

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



IoT Kit Hands-on  
トレーニング

Step 1：接続編

改版: 2015年 4月 5日

初版: 2014年 11月 26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 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](#_Toc416034243)

[0.1 はじめに 5](#_Toc416034244)

[0.2 学習内容 6](#_Toc416034245)

[1. Web アプリを通じた接続 7](#_Toc416034246)

[1.1 ASP.NET による Web アプリ 8](#_Toc416034247)

[1.2 .NET Micro Framework Emulator による接続 12](#_Toc416034248)

[1.3 IoT Kit による接続 17](#_Toc416034249)

[2. Event Hub を通じた接続 20](#_Toc416034250)

[2.1 Event Hub の作成 21](#_Toc416034251)

[2.2 IoT Kit を Event Hub につなぐ 23](#_Toc416034252)

[2.3 Event Hub の受信データ確認 26](#_Toc416034253)

|  |
| --- |
| 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最初のステップで行う実習の具体的な手順を解説します。学習を開始する前に、ポータルサイト <http://aka.ms/IoTKitHol> の記載内容、及び、Microsoft Virtual Academy から公開されている、以下の二つのコースを受講し、学習に必要な基礎知識を習得してください。

Note: Azureには無償の評価版が提供されています。リソース使用量に制限がありますが、簡単な評価を行う際には非常に便利です。Azureのアカウントをお持ちでなければ評価版にお申込みください。

また、MSDNサブスクリプション会員様は継続して無料で利用できる　**Microsoft Azure** 特典を提供しています。

**開発・テストでこそ使いたい Microsoft Azure**

<http://msdn.microsoft.com/ja-jp/dn818144.aspx>

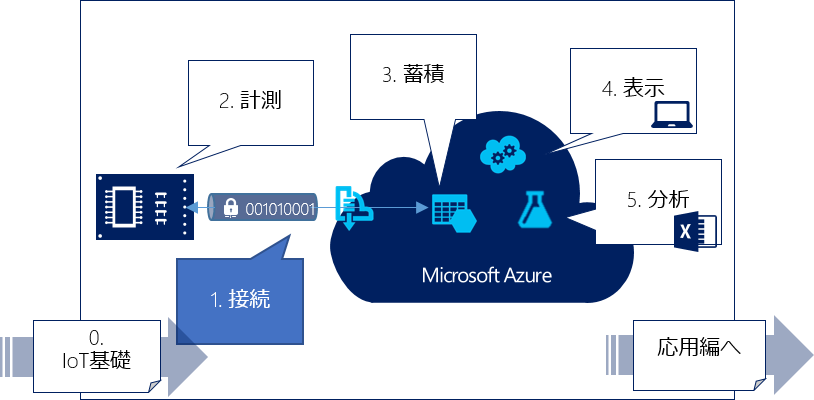
Note: この自習書に記載されている内容は**Microsoft Virtual Academy**で動画として公開されています。あわせてご覧ください。

**Microsoft Virtual Academy**

<http://www.microsoftvirtualacademy.com/>

## 学習内容

このステップでは、Internet of Things : IoTを構築するのに必要な、組込み機器や設備機器をクラウドにつなげる方法を学びます。



実習は以下の順番に従って行います。

1. Web アプリを通じた接続
   1. Web Siteによるクラウドサービス作成
   2. .NET Micro Framework Emulator による接続
   3. IoT Kit による接続
2. Event Hub を通じた接続
   1. Event Hub の作成
   2. IoT Kit による接続
   3. Event Hubの受信データ確認

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. Web アプリを通じた接続 |
| このステップでは、Azure Web Sites 上にASP.NET で作成した Web Service と、組込み機器を接続する方法を実習します。   * **ASP.NET による Web アプリ作成** * **.NET Micro Framework Emulator による接続** * **IoT Kit による接続** |

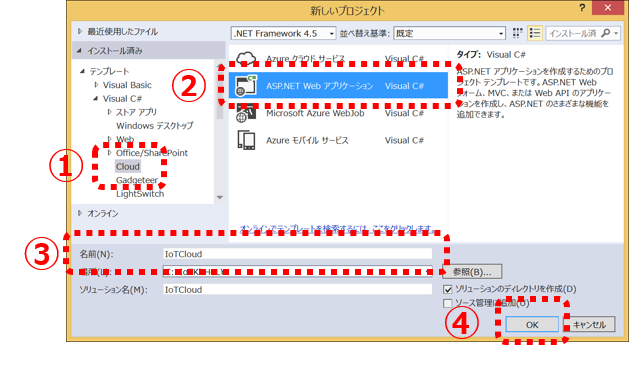
## ASP.NET による Web アプリ

最初に、Microsoft AzureのWeb Sitesの機能を使って、クラウド側で組込み機器を接続するための仕組みを作ります。

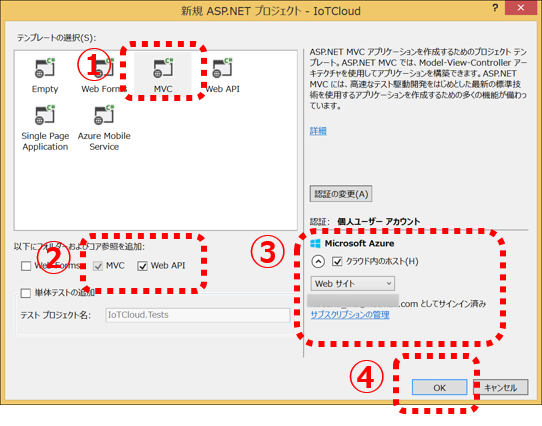
### ASP.NET プロジェクト作成

Visual Studio 2013を起動します。

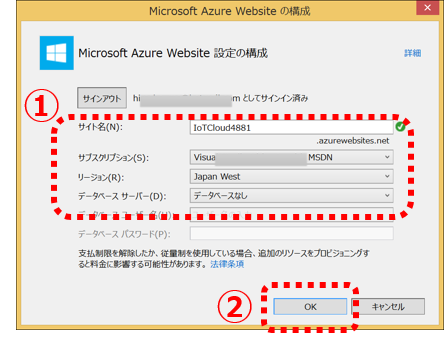
マウスで、メニューバーの“ファイル”をクリックし、“新規作成”→“プロジェクト”を選択します。



表示されたダイアログのテンプレートで、（①）“Visual C#”の“Cloud”を選択します。（②）“ASP.NET Web アプリケーション”を選択します。（③）名前の欄に、“IoTCloud” と入力し、（④）“OK”ボタンをクリックします。



表示されたダイアログの、（①）“テンプレートの選択”で、“MVC”を選択します。（②）“MVC”と“Web API”にチェックを入れます。（③）“サブスクリプションの管理”をクリックし、Microsoft Azureのサブスクリプションが使える、マイクロソフトアカウントや組織アカウントのメールアドレスで、サインインします。“クラウド内のホスト”をチェックし、コンボボックスで“Web サイト”を選択します。以上が完了したら、（④）“OK”をクリックします。

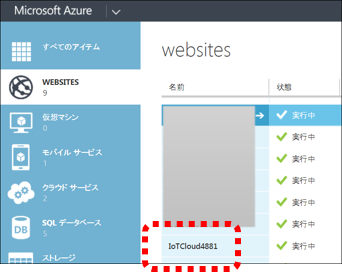


表示されたダイアログで、（①）“サイト名”“、“サブスクリプション”、“リージョン”、“データベースサーバー”を設定します。サイト名は、全世界で唯一でなければならないので、右側に緑色のチェックマークが表示される名前を設定してください。Microsoft Azureのサブスクリプション契約を複数持っている場合は、使用するサブスクリプションを適宜選択してください。“リージョン”は、このステップで作成するASP.NET Web アプリケーションが動く、データセンターの場所の選択です。折角なので、実習を行っている近くのリージョンを選択してください。“データベースサーバー”は“データベースなし”のままで構いません。設定が終わったら、（②）“OK”ボタンをクリックします。

※Microsoft Azure SDK 2.5.1から上のダイアログに、“App Service Plan”と“Resource Group”という2つの項目が追加されました。この項目ではそれぞれ、“Create New App Service Plan”と“Create New Resource Plan”を選択し、選択した後に表示される項目に適当な名前を入力するという作業を追加で行ってください。

これで、組込み機器が接続するためのクラウド向けWebアプリケーションプロジェクトのひな形が出来上がりました。

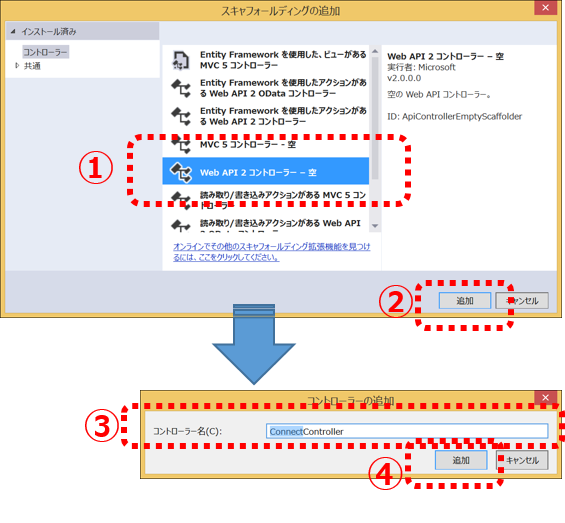
同時に、Microsoft Azure側でも、Web Sites に新しくWeb アプリが一つ追加されます。<http://manage.windowsazure.com> をインターネットブラウザで開いて、上の図の“サイト名”に入力した名前のアイテムが追加されていることを確認しましょう。



IoT Kit ハンズオントレーニングでは、今作成したWeb サービスをこのステップに続く以後のステップで拡張していきます。便宜上、“*IoTCloud*”という名前でこのプロジェクト、及び、Webサービスを呼ぶことにします。以降、“*IoTCloud*”と、この綴りでイタリック表記の名前が出てきたら、学習している各自が、このステップで作成したプロジェクト、及び、Webサービスを意味しているものと思ってくださいね。

### デバイス接続用コントローラの作成

Visual Studio に戻ります。“*IoTCloud*”プロジェクトを開いた状態で、ソリューションエクスプローラー内の、”*IoTCloud*“プロジェクトの下の”Controllers“フォルダーを右クリックし、”追加“→”コントローラを追加“を選択します。



表示されたダイアログで、（①）“Web API ２ コントローラ – 空”を選択し、（②）“追加”をクリックします。（③）“コントローラ名”に“ConnectController”と入力して、（④）“追加”ボタンをクリックします。“*IoTCloud*”プロジェクトの“Controllers”フォルダーに、“ConnectController.cs”というファイルが追加されます。

ConnectController.cs ファイルをソリューションエクスプローラーで、ダブルクリックして開きます。“ApiController”というクラスを継承した空の“ConnectController”クラスが定義されています。これを以下のように書き換えます。

public class ConnectController : ApiController

{

public string Get()

{

var header = this.Request.Headers;

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

return "Hello from Azure to " + deviceId;

}

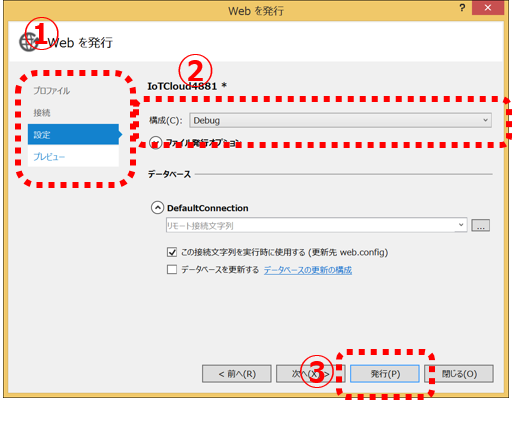
}

これで、組込み機器側からHTTPを通じて送信されてきたデータを受け取るための、Webサービス側のコーディングは終了です。

### Web アプリの公開

前段までで作成した、Web サービスをクラウド上で公開します。以降のステップのため、ここでは、“デバッグ”モードで発行します。

ソリューションエクスプローラーで、“*IoTCloud*”を右クリックし、“発行”を選択します。



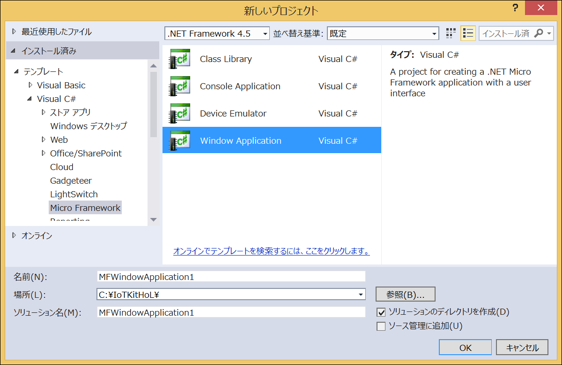
1. ）まず“設定”を選択し、（②）“構成”を“Debug”に変更します。そして、（③）“発行”をクリックします。Microsoft Azure側の実装が更新されます。これで、このステップの作業は完了です。

## .NET Micro Framework Emulator による接続

一つ前のステップで作成した“*IoTCloud*”にアクセスする組込み機器制御アプリを開発します。手始めに、PC上で動作する.NET Micro Framework Emulatorで接続を試みます。

### .NET Micro Framework プロジェクト作成

Visual Studio 2013を起動して、新規プロジェクトを作成します。プロジェクトテンプレートは、.NET Micro Framework の “Windows Application” テンプレートを使用します。



作成されたプロジェクトの参照に、“System.Http”と“System.IO”という名前の.NETコンポーネントを加えます。

作成されたプロジェクトの、Program.csファイルを開きます。このファイルの最後の、OnButtonUpというメソッドを以下のように変更します。

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

private void OnButtonUp(object sender, RoutedEventArgs evt)

{

ButtonEventArgs e = (ButtonEventArgs)evt;

// Print the button code to the Visual Studio output window.

Debug.Print(e.Button.ToString());

// 以下のコードを追加

**var request = HttpWebRequest.Create("http://egiotkitholservice.azurewebsites.net/api/DeviceConnect") as HttpWebRequest;**

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

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

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

**{**

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

**{**

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

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

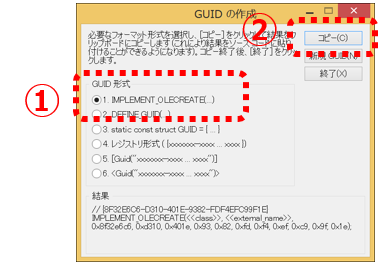
**Debug.Print(message);**

**}**

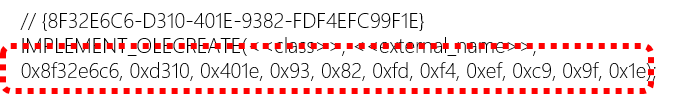
**}**

}

*[Guidのデータ列]*は、Visual Studio の“Guidの作成”ツールを使って生成したデータ列で置き換えます。“Guidの作成”ツールは、ツールメニューの同名の項目を選択すると表示されます。



まず、１の形式を選択し、“コピー”をクリックします。ノートパッドやテキストエディタにペーストし



赤い点線で囲った部分を*[Guidのデータ列]*にコピーします。

上記のコードをそのままテキストエディタで入力した場合、



のように、クラス名等でで、赤い波線が表示される個所があります。コンパイル時エラーとなります。これは、このクラスが属する名前空間がusing宣言されていなことに起因します。この場合は、赤い波線が表示されている文字列にカーソルを移動し、マウスをその文字列に重ねて左下に移動します。すると左下に“選択した項目をバインドするオプション”と表示されるので、黒い下向きの▼をマウスでクリックして、表示された“using …”を選択します。この操作でソースコードの先頭付近に必要なusing宣言が追加され、赤い波線は消え、コンパイルが通るようになります。”using…”が表示されない場合は、必要なコンポーネントが追加されていないことが原因です。今一度、System.HttpとSystem.IOが参照追加されているか確認しましょう。

### 接続テスト

以上で、クラウド側、組込み機器側の両方で準備が出来たので、接続試験を行います。

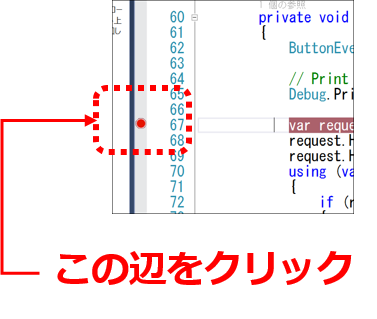
ここでは、Visual Studio を二つ起動して使います。それぞれで、“IotCloud”プロジェクトと、.NET Micro Framework Emulator向けプロジェクトを扱います。簡単のため、後者をNETMFアプリと呼ぶことにします。

まずはNETMFアプリの単体テストを行います。このステップを踏んで、組込み機器側のアプリケーションが正しく動作することを確認してから、クラウド側も含めたテストを行う、といった流れです。

先程作成したコードのHttpWebRequestのCreate（）メソッドの引数のURLの部分は、この一連のハンズオントレーニング向けのテスト用に用意した、Azure上で動作しているWeb サービスです。

NETMFアプリのプロジェクト側のVisual Studioで、Program.csを開き、ブレークポイントを設定します。

※“ブレークポイント”とは、プログラムのデバッグ時、実行を一旦止める行へのマークのことです。



ブレークポイントの設定は、設定したい行にカーソルを移動し、“F9”キーを押下するか、マウスでテキストエディタの左端の狭い帯をクリックすれば設定可能です。

var request = HttpWebRequest.Create…

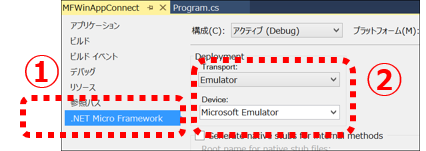
の行にブレークポイントを設定してください。

そして、NETMFアプリを実行します。“F5”キー（デバッグの開始）を押下等して、実行開始を指示します。



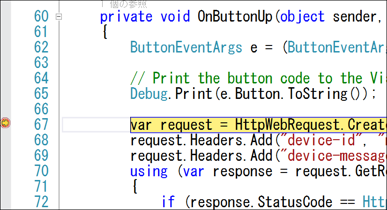
.NET Micro Framework のエミュレータが起動します。

※起動しない場合は、ソリューションエクスプローラで、プロジェクトアイコンを右クリックし、”プロパティ“を選択します。

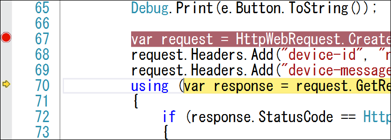


.NET Micro Frameworkをクリックし、“Transport”を“Emulator”に、“Device”を“Microsoft Emulator”に設定してください。

“Hello World”が表示されたら、赤い点線で囲まれたボタン、どれでもよいのでクリックしてください。クリックが終わった時点で、NETMFアプリを開いているVisual Studioが表示され、



設定しておいたブレークポイントの行がハイライトされ、実行が停止します。“F10”キー（ステップオーバー）押下で、処理を行ごとに進めていきます。



追加した一連のコードを順番に実行していき、response.StatusCodeがOKであることを確認し、StreamReaderで取り出した、Webサービスから送られたデータが、“Hello – *Guid*”であることを確認してください。Debug.Printで、”出力ビュー“にこの表示がされたら、動作確認終了です。

※この一連のコードを実行した際、Emulatorからdevice-idとして送信されたGUIDの値はAzure上に蓄積されます。送られたデータのうち、device-idとアクセスした時間以外のデータは一切保持しておりません。保持したデータは、このトレーニングの質向上の目的においてのみ使用します。

うまく動かない場合は、開発しているPCのネットワーク接続の確認や、<http://aka.ms/IoTKitHoL> にサービスの停止情報等をご確認ください。ブラウザで、<http://egiotkitholservice.azurewebsites.net/> が開けるかも確認をお願いします。

組込み機器側のアプリケーションが正しく動作することを確認した後は、クラウド側も含めた接続テストを行います。

デバッグをいったん終了（メニューバーの■ボタンをクリック）し、HttpWebRequestのCreateメソッドの引数の“http://...” の部分を、

、[http://*[IoTCloud]*.azurewebsites.net/api/Connect](http://[IoTCloud].azurewebsites.net/api/Connect)

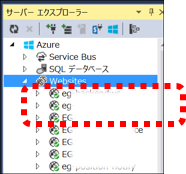
に変更します。”*[IotCloud]*”の部分を、1.1で作成したWebサイト名に置き換えてください。このテキストではWebサイト名は、“IoTCloud4881”という名前でMicrosoft Azure上で作成されていたので、この文字列は、<http://IoTCloud4881.azurewebsites.net/api/Connect> という文字列に置き換えることになります。

※プロジェクト名ではなく、Webサイト名であることに注意してください。

修正が完了したら、再度、“F5”等でデバッグ実行してください。

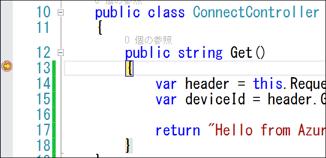
NETMFのプロジェクトをVisual Studioで開いてデバッグ実行したまま、もう一つVisual Studioを起動して、IoTCloudプロジェクトを開きます。

“IoTCloud”側で、ConnectController.cs の、“Get”メソッドにブレークポイントを設定します。そして、“サーバーエクスプローラ”ビュー上で（表示されていない場合は、メニューから“表示”→“サーバーエクスプローラ”を選択）、“Azure”→“Websites”と開き



発行済みのWebサイトを右クリックして、“デバッガーの接続”を選択します。すると、発行済みのサイトがブラウザで開き、Webサービスがデバッグモードで実行されます。

この状態で、Emulatorのボタンをクリックし、Emulator側のコードを順に実行していき、request.GetResponse()の行を実行すると、“*IotCloud*”のプロジェクトを開いているVisual Studioに切り替わり、



Getメソッドでブレークします。つまり、NETMFアプリ側でGetResponse（）メソッドをコールした際に、クラウド側にHTTP REST形式でのアクセスが行われ、結果、Microsoft Azure上の、“IotCloud”プロジェクトのConnectController.cs で実装されたクラスのGet()メソッドがコールされ、ブレークしたというわけです。“*IotCloud*”側のVisual StudioでF10でステップ実行をして、変数などの状態や、Headerに登録されて送られたIdやメッセージが届いていることを確認してみてください。Get()の実行がすべて終わると、NETMF側のVisual Studioが表示され、更にNETMF側のコードを実行していくと、“*IoTCloud*”側から送られたデータを取り出して、出力ビューに表示します。

以上で、.NET Micro Framework Emulatorによる、クラウドへの接続実習は完了です。

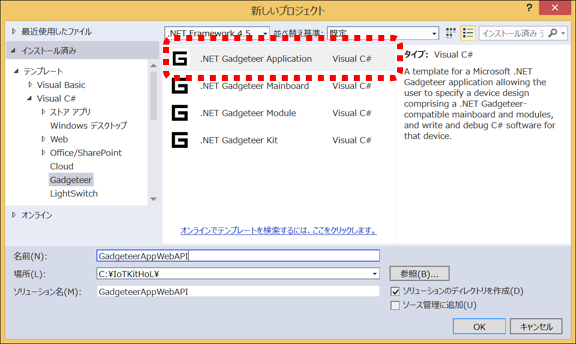
## IoT Kit による接続

次に、実際のハードウェアを使って、クラウドへの接続を行います。クラウド側は、1.1で作成したASP.NETで作成したWebアプリをそのまま利用します。

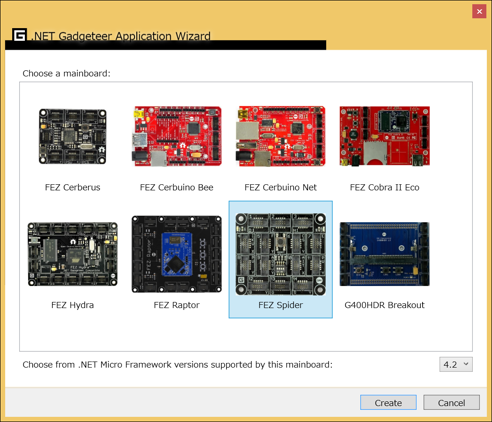
また、この実習では、EGIoTKit.Utilityライブラリを使うので、<https://github.com/ms-iotkithol-jp/Library>からライブラリをダウンロード（開いたページの右下の“Download ZIP”ボタンをクリック）し、ZIPファイルのプロパティで”ブロックの解除“を適用して、どこかに展開しておいてください。

### Gadgeteer プロジェクトの作成

Visual Studio 2013を起動して、新規プロジェクトを作成します。プロジェクトテンプレートは、Gadgeteer の “.NET Gadgeteer Application” テンプレートを使用します。

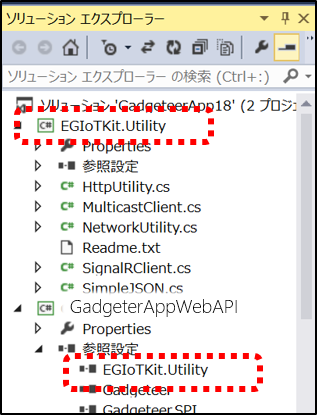


各自が利用するハードウェアに合わせてCPUボードを選択してください。CPUボードを選択した際の、右下のバージョンは、“4.3”を選択してください。

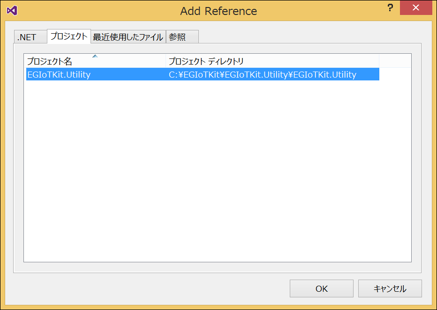


ソリューションエクスプローラで、出来上がったソリューションを右クリックして、“追加”→“既存のプロジェクト”を選択します。

表示されたダイアログで、展開したフォルダーに格納されているEGIoTKit.Utility\EGIoTKit.Utilityフォルダーの、EGIoTKit.Utility.csprojファイルを選択し、ソリューションに追加します。

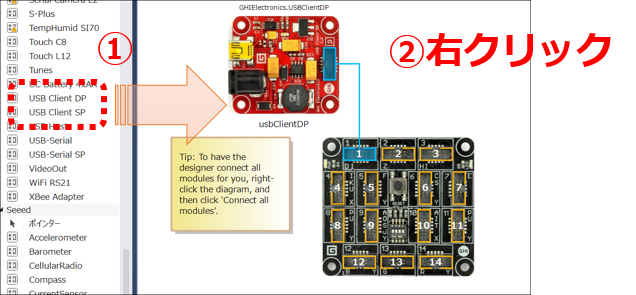


そして、追加したEGIoTKit.Utilityプロジェクトを、先ほど新規作成したプロジェクトの参照として追加します。ソリューションエクスプローラで、プロジェクトの“参照設定”を右クリックし、“参照の追加”を選択します。



“プロジェクト”タブを表示し、“EGIoTKit.Utility”プロジェクトを選択して、“OK”をクリックします。

※使用しているボードが“FEZ Spider”の場合、（①）ツールボックスから“USB Client DP”、及び、”Ethernet J11“をエディタ画面にドラッグ＆ドロップして、（②）エディタ上で右クリックし、”Connect All Module“を選択します。



Visual Studioのエディタ上で表示された結線通りにボードを配線してください。

Program.csファイルを開いて、“Program”クラスに、1-2で行った実習と同じく、Guidの値を格納するdeviceId変数を追加します。

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

// This method is run when the mainboard is powered up or reset.

void ProgramStarted()

次に、ProgramStarted()メソッドに以下のコードを追加します。

if (!ethernetJ11D.NetworkInterface.Opened)

{

ethernetJ11D.NetworkInterface.Open();

}

var ipAddress = EGIoTKit.Utility.NetworkUtility.Current.SetupNetwork();

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

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

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

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);

}

}

[IotCloud]の変更は、1-2 と同様に行ってください。上記のコードは1-2とほぼ同じです。

### 接続テスト

ハードウェアをUSBで開発しているPCに接続し、F5実行で、デバッグを開始します。後は、1-2の接続テストと同様の手順で、IoTKitハードウェア上で動作するアプリの接続テストをVisual Studioで行ってください。

|  |
| --- |
| 1. Event Hub を通じた接続 |
| このSTEPでは、Azureが提供するEvent Hub を通じて組込み機器を接続する方法を実習します。   * **Event Hub の作成** * **IoT Kit を接続** * **Event Hub の受信データ確認** |

## Event Hub の作成

まず、Microsoft Azure 上に、組込み機器とクラウドの接続をハイパフォーマンスに行うEvent Hub を作成します。

<http://manage.windowsazure.com> をブラウザで開き、サブスクリプション契約したアカウントでサインインします。



（①）左下の“＋新規”をクリックし、（②）“アプリケーション”→“Service Bus”→“イベントハブ”の順番に選択し、“カスタム作成”を選択します。



以下のように（①）“新しいイベントハブの追加”の項目に入力します。

* “イベントハブ名” →　“device”
* “地域” → 好きな地域を選んでください
* “名前空間” →　“新しい名前空間を作成する” を選択
* “名前空間名” →　※ネット上で一意の名前になるので、適宜、未登録の名前を入力します

“名前空間名”は、この実習書では、“*iot-eventhub-ns*”と以降表記します。各自が作成した名前空間名に適宜置き換えを行ってください。入力し終わったら、右下の→ボタンをクリックします。

以下のように（②）”イベントハブの構成“の項目に入力します。

パーティション数 → 実習用として“8”を入力

メッセージの保有期間 → 実習用として“１”を入力

実際のIoTシステム構築時には、この二つの値は、システムの特性、規模に応じて設定することになります。入力し終わったら、右下の✓ボタンをクリックします。

以上で、イベントハブが出来上がりました。以降のステップで使う、アクセスキーを確認しておきます。

左のタブの“Service Bus”を選択し、表示されたリストから先程作成した名前空間を選択し、ダブルクリックします。（①）“構成”をクリックして、



表示された、“RootManageSharedAccessKey”のプライマリーキーをコピーしておきます。

## IoT Kit を Event Hub につなぐ

次に実際のハードウェアを使ったEvent Hubへの接続を実習します。

### Amqp .NET Lite ライブラリをダウンロード

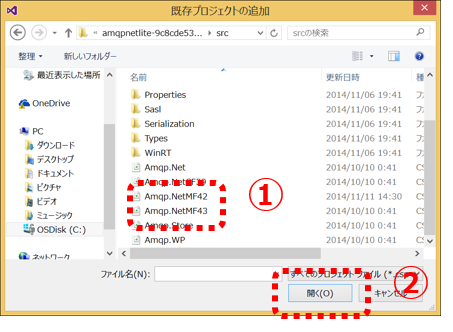
IoT Kit を Event Hub に接続するために、オープンソースで提供されている、AMQP.NET Liteを使用します。

<http://amqpnetlite.codeplex.com/>

の、“SOURCE CODE”タブをクリックして表示されたページの“Download”をクリックし、ZIPファイルをダウンロードします。ダウンロードしたZIPファイルは、適当な場所に解凍してください。

Gadgeteer プロジェクトへのライブラリの組み込み方法は、以下の通りです。

Visual Studio で作成した Gadgeteer プロジェクトをソリューションビューで表示し、ソリューションアイコンを右クリックします。プルダウンメニューで、“追加”→”既存のプロジェクト“を選択します。



解凍したしたAMQP .NET Liteのsrcフォルダーに入っている、Gadgeteerを.NET Micro Framework 4.2でアプリを開発する場合はAmqp.NetMF42 という名前のC#プロジェクトファイルを、.NET Micro Framework 4.3の場合は、Amqp.NetMF43を、選択します。さらに、1.3で説明したNETMFCloud プロジェクトも追加してください。

次に、Gadgeteer プロジェクトの参照設定を右クリックし、“参照の追加”を選択して追加したAmqp.NetMF4xプロジェクトを参照追加してください。

### IoT Kit を接続する

IoT Kit でEvent Hub に接続するアプリを開発します。1.3で解説した方法で、Gadgeteer プロジェクトを作成します。出来上がったプロジェクトにAMQP .NET Lite ライブラリを前節で説明した方法で組み込みます。

Program.cs ファイルを開き、Program クラスのProgramStartedメソッドの下に、いくつかのプロパティとAMQPSend という名前のメソッドを追加し、以下のようにコーディングします。

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;

void AMQPSend()

{

Address address = new Address(amqpAddress);

Connection connection = new Connection(address);

Session session = new Session(connection);

string eventhubName = "device";

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

var message = new Message(System.Text.UTF8Encoding.UTF8.GetBytes("Hello IoT Kit"));

message.ApplicationProperties = new ApplicationProperties();

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

amqpSender.Send(message);

}

最初の行の[Guidデータ列]は、1.2で説明したのと同様な方法で置き換えてください。

*[EventHubName]*は、Microsoft Azureポータルで作成したEvent Hubの名前です。2.1では“device”という名前で作成したので、

const string ehName=”device”;

となります。amqpAddressに設定する文字列ですが、[PolicyName]、[AccessKey]、[EventHubNS]は、2.1節でEvent Hubを作成した際に出てきたパラメータ群です。

PolicyName → RootManageSharedAccessKey

AccessKey → 2.1節でメモしたプライマリーキーのURLエンコーディングをかけた値

EventHubNS → Event Hubの名前空間名。2.1で作成した iot-eventhub-ns を使用

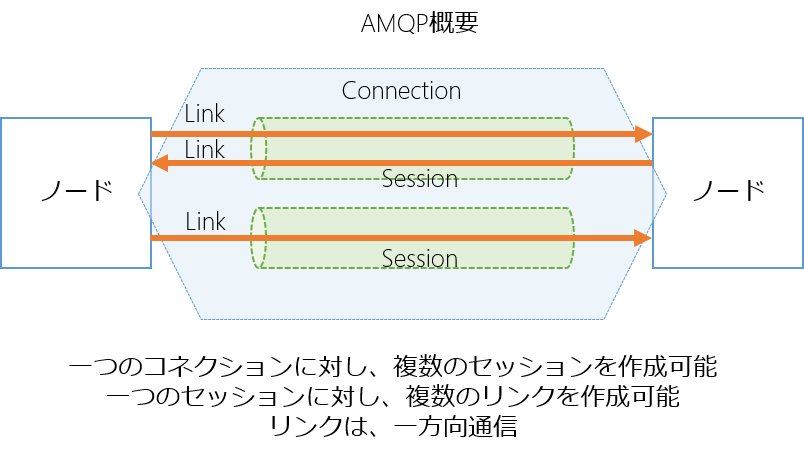
とします。結果としてamqpAddressの値は、

“amqps:://RootManageSharedAccessKey:[URLエンコード済みのAccessKey]@iot-eventhub-ns.servicebus.windows.net”

となります。

Address address = new Address(amqpAddress);

から始まるコードが、AMQPによる通信を実現します。AMQPは下図に示すようなプロトコルなので、



最初にConnection作成、Connectionに対してSessionを作成、最後に送信用のリンク（SenderLink）を作成します。SenderLinkを作成する際、“Partitions/1”と指定していますが、MVAオンライントレーニングで解説したように、Event Hubはパーティションを複数定義でき、送信用のLinkを作成する際には、どのパーティションに送るかを指定可能です。先に示したコードはIDが1のパーティションに送付するLinkを作成しています。AMQPについて詳しく知りたい方は、<http://www.amqp.org/> をご覧ください。

Linkを作成したら、送信するメッセージを作成します。先に示したコードでは、Messageクラスのインスタンスを作成し、Bodyにメッセージを、ApplicationPropertiesにDeviceIdやセンサー計測データを登録して送付することになります。以上の説明を踏まえ、先のコードを読んで、コーディングを行ってください。

URL Encoding

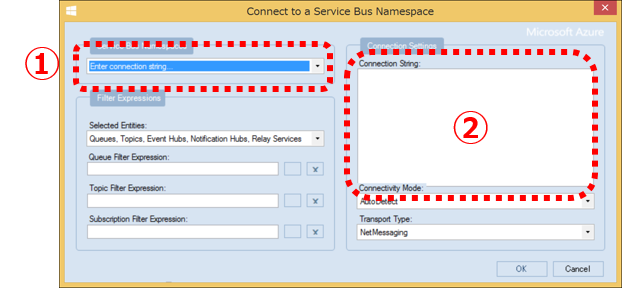
Webサービスにアクセスする際、利用できない文字を使う際に行われるエンコーディングです。<http://en.wikipedia.org/wiki/Percent-encoding> のPersent-encoding reserved characters に従って文字を置き換えます。アクセスキーによく出てくる、“＋”は“%2B”、“/”は“%2F”、”=”は”%3D”に、それぞれ置き換えます。

## Event Hub の受信データ確認

Event Hubで受信したデータは、そのままでは、何が受信されたか確認することができません。ここでは、オープンソースとして提供されているService Bus Explorerを使って、Event Hubに受信したデータを見る方法を学びます。

Service Bus Explorerは、<https://code.msdn.microsoft.com/windowsapps/Service-Bus-Explorer-f2abca5a> から公開されています。このサイトからC#のプロジェクト一式をダウンロードし、解凍してVisual Studioで開き、実行します。

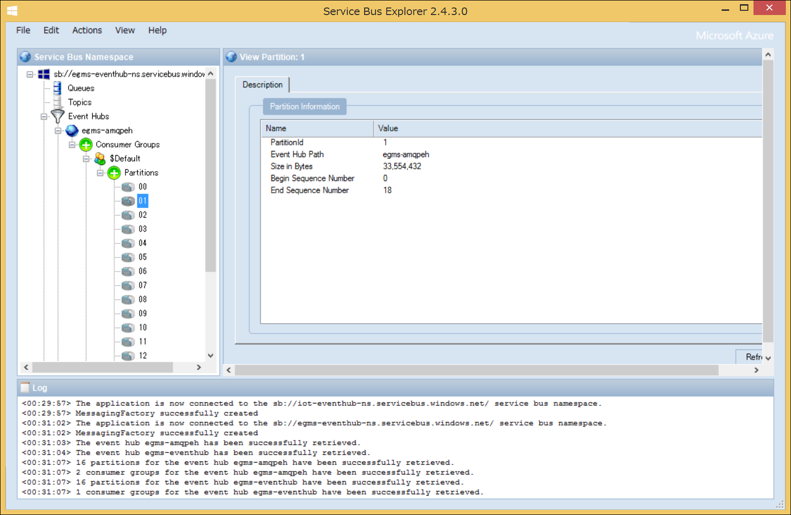
メニューの“File”→“Connect”を選択します。



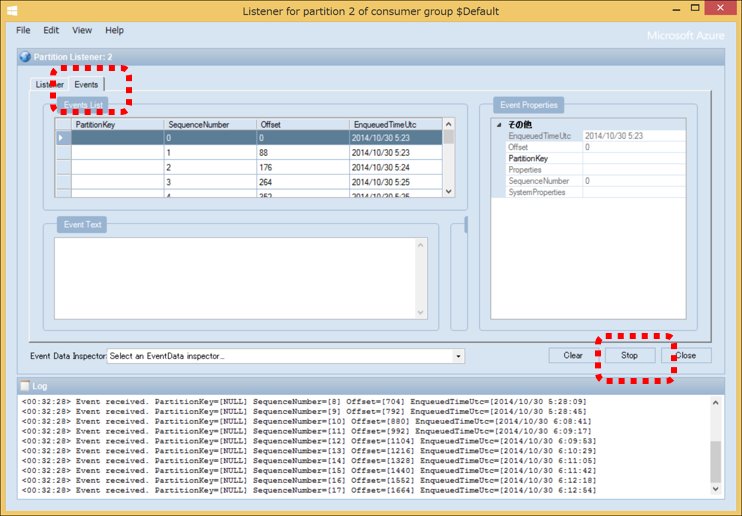
1. ）“Service Bus Namespace”を、“Enter Connection String”に選択します。（②）にService Bus の接続文字列をコピー＆ペーストします。接続文字列の取得は、Microsoft Azure のマネージメントポータルで、Service Busの項目を選択し、



接続情報をクリックします。表示されたダイアログの接続文字列情報をコピー＆ペーストし、“OK”をクリックします。



表示されたService Busの子要素を開いて、参照したいEvent Hubを表示します。中身を見たいEvent Hubのパーティションを右クリックして、“Create Partition Listener”を選択します。



右側の“Start”ボタンをクリックし（図は押下後“Stop”に変化した後の状態）、左上の“Event”タブをクリックします。IoT Kitからのデータが正しく受信されていれば、リストに受信データのリストが表示されます。各項目をクリックすると、受信データの詳細を表示できます。

以上で、本ステップの学習は完了です。以降のステップは、本ステップをベースに学習を進めていくので、今一度、動作確認、復習を行って、説明内容を血肉に浸透させてください。