します。

まずは、新しくGadgeteerプロジェクトを一つ作って、1.1、1.2の手順で、AccelG248とTempHumidSI70のセンサー計測コードを追加してください。このプロジェクト名は、“**GadgeteerAppSensorEH**”とします。以降の学習ステップでも再利用していきます。

次に、ステップ1の手順に従って、Amqp .NET Liteライブラリをプロジェクトに組み込んでください。

Program.csのProgramクラスに以下のコードを追加します。ProgramStarted（）メソッドの下に追加してください。

Guid deviceId = new Guid(*[Guidデータ列]*);

const string ehName = "*[EventHubName]*";

const string amqpAddress = "amqps://*[PolicyName]*:*[AccessKey]*@*[EventHubNS]*.servicebus.windows.net";

SenderLink amqpSender;

GT.Timer uploadTimer;

void SetupEventHub()

{

Address address = new Address(amqpAddress);

Connection connection = new Connection(address);

Session session = new Session(connection);

amqpSender = new SenderLink(session, "send-link" + eventhubName, eventhubName + "/Partitions/1");

}

このコードは、Step1で説明したEvent Hubへの接続部分をまとめたメソッドと、センサー計測データを定期的にアップロードするためのタイマー変数の定義です。

次に、ProgramStarted()メソッドにSetupEventHub()のコールとアップロードタイマーの初期化コードをセンサー計測タイマー初期化の次に加えます。

measureTimer = new GT.Timer(2000);

measureTimer.Tick += measureTimer\_Tick;

measureTimer.Start();

**SetupEventHub();**

**uploadTimer = new GT.Timer(5000);**

**uploadTimer.Tick += uploadTimer\_Tick;**

**uploadTimer.Start();**

そして、アップロードのコードを加えます。

void uploadTimer\_Tick(GT.Timer timer)

{

uploadTimer.Stop();

var message = new Message(System.Text.UTF8Encoding.UTF8.GetBytes("Sending Sensor"));

message.ApplicationProperties = new ApplicationProperties();

message.ApplicationProperties["DeviceId"] = deviceId.ToString();

lock (this)

{

message.ApplicationProperties["AccelX"] = lastAccelX;

message.ApplicationProperties["AccelY"] = lastAccelY;

message.ApplicationProperties["AccelZ"] = lastAccelZ;

message.ApplicationProperties["Temperature"] = lastTemperature;

}

amqpSender.Send(message);

uploadTimer.Start();

}

Step 1で学習した送信方法の基本を利用し、ApplicationPropertiesに値をセットすることにより計測データを送信します。コーディングはこれで終了です。

IoT Kit上で開発したアプリを動かし、Step 1で学習した方法で、Event Hubにデータが入っているか確認してください。

|  |
| --- |
| 1. クラウドからデータを受信する |
| このSTEPでは、Azureから送られたデータをEvent Hubを通じてIoT Kitで受信する方法を実習します。   * **IoT Kit に Event Hubからデータを受信する** * **Event Hubにメッセージを送信するAPIをWeb APIで追加する** |

## IoT Kit に Event Hub からデータを受信する

直前のステップで作成したGadgeteerAppSensorEH にAzureからのデータをEvent Hub を通じて受信するコードを加えます。

この実習では、データ受信用に“deviceCommand”という名前のConsumerGroupを作成し、Idが5のPartition で受信するよう実装します。

まず、Program.csファイルのProgramクラスに受信で使うReceiverLinkプロパティとConsumerGroup名を指定する変数を追加します。

Guid deviceId = new Guid(*[Guidデータ列]*);

const string ehName = "*[EventHubName]*";

const string amqpAddress = "amqps://*[PolicyName]*:*[AccessKey]*@*[EventHubNS]*.servicebus.windows.net";

SenderLink amqpSender;

**ReceiverLink** **amqpReceiver;**

**const string receiverConsumerGroup = "deviceCommand";**

GT.Timer uploadTimer;

次に、SetupEventHub()メソッドに、amqpReceiverの初期化コードを追加します。

void SetupEventHub()

{

Address address = new Address(amqpAddress);

Connection connection = new Connection(address);

Session session = new Session(connection);

amqpSender = new SenderLink(session,

"send-link" + eventhubName, eventhubName + "/Partitions/1");

**amqpReceiver = new ReceiverLink(session,**

**"receive-link:" + eventhubName,**

**eventhubName + "/ConsumerGroups/" + receiverConsumerGroup**

**+ "/Partitions/" + "5");**

}

Azureからの受信は、ReceiverLinkクラスのReceive（）メソッドをコールすることにより行います。このメソッドはコールすると、メッセージが送られてくるまで処理がブロックされるので、受信用のスレッドを作成し、そのスレッド上でメッセージ受信を繰り返す処理を実装します。

void ReceiveEventHub()

{

int index = 0;

int receiveCredit = 5;

for (; ; )

{

if ((index++ % receiveCredit) == 0)

{

amqpReceiver.SetCredit(receiveCredit);

}

try

{

var message = amqpReceiver.Receive();

if (message != null)

{  
 var enqueTime = message.MessageAnnotations[new Symbol("x-opt-enqueued-time")];  
 amqpReciever.Accept(message);

if (lastReceivedTimestamp <(DateTime) enqueTime)

{

amqpReceiver.Accept(message);  
 lastReceivedTimestamp = (DateTime)enqueTime;

if (message.Body != null)

{

var dataBodyBinary = (Data)message.Body;

if (dataBodyBinary.Binary != null)

{

var dataChars = System.Text.UTF8Encoding.UTF8.GetChars(dataBodyBinary.Binary);

var data = new string(dataChars);

if (data == "command")

{

if (message.ApplicationProperties != null)

{

foreach (var prop in message.ApplicationProperties.Map)

{

if (prop is System.Collections.DictionaryEntry)

{

var de = (System.Collections.DictionaryEntry)prop;

Debug.Print(de.Key.ToString() + ":" + de.Value.ToString());

if (de.Key.ToString() == "l-chika")

{

var duration = de.Value.ToString();

// ボード上のデバッグ用LEDを指定された期間でLチカする

}

}

}

}

}

}

}

}

}

catch (Exception ex)

{

Debug.Print(ex.ToString());

}

}

}

EventHubを通じて送られてくるメッセージを、for(;;)でループを回しながら受信します。SetCredit（）というメソッドをコールし、以降受信するメッセージの数を宣言します。ここでは5回を指定しています。上のコードでは、DataBodyプロパティから送信されたメッセージを取り出して、System.Text.UTF8Encoding.UTF.GetChars（）メソッドを使ってデコードし、送られてきたメッセージが”command”なら、メッセージのApplicationPropertiesから、更に値を取得する処理を例示しています。

IoT Kit のボード上に装備されているデバッグ用のLEDを、ApplicationPropertiesに格納されて送られてきた時間間隔で点滅させるコードは、皆さん自分で考えて実装してみてください。

最後に、ReceiveEventHub（）メソッドをスレッドで実行するコードを、SetupEventHub（）メソッドの最後の行に加えます。

void SetupEventHub()

{

Address address = new Address(amqpAddress);

Connection connection = new Connection(address);

Session session = new Session(connection);

amqpSender = new SenderLink(session,

"send-link" + eventhubName, eventhubName + "/Partitions/1");

amqpReceiver = new ReceiverLink(session,

"receive-link:" + eventhubName,

eventhubName + "/ConsumerGroups/" + receiverConsumerGroup

+ "/Partitions/" + "5");

**var thread = new Thread(ReceiveEventHub);**

**thread.Start();**

}

以上で、EventHubからデータを受信するコードの実装は完了です。IoT Kitでアプリケーションを実行し、動作を確認してください。

ReceiveEventHub（）メソッド内のReceive()メソッドをコールした時点でブロックされるのが正しい動きです。

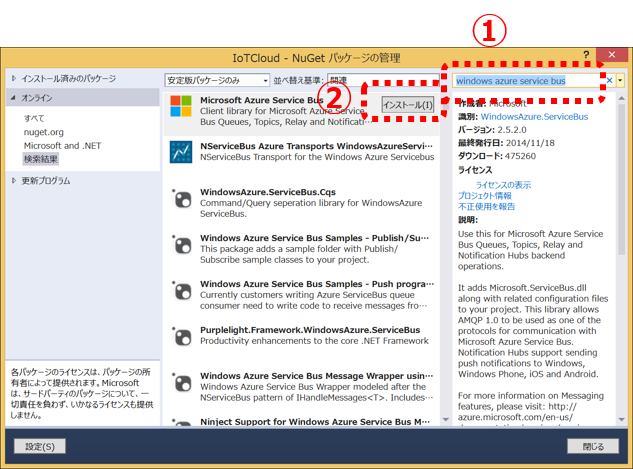
## Event Hub にメッセージを送信するAPIを追加

このステップでは、WebサービスからIoT KitにEvent Hubを通じてメッセージを送信する機能を作成します。2.1で改良した“*IoTCloud*”プロジェクトに機能を追加します。

まず、“*IoTCloud*”プロジェクトをVisual Studioで開きます。WebアプリでのEvent Hubの操作は、WindowsAzure Service Bus SDKを使用します。

インストール方法は以下の通りです。

ソリューションエクスプローラで、“参照設定”を右クリックし、“NuGet パッケージの管理”を選択します。



表示されたダイアログの検索ボックスに“Windows Azure Service Bus”と入力し、表示された項目右横の“インストール”ボタンをクリックします。ライセンスを同意してインストールを完了します。

次に、Web.configファイルをエディタで開きます。このSDKをインストールすると、Web.configの<appSettings>…</appSettings>に、Windows.ServiceBus.ConnectionStringのKeyが追加されます。これに、Service Busの名前空間、アクセスキーを追加します。

<add key="Microsoft.ServiceBus.ConnectionString" value="Endpoint=sb://*[Servie Bus Namespace]*.servicebus.windows.net;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=*[AccessKey]*" /></appSettings>

Step 1.の2.1で解説している方法を参考にして、[Service Bus Namespace]と[AccessKey]を置き換えてください。

前節で説明したように、IoT Kitは、“deviceCommand”というConsumer Groupを通じてメッセージを受け取ることを想定した実装を追加しました。このConsumer Groupを作成するコードを、Startup.csに加えます。

StartupクラスのConfiguration（）メソッドに以下のようにコードを追加します。このメソッドは、Webサービスが起動したときにコールされ実行されます。

public async void Configuration(IAppBuilder app)

{

ConfigureAuth(app);

**var ehCS = CloudConfigurationManager.GetSetting("Microsoft.ServiceBus.ConnectionString");**

**var namespaceManager = NamespaceManager.CreateFromConnectionString(ehCS);**

**var ehDescrip = await namespaceManager.GetEventHubAsync(eventhubName);**

**var cdDeviceCommand = await namespaceManager.CreateConsumerGroupIfNotExistsAsync(eventhubName, cgDeviceCommandName);**

}

**const string eventhubName = "device";**

**const string cgDeviceCommandName = "deviceCommand";**

CreateConsumerGroupIfNotExitsAsync（）を使い、“deviceCommand”という名前のConsumerGroupが無ければ、新たに作成します。

次に、Controllersフォルダーに、“LChikaController.cs”という名前で、コントローラクラスを追加します。”Controllers“フォルダーをソリューションエクスプローラで右クリックし、”追加“→”コントローラ“で簡単に追加できます。

出来上がったファイルに以下のようにコーディングします。

public class LChikaController : ApiController

{

**const string eventhubName = "device";**

**public async void Get([FromUri] string duration)**

**{**

**var ehCS = CloudConfigurationManager.GetSetting("Microsoft.ServiceBus.ConnectionString");**

**var client = EventHubClient.CreateFromConnectionString(ehCS, eventhubName);**

**string message = "command";**

**var data = new EventData(Encoding.UTF8.GetBytes(message));**

**data.Properties.Add("l-chika", double.Parse(duration));**

**var sender = await client.CreatePartitionedSenderAsync("5");**

**await sender.SendAsync(data);**

**}**

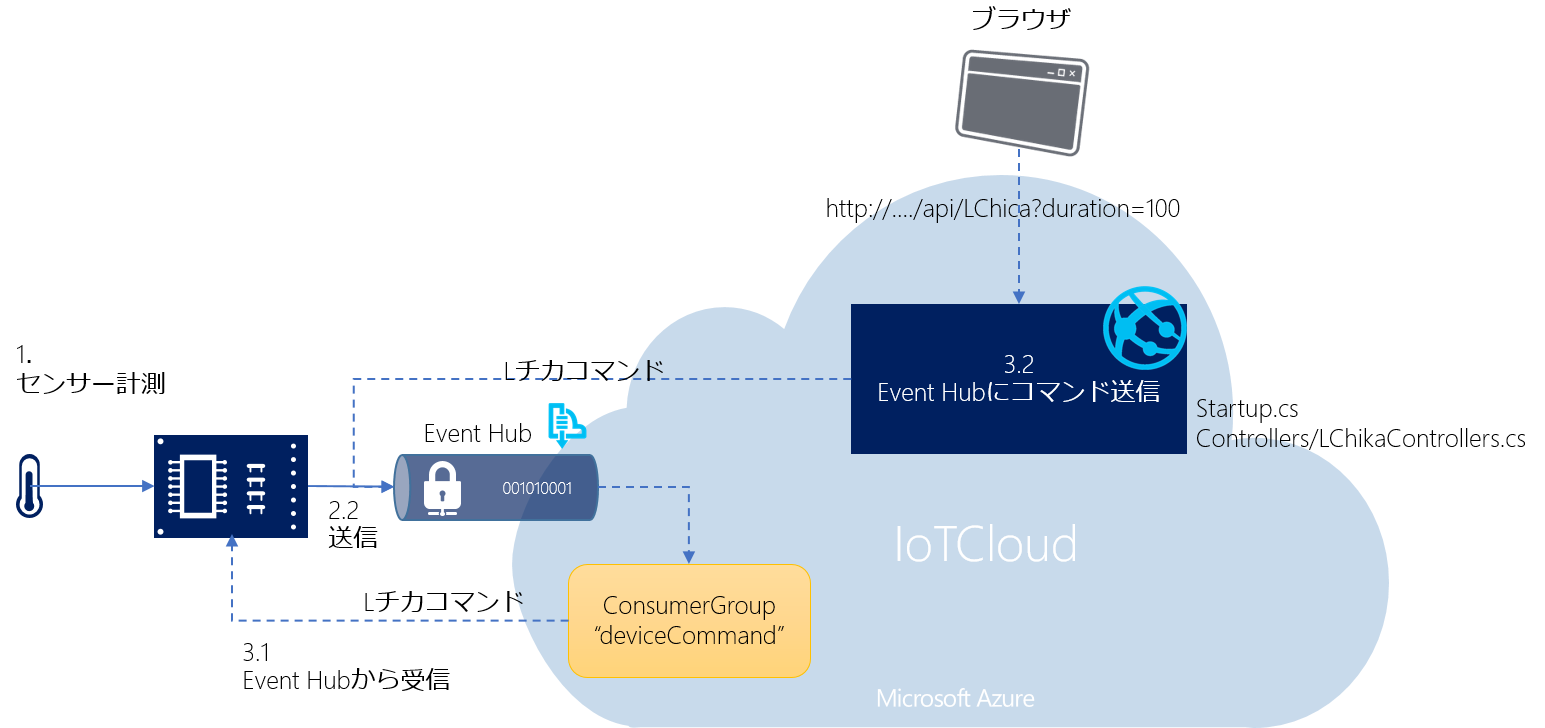
}

EventHubClientクラスのCreateFromConnectionString（）メソッドで、Event Hubにアクセスするクライアントが取得できます。後は送信するメッセージを組み立てて、IoT Kitが受信を試みているIdが5のパーティションにメッセージを送信します。Get（）メソッドの引数durationに”[FromUri]”というアトリビュートを付与しています。これを付与することにより、

http://*[IoTCloud]*.azurewebsites.net/api/LChika?duration=100

といった形式で、ブラウザからこのサービスを起動することができるようになります。LChika＝Lチカ、duration=チカチカ間隔…ということで、IoT Kit側にLチカの時間間隔を送信することができます。

説明とやることが多く混乱してしまっている人のために、データの流れと各作業の関係を図示しておきます。どこで何をやったか、どこに何が実装されているか、確認してみてください。



実装作業が終わったら、これまでの説明を参考に、デバッグやテストを行ってください。

以上で、センサーの計測、計測データの送信、クラウドからのコマンド受信の実装方法の説明は完了です。

|  |
| --- |
| 1. 試してみよう |
| このステップの学習で、センサー計測、計測データのクラウドへの送信、クラウドからIoT Kitへのコマンド受信の実装方法を学びました。  計測の時間間隔や、アップロードの頻度、クラウドに送信したデータを基にした、IoT Kitへのフィードバック命令など、色々と考えて、拡張してみてください。 |