基般

3 Windows Phone によるセンサプログラミング

太田　寛
日本マイクロソフト（株）

Windows Phone とセンサ

■ センサを活用したアプリケーション開発

過去，マイクロソフトはスマートフォン向けに，Windows Mobile 5.x や Windows Mobile 6.x を提供していました．本稿で取り上げる Windows Phone は，過去の Windows Mobile の系列のバージョンアップではなく，まったく新しくゼロから開発されたスマートフォン向け OS です．2010 年に Windows Phone 7.0 が米国を含む数カ国でリリースされ，日本でも最新バージョン Windows Phone 7.1 を搭載するスマートフォンが 2011 年 9 月にリリースされました．アプリケーション開発向けに提供されるプログラミング環境やライブラリ群も Windows Mobile 時代から一新されています．本文中に出てくる Windows Phone に関する説明はすべて，最新版の Windows Phone 7.1 に基づくものであることをご承知おきください．

Windows Phone にはスマートフォンで有用なさまざまなセンサが搭載されています．Windows Phone のアプリケーションは，マイクロソフトが提供する Windows Phone SDK 7.1 という開発キットを使って開発を行います．SDK で提供されるクラスライブラリや開発環境を使って，端末の位置や動き，向きなどを取り込んだアプリケーションをすばやく開発できます．また，一貫性ある API 群により，高機能なグラフィクス描画やカメラ機能，ソーシャル機能をはじめとする，Windows Phone 端末で提供されるさまざまな機能との連携も容易です．

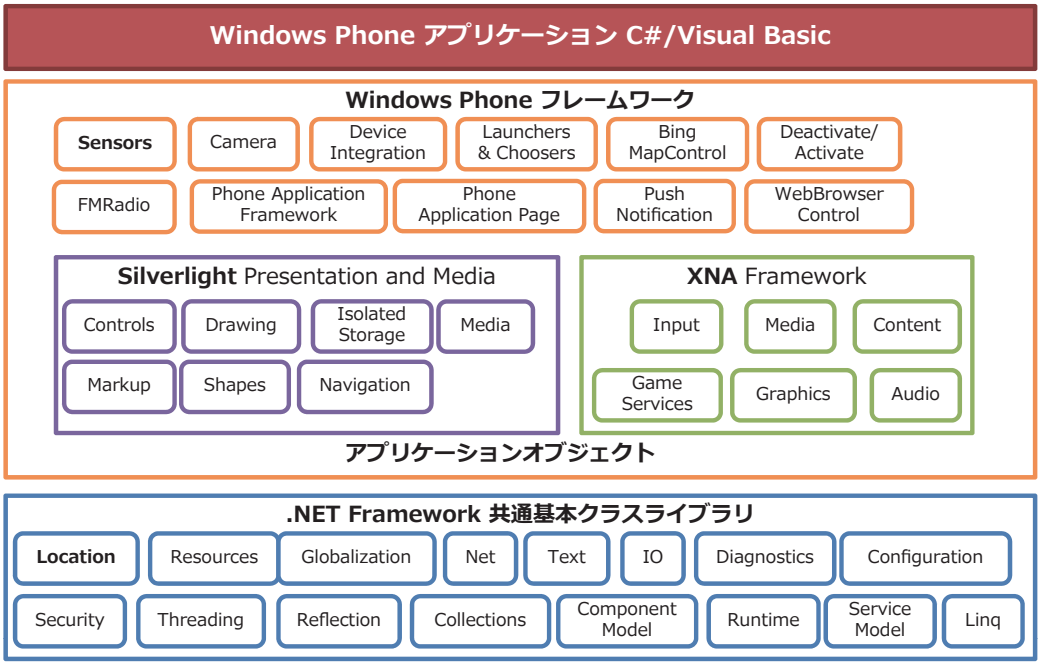
PC 向けの Windows 7 や Android OS とは異なり，Windows Phone OS そのものはスマートフォン端末メーカのみに提供され，OS 単体では一般向けには提供されていません．開発したアプリケーションを，実際の端末で実行するには，通信キャリアから販売されている Windows Phone 端末が必要です．2011 年 10 月現在で，日本では，富士通東芝モバイルコミュニケーションズが開発した IS12T が au から販売されているので，実端末で試したい場合は，この端末を使うのがよいでしょう．全世界ではすでに 35 カ国で 21 機種の Windows Phone 端末が販売されています．

■ 搭載されているセンサ

Windows Phone では，文献 1）により，端末が満たすべきハードウェア仕様が決められています．表 -1 にセンサに関する規定を示します．

センサ	概要	備考
A-GPS	GPS センサ	標準装備
Accelerometer	加速度センサ	標準装備
Compass	方位センサ	オプション装備
Gyro	ジャイロセンサ	オプション装備

表 -1　搭載センサー覧



※太字で示した，“**Sensors**”，“**Location**”は，本稿で取り上げるライブラリを示す

図 -1　Windows Phone アプリケーションアーキテクチャ

センサの中で，標準装備と書いてあるものは，Windows Phone を採用しているすべての端末で利用可能，オプション装備のものは，端末によって装備の有無が異なります．どのセンサを載せるかはスマートフォンメーカ自身が決めることなので，アプリケーションがさまざまな機種で動作するよう，オプション装備のセンサが端末にない場合でも，正しく動くアプリケーションを開発するようにしましょう．

■ Windows Phone のアプリケーション開発

各センサの詳細に入る前に，Windows Phone アプリケーション開発の基本事項を，簡単に紹介しておきます．Windows Phone は，図 -1 に示すようなアーキテクチャをしています．

ユーザ Windows Phone アプリケーションの開発言語は C# か Visual Basic のどちらかを選択可能です．コンパイルされたアプリケーションコードは，Java が JVM で動くのと同様に，.NET Framework のバーチャルマシン上で動作します．

アプリケーション開発用に，Windows Phone フレームワークが提供されています．Windows Phone フレームワークには，センサをはじめとするスマートフォン専用機能向けのクラスライブラリに加え，Silverlight と XNA Framework が含まれています．

Silverlight は，ボタンやテキストボックス，リスト，キャンバスなどユーザインタフェースを構築するための部品（コントロール）を提供するとともに，アプリケーションのライフサイクルやアプリケーションの基本骨格（フレームワーク），ページのナビゲーション機能等を提供します．XNA Framework は，高機能グラフィクスや音楽などを扱うための機能を提供するフレームワークです．Windows Phone 7.1 のアプリケーションは，Silverlight，もしくは，XNA Framework，2 つのフレームワークの組合せをベースに開発を行います．Windows Phone の基本的なアプリケーション動作モデル，Silverlight と XNA Framework といった基本事項は，文献 2）を参照してください．

一般の開発者が手を入れることができるのは Windows Phone フレームワークより上の層だけで，OS を含む Windows Phone フレームワーク以下の層は一切手を入れることはできません．アプリケーションコードは一種のサンドボックス環境で実行されます．このような構成は，一般的なアプリケーション開発において，不正なコードの紛れ込みによるセキュリティの脆弱性や，システムの不安定性の要因を排除することに寄与しています．

センサをはじめとする，アプリケーション開発に必要な諸機能は，カテゴリごとにコンポーネント化されて提供されています．Windows Phone のアプリケーション開発では，必要に応じて適切なコンポーネントを組み込んでアプリケーション開発を行います．

開発環境一式は，マイクロソフトから無償で提供されている Windows Phone SDK 7.1 にすべて含まれています．SDK は，文献 3）から無償で入手可能です．

このページからリンクをたどり，SDK 一式のインストールを行います．SDK をインストールすると，以下のアプリケーション開発用ツールが PC にインストールされます．

- Visual Studio 2010 Express for Windows Phone
  アプリケーションロジック開発環境．主に開発者が使うツール
- Windows Phone Emulator
  PC 上でアプリケーションの実行確認を行うための Emulator
- Expression Blend SDK for Windows Phone
  アプリケーションの見栄え・デザインを作成する環境．主にデザイナが使うツール
- Application Deployment
  テスト環境で，アプリケーションを実端末にインストールするためのツール
- Windows Phone Developer Registration
  テスト向けに実端末を登録するツール

Emulator は PC のバーチャルマシン上で実行されています．Windows Phone OS のバイナリコードがそのまま動いているので，アプリケーションプログラムは，実端末と同じ実行環境での動作チェックが可能です．開発したアプリケーションを，実端末上で実行するには，Windows Phone App Hub での開発者登録（有償）が必要です．登録を済ませれば，実端末上での実行だけでなく，マイクロソフトが運営する Marketplace に開発したアプリケーションを登録し，世界中の Windows Phone ユーザに配布・販売が可能です．開発

クラス名	計測可能なデータ種別	データ形式	備考
Accelerometer	端末にかかる加速度	3 軸 ※重力含む	標準
Gyroscope	端末にかかる角速度	3 軸：ラジアン／秒	オプション
Compass	真の北極からの角度	0 ～ 360 度	オプション
	磁極の北極からの角度	0 ～ 360 度	
	磁極の方向	3 軸	
Motion	端末にかかる加速度	3 軸 ※重力なし	オプション
	端末にかかる角速度	3 軸：ラジアン／秒	
	端末の姿勢	3 軸：ラジアン 3 × 3 の変換行列 クオータニオン	
	重力の方向	3 軸	
GeoCoordinateWatcher	緯度，経度，高さ，誤差	度，メータ	標準

表 -2　センサ関連クラスと計測可能なデータ

者登録からアプリケーション登録までの一連の流れや開発者登録に必要な金額は，文献 3）を参照してください．

■ センサ関連クラスライブラリ

Windows Phone SDK 7.1 で提供されるセンサ向けのクラスライブラリと計測可能な値を表 -2 にまとめます．

表 -2 から分かるように，表 -1 で紹介したセンサデバイスと，センサを扱うクラスは 1 対 1 には対応していません．Accelerometer と Gyroscope，Compass は，それぞれ表 -1 の Accelerometer，Gyro，Compass と対応していますが，Motion クラスは，センサデバイスとして装備される Accelerometer と Gyro，Compass の計測値から論理的に計算された値をアプリケーションに提供するクラスです．表 -2 で挙げているセンサ計測値のうち，3 軸の計測値の座標系を図 -2 に示します．

また，GeoCoordinateWatcher は位置を取得するためのクラスですが，スマートフォンの位置検出の手段としては GPS だけでなく，携帯基地局からの情報取得や Wi-Fi ルータ，Web サービスからの取得などさまざまな方法があるので，GPS を直接連想するようなクラス名ではなく，それらのサービスを統合した上で，位置情報取得サービスとしてのクラスとして位置付けられています．

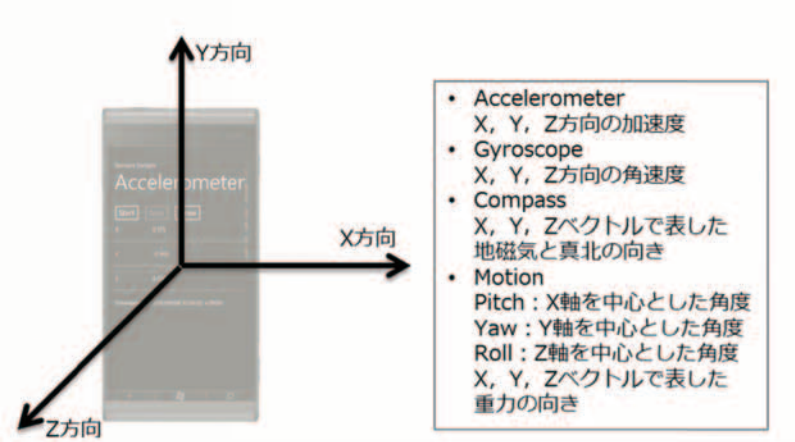


図 -2　センサ計測値の座標

一般的なオブジェクト指向言語がそうであるように，表 -2 に挙げたクラスはすべて，特定の名前空間に属しています．Accelerometer から Motion までの 4 つのクラスは，Microsoft.

メンバ名	種別	内容
IsSupported	スタティックプロパティ	センササポート状況
State	プロパティ	センサデバイスの状態
CurrentValue	プロパティ	計測データの現在値
IsDataValid	プロパティ	計測データの確かさ
TimeBetweenUpdates	プロパティ	CurrentValueChanged イベントを発生する時間間隔
Start	メソッド	計測開始
Stop	メソッド	計測終了
CurrentValueChanged	イベント	センサ計測値を受信するためのイベント

表 -3　センサクラスのメンバ

Devices.Sensors 名前空間に属し，GeoCoordinateWatcher は，System.Device.Location 名前空間に属しています．この 2 つの名前空間に属するクラス群は，それぞれの名前空間と同名のコンポーネントにパッケージされています．

Windows Phone のアプリケーション開発で使えるすべてのクラスライブラリに関する詳細は，文献 4）を参照してください．

センサプログラミングの基本

Accelerometer，Gyroscope，Compass，Motion の基本的な使い方を解説します．

■ センサクラスとクラスのメンバ

センサに関するクラスはすべて，ほぼ同じ形式のメンバで構成されています．それらを表 -3 に示します．

センサを操作するすべてのクラスは，表 -3 に挙げたメンバを持っています．

まず，IsSupported プロパティです．繰り返しになりますが，Accelerometer を除く 3 つのセンサ種別は，すべての機種でサポートされているとは限りません．搭載されている場合は，クラスのプロパティとして用意されている IsSupported の値が true になります．

State プロパティは，センサデバイスの状態を保持するプロパティです．この値が Ready になっているときだけ，計測データを取得可能です．

CurrentValue プロパティはセンサが計測しているデータの現在値です．プログラムのロジックの中でセンサの値を取得してすぐ使う（同期的と呼ぶ）場合にこのプロパティを利用します．このプロパティは，*XXX* の部分を各センサクラスの名前とすると，*XXX*Reading という型名を持っており，それぞれのセンサクラスの計測対象のプロパティを保持します．

IsDataValid プロパティは，CurrentValue の確かさを示しています．たとえばセンサデバイスが初期化中などの理由で正しい値が得られていない場合は，このプロパティは false になります．センサ計測値の信頼性を求められるアプリケーションの場合は，IsDataValid を確認して，正しい値が同期的に取得できているか確認が必要です．

TimeBetweenUpdates は，計測の時間間隔を指定します．Windows Phone では，前述の同期的なセンサアクセス方法のほかに，センサがデータを計測した時点で，センサ側から通知を受ける非同期的な動作モデルも用意されています．アプリケーションの要件に合わせてこのプロパティの値を調整し，計測時間間隔の精度の調整や，センサデバイスの無駄な実行を防ぎます．

■ 計測データの取得

センサの計測は，Start メソッドをコールすることによって開始し，Stop メソッドをコールすることにより終了します．CurrentValue からの値取得も Start メソッドコールから Stop メソッドコールの間でだけ可能です．一般的にセンサデバイスの実行には，より多くの電力を必要とし電池の減りを早めるので，必要最低限な範囲で小まめに Start と Stop をコールしなければなりません．

センサを使い終わったら Dispose メソッドをコールします．C# も Visual Basic もガベージコレクションによるメモリ解放機構は持っていますが，OS が内部的にセンサ処理のために確保したリソースは自動的には解放されません．Dispose メソッドを明示的にコールして解放してやる必要があります．

センサのデータ計測をトリガに，その計測値を受信して何らかの処理を開始するような動作モデルを非同期的といいます．Java の場合，Listener という概念があり，あらかじめハンドラを登録して後でコールしてもらう場合は，あらかじめ決まったインタフェースのクラスを用意してインスタンスを作成し，addEventListener 等のメソッドで登録するという煩わしいコーディングが必要ですが，C# や Visual Basic ではイベントという概念が言語仕様として用意されているので，非同期的にセンサの計測値を取得する場合には，CurrentValueChanged イベントにハンドラメソッドを登録し，センサからの計測値をハンドラメソッドの引数を通じて受信する，というコードを記述するだけですみます．CurrentValueChanged イベントに登録するハンドラメソッドには，SensorReadingEventArgs<*XXX*Reading> という型の引数が用意されています．*XXX*Reading の *XXX* の部分はそれぞれのセンサクラス名におきかわります．このデータ型は同期的にセンサ計測値を取得する場合に利用する CurrentValue と同じ型なので，同期的にセンサ計測値を参照するときと同じコードで非同期の場合も計測データを参照することが可能です．

センサチュートリアル

以上，センサクラスのメンバをざっと説明してきました．ここでは，センサを使ったプログラムの流れを，C# による Silverlight を使った Windows Phone アプリケーション開発で解説します．

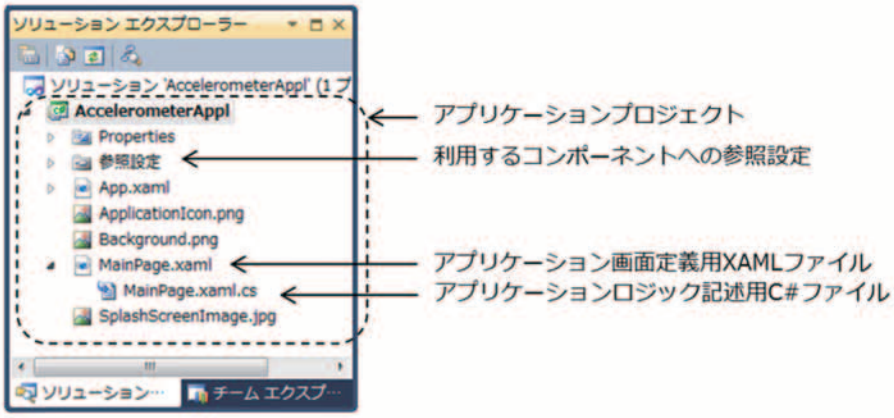


図 -3　Windows Phone アプリケーション開発ファイル一式

■ Accelerometer を使った簡単なセンサプログラミング

Visual Studio を起動し，ファイルメニューで " 新規作成→プロジェクト " を選択して，Visual Studio のアプリケーション開発単位であるプロジェクトを作成します．この操作により，プロジェクトの種別ごとに用意されたプロジェクトテンプレートを選択するダイアログが表示されます．

Visual Studio では，あらかじめ用意されたテンプレートを選択して最小限の雛形を出発点として開発を行います．ここでは，"Visual C#" → "Silverlight for Windows Phone" の "Windows Phone アプリケーション " を選択し，" 新しいプロジェクト " ダイアログの名前の項目に，適当なプロジェクト名を入力して "OK" ボタンをクリックします．対象とする Windows Phone OS のバージョンを選択するダイアログが表示されるので，"Windows Phone OS 7.1" を選択します．

必要なファイル一式と設定を持ったプロジェクトができあがり，アプリケーション開発の起点となるファイル一式が生成されます．自動生成されるファイル一式を図 -3 に示します．

図 -3 の " ソリューションエクスプローラー " は，Visual Studio の開発単位であるソリューションに含まれるファイルやリソースを表示するためのビューワです．

開発環境の準備が整うと，MainPage.xaml というファイルが " デザイナー " ビューワで開かれた状態になります．

Silverlight による開発では，ユーザインタフェースのデザインは，XAML と呼ばれる XML ベースの宣言型プログラミングで行います．XAML や宣言型プログラミングなど概念を言葉だけで説明することは非常に難しいので，図 -4 を参考に，Visual Studio の無償版等を利用して実際に試してみてください．

ツールボックスから Button を 2 つと TextBlock を 1 つ，図 -5 のようにマウスによるドラッグアンドドロップで配置し，位置と大きさを微調整します．すると右側の XAML のテキストの内容が，左側の絵の変更に応じて変わります．

XAML 側（図 -4 右側の破線で囲まれた部分）で，Button の Name と Content を左の Button は "buttonStart" と "Start"，右側の Button は，"buttonEnd" と "End" に変更します．すると絵側（図 -4 左側の破線で囲まれた部分）の表示が変わります．絵と XAML テキストは

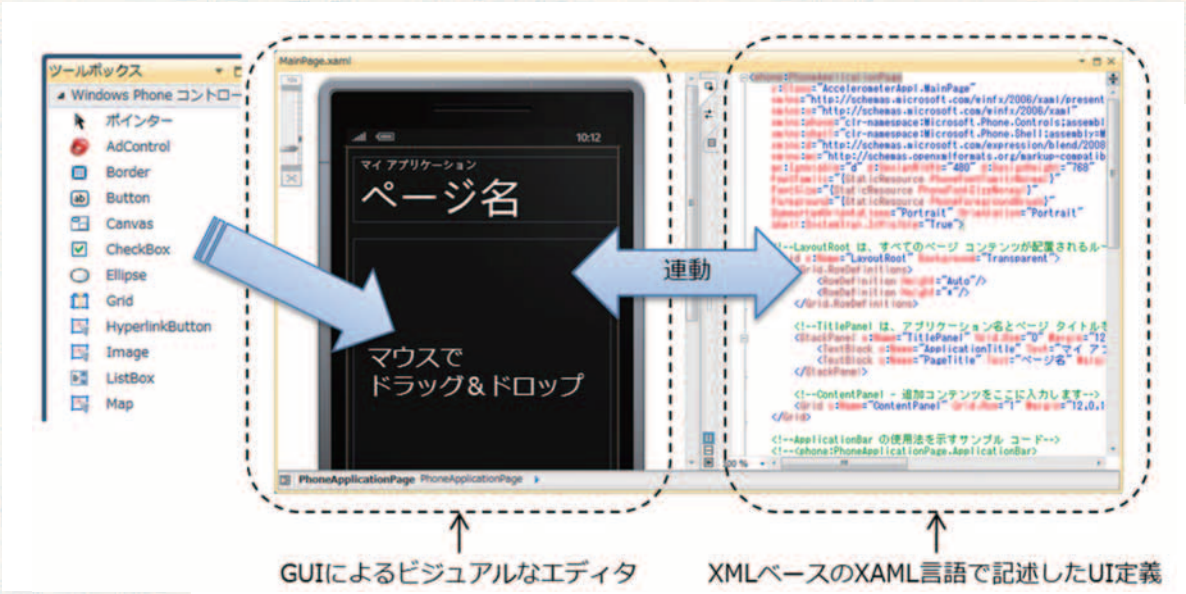


図 -4　MainPage.xaml とエディタ

図 -5　必要な UI コントロールの配置

連動していて，XAML テキストを変更すればそれに従って絵が変わるのが分かるでしょう．

絵の Start Button 上でマウスをダブルクリックすると，MainPage.xaml.cs という名前の C# ソースファイルが開かれ，buttonStart_Click というメソッドにカーソルが移動します．もう一度 MainPage.xaml に戻って，End Button をダブルクリックします．すると今度は buttonStop_Click というメソッドが新たに追加されて，カーソルがそこに移動します．この C# ソースファイルをコードビハインドと呼び，このファイルにアプリケーションの処理ロジックを記述していきます．MainPage.xaml に戻り，Button タグの中身をよく見るとそれぞれに “Click=” という属性が追加されています．これが，Button がクリックされたときに，どのメソッドを実行するかを示す XAML 上での記述です．

MainPage.xaml.cs にできた buttonStart_Click メソッドに，

```
textBlock1.Text = “Start”;
```

図-6　センサプログラミングに必要なコンポーネントの参照設定

buttonStop_Click メソッドに，

　textBlock1.Text = “Stop”;

とそれぞれ書いて，F5 キーを押すと，Emulator が起動され，XAML でデザインされた画面が表示されます．Start ボタンと Stop ボタンをマウスでクリックして，動作を確認してみてください．

　Silverlight のアプリケーションは，UI 上の部品に対する刺激やセンサからの更新などをトリガとして動作する非同期的なプログラミングモデルが基本です．

　次に，アプリケーションでセンサを利用するために Microsoft.Devices.Sensors アセンブリをプロジェクトに追加します．

　図-6 に示すように，ソリューションエクスプローラーでプロジェクトの下の参照設定を右クリックし“参照の追加”を選択します．表示されたダイアログの .NET タブを選択し，Microsoft.Devices.Sensors を選択して OK をクリックします．センサの諸クラスは，測定データパラメタの型として XNA Framework で定義されているデータ型を利用しているので，同じ操作で，Microsoft.Xna.Framework も追加しておきます．

　次に，コードビハインド側の MainPage.xaml.cs のコードを書いていきます．後述の作業がすべて終わった状態のコードは以下の通りになります．リストは，namespace 宣言で囲まれたブロックの中だけを示しています．

```
// Part Ⅰ ：使用する名前空間の宣言
using Microsoft.Devices.Sensors;
using Microsoft.Xna.Framework;
// Part Ⅰ ：終わり

public partial class MainPage : PhoneApplicationPage
{
    private Accelerometer sensor;     // Part Ⅱ ：メンバ変数宣言

    // コンストラクタ
    public MainPage()
    {
        InitializeComponent();

        // Part Ⅲ ：センサインスタンス作成とイベントハンドラ登録
        sensor = new Accelerometer();
        sensor.CurrentValueChanged +=
```

```
            new EventHandler<SensorReadingEventArgs<AccelerometerReading>>(
                sensor_CurrentValueChanged);
        // Part Ⅲ ：終わり
    }

    void sensor_CurrentValueChanged(object sender,
        SensorReadingEventArgs<AccelerometerReading> e)
    {
        // Part Ⅳ ：TextBlock に計測値を表示
        Deployment.Current.Dispatcher.BeginInvoke(() =>
        {
            Vector3 accel = e.SensorReading.Acceleration;
            textBlock1.Text =
                String.Format("X={0:0.000},Y={1:0.000},Z={2:0.000}",
                accel.X, accel.Y, accel.Z);
        });
        // Part Ⅳ ：終わり
    }

    private void buttonStart_Click(object sender, RoutedEventArgs e)
    {
        sensor.Start();
    }

    private void buttonStop_Click(object sender, RoutedEventArgs e)
    {
        sensor.Stop();
    }

    private void PhoneApplicationPage_Unloaded(object sender, RoutedEventArgs e)
    {
        sensor.Dispose();
    }
}
```

まず，MainPage.xaml.cs を開き，クラス名（MainPage）の直前に，使用する名前空間を追加（Part Ⅰ）し，MainPage クラスのメンバ変数として Accelerometer 型の変数を追加（Part Ⅱ）します．

Visual Studio には，コーディング入力の際に，入力候補の提示や定型的なコードを補完してくれる，便利なインテリセンスという機能があります．無理やり自分で一文字一文字入力しようとせず，Visual Studio に適時表示される指示に従い，より少ないキー入力でコーディングを行ってください．

リストの中で太字で示した部分が，手入力が必要な部分です．

コンストラクタ MainPage() でセンサのインスタンスを作成し，sensor 変数にイベントハンドラを登録します（Part Ⅲ）．イベントハンドラ用のメソッドは，Visual Studio の入力機能を使って自動生成するのが楽です．

生成されたハンドラに，XAML で定義した TextBlock をセンサから通知された値で更新するコードを加えます（Part Ⅳ）．

ハンドラの引数には，SensorReading というプロパティがあり，Timestamp という計測された時間と Acceleration という，X 軸，Y 軸，Z 軸方向の端末に働いている加速度の計測値が，それぞれ格納されています．別のセンサの場合には Acceleration の代わりに

図 -7　デバッグターゲットの選択と実行開始

それぞれの計測データを保持する型のデータが入っています．加速度の場合，地球の重力加速度を 1 とする単位系を採用しています．センサから取得した値を使って Silverlight のコントロールを更新する場合には，ハンドラをコールするスレッドは UI スレッドとは異なるため注意が必要です．Windows 系に限らず一般的な UI フレームワークでは，UI スレッドとは別のスレッドからの UI 上のコントロールへのアクセスは禁じられており，そのまま実行すると例外が発生します．ここでは，TextBlock のプロパティ更新処理を，Deployment.Current.Dispatcher.BeginInvoke(() => {…}); で囲むことにより，UI スレッドに処理を委譲しています．

後は，次に buttonStart_Click メソッドにセンサの計測開始用コードを記述し，buttonStop_Click メソッドにセンサの計測終了用のコードを記述します．

最後に，センサの後始末のコードを追加します．まず，MainPage.xaml を開き，XAML テキスト側で，1 行目にカーソルを移動します．プロパティビューでイベントをクリックし，Unloaded の右側の空欄をクリックします．自動生成された PhoneApplicationPage_Unloaded メソッドには Dispose() メソッドコールを記述します．こうすることにより，ページが閉じるとき確実にセンサが Dispose されることになります．

これで準備は完了です．**図 -7** のように，実行ターゲットのコンボボックスから Windows Phone Emulator を選択し，F5 キーを押下してデバッグを開始します．

Windows Phone Emulator が起動し，アプリケーションが Emulator に配置され，実行を開始します．

Start ボタンをマウスでクリックするとセンサ計測が開始します．Phone Emulator の右横のメニューバーの “>>” をクリックすると，**図 -8** のような加速度をシミュレートするツールが表示されます．赤い点をマウスでグリグリやると，端末の移動に従って Accelerometer に加速度データが供給され，実端末がなくてもロジックの検証が行えます．

以上，Accelerometer の使い方を解説しました．ほかの Gyroscope，Compass，Motion も CurrentValue，および，e.SensorReading のデータ型と保持しているプロパティの種類が変わるだけで，プログラムの流れはまったく同じです．

ただし残念ながら Emulator は，Gyroscope, Compass, Motion はサポートしていません．

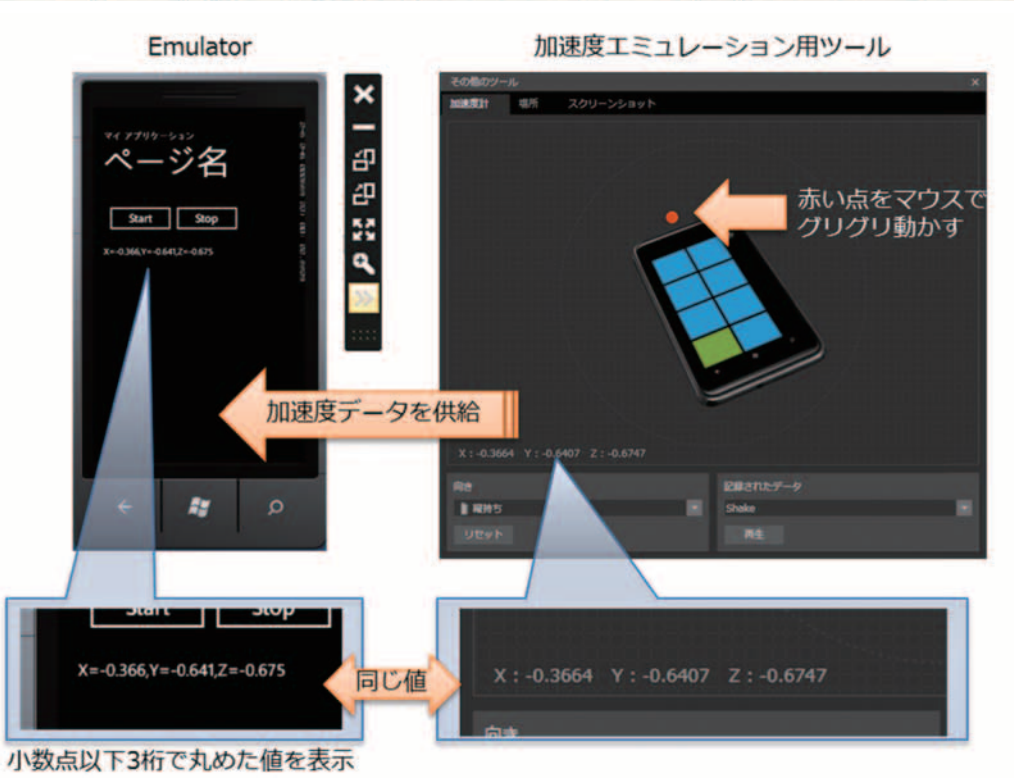


図 -8　Windows Phone Emulator 環境

これらのセンサを使う場合には実端末上での動作検証が必要です．

ほかのセンサの使い方を示すサンプルを文献 3），6）から公開しているので，そちらを参照してください．そこで紹介されているアプリケーションは Sensor Checker という名前で Marketplace にも登録されているので，開発者登録していない一般ユーザの端末でもインストールして実行が可能です．

■ 位置情報とコンパスの例

次に，GeoCoordinateWatcher と Compass を組み合わせたプログラムを解説します．Accelerometer の例で紹介した通りセンサプログラミング自体は非常にシンプルで簡単です．ここでは，Windows Phone のネットワーク上のサービス連携機能の強みとセンサの組合せのサンプルを解説します．このサンプルでは，マイクロソフトが提供する地図サービス BingMap と GeoCoordinateWatcher，Compass を組み合わせて使います．

まず，Visual Studio で Silverlight for Windows Phone の Windows Phone アプリケーションテンプレートを使ってプロジェクトを作成し，参照設定に以下の 4 つのコンポーネントを追加します．

- Microsoft.Devices.Sensors
- Microsoft.Xna.Framework
- System.Device　　※ GeoCoordinateWatcher を含むコンポーネント

- Microsoft.Phone.Controls.Maps　※地図コントロールを含むコンポーネント

MainPage.xaml の頭のほうの xmlns:…という一連のコードの最後に，xmlns:map の定義を加えます．Xmlns:map= までキーボード入力すると，候補リストが表示されるので，Microsoft.Phone.Controls.Maps を含む項目を選択すれば入力は完了です．

```
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc=http://schemas.openxmlformats.org/markup-compatibility/2006

xmlns:map="clr-
namespace:Microsoft.Phone.Controls.Maps;assembly=Microsoft.Phone.Controls.Maps"

mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
```

リスト上は xmlns:map が 2 行にわたって書いてありますが，実際には 1 行で書いてください．

そして “ContentPanel - 追加コンテンツをここに入力します ” というコメントの直下の Grid タグに，地図を書くためのキャンバス（タグ名は Canvas）をおき，その子要素に map:Map タグを追加します．Compass から取得したデータを元にこの地図を回転するための仕掛け（RotateTransform タグ）をコーディングします．

```
<!--ContentPanel - 追加コンテンツをここに入力します-->
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <Canvas>
        <map:Map x:Name="myMap" ZoomLevel="15"  Width="400" Height="400"
            Canvas.Left="28" Canvas.Top="50"
            CredentialsProvider="CredentialKey">
            <map:Map.RenderTransform>
                <RotateTransform x:Name="mapRotation" CenterX="200" CenterY="200"
                    Angle="0"/>
            </map:Map.RenderTransform>
        </map:Map>
    </Canvas>
</Grid>
```

BingMap サービスの利用は登録制になっており，リストの中で *CredentialKey* と記されている部分は，**http://www.bingmapsportal.com/** のサイトで開発者登録し，アプリケーションを登録したときに発行されるキーで置き換えます．開発者登録，アプリケーション登録は無償です．

次に，MainPage.xaml.cs に対して制御ロジックを追加します．一通りコーディングが終わった状態をリストに示します．

```
using System.Device.Location;
using Microsoft.Devices.Sensors;
using Microsoft.Xna.Framework;
using Microsoft.Phone.Controls.Maps;

public partial class MainPage : PhoneApplicationPage
{
    private Compass compass;
    private GeoCoordinateWatcher gcWatcher;

    // コンストラクタ
    public MainPage()
    {
        InitializeComponent();

        if (Compass.IsSupported)
        {
            compass = new Compass();
            compass.CurrentValueChanged +=
                new EventHandler<SensorReadingEventArgs<CompassReading>>(
                    compass_CurrentValueChanged);
            compass.Start();
        }
        gcWatcher = new GeoCoordinateWatcher();
        gcWatcher.PositionChanged +=
            new EventHandler<GeoPositionChangedEventArgs<GeoCoordinate>>(
                gcWatcher_PositionChanged);
        gcWatcher.Start();
    }

    // Part Ⅰ：地図位置の更新
    void gcWatcher_PositionChanged(object sender,
        GeoPositionChangedEventArgs<GeoCoordinate> e)
    {
        Deployment.Current.Dispatcher.BeginInvoke(() =>
        {
            myMap.Center = e.Position.Location;
        });
    }
    // Part Ⅰ：終わり

    // Part Ⅱ：方位の更新
    void compass_CurrentValueChanged(
        object sender, SensorReadingEventArgs<CompassReading> e)
    {
        Deployment.Current.Dispatcher.BeginInvoke(() =>
        {
            mapRotation.Angle = 360 - e.SensorReading.TrueHeading;
        });
    }
    // Part Ⅱ：終わり

    private void PhoneApplicationPage_Unloaded(object sender, RoutedEventArgs e)
    {
        if (compass != null)
        {
            compass.Stop();
            compass.Dispose();
        }
        gcWatcher.Stop();
        gcWatcher.Dispose();
    }
}
```

図 -9　Windows Phone Emulator 環境

上のコードは，Accelerometer の場合とまったく同じ要領でコードを書いていきます．

コード中の myMap と mapRotation は，MainPage.xaml ファイルに書かれた，それぞれ，map:Map タグと RotationTransform タグの x:Name に記されている文字列です．XAML 側で名前付けした要素は，コードビハインドの C# 側ではクラスのメンバ変数としてアクセスできます．

コードを見て大体察しはつくと思いますが，GeoCoordinateWatcher が送ってきた現在位置の緯度，経度情報を含む Position.Location を myMap の中心を保持するプロパティ Center に代入することにより，表示された地図の中心を現在位置に変えています（Part Ⅰ）．

それから，Compass から送られてきた真の北極との角度（SensorReading.TrueHeading）で mapRotation の Angle プロパティを更新することにより，端末が向いている方向に合わせて地図が回転します（Part Ⅱ）．

これらのコードでは，センサから受信したイベント引数のプロパティをそのまま Map や RotationTransform のプロパティに代入できているところが実は大きなポイントです．この 2 つのセンサだけに限らず，ほかのセンサも Silverlight のコントロールや XNA Framework の 3D 描画オブジェクトのプロパティに，センサデータをそのまま型変換せずに代入できるような一貫したクラス設計がなされているのも Windows Phone SDK 7.1 の特徴です．

Unloaded イベントのハンドラを追加して，センサの後始末コードを追加してできあがりです．

F5 キーで実行を開始すると，図 -9 のように BingMap による地図が Emulator 上で表示されます．Windows Phone SDK 7.1 では，加速度と同様現在位置のエミュレーションが可能です．加速度を供給するパネルの“場所”タブを選択し，地図上の位置を更新すると，

今作成したアプリケーションの地図の中心がそれにつれて変わります．

Map コントロールは，**http://www.bing.com** で公開されているすべての UI 操作の利用が可能なので実端末で実行するとタッチ UI で地図の拡大や縮小が可能です．

残念ながら Emulator では Compass はサポートされていないので，方位による地図の回転のテストはできません．実端末で実行するには，USB で PC に接続します．PC と Windows Phone 端末の接続を行うための Zune アプリケーションが PC 上で起動し，実端末と PC の接続が完了します．

Visual Studio の実行ターゲットを “Windows Phone Device” に変えて，F5 キーでデバッグを開始すると，今度は実端末上にアプリケーションが配置され，アプリケーションが起動されます．**図 -10** に示すように，Compass の計測値に応じて地図が回転します．

もちろん，Emulator の場合と同じく，ブレークポイントによる実行の停止やステップ実行などによるロジック確認が可能です．

実際にやってみると，Compass の更新頻度が高く地図が見づらいので，Compass の計測頻度を，下のリストのように太字の行を加えて変更します．このコードでは計測の間隔を 2 秒に設定しています．ちなみに IS02T の場合，デフォルトの更新頻度は 25ms です．

```
if (Compass.IsSupported)
{
    compass = new Compass();
    compass.CurrentValueChanged +=
        new EventHandler<SensorReadingEventArgs<CompassReading>>(
            compass_CurrentValueChanged);

    compass.TimeBetweenUpdates = TimeSpan.FromSeconds(2);

    compass.Start();
}
```

これで試すと，2 秒ごとに地図の回転が更新され，だいぶ見やすくなってはいるのですが，方向が変わるとカクカクと動いて見栄えがよくはありません．見栄えを良くするために角度の動きを補間するアニメーションを追加します．

Compass_CurrentValueChanged メソッドの BeginInvoke ブロックの中を以下のコードで置き換えます．

図 -10　実端末での実行

```
Deployment.Current.Dispatcher.BeginInvoke(() =>
{
    Duration duration = new Duration(TimeSpan.FromSeconds(1));

    DoubleAnimation animation = new DoubleAnimation();
    animation.Duration = duration;
    animation.To = e.SensorReading.TrueHeading;

    Storyboard sb = new Storyboard();
    sb.Children.Add(animation);
    sb.Duration = duration;
    Storyboard.SetTarget(animation, mapRotation);
    Storyboard.SetTargetProperty(
        animation,
        new PropertyPath(RotateTransform.AngleProperty));

    sb.Begin();
});
```

紙面の関係でここではアニメーションの詳細は説明しませんが，DoubleAnimation と StoryBoard の 2 つのクラスを使って簡単にアニメーションを入れ込むことができます．

実際に実端末で動かせば，1 秒間で前の角度から新しい角度にスムーズに移動する表示ができるようになります．

以上，Windows Phone のセンサプログラミングを解説しました．Emulator をはじめとする開発環境，一貫した API により，センサを扱うプログラムの作成は非常に簡単です．

しかし，単にセンサの計測値を取得して表示するアプリケーションには何の価値もありません．アプリケーションの価値からすれば，計測データをどう意味付け，活用して魅力的なアプリケーションを開発するか，という点に注力することが非常に重要です．筆者のブログ（文献 6））や文献 5）でも，センサプログラミングの解説だけでなく，センサを活用するアイディアや，Windows Phone に関する技術情報も公開しているので，そちらも参考にしてください．

参考文献

1）Windows Phone 7.1 Hardware Specification，http://msdn.microsoft.com/en-us/library/ff637514(v=VS.92).aspx
2）Fundamental Concepts for Windows Phone，http://msdn.microsoft.com/en-us/library/ff967549(v=VS.92).aspx
3）Windows Phone デベロッパーサイト，http://msdn.microsoft.com/ja-jp/windowsphone/
4）Windows Phone Class Library Reference，http://msdn.microsoft.com/en-us/library/ff626516(v=VS.92).aspx
5）マイクロソフト UX チーム UX-TV サイト，http://msdn.microsoft.com/ja-jp/hh162048
6）筆者のブログ，http://blogs.msdn.com/hirosho/

（2011 年 10 月 17 日受付）

太田　寛　hirosho@microsoft.com
マイクロソフトのエバンジェリスト．開発者向けに新規技術・製品の普及啓発に携わる．デバイス間，およびデバイスとクラウド連携，小型組込み機器向けプラットフォームやセンサテクノロジーの活用などで，講演，記事多数．