

Kinect Camp with TMCN

Kinect入門ハンズオン

Microsoft MVP Windows Platform Development

Oracle ACE for RDBMS

TMCNテクニカルエヴァンジェリスト

初音玲

Twitter : @hatsune_

Blog : <http://hatsune.hatenablog.jp/>

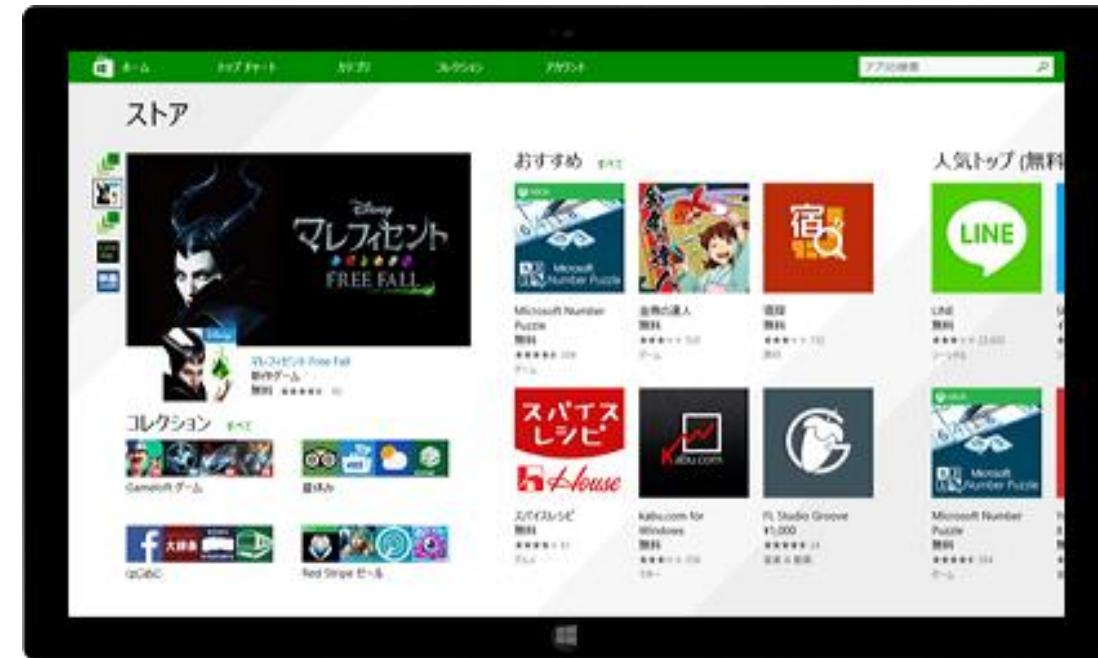
ハンズオンの概要

1. アプリ作成の基礎を学びます
2. Kinect対応アプリを作ります
3. Kinect Studioを利用してデバックを楽にします
4. じゃんけんWindowsストアアプリを作ります

1. アプリ作成の基礎

Windows ストア アプリケーション

- Windows デバイス上で動作
- Windows ストアで配布
- ビューとロジック分離開発
- Windows 10でも同様



詳しい内容は下記 Web ページをご覧ください

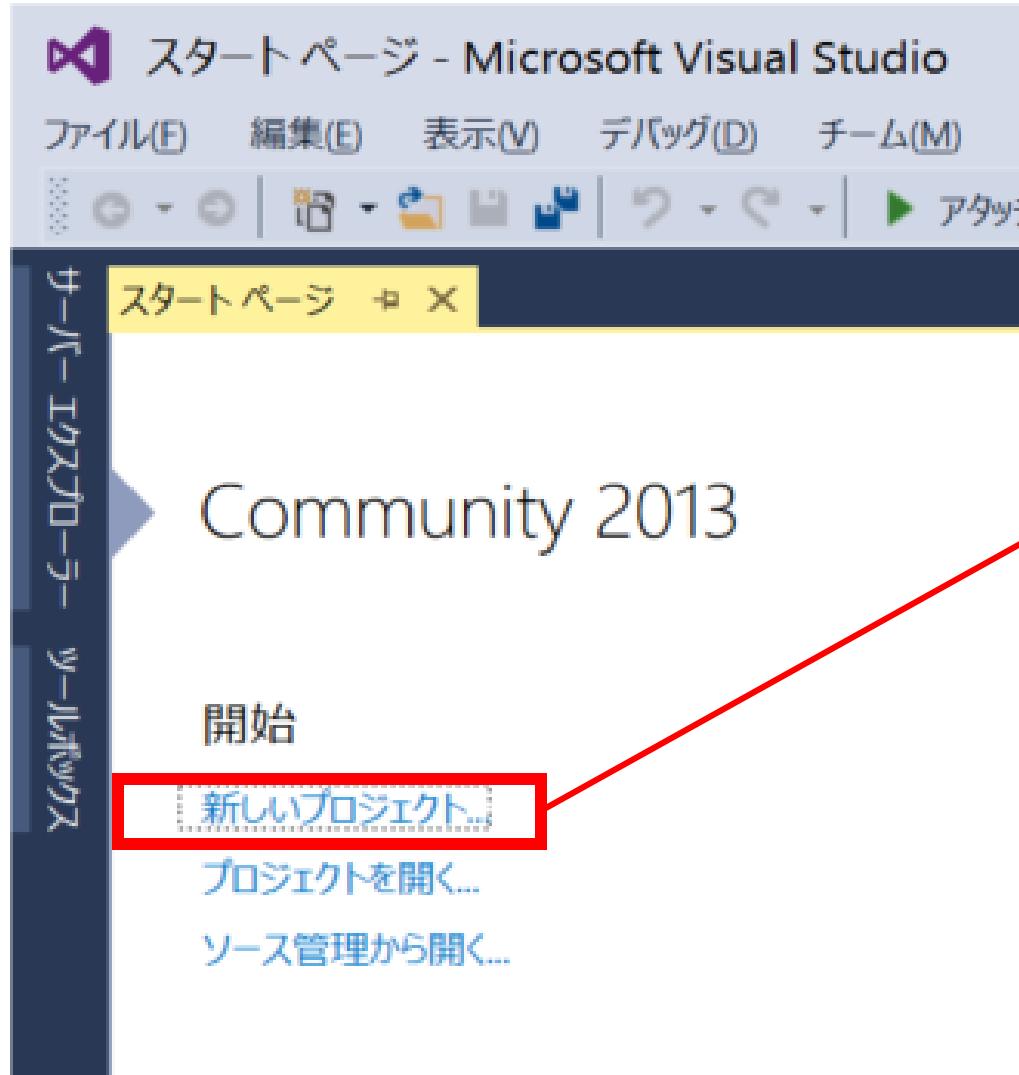
<http://msdn.microsoft.com/ja-jp/library/windows/apps/dn726767.aspx>

チェック

- Microsoft Windows 8.1 (x64)
- Visual Studio Community 2013
- Kinect for Windows SDK v2.0

<http://www.microsoft.com/en-us/kinectforwindows/>
⇒ [download]

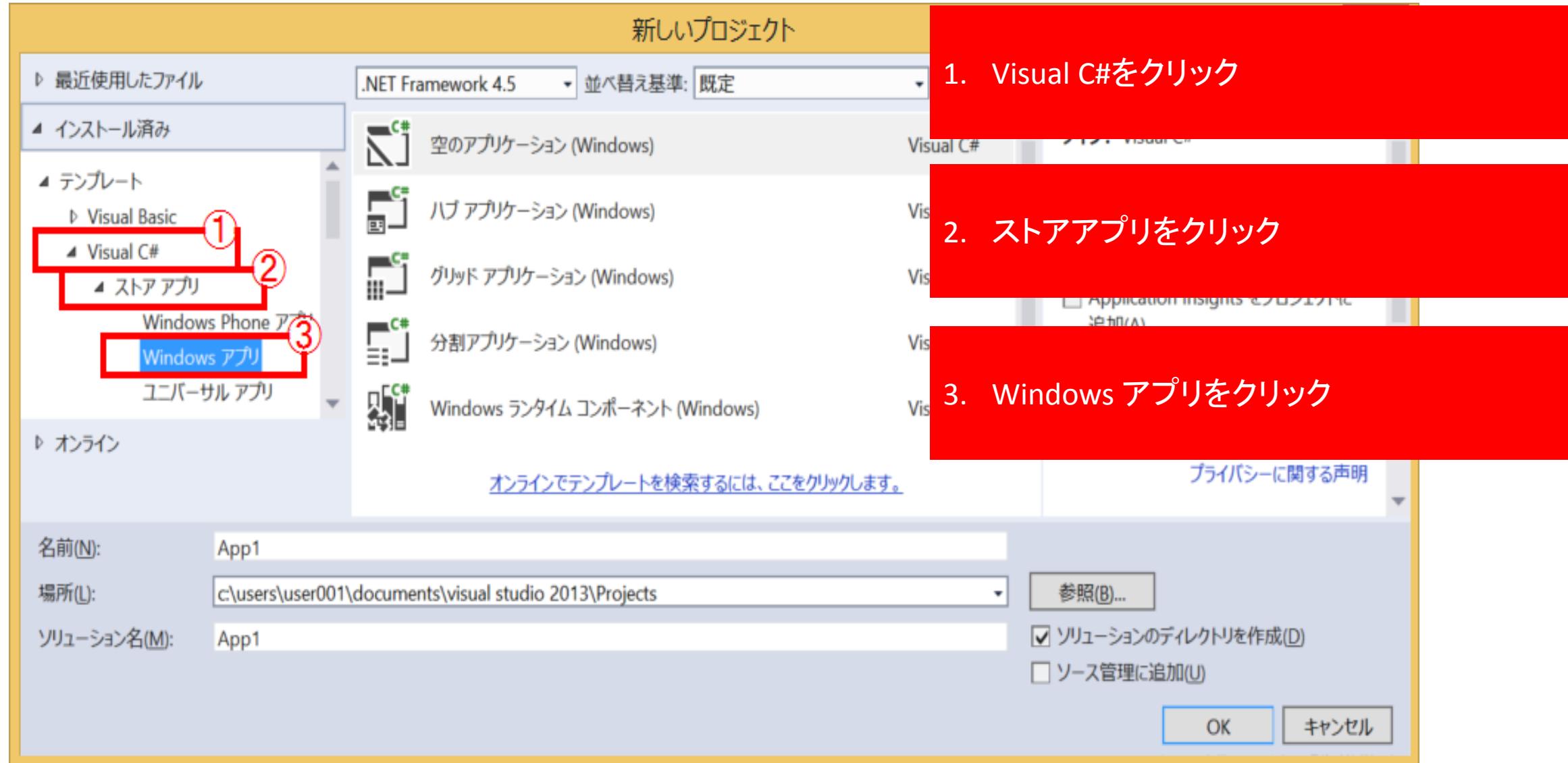
新しいプロジェクト開始



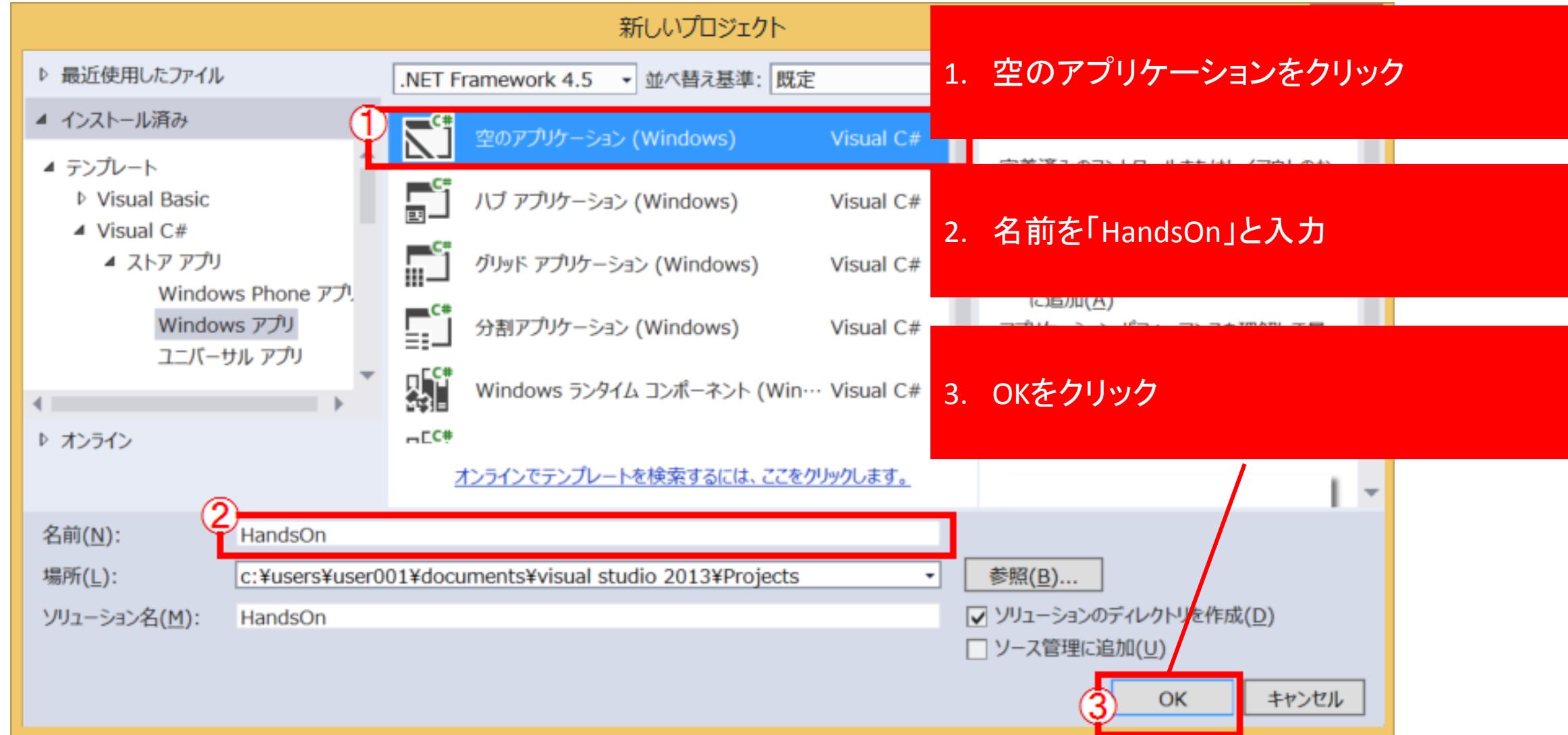
1. Visual Studioを立ち上げます。

2. 新しいプロジェクトをクリックします。

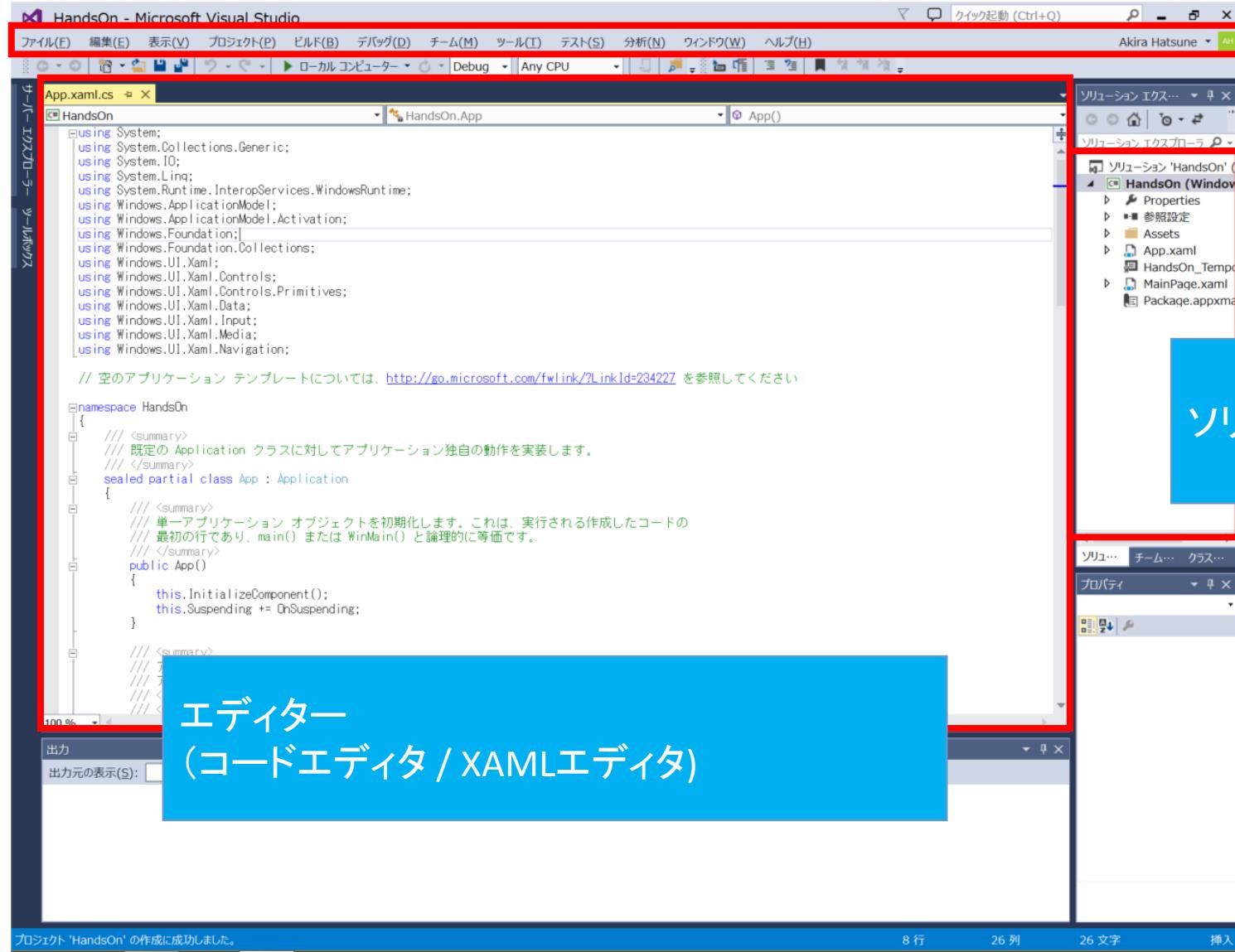
[新しいプロジェクト]ダイアログ



テンプレートを選択してプロジェクトを作成



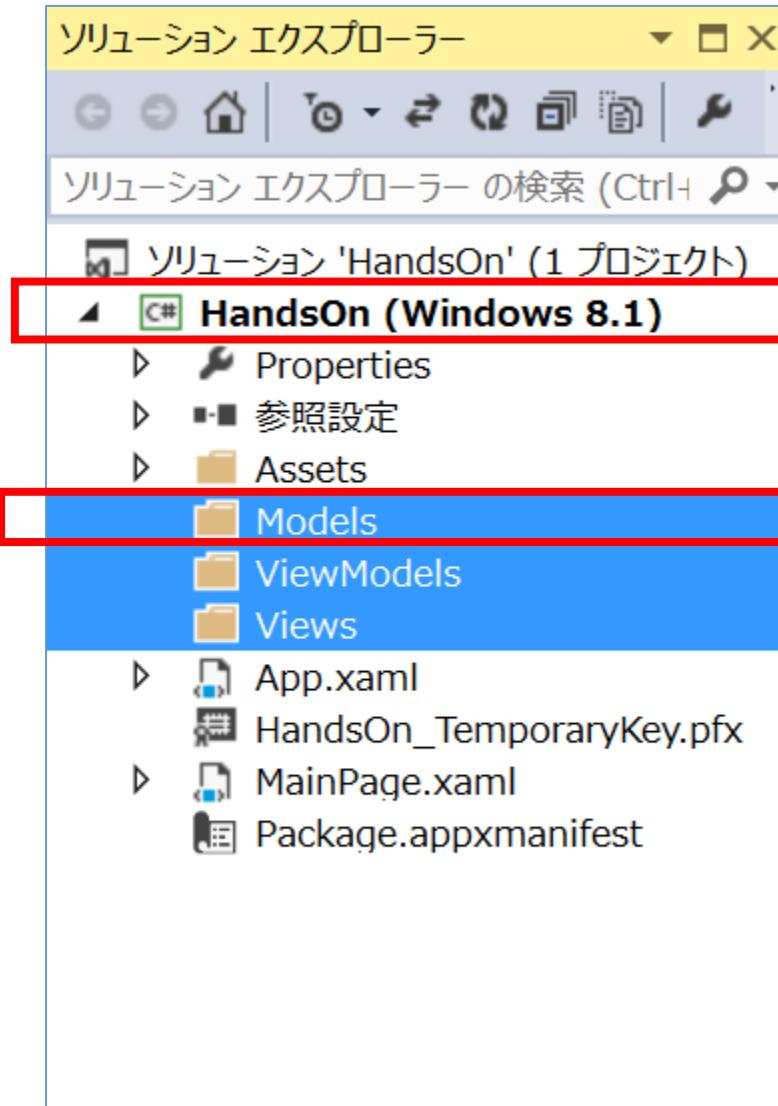
主な名称



メニューバー

ソリューションエクスプローラー

3つのフォルダを作成します



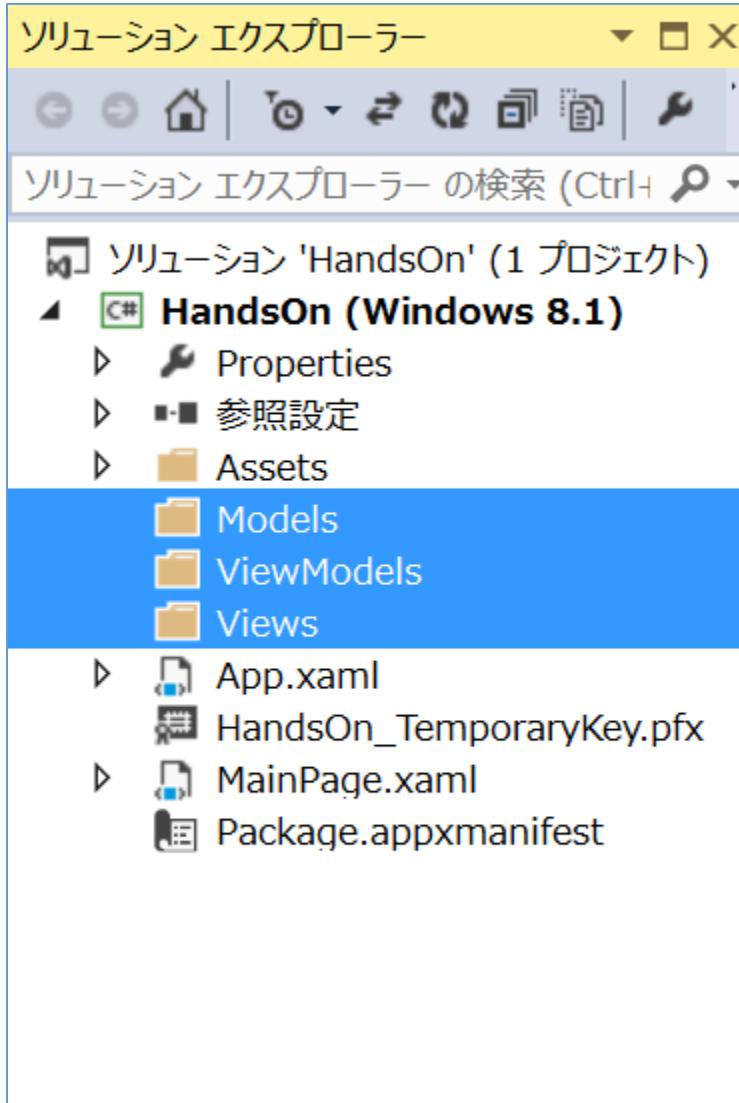
1. プロジェクト名を右クリック

2. [追加]→[新しいフォルダ]とクリック

3. 「Models」と名前を入力

同じように「ViewModels」「Views」を作成

3つのフォルダを作成



■ Models

- Kinectと接続するコードを記述

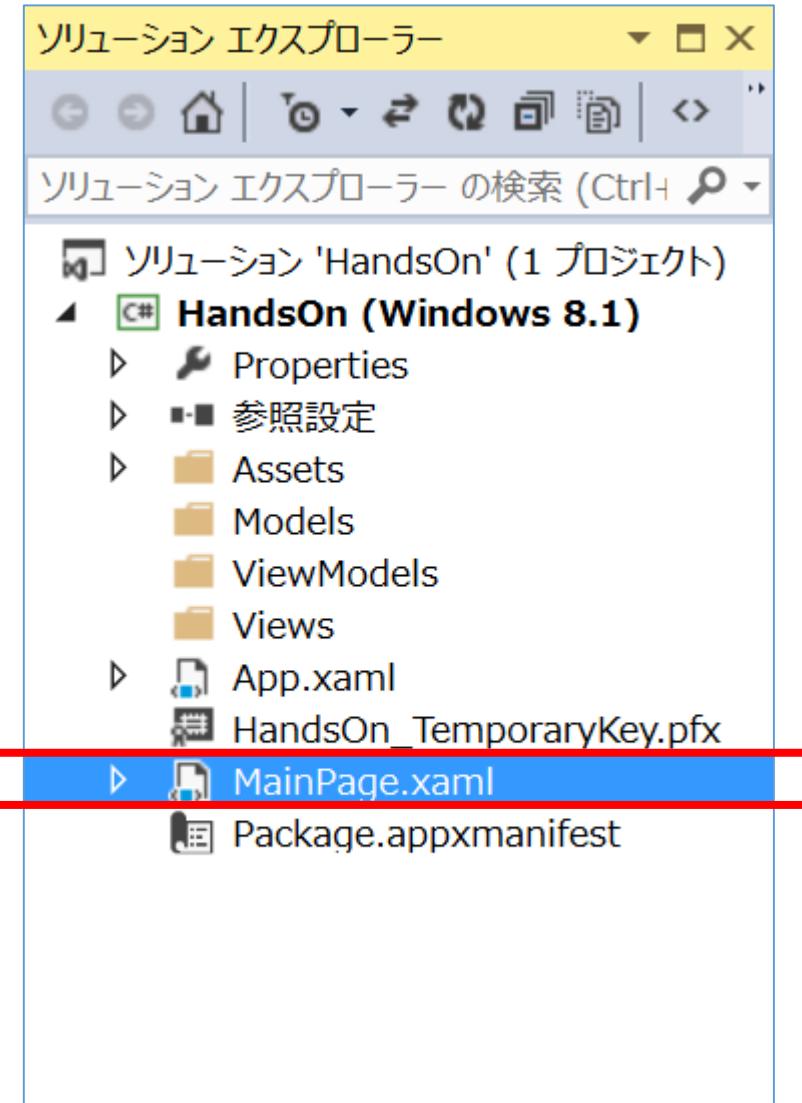
■ ViewModels

- 複数のModelをまとめること

■ Views

- 画面定義などを記述

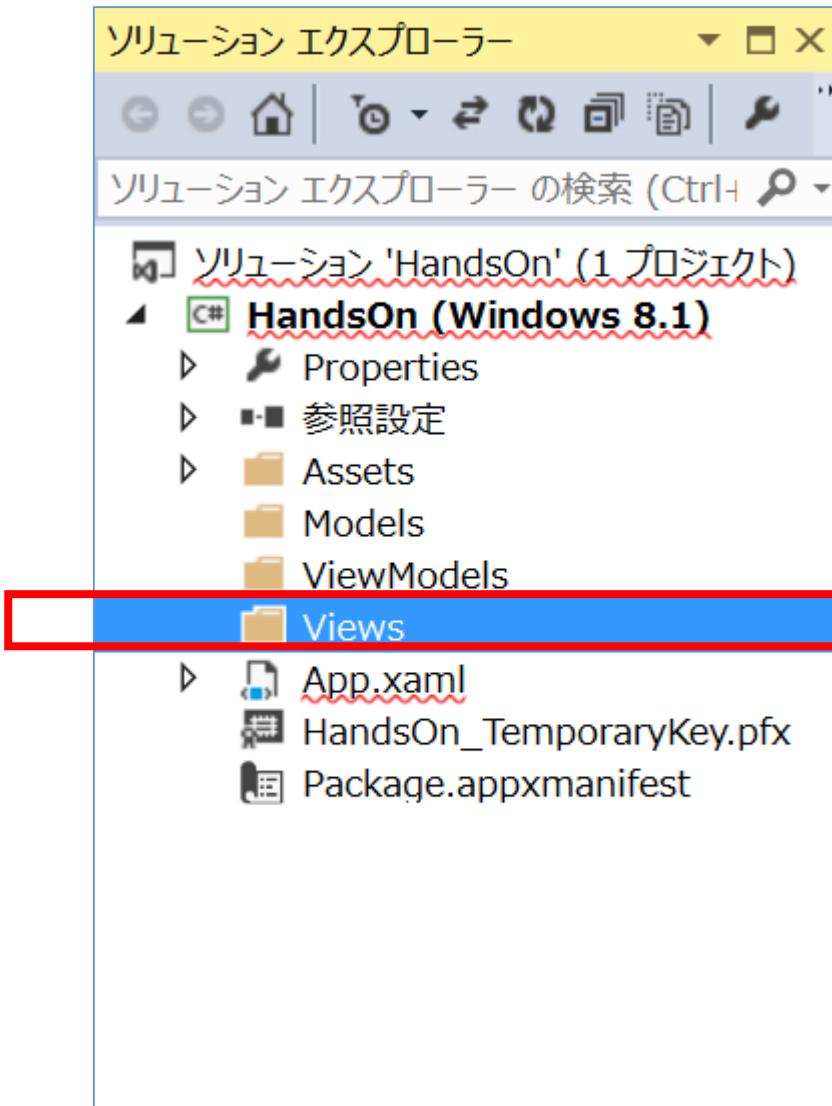
不要なファイルの削除



1. 「MainPage.xaml」を右クリック

2. [削除]をクリック

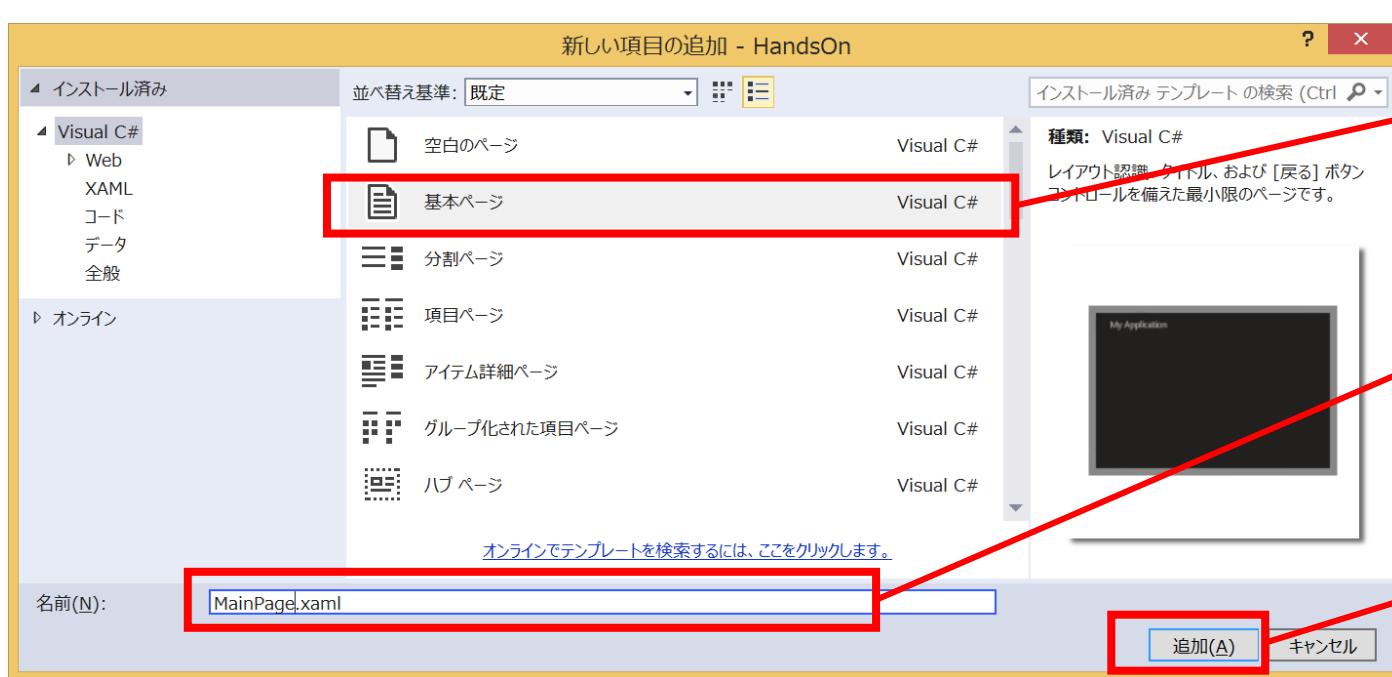
Viewsフォルダに MainPage.xaml を作成



1. 「Views」を右クリック

2. [追加]→[新しい項目]をクリック

Views フォルダに MainPage.xaml を作成

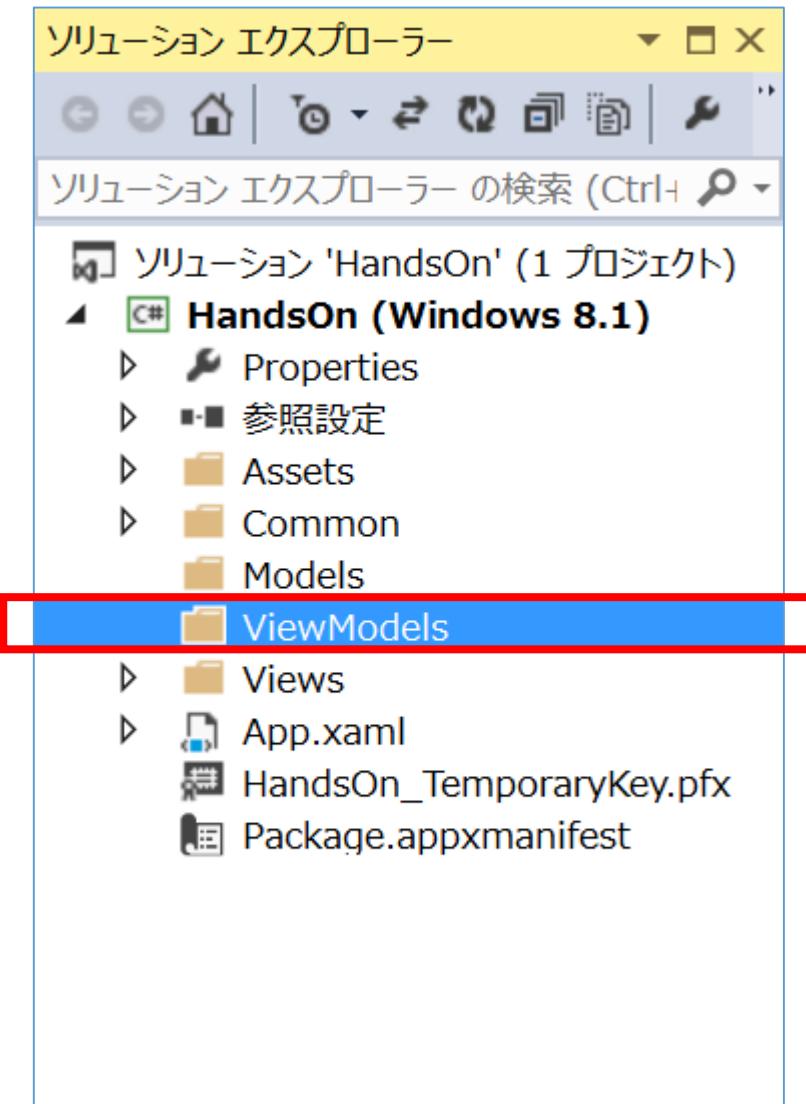


1. 「基本ページ」をクリック

2. 名前を「MainPage.xaml」と入力

3. [追加]をクリック

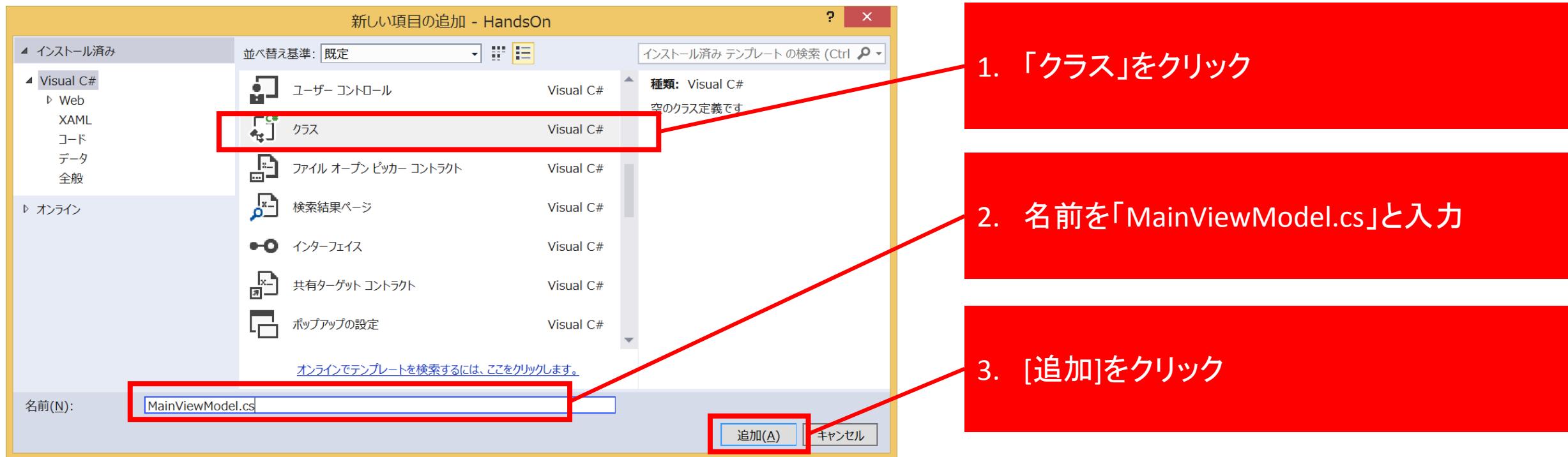
ViewModels フォルダに MainViewModel.cs を作成



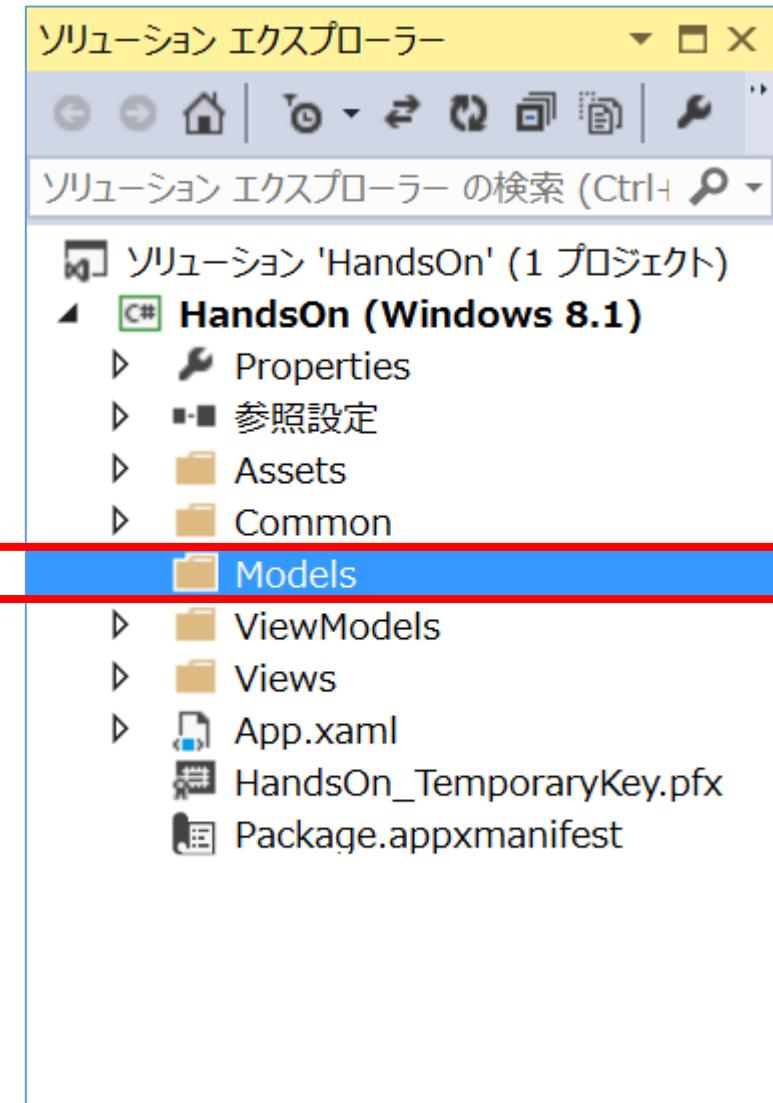
1. 「ViewModels」を右クリック

2. [追加]→[クラス]をクリック

ViewModels フォルダに MainViewModel.cs を作成



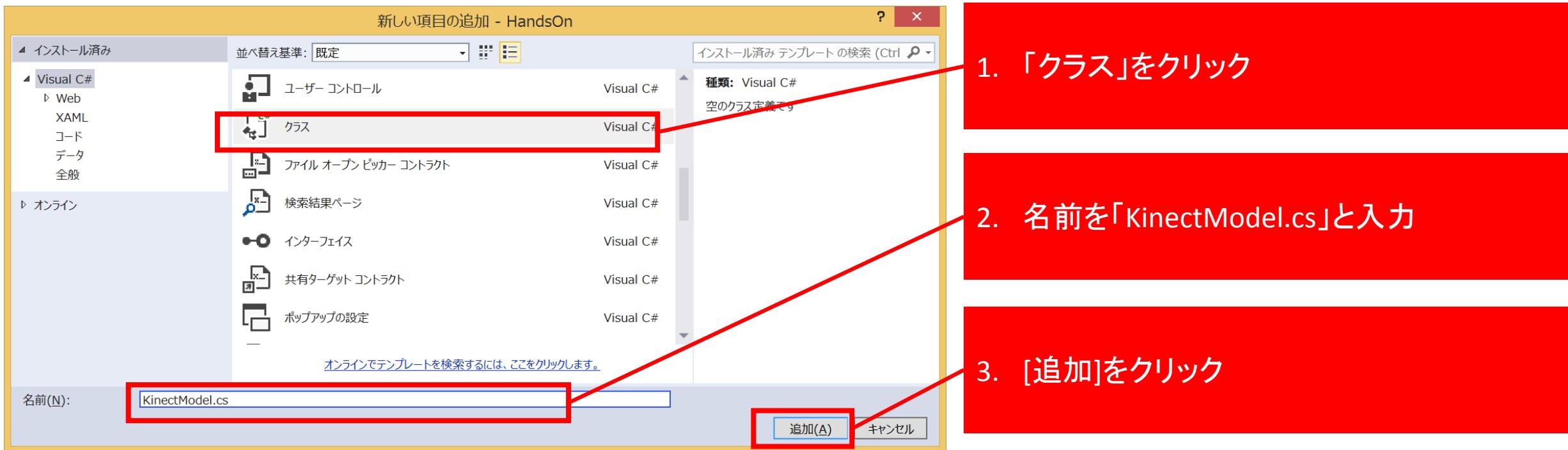
Models フォルダに KinectModel.cs を作成



1. 「Models」を右クリック

2. [追加]→[クラス]をクリック

ModelsフォルダにKinectModel.csを作成



ここでビルト

1. [F5]キーを押す

コンパイルエラーの対応

1. エラー表示をダブルクリック



コンパイルエラーの対応

The screenshot shows the Visual Studio IDE with the project 'HandsOn' open. The code editor displays the file `App.xaml.cs`. A tooltip is visible at the bottom of the screen, listing options for completing the current identifier. The option `' MainPage' のクラスを生成する(C)` is highlighted with a yellow background.

Red arrows and numbered steps are overlaid on the image to guide the user through the process:

1. 青い波線の左端にマウスを移動
2. [▼]をクリック
3. [HandsOn.Views.MainPage]をクリック

The code in the editor is as follows:

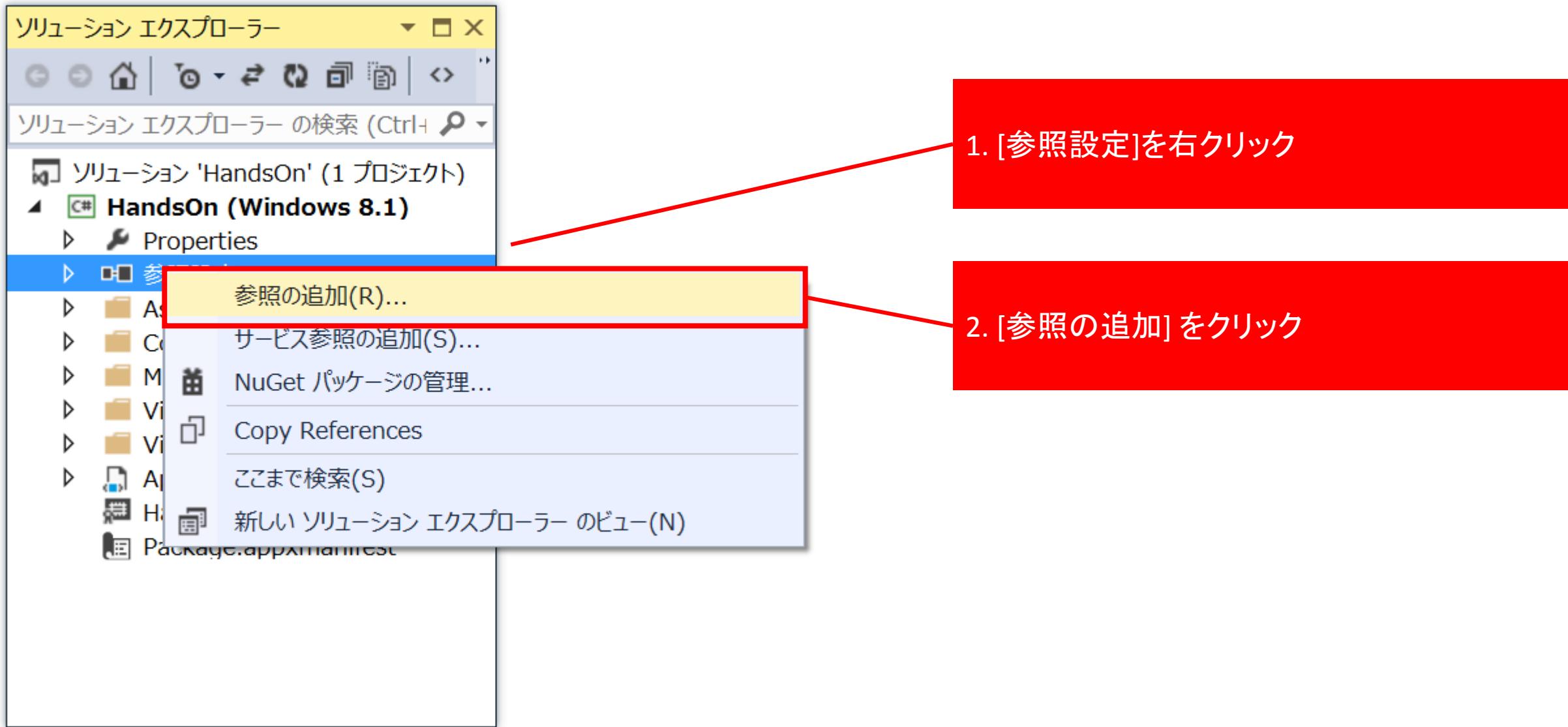
```
74     if (rootFrame.Content == null)
75     {
76         // ナビゲーションの履歴スタックが復元されているか
77         // このとき、必要な情報をナビゲーション パラメータ
78         // 作成します
79         rootFrame.Navigate(typeof(MainPage), e.Arguments);
80     }
81     // 現在のルートフレームがアクティブであることを確認してから
82     // using HandsOn.Views;
83
84     HandsOn.Views.MainPage
```

ここでビルト

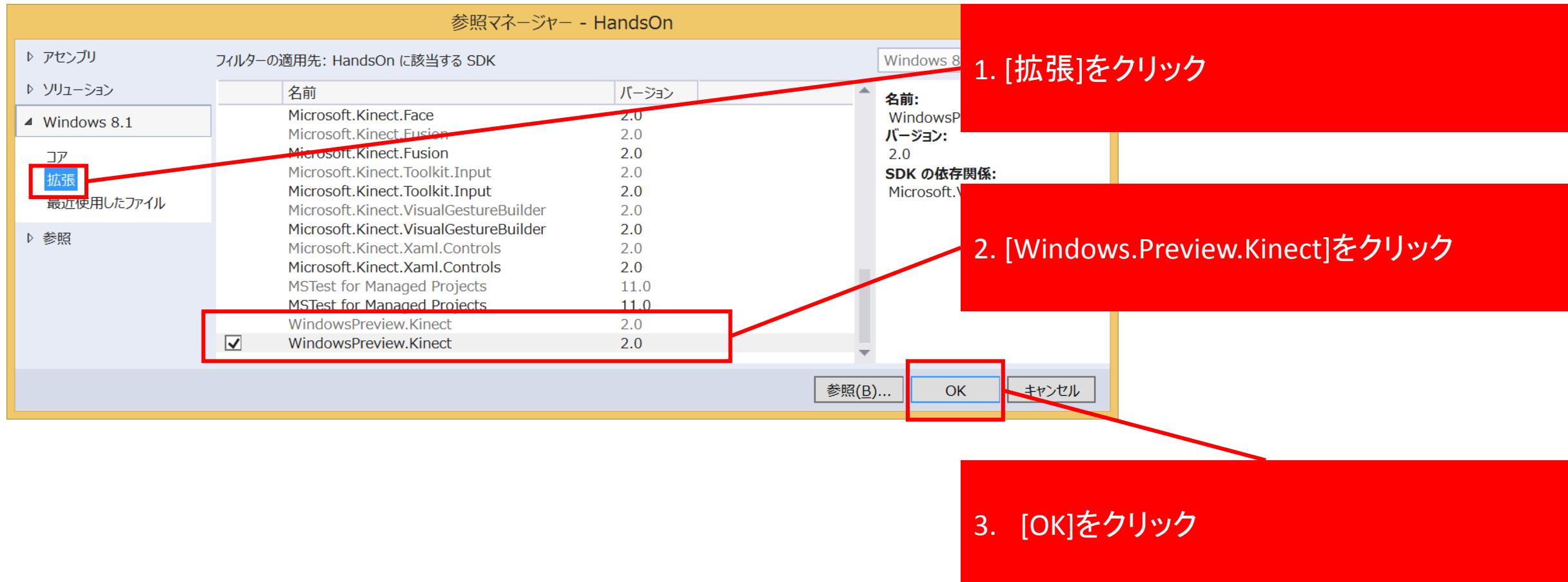
1. [F5]キーを押す

2. 簡単なKinect対応アプリを作成

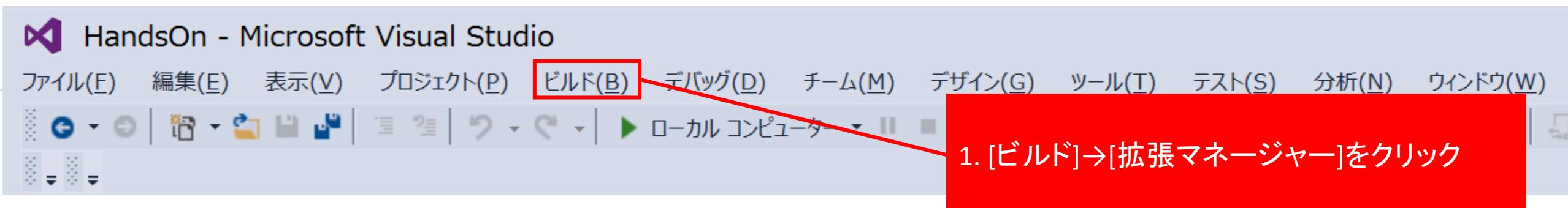
Kinect 関連のモジュールを追加します



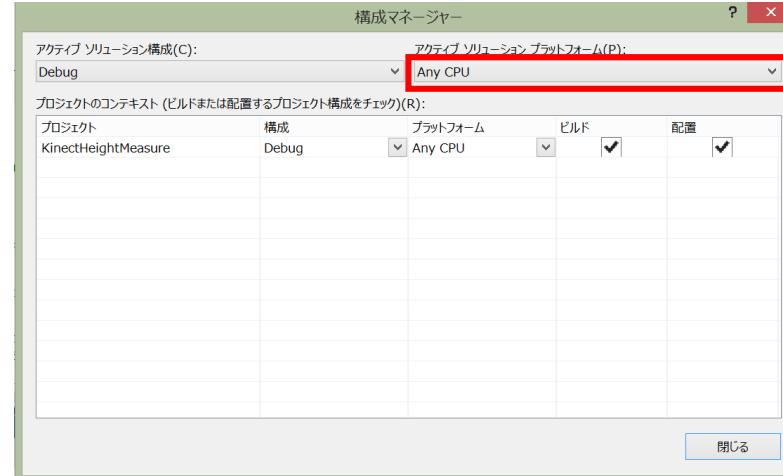
Kinect 関連のモジュールを追加します



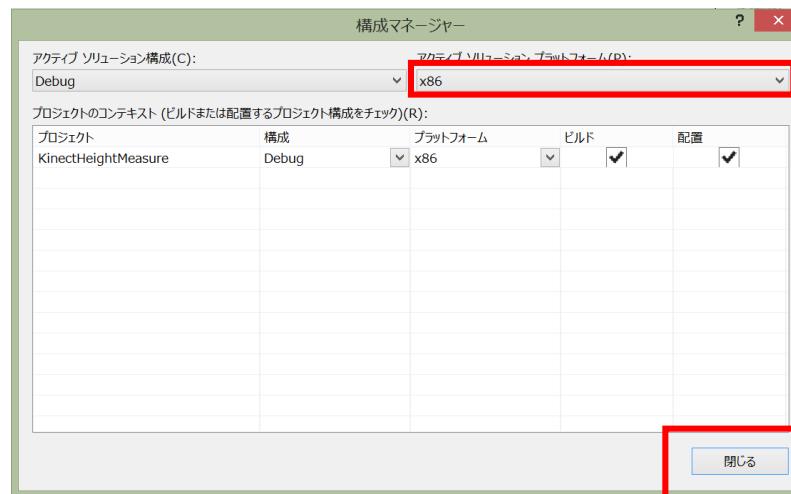
ビルドの構成を変更します



ビルドの構成を変更します



1. 左のような画面が表示されます。

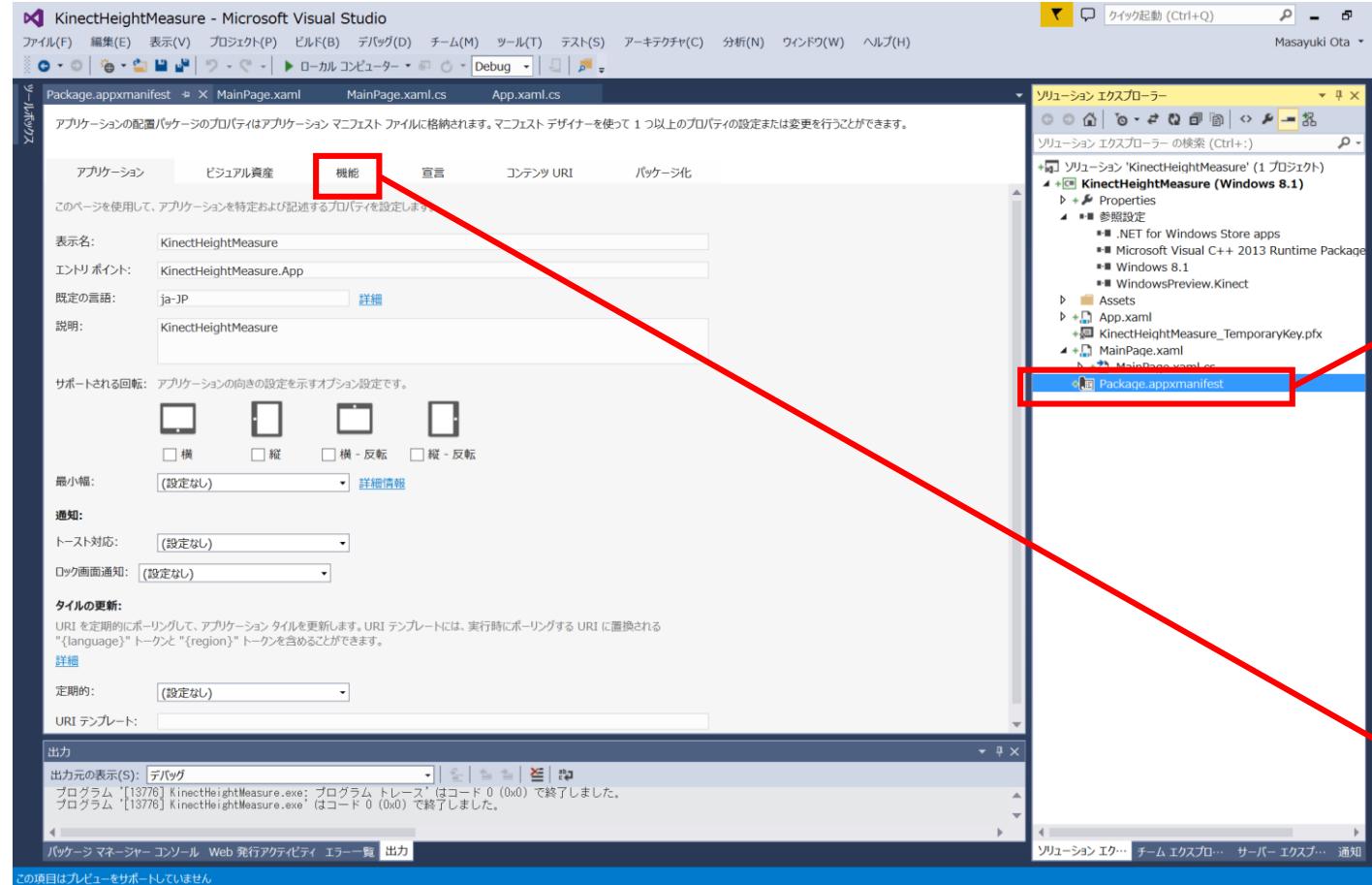


2. ここをクリックし、[x86]を選択します。

3. 左のような画面が表示されます。

4. [閉じる] ボタンをクリックし、
設定を終了します。
これにてビルド構成の変更は完了です。

Web カメラとマイクへのアクセスを許可します

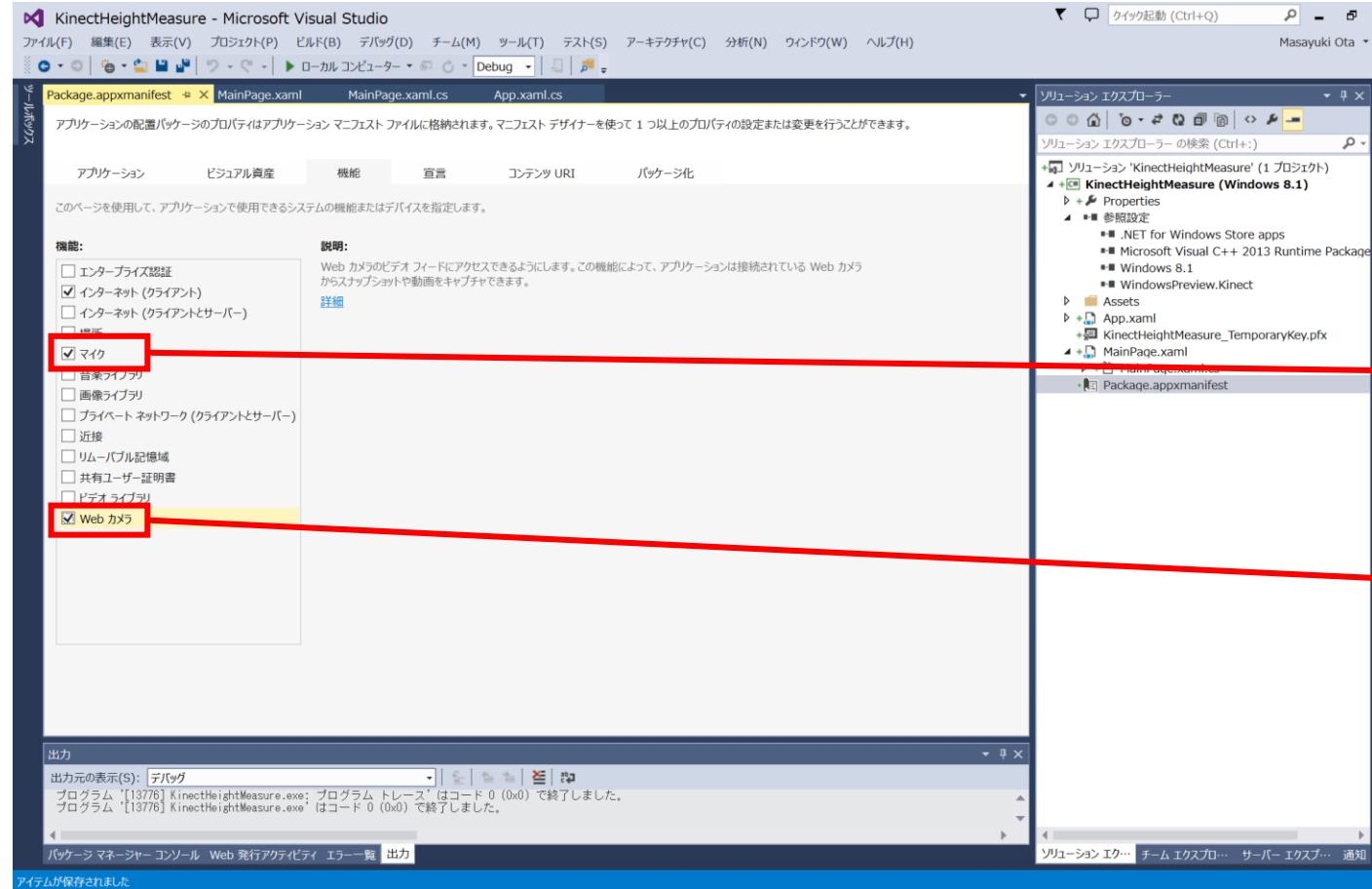


1. 「Package.appxmanifest」を右クリック

2. [開く]メニューをクリック

3. [機能]をクリック

Web カメラとマイクへのアクセスを許可します



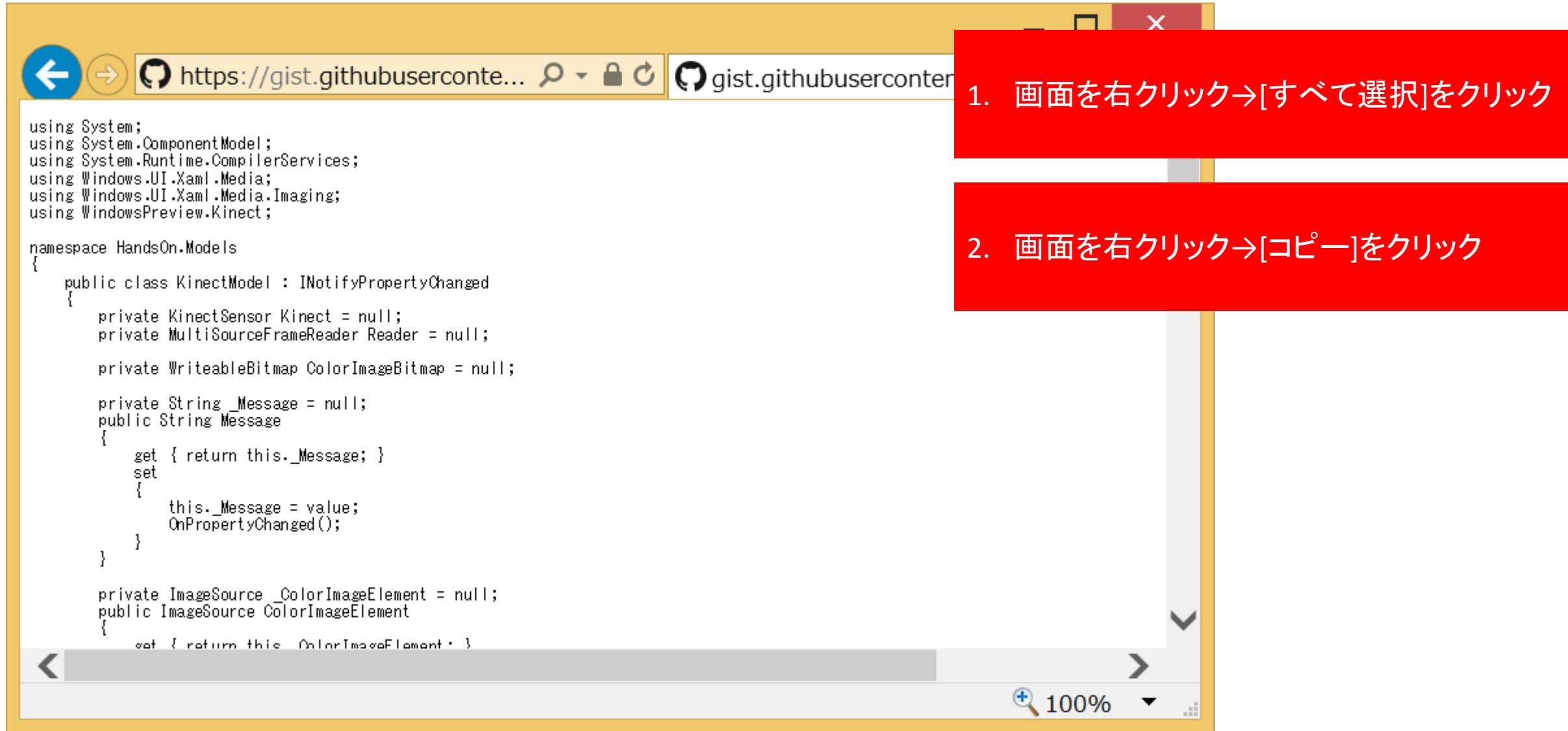
1. [マイク] の隣にチェック

2. [Web カメラ] の隣にチェック
Kinectにアクセスが可能になりました。

KinectModel.csを作成

<https://gist.githubusercontent.com/hatsunea/1ff576d9f9ff517f0281/raw/96851c3f80b68c3b613bd6cc81536711c77737df/KinectModel.cs>

<http://bit.ly/1APQMwe>



Visual StudioのKinectModel.cs

1. [編集]→[すべて選択]をクリック

2. [編集]→[貼り付け]をクリック

```
1  using System;
2  using System.ComponentModel;
3  using System.Runtime.CompilerServices;
4  using Windows.UI.Xaml.Media;
5  using Windows.UI.Xaml.Media.Imaging;
6  using WindowsPreview.Kinect;
7
8  namespace HandsOn.Models
9  {
10     public class KinectModel : INotifyPropertyChanged
11     {
12         private KinectSensor Kinect = null;
13         private MultiSourceFrameReader Reader = null;
14
15         private WriteableBitmap ColorImageBitmap = null;
16
17         private String _Message = null;
18         public String Message
19         {
20             get { return this._Message; }
21             set
22             {
23                 this._Message = value;
24                 OnPropertyChanged();
25             }
26         }
27     }
28 }
```

MainViewModel.csを作成

<http://bit.ly/1cCU2iK>



The screenshot shows a browser window with the URL <https://gist.github.com/...>. The page content is a C# class definition:

```
using System;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using Windows.UI.Xaml.Media;

namespace HandsOn.ViewModels
{
    public class MainViewModel : INotifyPropertyChanged
    {
        private Models.KinectModel Model;

        public MainViewModel()
        {
            this.Model = new Models.KinectModel();
            this.Model.PropertyChanged += Model_PropertyChanged;
        }

        public String Message
        {
            get { return this.Model.Message; }
            set
            {
                this.Model.Message = value;
            }
        }

        public ImageSource ColorImageElement
        {
            get { return this.Model.ColorImageElement; }
            set
            {
                this.Model.ColorImageElement = value;
            }
        }
    }
}
```

1. 画面を右クリック→[すべて選択]をクリック

2. 画面を右クリック→[コピー]をクリック

Visual StudioのMainViewModel.cs

The screenshot shows the Visual Studio IDE with the file `MainViewModel.cs` open. The code implements a `INotifyPropertyChanged` interface, using a private field `Model` of type `KinectModel` and a public property `Message`. The code is annotated with line numbers from 1 to 25.

```
1  using System;
2  using System.ComponentModel;
3  using System.Runtime.CompilerServices;
4  using Windows.UI.Xaml.Media;
5
6  namespace HandsOn.ViewModels
7  {
8      public class MainViewModel : INotifyPropertyChanged
9      {
10         private Models.KinectModel Model;
11
12         public MainViewModel()
13         {
14             this.Model = new Models.KinectModel();
15             this.Model.PropertyChanged += Model_PropertyChanged;
16         }
17
18         public String Message
19         {
20             get { return this.Model.Message; }
21             set
22             {
23                 this.Model.Message = value;
24             }
25         }
26     }
27 }
```

Two red callout boxes provide instructions:

1. [編集]→[すべて選択]をクリック
2. [編集]→[貼り付け]をクリック

ここでビルト

アプリが無事立ち上がるが何も起きない

1. [F5]キーを押す

MainPage.xamlの編集

```
<Grid.RowDefinitions>
    <RowDefinition Height="140"/>
    <RowDefinition Height="*"/>
</Grid.RowDefinitions>
```

こう書かれている
部分を探す

ここにコードを書く

MainPage.xamlの編集

```
<Grid.RowDefinitions>
    <RowDefinition Height="140"/>
    <RowDefinition Height="*"/>
</Grid.RowDefinitions>

<Grid Grid.Row="1">
    <Grid>
        <Viewbox Height="700" Width="1366">
            <Image Source="{Binding ColorImageElement}" Stretch="UniformToFill" />
        </Viewbox>
    </Grid>
    <TextBlock VerticalAlignment="Bottom" Text="{Binding Message}"
        TextWrapping="Wrap" Foreground="Orange" Margin="10,0,0,0"
        Style="{StaticResource SubheaderTextBlockStyle}"/>
</Grid>
```

MainPage.xaml.csの編集

```
private ViewModels.MainViewModel ViewModel = new ViewModels.MainViewModel();
```

```
public MainPage()
{
    this.InitializeComponent();
    this.navigationHelper = new NavigationHelper(this);
    this.navigationHelper.LoadState += navigationHelper_LoadState;
    this.navigationHelper.SaveState += navigationHelper_SaveState;
```

```
    this.DataContext = this.ViewModel;
```

```
}
```

```
private void navigationHelper_LoadState(object sender, LoadStateEventArgs e)
```

```
{
```

```
    this.ViewModel.KinectStart();
```

```
}
```

```
private void navigationHelper_SaveState(object sender, SaveStateEventArgs e)
```

```
{
```

```
    this.ViewModel.KinectStop();
```

```
}
```

ここでビルト

1. [F5]キーを押す

Kinectをつないでみよう

ここでビルド

Kinectカメラの映像が表示で来たら成功！

1. [F5]キーを押す

3. Kinect Studio を利用してデバックを楽にします

Kinect Studio 開発で苦労するポイント

■ アプリケーション開発・デバッグ時に Kinect センサーが必要

- SDK、開発環境は無償で入手できますが、Kinect の購入が必要
- 持ち運ぶために鞄にある程度のスペースが必要

■ センシングするために十分な場所が必要

■ 実行・デバッグの度にセンシングが必要

Kinect Studio とは

Kinect 開発を楽にするエミュレーター

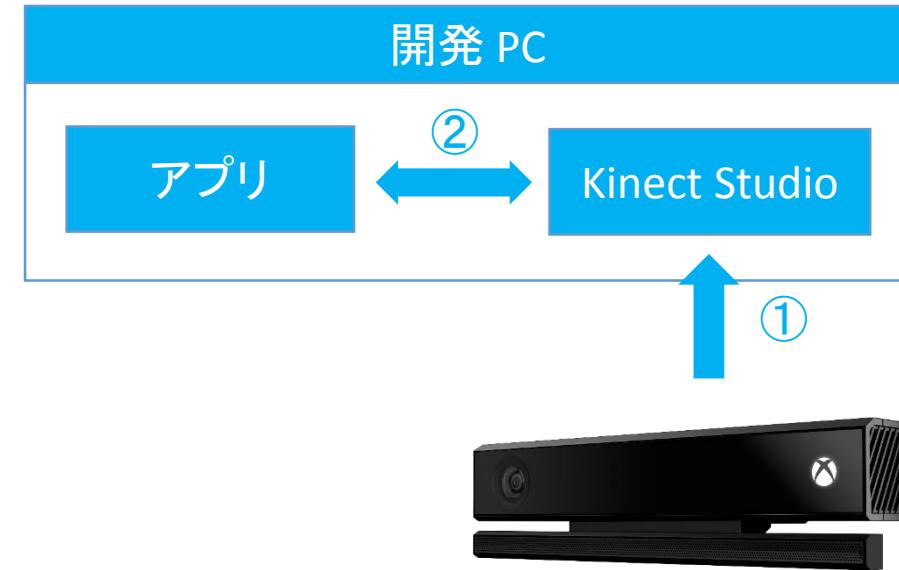
通常の開発時

1. Kinect からデータをアプリへ渡します

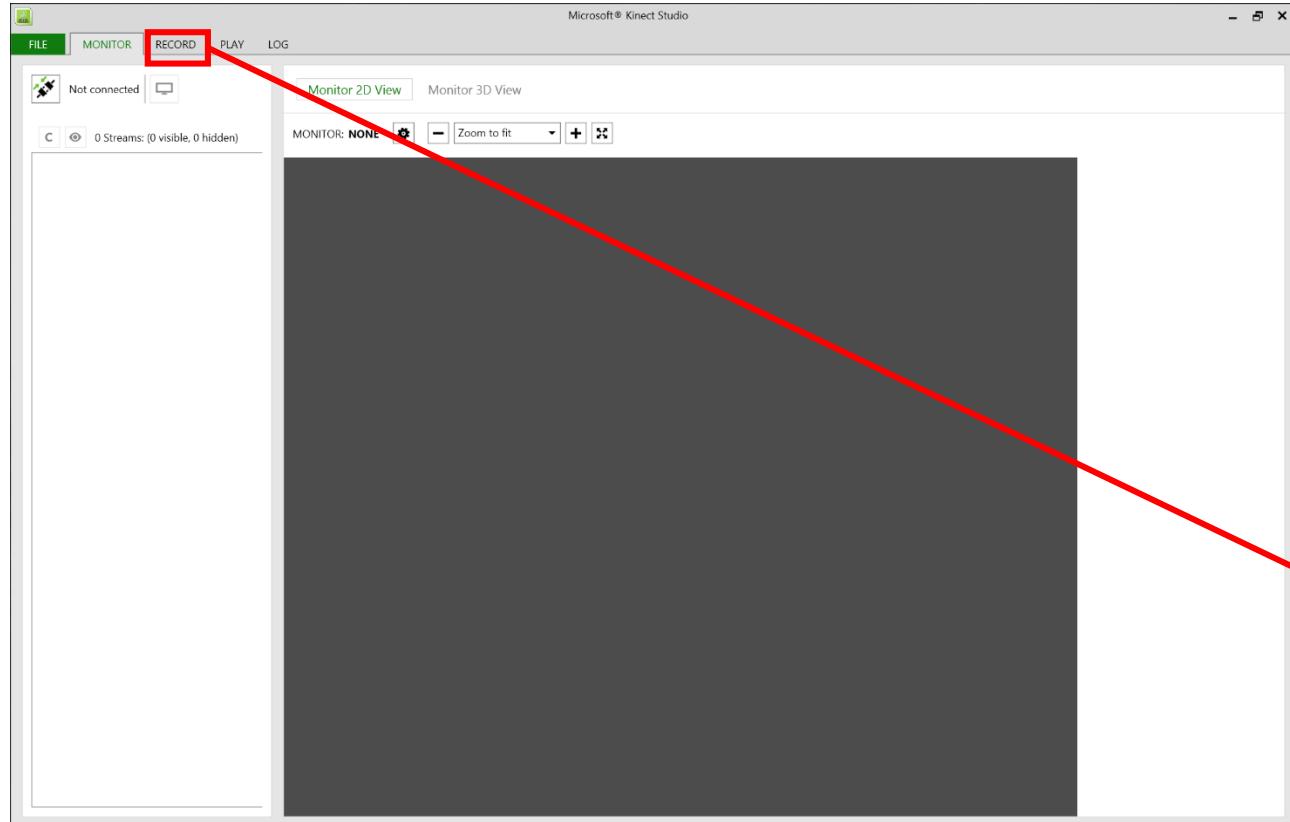


Kinect

1. Kinect からデーターを取得します
2. Kinect Studio のデータをアプリへ渡します

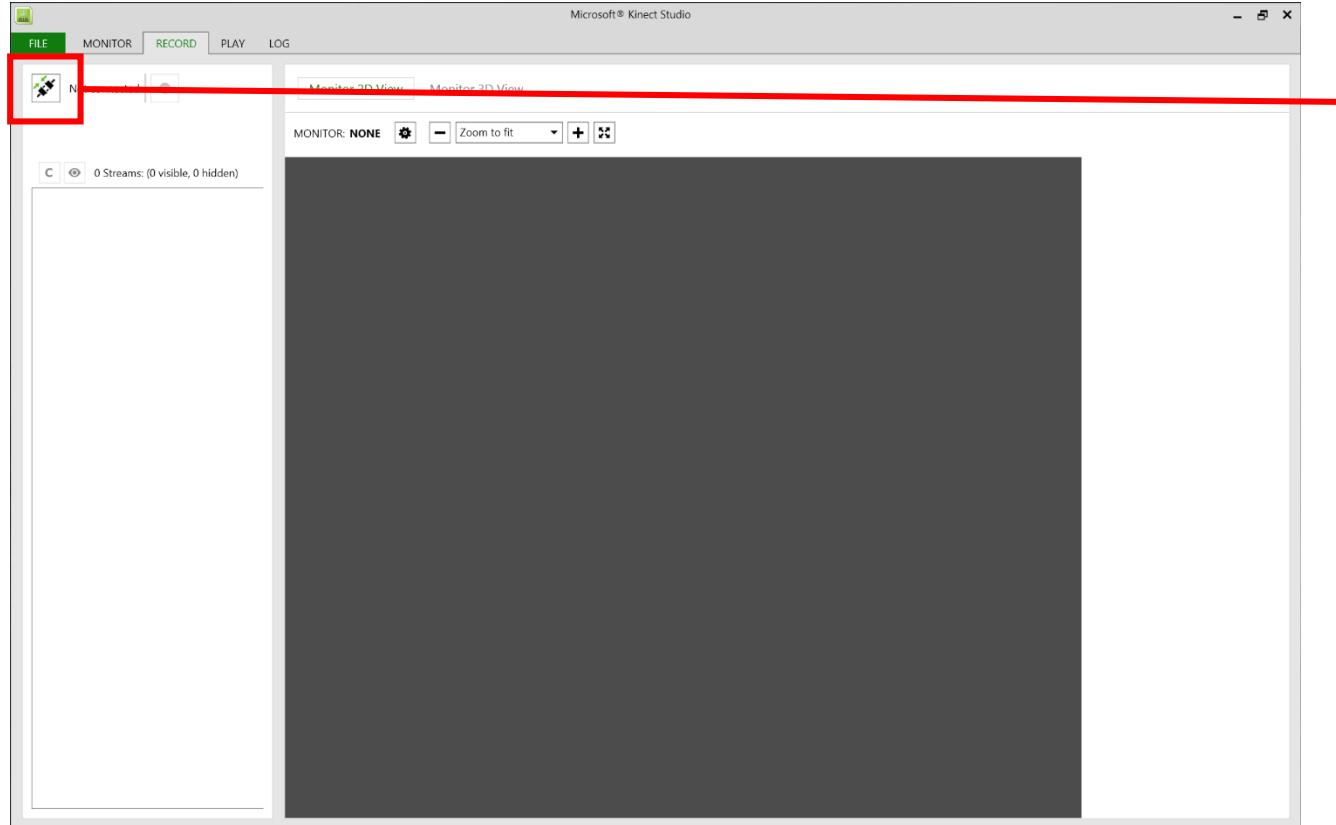


Kinect Studio を起動する



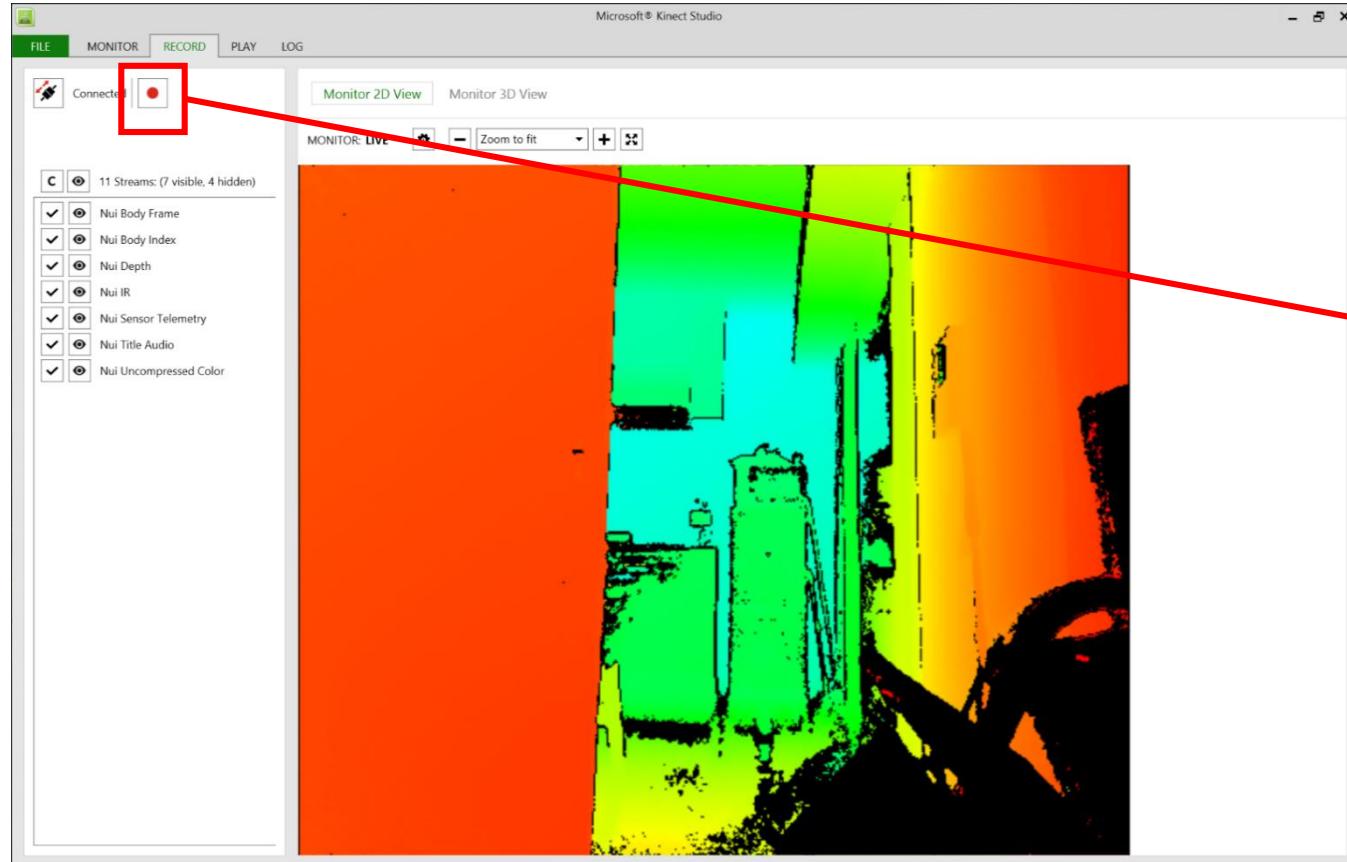
1. キーボード上で [Windows キー] を押します。
2. スタート画面が表示されるので、"Kinect" と入力し、アプリケーションを検索します。
3. "Kinect Studio v2.0" を選択し、起動します。
4. 左の画面が表示されます。
5. [Record] タブをクリックします。

Kinect からデータを取得する



1. Kinect が PC に接続されていることを確認のうえ、ここをクリックします。

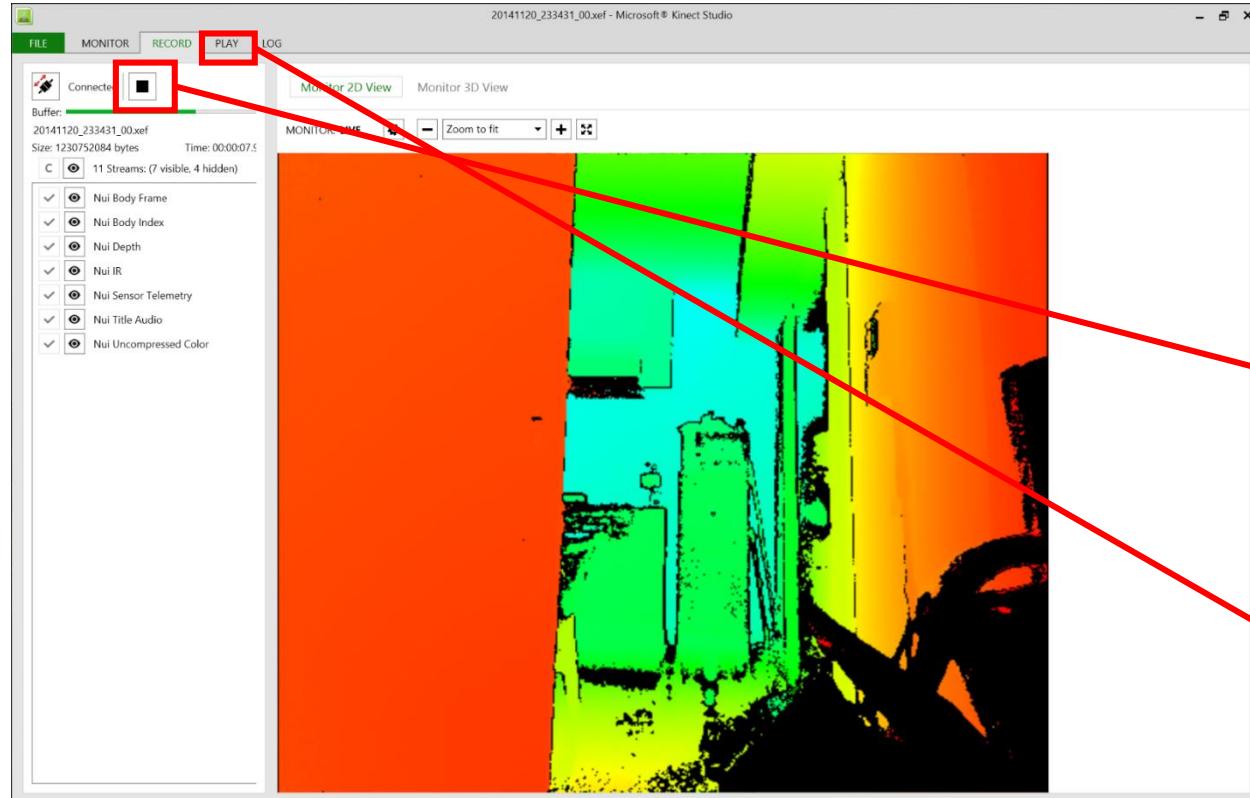
Kinect からデータを取得する



1. 画面が左のようになります

2. ここをクリックします。

Kinect からデータを取得する

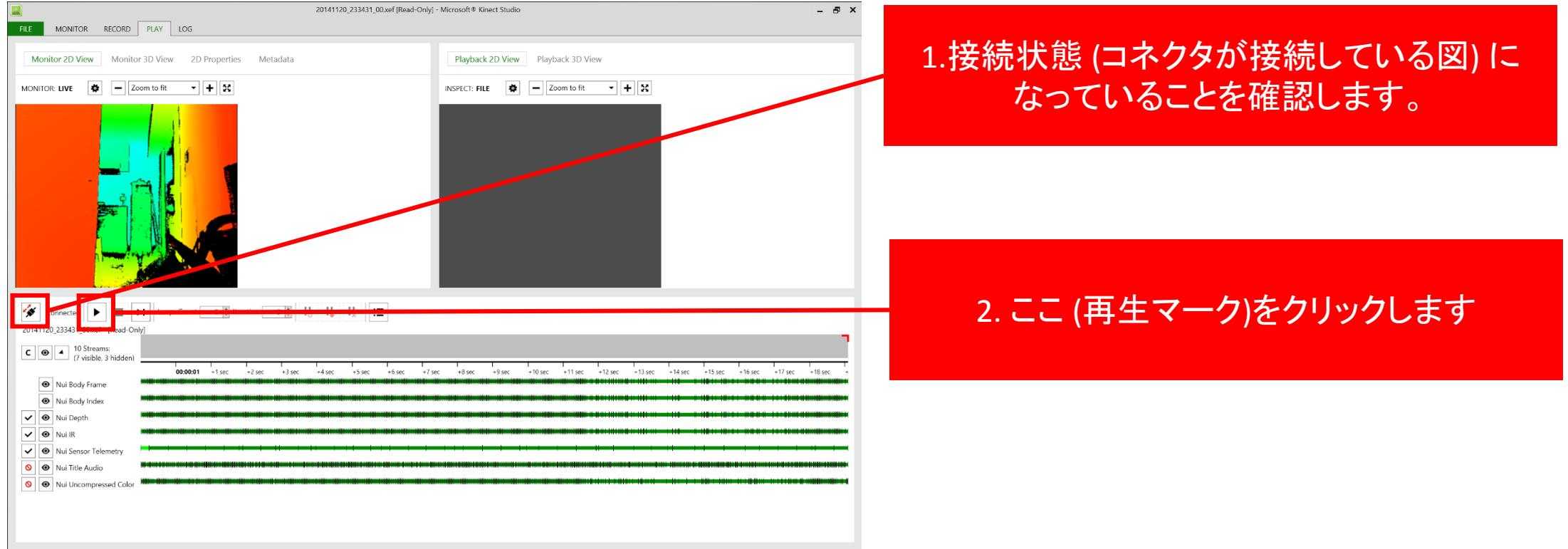


1. データの取得が開始されますので、Kinect の前で動きます。

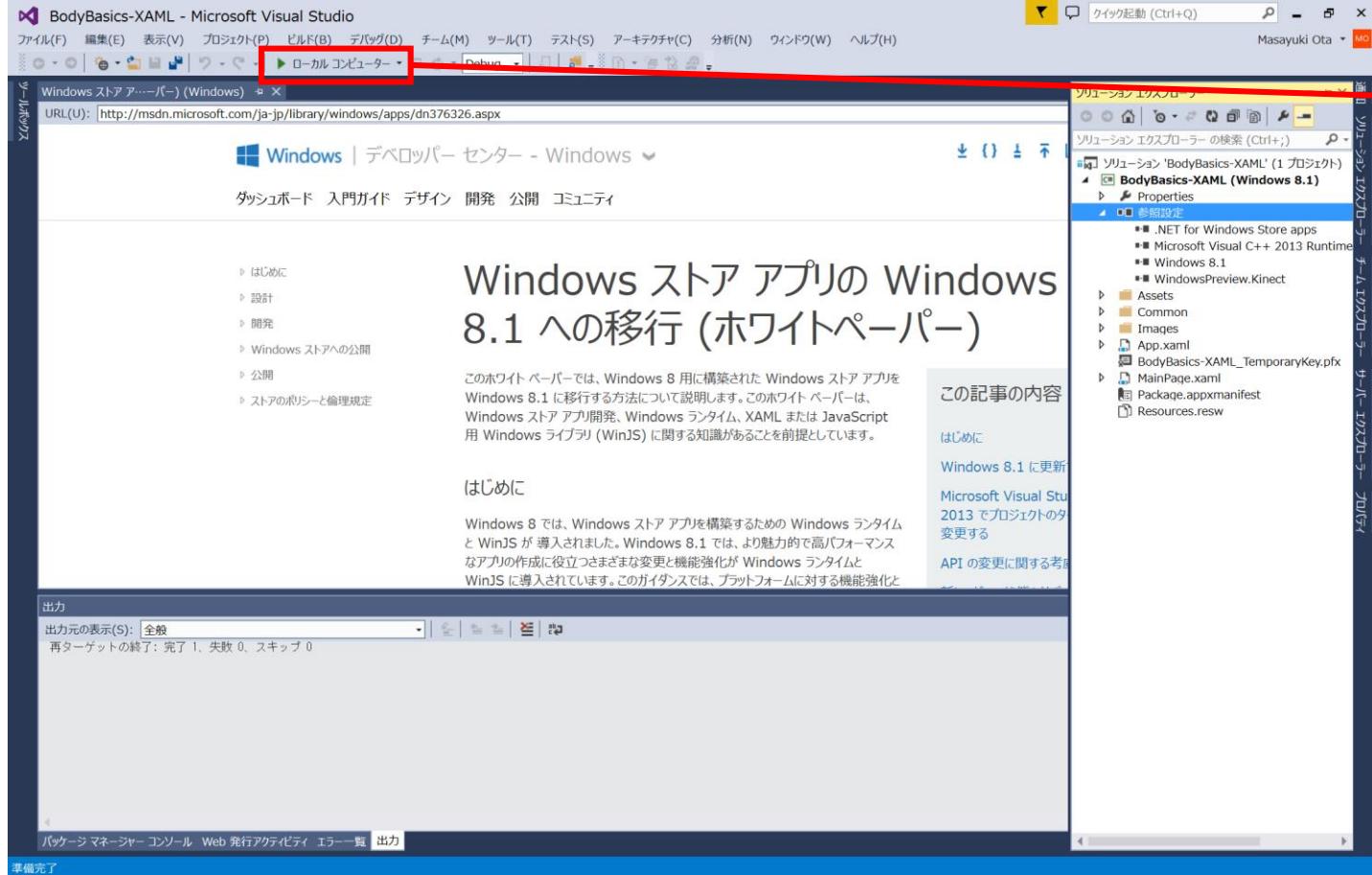
2. 目的のデーターを取得し終えましたら、ここをクリックします。

3. [Play] タブをクリックします

取得したデータでデバッグする

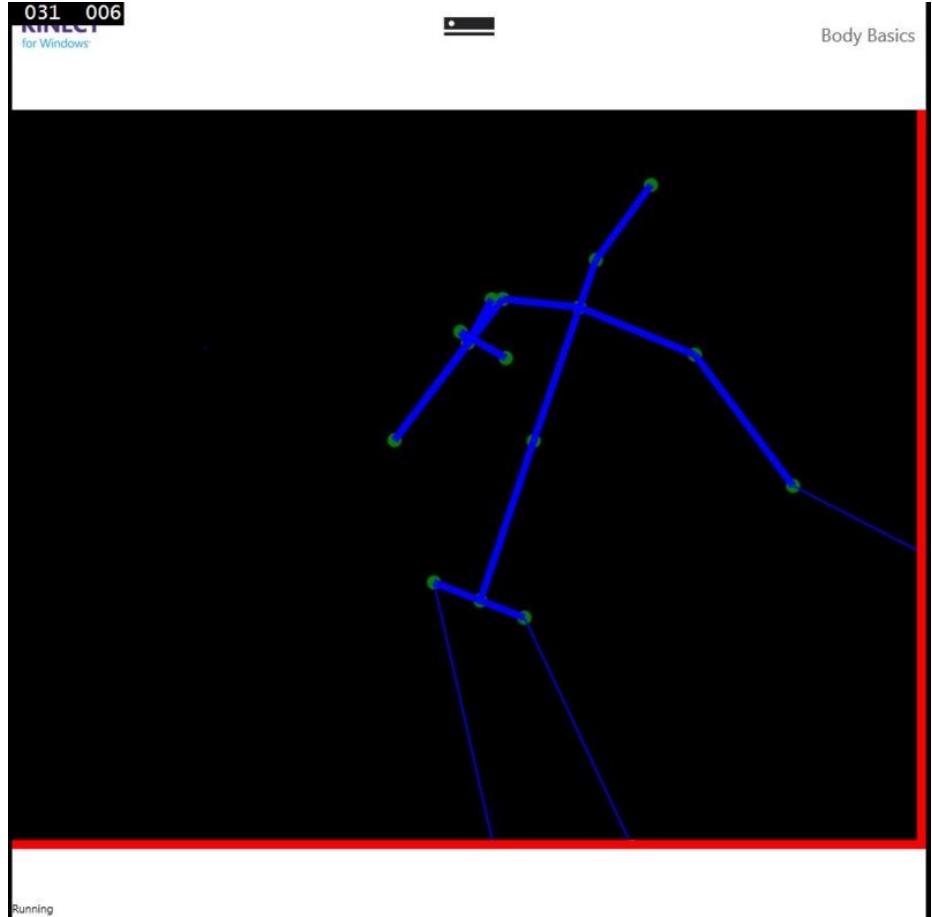


Visual Studio でサンプルを実行する



1. キーボード上で [F5] キーを押すか、
ここをクリックします。

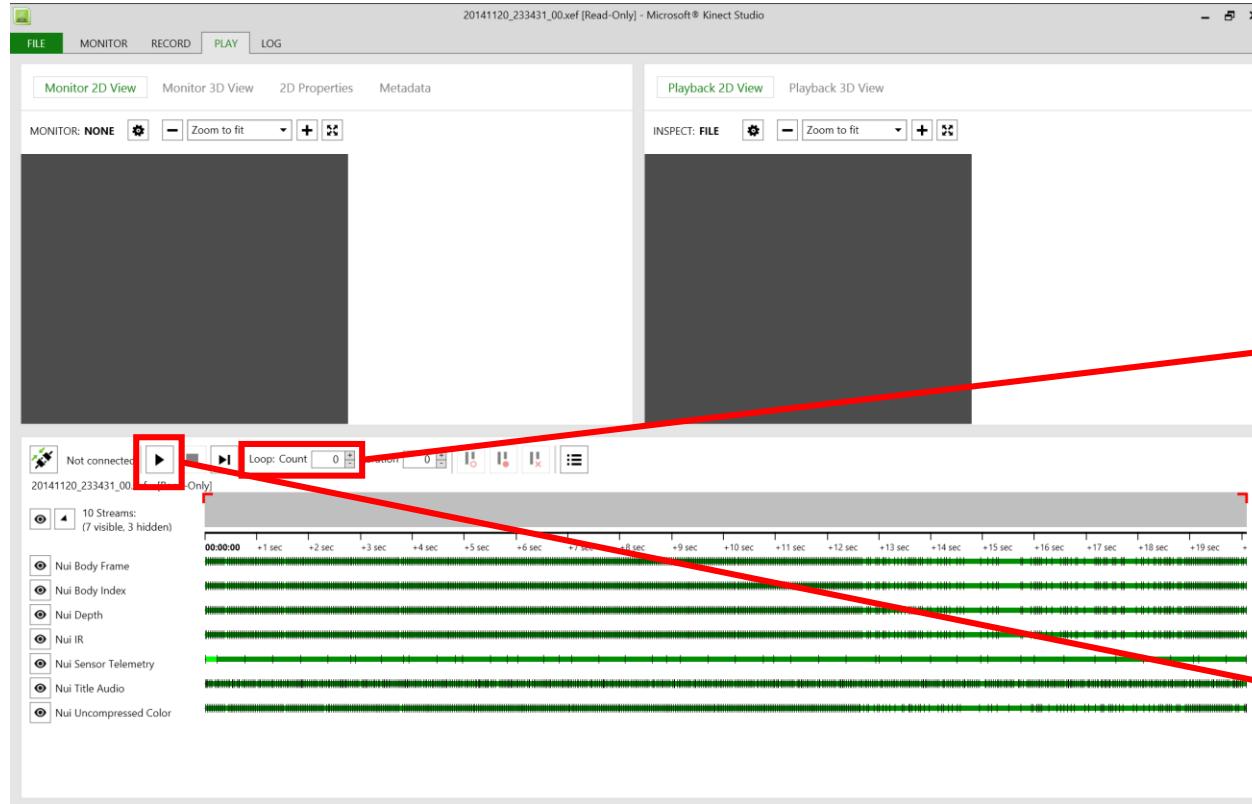
Visual Studio でサンプルを実行する



Kinect Studio からのデータを元に
サンプルプログラムのデバッグを行うことができます。

手元に Kinect が無い時に Kinect 開発をする場合には
Kinect Studio を使って快適に開発を進めてみてください。

補足情報



1. デバッグをより快適に実行するために同じデータを繰り返して再生することができます。

2. [Loop Count] の値を “3” など任意の値に設定します。

3. ここをクリックし、デバッグを進めてください。

4. じゃんけんアプリを作成

じゃんけんアプリの仕組み

- ハンズオンで作成したKinectカメラアプリを拡張
 - <http://bit.ly/1QcMANJ>
- 右手のボーンを取得
- 右手のボーンのステータスを判定
- 判定結果をじゃんけんにマッピング

KinectModel.csに じゃんけんを返すインターフェイスを追加

```
public enum ResultState
{
    Unknown = HandState.Unknown,
    Lock = HandState.Closed,
    Paper = HandState.Open,
    Scissors = HandState.Lasso
}

private ResultState _Result = (ResultState)HandState.Unknown;
public ResultState Result
{
    get { return this._Result; }
    set
    {
        this._Result = value;
        OnPropertyChanged();
    }
}
```

1. ColorImageElementの定義の後の
38行目に追記

<http://bit.ly/1KfZ0o7> (文字化けしてたらUTF8)



The screenshot shows a browser window with the URL <https://gist.githubusercontent.com/hatsunea/1ff576d9f9ff517f0281/raw/edc>. The page content is a C# script for skeleton tracking:

```
//骨格情報を取得する
using (var frame = multiSourceFrame.BodyFrameReference.AcquireFrame())
{
    if (frame != null)
    {
        Body targetBody = null;
        var bodies = new Body[frame.BodyCount];

        frame.GetAndRefreshBodyData(bodies);

        //一番近い左手を結果に設定する
        foreach (var item in bodies)
        {
            if (item.IsTracked && item.Joints[JointType.HandRight].TrackingState != TrackingState.TrackingFailure)
            {
                if (targetBody == null || targetBody.Joints[JointType.HandRight].Position.Z < item.Joints[JointType.HandRight].Position.Z)
                {
                    targetBody = item;
                }
            }
        }
        if (targetBody != null)
        {
            this.Result = (ResultState)targetBody.HandRightState;
        }
    }
}
```

1. 画面を右クリック→[すべて選択]をクリック

2. 画面を右クリック→[コピー]をクリック

Visual StudioのKinectModel.cs

The screenshot shows the Visual Studio IDE with the KinectModel.cs file open. The code editor displays C# code for handling Kinect frame data. A red box highlights line 141, and two numbered instructions are overlaid on the right side of the screen.

```
117         if (multiSourceFrame != null)
118     {
119         //カラー情報を取得する
120         using (var frame = multiSourceFrame.ColorFrameReference.)
121         {
122             if (frame != null)
123             {
124                 var frameDescription = frame.FrameDescription;
125                 if ((frameDescription.Width == this.ColorImageBitmap.Pix
126                     {
127                         if (frame.RawColorImageFormat == ColorImageFormat.Bg
128                         {
129                             frame.CopyRawFrameDataToBuffer(this.ColorImageBi
130                         }
131                         else
132                         {
133                             frame.CopyConvertedFrameDataToBuffer(this.ColorI
134                         }
135                         this.ColorImageBitmap.Invalidate();
136                         this.ColorImageElement = this.ColorImageBitmap;
137                     }
138                 }
139             }
140         }
141     }
142 }
```

1. 141行目で改行
2. [編集]→[貼り付け]をクリック

Visual StudioのMainViewModel.cs

```
public ImageSource ColorImageElement
{
    get { return this.Model.ColorImageElement; }
    set
    {
        this.Model.ColorImageElement = value;
    }
}

public HandsOn.Models.KinectModel.ResultState Result
{
    get { return this.Model.Result; }
    set
    {
        this.Model.Result = value;
        OnPropertyChanged();
    }
}
```

1. ColorImageElementの定義の後に追記

Visual Studioの MainPage.xaml

1. Messageの定義の後に追記

```
<TextBlock VerticalAlignment="Bottom" Text="{Binding Message}"
    TextWrapping="Wrap" Foreground="Orange" Margin="10,0,0,0"
    Style="{StaticResource SubheaderTextBlockStyle}"/>
<TextBlock VerticalAlignment="Center" HorizontalAlignment="Center"
    Text="{Binding Result}"
    TextWrapping="Wrap" Foreground="Orange" Margin="10,0,0,0"
    Style="{StaticResource HeaderTextBlockStyle}"/>
</Grid>
```

FIN

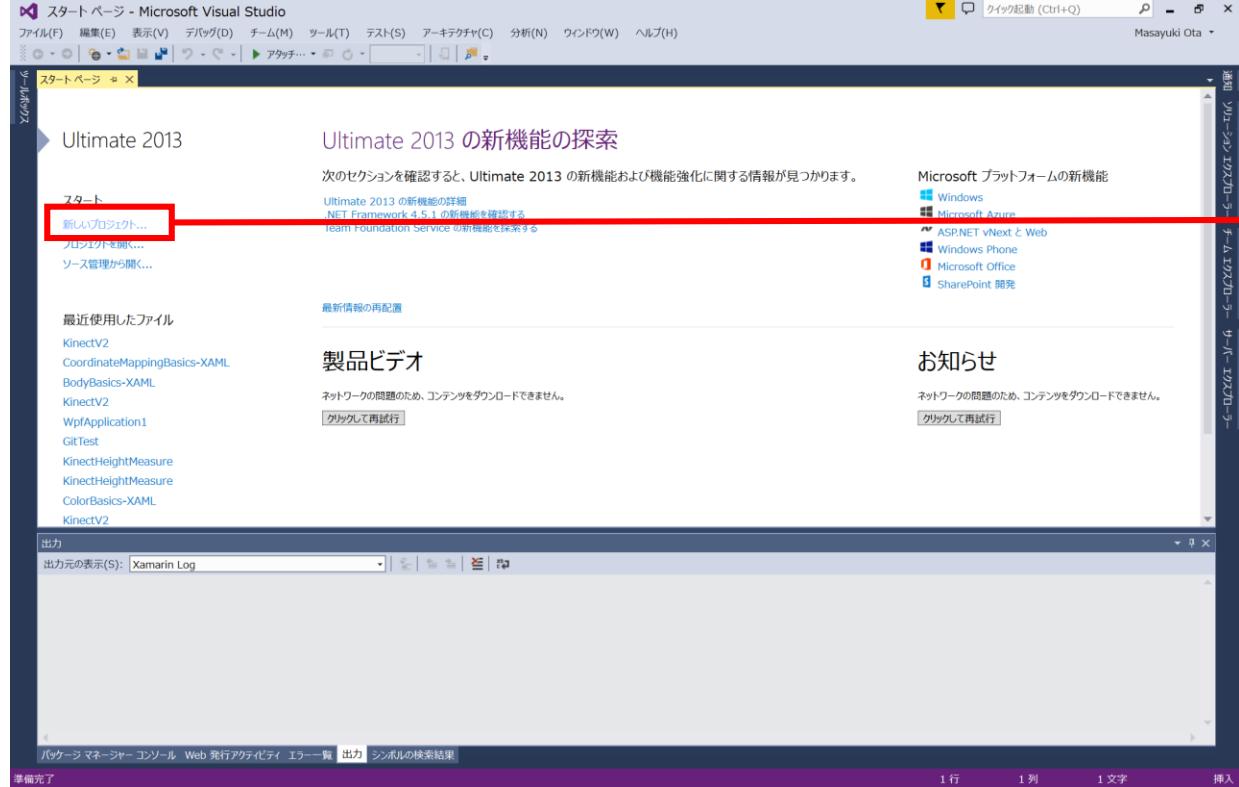


3. 身長を計測するWindowsストアアプリを作ります (90分)

1. Visual Studio のプロジェクトを作成します
2. 画面に RGB カメラからの出力を表示します
3. 骨格情報から身長を計測し画面に表示します
4. (Option) BodyIndex frame を使ってより正確に身長を計測します

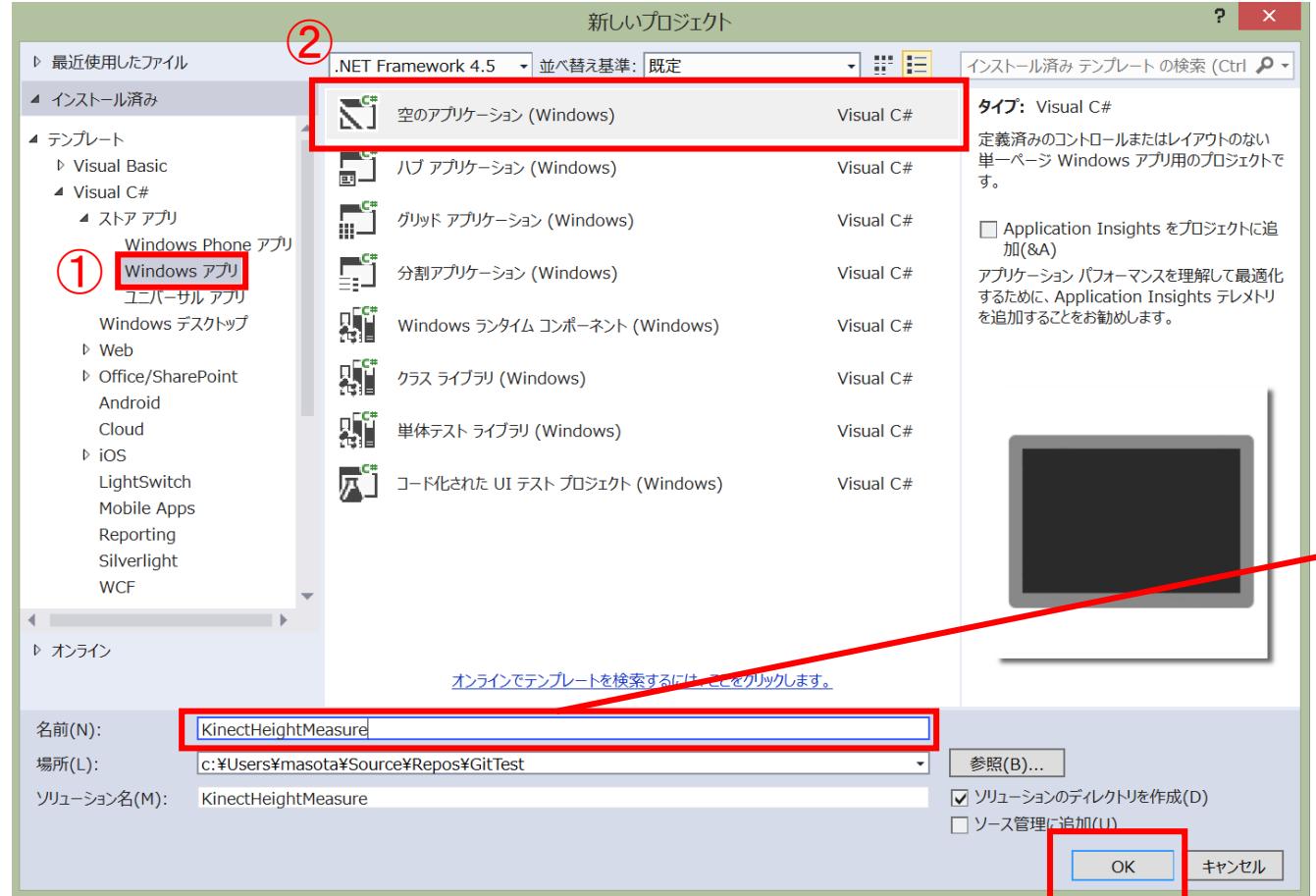
1、2は画面操作の手順も多いため手順書に記載しております。
3、4はコードの分量が多くPowerPointよりもコード上のコメントの方が理解しやすいため、ハンズオンの中でソリューションファイルごと提供いたします。

Visual Studio を起動します



1. [新しいプロジェクト] をクリックします。

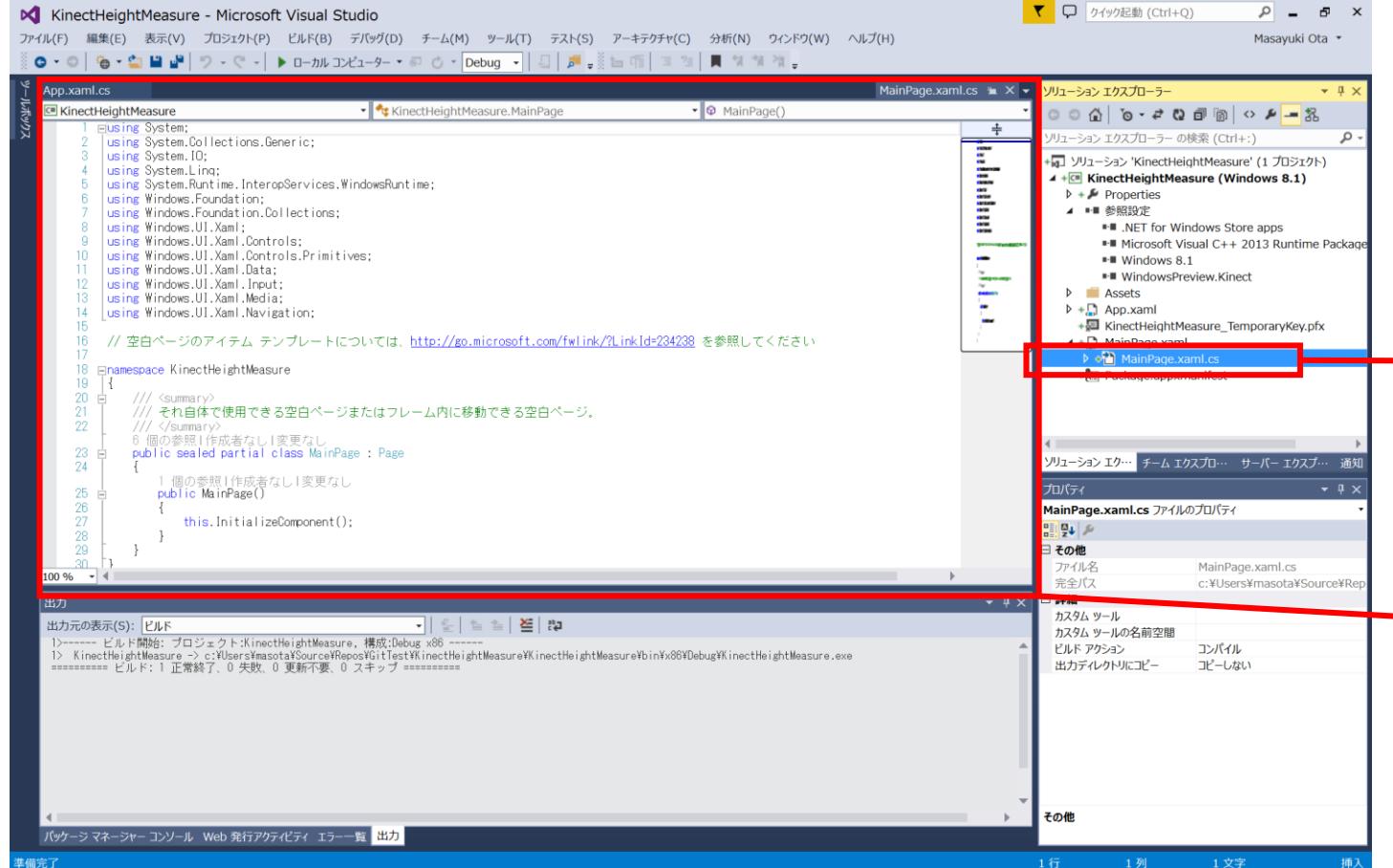
プロジェクトを作成します



①②の順で選択し、“Windows アプリ”の“空のアプリケーション”を選択します。

③ “KinectHeightMeasure”という名前を入力します。
④ [OK] ボタンをクリックします。

Kinect に接続するプログラムを追加する



1. “MainPage.xaml.cs” をダブルクリックします。

2. 画面左側に “MainPage.xaml.cs” のソースコードが表示されます。

Kinect に接続するプログラムを追加する

```
10 using Windows.UI.Xaml.Controls.Primitives;
11 using Windows.UI.Xaml.Data;
12 using Windows.UI.Xaml.Input;
13 using Windows.UI.Xaml.Media;
14 using Windows.UI.Xaml.Navigation;
15 using WindowsPreview.Kinect;
16 // 空白ページのアイテム テンプレートについては、http://go.microsoft.com/fwlink/?LinkId=234238
17
18 namespace KinectHeightMeasure
19 {
20     /// <summary>
21     /// それ自体で使用できる空白ページまたはフレーム内に移動できる空白ページ。
22     /// </summary>
23     6 個の参照 | 作成者なし | 変更なし
24     public sealed partial class MainPage : Page
25     {
26         private KinectSensor kinect;
27         1 個の参照 | 作成者なし | 変更なし
28         public MainPage()
29         {
30             this.InitializeComponent();
31         }
32     }
33 }
```

1. Kinect の name space を追加します。

2. このクラス内から参照するために変数を追加します。

Kinect に接続するプログラムを追加する

```
namespace KinectHeightMeasure
{
    /// <summary>
    /// それ自体で使用できる空白ページまたはフレーム内に移動できる空白ページ。
    /// </summary>
    6 個の参照 | 作成者なし | 変更なし
    public sealed partial class MainPage : Page
    {
        private KinectSensor kinect;

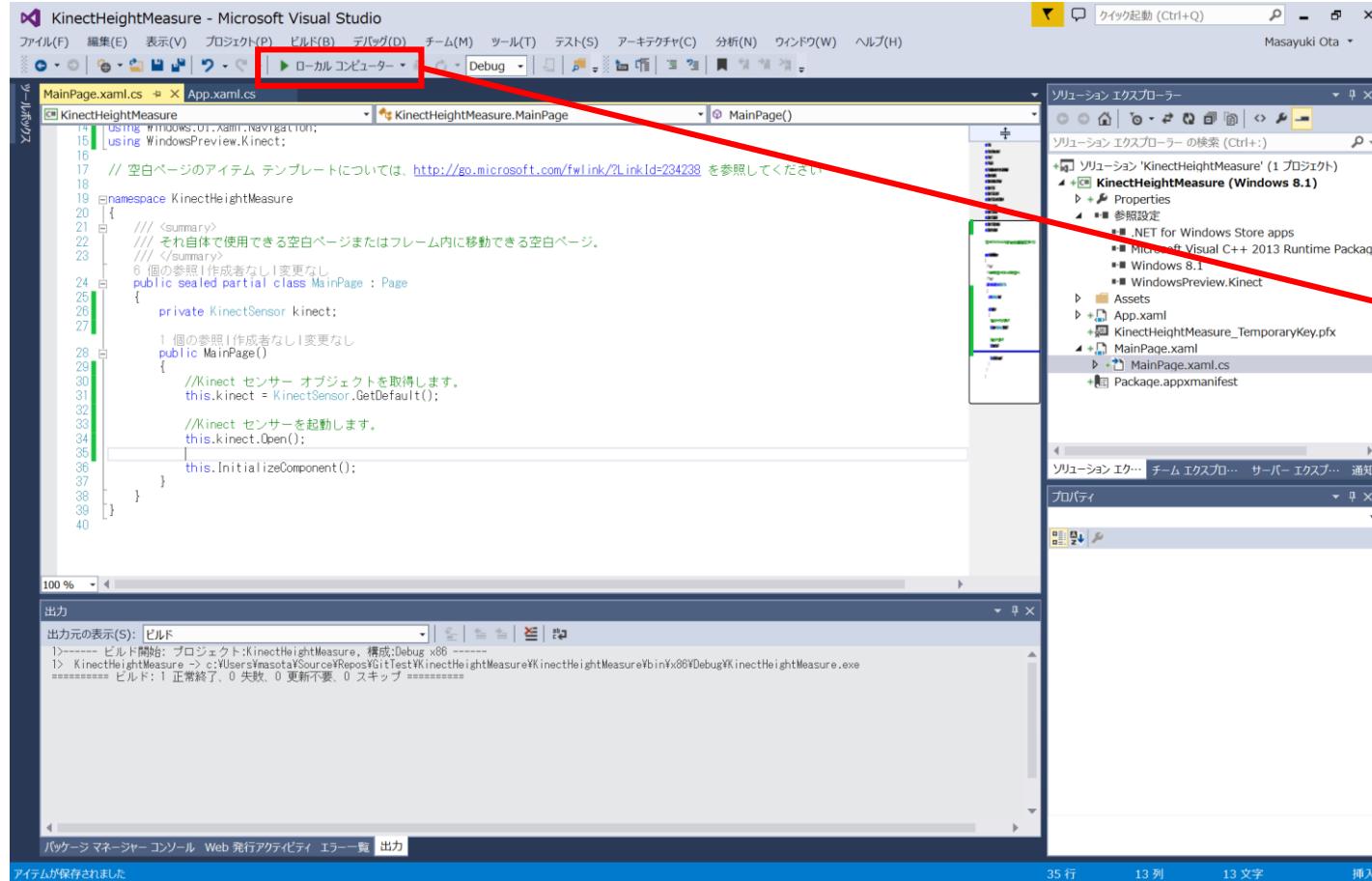
        1 個の参照 | 作成者なし | 変更なし
        public MainPage()
        {
            //Kinect センサー オブジェクトを取得します。
            this.kinect = KinectSensor.GetDefault();

            //Kinect センサーを起動します。
            this.kinect.Open();

            this.InitializeComponent();
        }
    }
}
```

Kinect センサー オブジェクトを取得後、
起動しています。

Kinect に接続するプログラムを追加する



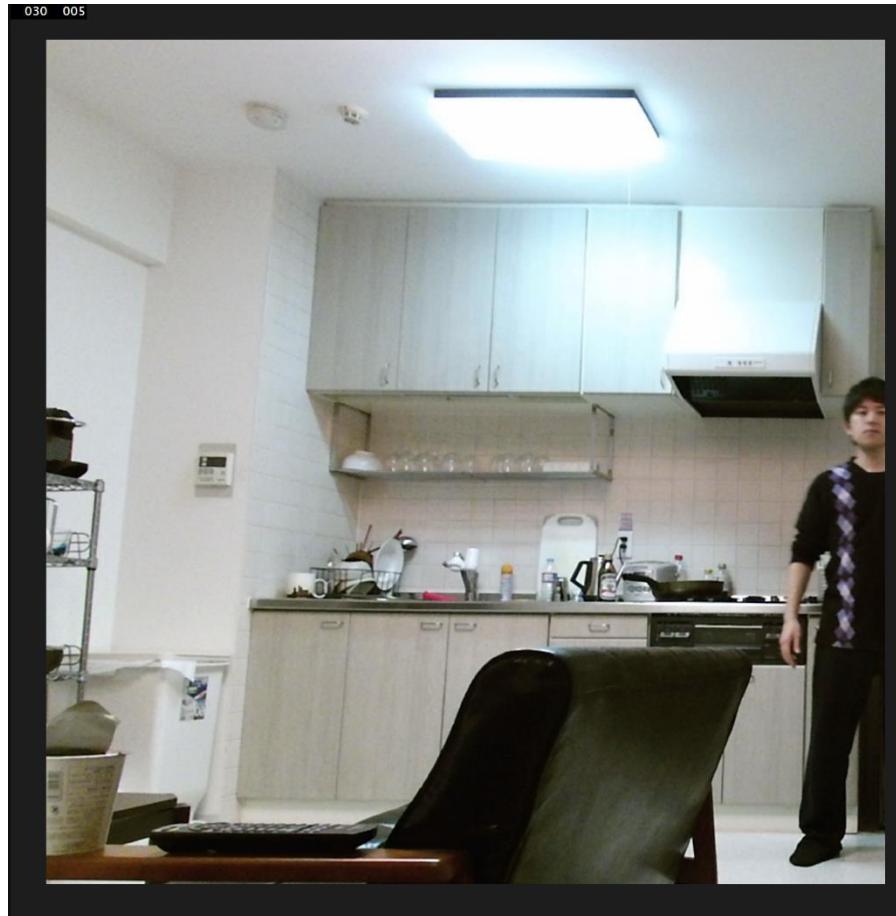
1. キーボード上で [F5] キーを押すか、このボタンをクリックしデバッグ実行します。
2. Kinect センサーを見てください。赤外線のビームが照射されている(起動している)ことが確認できます。

3. 身長を計測するWindowsストアアプリを作ります (90分)

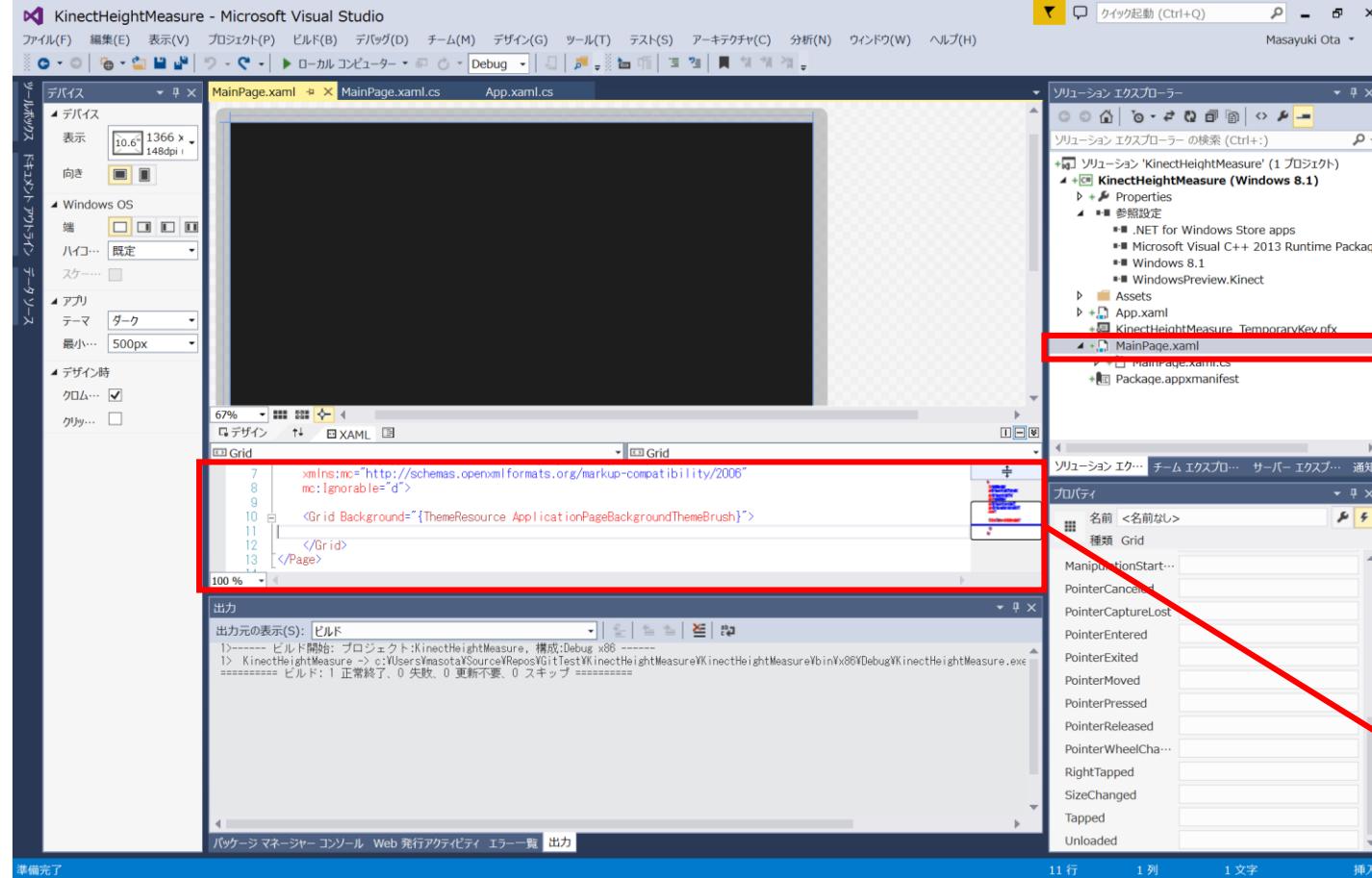
1. Visual Studio のプロジェクトを作成します
2. 画面に RGB カメラからの出力を表示します
3. 骨格情報から身長を計測し画面に表示します
4. (Option) BodyIndex frame を使ってより正確に身長を計測します

1、2は画面操作の手順も多いため手順書に記載しております。
3、4はコードの分量が多くPowerPointよりもコード上のコメントの方が理解しやすいため、ハンズオンの中でソリューションファイルごと提供いたします。

この手順で出来上がるもののイメージ



画面に表示領域を追加します



1. “ MainPage.xaml ” をダブル クリックします。

2. 左図のようにデザイン画面が表示されます。

3. ここを編集していきます

画面に表示領域を追加します

```
1 <Page  
2     x:Class="KinectHeightMeasure.MainPage"  
3     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
4     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
5     xmlns:local="using:KinectHeightMeasure"  
6     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  
7     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  
8     mc:Ignorable="d">  
9  
10    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">  
11        <Image Name="theImage" Margin="40,40,40,40" Stretch="None"/>  
12    </Grid>  
13 </Page>
```

画像を表示するための Image タグを挿入します。名前は “theImage” です。プログラムからは、this.theImage でアクセスできます。

画面描画のためのプログラムを書きます

```
15  using WindowsPreview.Kinect;
16  using Windows.UI.Xaml.Media.Imaging;
17
18 // 空白ページのアイテム テンプレートについては、http://go.microsoft.com/fwlink/?LinkId=234238 を参照してください
19
20 namespace KinectHeightMeasure
21 {
22     /// <summary>
23     /// それ自体で使用できる空白ページまたはフレーム内に移動できる空白ページ。
24     /// </summary>
25     4 個の参照 | 作成者なし | 変更なし
26     public sealed partial class MainPage : Page
27     {
28         private KinectSensor kinect;
29
30         /// <summary>
31         /// 複数の frame (カラー、深さ、骨格) を取得する Reader のための変数
32         /// </summary>
33         private MultiSourceFrameReader multiReader = null;
34
35         /// <summary>
36         /// colorFrame の設定値 (横幅や、ピクセルごとのビット数など) を確認するためのオブジェクトを格納する変数
37         /// </summary>
38         private FrameDescription colorFrameDescription = null;
39
40         /// <summary>
41         /// 書き込みおよび更新が可能なビットマップデータの格納先
42         /// </summary>
43         private WriteableBitmap bitmap = null;
44
45         0 個の参照 | 作成者なし | 変更なし
46         public MainPage()
47         {
48             //Kinect センサー オブジェクトを取得します。
49             this.kinect = KinectSensor.GetDefault();
50
51             //Kinect センサーを起動します。
52             this.kinect.Open();
53         }
54     }
55 }
```

1. ビットマップデータの格納オブジェクト (WritableBitmap) を利用するために namespace を追加します。

2. RGB カメラのみ利用する場合には、 ColorFrameReader を利用しますが、ハンズオンの後半で骨格情報なども利用するので、複数の情報をインプットできる MultiSourceFrameReader を利用します。

3. 後ほど使いまわす RGB カメラの属性情報 を格納する変数を宣言します。

4. 画像データの格納先の変数を宣言します。

画面描画のためのプログラムを書きます

```
0 個の参照 | 作成者なし | 変更なし
public MainPage()
{
    //Kinect センサー オブジェクトを取得します。
    this.kinect = KinectSensor.GetDefault();

    //複数 frame (カラー画像、骨格情報、人の認識) 用の Reader を取得します。
    this.multiReader = this.kinect.OpenMultiSourceFrameReader(FrameSourceTypes.Color);

    //frame 到着時のイベントハンドラを登録 (カラー画像、骨格情報、人の認識時のすべてのイベントを同じハンドラで処理します。)
    this.multiReader.MultiSourceFrameArrived += multiReader_MultiSourceFrameArrived;

    //取得する画像の情報形式を BGRA (RGB と透明度を表す Alpha) フォーマットとして定義。
    colorFrameDescription = this.kinect.ColorFrameSource.CreateFrameDescription(ColorImageFormat.Bgra);

    //表示用の bitmap データ作成します。
    this.bitmap = new WriteableBitmap(colorFrameDescription.Width, colorFrameDescription.Height);

    //Kinect センサーを起動します。
    this.kinect.Open();

    this.InitializeComponent();
    theImage.Source = this.bitmap;
}
```

1. 複数のリソースを読み込む
MultiSourceFrameReader を変数に格納します。

2. イベントハンドラを宣言します。
なお、Visual Studio で “ += ” の部分まで書いていただくとイベントハンドラ名が推薦されるので、Tab を 2 回押して自動生成してください。

画面描画のためのプログラムを書きます

```
0 個の参照 | 作成者なし | 変更なし
public MainPage()
{
    //Kinect センサー オブジェクトを取得します。
    this.kinect = KinectSensor.GetDefault();

    //複数 frame (カラー画像、骨格情報、人の認識) 用の Reader を取得します。
    this.multiReader = this.kinect.OpenMultiSourceFrameReader(FrameSourceTypes.Color);

    //frame 到着時のイベントハンドラを登録 (カラー画像、骨格情報、人の認識時のすべてのイベントを同じハンドラで処理します。)
    this.multiReader.MultiSourceFrameArrived += multiReader_MultiSourceFrameArrived;

    //取得する画像の情報形式を BGRA (RGB と透明度を表す Alpha) フォーマットとして定義。
    colorFrameDescription = this.kinect.ColorFrameSource.CreateFrameDescription(ColorImageFormat.Bgra);

    //表示用の bitmap データ作成します。
    this.bitmap = new WriteableBitmap(colorFrameDescription.Width, colorFrameDescription.Height);

    //Kinect センサーを起動します。
    this.kinect.Open();

    this.InitializeComponent();
    theImage.Source = this.bitmap;
}
```

3. Kinect が取り込む RGB カメラの情報
(幅・高さなど) が格納されます。

4. Kinect が取り込む RGB カメラの幅・高さを
考慮して格納先を生成します。

5. 画面に追加した Image タグの画像ソースに
先ほど生成した WritableBitmap を指定しま
す。この紐づけにより、WritableBitmap を変
更すると表示が変わることになります。

画面描画のためのプログラムを書きます

```
1 個の参照 | 作成者なし | 変更なし
void multiReader_MultiSourceFrameArrived(MultiSourceFrameReader sender, MultiSourceFrameArrivedEventArgs args)
{
    bool colorFrameProcessed = false;

    //複数 frame データへの参照を取得します。
    var reference = args.FrameReference.AcquireFrame();

    using (ColorFrame colorframe = reference.ColorFrameReference.AcquireFrame())
    {
        if (colorframe != null)
        {
            FrameDescription colorFrameDescription = colorframe.FrameDescription;

            //データを検証したうえで bitmap データを更新する。
            if ((colorFrameDescription.Width == this.bitmap.PixelWidth) && (colorFrameDescription.Height == this.bitmap.PixelHeight))
            {
                if (colorframe.RawColorImageFormat == ColorImageFormat.Bgra)
                {
                    colorframe.CopyRawFrameDataToBuffer(this.bitmap.PixelBuffer);
                }
                else
                {
                    colorframe.CopyConvertedFrameDataToBuffer(this.bitmap.PixelBuffer, ColorImageFormat.Bgra);
                }
                colorFrameProcessed = true;
            }
        }
    }
    //color frame が取得できている場合には表示する。
    if (colorFrameProcessed)
    {
        this.bitmap.Invalidate();
    }
}
```

2つ前のスライドで追加したイベントハンドラー (multiReader_MultiSourceFrameArrived) 内を編集します。
このイベントハンドラーは、Kinect が新しい情報を取得するたびに呼び出されます。

画面描画のためのプログラムを書きます

```
1 個の参照 | 作成者なし | 変更なし
void multiReader_MultiSourceFrameArrived(MultiSourceFrameReader sender, MultiSourceFrameArrivedEventArgs args)
{
    bool colorFrameProcessed = false;

    //複数 frame データへの参照を取得します。
    var reference = args.FrameReference.AcquireFrame();

    using (ColorFrame colorframe = reference.ColorFrameReference.AcquireFrame())
    {
        if (colorframe != null)
        {
            FrameDescription colorFrameDescription = colorframe.FrameDescription;

            //データを検証したうえで bitmap データを更新する。
            if ((colorFrameDescription.Width == this.bitmap.PixelWidth) && (colorFrameDescription.Height == this.bitmap.PixelHeight))
            {
                if (colorframe.RawColorImageFormat == ColorImageFormat.Bgra)
                {
                    colorframe.CopyRawFrameDataToBuffer(this.bitmap.PixelBuffer);
                }
                else
                {
                    colorframe.CopyConvertedFrameDataToBuffer(this.bitmap.PixelBuffer, ColorImageFormat.Bgra);
                }
                colorFrameProcessed = true;
            }
        }
    }

    //color frame が取得できている場合には表示する。
    if (colorFrameProcessed)
    {
        this.bitmap.Invalidate();
    }
}
```

1. 実際に届いたデータの形式(幅・高さなど)と
初めに取得した設定値を比較し、正しいデータが届いた場合のみ次の処理に進みます。

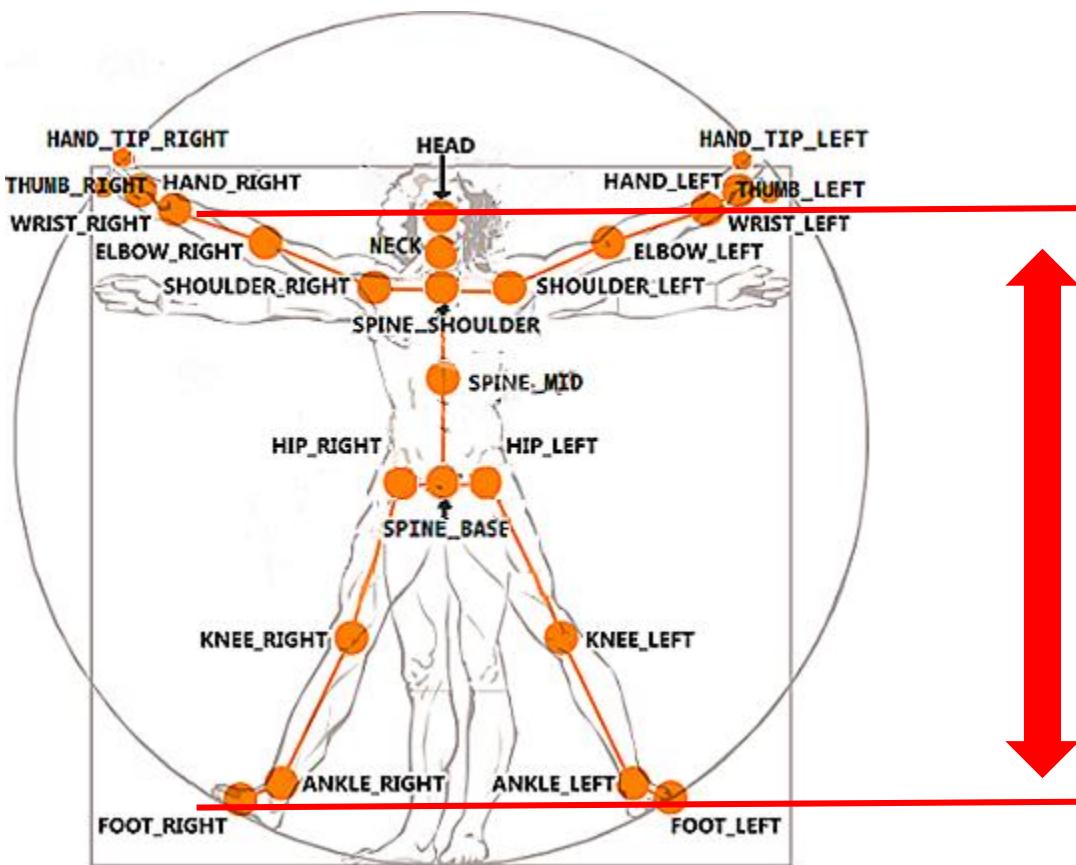
2. WritableBitmap が格納する形式 (Bgra) へ
データを統一しています。

3. データを更新します。このタイミングで画面
の表示が変更されます。

3. 身長を計測するWindowsストアアプリを作ります (90分)

1. Visual Studio のプロジェクトを作成します
2. 画面に RGB カメラからの出力を表示します
3. 骨格情報から身長を計測し画面に表示します
4. (Option) BodyIndex frame を使ってより正確に身長を計測します

身長計測の仕組み



Kinect が読み取る頭の座標と足の座標の
Y 座標の差異から身長を測定し表示します。

体の情報を保存する配列を宣言します

```
3 個の参照 | 作成者なし | 変更なし  
public sealed partial class MainPage : Page  
{  
    //Kinect センサー用の変数  
    private KinectSensor kinect = null;
```

```
//複数の frame (カラー、深さ、骨格) を取得する Reader のための変数  
private MultiSourceFrameReader multiReader = null;
```

```
/// <summary> ...  
private WriteableBitmap bitmap = null;
```

```
/// <summary> ...  
private Body[] bodies;
```

Kinect が読み取った体の情報を保存するため
の配列を宣言します。

体の情報を保存する配列を初期化します

```
public MainPage()
{
    //Kinect センサー オブジェクトを取得します。
    this.kinect = KinectSensor.GetDefault();

    //センサーが認識する体の総数分の配列を作成します。
    this.bodies = new Body[this.kinect.BodyFrameSource.BodyCount];

    //複数 frame 用の Reader を取得します。
    this.multiReader = this.kinect.OpenMultiSourceFrameReader(FrameSourceTypes.Color | FrameSourceTypes.Body);

    //frame 到着時のイベントハンドラを登録 (RGB、深度、骨格すべてを同じハンドラで処理します。)
    this.multiReader.MultiSourceFrameArrived += multiReader_MultiSourceFrameArrived;

    //取得する画像の情報形式を BGRA (RGB と透明度を表す Alpha) フォーマットとして定義。
    FrameDescription colorFrameDescription = this.kinect.ColorFrameSource.CreateFrameDescription(ColorImageFormat.Bgra);

    //表示用の bitmap データ作成
    this.bitmap = new WriteableBitmap(colorFrameDescription.Width, colorFrameDescription.Height);

    //Kinect センサーを起動します。
    this.kinect.Open();

    //画面上のコンポーネントを初期化します。
    this.InitializeComponent();
    theImage.Source = this.bitmap;
}
```

1. BodyCount は Kinect が読み取れる最大人数である 6 が設定されています。
そのため、この処理は Kinect で扱える体の最大数の配列を作成しています。

2. MultiSourceFrameReaader が骨格情報を読み取れるように設定しています。

骨格情報を読み取るための処理を加えます

```
void multiReader_MultiSourceFrameArrived(MultiSourceFrameReader sender, MultiSourceFrameArrivedEventArgs args)
{
    bool colorFrameProcessed = false;

    //複数 frame データへの参照を取得します。
    var reference = args.FrameReference.AcquireFrame();

    Color frame (に関する操作)

    #region 骨格情報を操作して身長を計測する
    using (BodyFrame bodyframe = reference.BodyFrameReference.AcquireFrame())
    {
        if(bodyframe != null)
        {
            //保持するデータを最新のものに更新する
            bodyframe.GetAndRefreshBodyData(this.bodies);
            //身長計測用のメソッドを呼び出します。
            this.MeasureHeight(bodyframe);
        }
    }
    endregion
}
```

1. 色の情報と同じようにフレームを取得します。

2. 体の情報の入る配列の情報を更新します。

3. MeasureHeight 関数は次のページ以降で定義します。

骨格情報を読み取るための処理を加えます

```
private void MeasureHeight(BodyFrame bodyframe)
{
    //頭の骨格情報
    Joint head_joint;
    //左足の骨格情報
    Joint footLeft_joint;
    //右足の骨格情報
    Joint footRight_joint;

    foreach (Body body in bodies)
    {
        if (body.IsTracked)
        {
            head_joint = body.Joints[JointType.Head];
            footLeft_joint = body.Joints[JointType.FootLeft];
            footRight_joint = body.Joints[JointType.FootRight];

            //もし各種ジョイントが取得できていない場合には、処理を中断します。
            if (head_joint.TrackingState != TrackingState.Tracked || footLeft_joint.TrackingState != TrackingState.Tracked || footRight_joint.TrackingState != TrackingState.Tracked)
                return;

            //Y 座標の小さいほうの足を取得する
            Joint foot_joint;
            if(footLeft_joint.Position.Y > footRight_joint.Position.Y)
            {
                foot_joint = footRight_joint;
            }
            else
            {
                foot_joint = footLeft_joint;
            }

            //身長を計測する
            double height = Math.Abs(head_joint.Position.Y - foot_joint.Position.Y)*100;

            //身長を表示します。
            this.height_textblock.Text = height.ToString("#0.0cm");

            //頭と足の位置をわかりやすく表示するメソッドを呼び出します
            this.DrawHeight(head_joint, foot_joint);

            //今回は、Kinect にトラックされている 1 名を対象に処理を実施します。
            break;
        }
    }
}
```

Measureheight の定義の全体像を記載します。
分量が多いいため、線で区切っている部分ごとに解説します。

Measureheight (1/4)

```
private void MeasureHeight(BodyFrame bodyframe)
{
    //頭の骨格情報
    Joint head_joint;
    //左足の骨格情報
    Joint footLeft_joint;
    //右足の骨格情報
    Joint footRight_joint;
```

頭と足の骨格情報を保存する変数を宣言します。

Measureheight (2/4)

1. 変数に入っている体の情報をひとつづつ処理します。

```
foreach (Body body in bodies)
{
    if (body.IsTracked)
        head_joint = body.Joints[JointType.Head];
        footLeft_joint = body.Joints[JointType.FootLeft];
        footRight_joint = body.Joints[JointType.FootRight];

    //もし各種ジョイントが取得できていない場合には、処理を中断します。
    if (head_joint.TrackingState != TrackingState.Tracked || footLeft_joint.TrackingState != TrackingState.Tracked || footRight_joint.TrackingState != TrackingState.Tracked)
        return;
}
```

2. 配列内には空の情報もあるので、体が Track されていると判定されたもののみ処理します

3. 障害物などで足・頭の座標が取れない場合には正確な値が取れないので、処理を中断します。

Measureheight (3/4)

```
//Y 座標の小さいほうの足を取得する
Joint foot_joint;
if(footLeft_joint.Position.Y > footRight_joint.Position.Y)
{
    foot_joint = footRight_joint;
}
else
{
    foot_joint = footLeft_joint;
}
```

最終的に頭と足の差分を取り出しますが、
比較するためによりY座標の低い足を取り出します。

Measureheight (4/4)

```
//身長を計測する  
double height = Math.Abs(head_joint.Position.Y - foot_joint.Position.Y)*100;
```

1. 骨格情報は double 型の m(メートル) 単位で出力されるので、cm に出力するために 100 をかけています。

```
//身長を表示します。  
this.height_textblock.Text = height.ToString("#0.0cm");
```

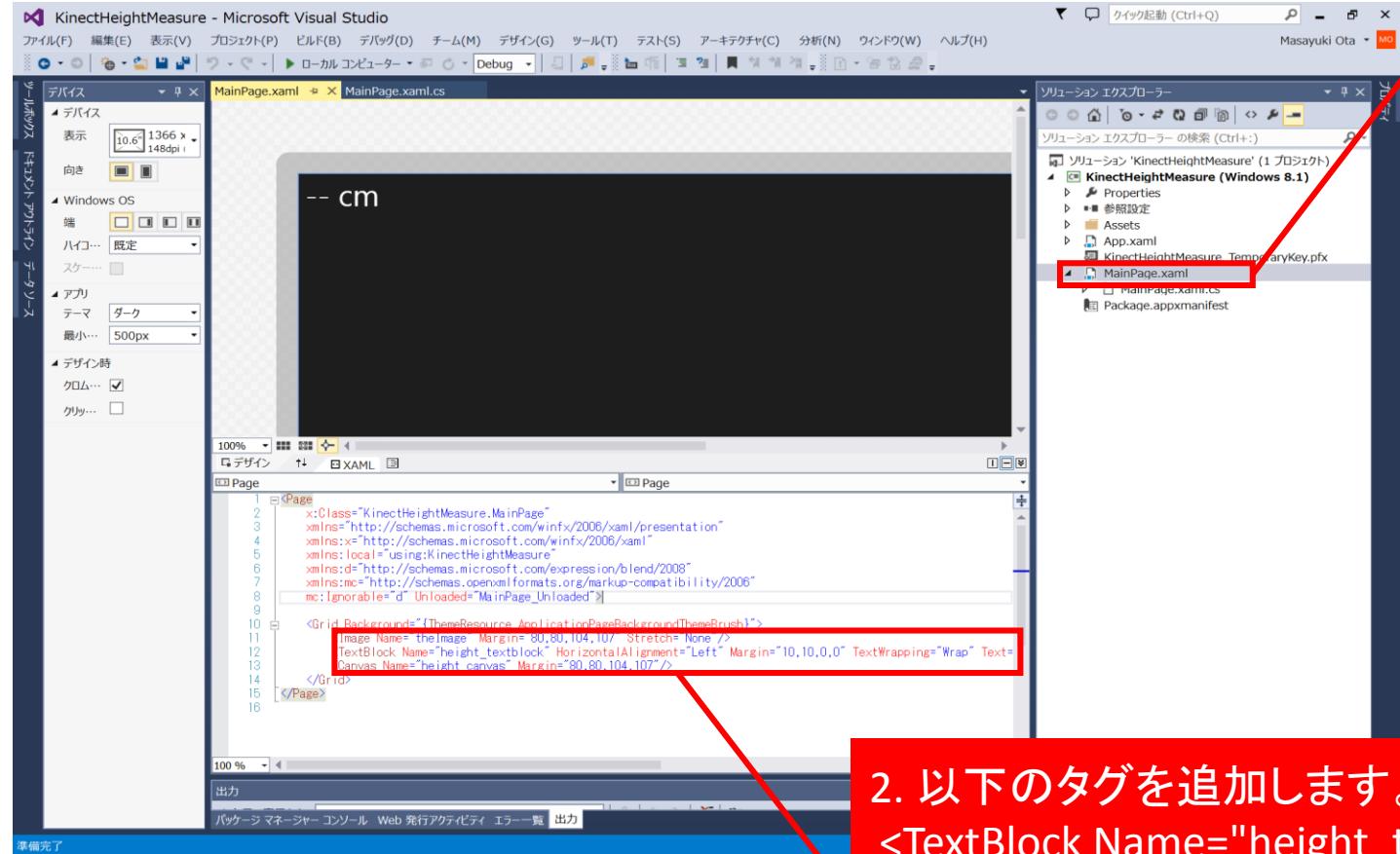
2. height_textblock および DrawHeight は次以降のスライドで説明します。

```
//頭と足の位置をわかりやすく表示するメソッドを呼び出します  
this.DrawHeight(head_joint, foot_joint);
```

```
//今回は、Kinect にトラックされている 1 名を対象に処理を実施します。  
break;
```

3. Foreach で複数の体情報を取りだしてしまいますが、最初に認識した体のみで計算したいので、あえて break しています。

身長情報を表示するための UI 追加



1. “ MainPage.xaml ” をダブル クリックし、
デザイン画面を開きます。

2. 以下のタグを追加します。
<TextBlock Name="height_textblock" HorizontalAlignment="Left"
Margin="10,10,0,0" TextWrapping="Wrap" Text="-- cm"
VerticalAlignment="Top" Height="65" Width="275" FontSize="48"/>
<Canvas Name="height_canvas" Margin="80,80,104,107"/>

身長情報を表示するための UI 追加

```
1 <Page  
2     x:Class="KinectHeightMeasure.MainPage"  
3     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
4     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
5     xmlns:local="using:KinectHeightMeasure"  
6     xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  
7     xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  
8     mc:Ignorable="d" Unloaded="MainPage_Unloaded">  
9  
10    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">  
11        <Image Name="theImage" Margin="80,80,104,107" Stretch="None"/>  
12        <TextBlock Name="height_textblock" HorizontalAlignment="Left" Margin="10,10,0,0" TextWrapping="Wrap" Text="-- cm" VerticalAlignment="Top" Height="65"  
13        <Canvas Name="height_canvas" Margin="80,80,104,107"/>  
14    </Grid>  
15 </Page>  
16
```

先ほどの変更で上記のようになります。TextBlock は画面上切れていますが、以下の値が記載されています。

```
<TextBlock Name="height_textblock" HorizontalAlignment="Left"  
Margin="10,10,0,0" TextWrapping="Wrap" Text="-- cm"  
VerticalAlignment="Top" Height="65" Width="275" FontSize="48"/>
```

計算した身長情報を表示します

```
private void DrawHeight(Joint head,Joint foot)
{
    //頭と足の骨格座標からイメージ座標へ座標を変換します。
    ColorSpacePoint head_point = kinect.CoordinateMapper.MapCameraPointToColorSpace(head.Position);
    ColorSpacePoint foot_point = kinect.CoordinateMapper.MapCameraPointToColorSpace(foot.Position);

    //四角形を生成する
    Rectangle rec = new Rectangle()
    {
        Width = 5,
        Height = foot_point.Y - head_point.Y,
        Fill = new SolidColorBrush(Colors.Red),
        Margin = new Thickness(head_point.X + 100, head_point.Y,0,0)
    };

    //キャンバスをクリアにする
    height_canvas.Children.Clear();

    //キャンバスに四角形を追加する。
    height_canvas.Children.Add(rec);

    //身長の表示を四角形の横に場所に移動させる。
    this.height_textblock.Margin = new Thickness(rec.Margin.Left + 25, rec.Margin.Top + rec.Height / 2, 0, 0);
}
```

1. DrawHeight メソッドを宣言します。

2. 骨格情報は 3 次元ですが、表示は 2 次元となります。2 次元では遠くの人が背が低く見えてしまうなど正確な値が取れなくなるので、そのような問題を回避するための座標変換の仕組みが CoordinateMapper という名称で用意されています。

今回は、MapCameraPointToColor という骨格情報などの 3 次元情報を 2 次元の RGB 表示にするための座標変換をしています。

3. 身長を計測するWindowsストアアプリを作ります (90分)

1. Visual Studio のプロジェクトを作成します
2. 画面に RGB カメラからの出力を表示します
3. 骨格情報から身長を計測し画面に表示します
4. (Option) BodyIndex frame を使ってより正確に身長を計測します

以下の URL にアクセスしソースコードをダウンロードしてください。また、ここまで終了した方は手を上げて大田までお知らせください。
URL : <http://1drv.ms/11AysWA>

サンプル プロジェクトについて

ダウンロードいただいた zip ファイル内には以下 3 つのフォルダーが含まれています。

■ KinectHeightMeasure_RGB

- 画面に RGB カメラからの出力を表示する部分までのサンプルです

■ KinectHeightMeasure_body

- 骨格情報をを利用して身長を測り、出力する部分までのサンプルです

■ KinectHeightMeasure_complete

- より正確に身長を測るサンプルです