Windows Presentaion Foundation 4.5 入門

# はじめに

## 本書の目的

2012年8月に.NET Framework 4.5がリリースされました。対応する開発環境としてVisual Studio 2012もリリースされ、Windows 8の時代に対応するアプリケーション開発の環境が整ってきています。Windows 8で動くアプリケーションには大別してWindows ストア アプリとデスクトップアプリケーションの2種類があります。Windows ストア アプリが注目されがちですが、デスクトップアプリケーションも従来と変わらず重要なファクターになります。今後は、デスクトップアプリケーションにもタッチ対応スクリーンへの対応や、拡大されたときの表示などに対応することが求められます。それに対応するためには従来のWindows FormよりもWindows Presentation Foundation(以下WPF)のほうが有利になります。

本書では、日本語としてまとまった情報がMSDN以外にあまりないWPF 4.5の現状を著者の自習も兼ねながらまとめることを目的としています。そのため、間違った情報を含んでいる可能性あるため、その際は以下の連絡先に連絡ください。

大田　一希  
 [k.ota.0130@gmail.com](mailto:k.ota.0130@gmail.com)

## 本書の対象者

本書は、以下のような方を意識して書いています。

* WPF 4.5の学習をしたい方
* C#についての基本的な知識については知っている方  
  具体的には以下のキーワードについて知っている方
  + LINQ
  + async, await
  + ラムダ式
* Visual Studio 2012の基本的な操作方法について理解しているかた
  + プロジェクトの新規作成やクラスなどの新規作成方法
  + 参照の追加方法
  + NuGetを使った参照の追加方法など

## 執筆環境

本書は、以下の環境で作成しています。

* Windows 8 Pro 64bit
* Visual Studio 2012 Ultimate  
  (おそらくExpress Editionでも同様に作成可能なものが主になると思います)

# WPFとは

MSDNのWPFの概要の章に以下のように説明されています。

WPFの概要より抜粋 http://msdn.microsoft.com/ja-jp/library/aa970268.aspx  
Windows Presentation Foundationとは、魅力的な外観のユーザーエクスペリエンスを持つWindows クライアント アプリケーションを作成するための次世代プレゼンテーション システムです。

.NET Framework 3.0から搭載されているWPFの概要をそのまま引き継いでいるためだと思いますが、次世代のプレゼンテーションシステムというよりは、現在求められようとしているデスクトップアプリケーションを開発するために必要な機能が含まれたプラットフォームという言い方のほうがしっくりくると思います。WPFが登場してからのGUIアプリケーションを作るプラットフォームがすべてWPFの考えを踏襲して作られていることからも、このことが伺えます。

* Silverlight  
  WPF/Eという名前で開発され、Silverlightという名前でリリースされた。WPFと同じプログラミングモデルで開発が可能。
* Silverlight for Windows Phone  
  SilverlightをWindows Phone向けにしたもの
* Windows Runtime  
  Windows ストア アプリ開発のためのプラットフォームで、WPFと同じプログラミングモデルで開発が行えるスタイルが提供されている。
* Windows Phone Runtime  
  Windows RuntimeとSilverlight for Windows Phoneを混ぜたような雰囲気を醸し出しているプラットフォーム。当然WPFと同じプログラミングモデルで開発が行える。

## WPFのプログラミングモデル

WPFで確立されたプログラミングモデルをベースに、最近のGUIアプリケーション開発のためのプラットフォームが作成されていることについて紹介しました。では、WPFのプログラミングモデルとはどのようなものか、ここで簡単に説明したいと思います。WPFのアプリケーションは、主に以下のような形で作成されます。(カッコ内は記述言語)

画面

UI

(XAML)

コードビハインド（C#）

ロジック

データモデル

Etc…

（C#）

データバインディング

メソッド呼び出し

XAML（ザムルと読みます）と、C#によって画面を作成し、それとロジックやデータモデルをデータバインディングやメソッド呼び出しによって連携させて１つのアプリケーションとして作成します。

## Hello world

ここでは、Visual Studio 2012を使って簡単なHello worldアプリケーションを作成して、WPFのアプリケーション作成の流れと、アプリケーションの構成要素がどんなものか実際に見て行こうと思います。

Visual Studio 2012でWPFアプリケーションを新規作成すると以下のような画面になります。



参照設定に、PresentationCoreとPresentationFrameworkとWindowsBaseとSystem.Xamlという4つが追加されています。この4つがWPFのクラスを含むアセンブリになります。そのほかに、App.xaml(xaml.csとペア)やMainWindow.xaml(xaml.csとペア)が作成されています。

### Appクラス

App.xamlは、以下のような内容のXAMLで書かれたファイルになります。XAMLは、WPFでは主にGUIを記述するための言語として使われますがApp.xamlでは、GUIではなくアプリケーション全体を制御するクラスを定義しています。App.xamlのコードを以下に示します。

<Application x:Class="HelloWorld.App"

             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

             StartupUri="MainWindow.xaml">

    <Application.Resources>

    </Application.Resources>

</Application>

見ていただければわかると思いますがXAMLはXMLをベースとして作られています。XML名前空間やXMLの開始タグや閉じタグがあります。x:Classは、このXAMLと対になるコードビハインドのクラスを表しています。HelloWorld.Appクラスは、App.xaml.csの中に以下のように定義されています。

using System;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Linq;

using System.Threading.Tasks;

using System.Windows;

namespace HelloWorld

{

    /// <summary>

    /// App.xaml の相互作用ロジック

    /// </summary>

    public partial class App : Application

    {

    }

}

初期状態では、何も定義されていません。このApp.xamlとApp.xaml.csは、コンパイル時に1つのクラスとして解釈されます。そのため、App.xamlで定義してあることと、App.xaml.csで記述したコードが1つAppクラスになります。Appクラスは、従来のアプリケーションでいうところのMainメソッドを持つエントリポイントのクラスになります。

App.xamlで重要な点は、StartupUri属性でMainWindow.xamlを指定している点です。StartupUriで指定したウィンドウを起動時に表示するようになっているため、このアプリケーションを実行するとMainWindow.xamlが表示されます。

### MainWindow.xaml

MainWindow.xamlは、Windowを定義したXAMLになります。

<Window x:Class="HelloWorld.MainWindow"

        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

        Title="MainWindow" Height="350" Width="525">

    <Grid>

    </Grid>

</Window>

Windowタグが、WPFのウィンドウを表します。そしてタイトルにMainWindowと設定してあり、高さと幅が350と525に設定されていることが確認できます。Windowの中にはGridというタグが定義されています。App.xamlと同様にx:Classという属性でコードビハインドのクラスが指定されています。HelloWorld.MainWindowクラスのコードを以下に示します。

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace HelloWorld

{

    /// <summary>

    /// MainWindow.xaml の相互作用ロジック

    /// </summary>

    public partial class MainWindow : Window

    {

        public MainWindow()

        {

            InitializeComponent();

        }

    }

}

コンストラクタで呼び出されているInitializeComponentメソッドは、XAMLで定義された情報を使用するために必須のメソッドです。このメソッドの呼び出しを忘れると、XAMLで定義した情報が使用できなくなるので気を付けてください。

### デザイナによる画面の設計

Visual Studio 2012には、WPFアプリケーションの画面デザインを行うためのデザイナがついています。主にツールボックスとデザイナとドキュメントアウトラインとプロパティを使用します。



WPFで使用できるコントロールの一覧です。

ドラッグアンドドロップでデザイナにコントロールを置くことができます。

デザイナのコントロールの親子関係をツリー状に表示します

画面デザインを行います

デザイナで選択した要素のプロパティを設定します

ツールボックスからボタンを画面にドラッグアンドドロップして、位置と大きさを調整して以下のように画面に配置します。



ボタンのプロパティを以下のように設定しています。

|  |  |
| --- | --- |
| プロパティ名 | 説明 |
| x:Name | コントロールをコードビハインドから利用するための変数名。プロパティの名前か、ドキュメント アウトラインで対象をダブルクリックすることで設定可能。  helloWorldButtonと設定しています。 |
| Content | 表示文字列。Hello worldと設定しています。 |

その他のHorizontalAlignmentプロパティなどはデザイナ上などで自動的に設定されたものなので、ここでは割愛します。

### イベントハンドラの追加とコードの記述

作成したボタンをダブルクリックすると、ボタンのクリックイベントが作成されます。プロパティのイベントからもWindows Formアプリと同じ要領でイベントを作成できます。helloWorldButton\_Clickというメソッドが作成されるので以下のようにメッセージボックスを表示するコードを追加してください。

public partial class MainWindow : Window

{

    public MainWindow()

    {

        InitializeComponent();

    }

    private void helloWorldButton\_Click(object sender, RoutedEventArgs e)

    {

        MessageBox.Show("Hello world");

    }

}

### コンパイルして実行

アプリケーションを実行すると、以下のようにボタンの置いてある画面が表示されます。ボタンを押すとメッセージボックスが表示されます。



### Mainメソッドはどこにいった？

Hello worldを作る手順の中でAppクラスがWPFにおけるMainメソッドを持つエントリポイントのようなクラスであるという説明を行いましたが、これについてもう少し詳しく説明したいと思います。App.xamlとApp.xaml.csがコンパイルされる際に、以下のようなコードがコンパイラによって生成されます。このコードを見るには、ソリューションエクスプローラですべてのファイルを表示するように設定して「obj→Debug→App.g.cs」というコードを開きます。



namespace HelloWorld {

/// <summary>

/// App

/// </summary>

public partial class App : System.Windows.Application {

    /// <summary>

    /// InitializeComponent

    /// </summary>

    [System.Diagnostics.DebuggerNonUserCodeAttribute()]

    [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]

    public void InitializeComponent() {

        #line 4 "..\..\App.xaml"

        this.StartupUri = new System.Uri("MainWindow.xaml", System.UriKind.Relative);

        #line default

        #line hidden

    }

    /// <summary>

    /// Application Entry Point.

    /// </summary>

    [System.STAThreadAttribute()]

    [System.Diagnostics.DebuggerNonUserCodeAttribute()]

    [System.CodeDom.Compiler.GeneratedCodeAttribute("PresentationBuildTasks", "4.0.0.0")]

    public static void Main() {

        HelloWorld.App app = new HelloWorld.App();

        app.InitializeComponent();

        app.Run();

    }

}

このように、WPFアプリケーションではMainメソッドはコンパイラによって生成されています。

## 全てC#でHello world

XAMLとC#を使ってHello worldアプリケーションを作成しました。ここでは、このHello worldアプリケーションをC#のみで作成します。通常は、画面はXAMLで記述しますしXAMLで記述することを推奨します。ただ、XAMLで書けることは、ほぼ全てC#で記述できます。

C#でクラス ライブラリのプロジェクトを新規作成します。ここではCodeHelloWorldという名前で作成しました。参照設定にWPFで必要な以下の4つの参照を追加します。

* PresentationCore
* PresentationFramework
* WindowsBase
* System.Xaml

MainWindowという名前のクラスを作成して、Windowクラスを継承させます。

namespace CodeHelloWorld

{

    using System.Windows;

    class MainWindow : Window

    {

    }

}

Hello worldアプリケーションで作成した画面をコードで組み立てます。InitializeComponentというメソッド内でWindow内のコントロールを組み立てています。基本的にXAMLで設定している内容と1対1に対応していることが確認できます。

namespace CodeHelloWorld

{

    using System.Windows;

    using System.Windows.Controls;

    class MainWindow : Window

    {

        private Button helloWorldButton;

        private void InitializeComponent()

        {

            // Windowのプロパティの設定

            this.Title = "MainWindow";

            this.Height = 350;

            this.Width = 525;

            // Buttonの作成

            this.helloWorldButton = new Button

            {

                Content = "Hello world",

                HorizontalAlignment = HorizontalAlignment.Left,

                VerticalAlignment = VerticalAlignment.Top,

                Margin = new Thickness(10, 10, 0, 0),

                Width = 100

            };

            this.helloWorldButton.Click += helloWorldButton\_Click;

            // Gridの作成

            var grid = new Grid();

            grid.Children.Add(this.helloWorldButton);

            // gridをWindowに設定

            this.Content = grid;

        }

        public MainWindow()

        {

            this.InitializeComponent();

        }

        private void helloWorldButton\_Click(object sender, RoutedEventArgs e)

        {

            MessageBox.Show("Hello world");

        }

    }

}

MainWindowクラスが出来たのでAppクラスを作成します。XAMLを使ったAppクラスではStartupUriで開始時に表示するWindowのXAMLのURIを指定していましたが、ここではXAMLを使っていないのでMainWindowの表示をAppクラスのStartupイベントで明示的に行っています。

namespace CodeHelloWorld

{

    using System;

    using System.Windows;

    class App : Application

    {

        private void InitializeComponent()

        {

            // StartupUriは使えないのでStartupイベントを使う

            this.Startup += App\_Startup;

        }

        private void App\_Startup(object sender, StartupEventArgs e)

        {

            // ウィンドウを作成して表示させるコードを明示的に書く

            var w = new MainWindow();

            w.Show();

        }

        [STAThread]

        public static void Main(string[] args)

        {

            // Appクラスを作成して初期化して実行

            var app = new App();

            app.InitializeComponent();

            app.Run();

        }

    }

}

一通りのクラスが出来たので、プロジェクトのプロパティをクラスライブラリからWindowsアプリケーションに変更して実行します。実行すると、XAMLを使った時とおなじ見た目と動作のアプリケーションが起動します。



ここで伝えたかったのは、C#のコードでもXAMLでも同じようにWPFのアプリケーションが作れるという点です。そのことを知ったうえで、XAMLの特性を理解し、どういうときにXAMLで記述し、どういうときにC#で記述すべきなのかということを考えることが必要だということを意識して今後を読み進めてください。

## WPFを構成するものを考えてみる

XAMLとC#で作成するケースと、C#だけで作成するケースのHello worldを2つ作成しました。その過程で説明しましたが、XAMLで記述できることは、ほぼ全てC#でも記述できます。その理由についてWPFの構成するものを交えて説明します。

### WPFはクラスライブラリ？

WPF = クラスライブラリというのは言い過ぎかもしれませんが、WPFはPresentationFrameworkとPresentationCoreとWindowsBaseの3つのアセンブリに入っているクラスから構成されています。WPFは、通常のCLRのクラスと同じように親をたどっていくと最終的にSystem.Objectにたどり着くクラス群から構成されています。その中に、ボタンを表すButtonクラスやウィンドウを表すWindowクラスやボタンなどのコントロールの表示位置を決めるGridクラスなど様々なものが含まれています。基本的に、これらのクラスをインスタンス化してプロパティを設定して繋いでいくことで、WPFの画面は作成できます。

### XAMLは何？

WPFの画面を全てC#で記述できるということは、XAMLは不要なのでは？という疑問がわいてきます。確かにXAMLが無くてもWPFアプリケーションの作成は出来ます。XAMLは、C#で記述するよりもオブジェクトのプロパティの設定や複雑なオブジェクトの組み立てを宣言的に記述できるという点でC#より優れています。

XAMLは、オブジェクトのインスタンス化という領域に特化したドメイン固有言語という見方が出来ます。画面の構築は、基本的に画面を構成するオブジェクトのインスタンス化が主な仕事になります。C#で記述したHello worldの画面構築のコードをもう一度以下に示します。

private void InitializeComponent()

{

    // Windowのプロパティの設定

    this.Title = "MainWindow";

    this.Height = 350;

    this.Width = 525;

    // Buttonの作成

    this.helloWorldButton = new Button

    {

        Content = "Hello world",

        HorizontalAlignment = HorizontalAlignment.Left,

        VerticalAlignment = VerticalAlignment.Top,

        Margin = new Thickness(10, 10, 0, 0),

        Width = 100

    };

    this.helloWorldButton.Click += helloWorldButton\_Click;

    // Gridの作成

    var grid = new Grid();

    grid.Children.Add(this.helloWorldButton);

    // gridをWindowに設定

    this.Content = grid;

}

プロパティの設定とオブジェクトの組み立てしか行っていません。同じ画面を構築するためのXAMLのコードをもう一度以下に示します。

<Window x:Class="HelloWorld.MainWindow"

        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

        Title="MainWindow" Height="350" Width="525">

    <Grid>

        <Button x:Name="helloWorldButton" Content="Hello world" HorizontalAlignment="Left" Margin="10,10,0,0" VerticalAlignment="Top" Width="100" Click="helloWorldButton\_Click"/>

    </Grid>

</Window>

Windowの下にGridがありGridの下にButtonがあるというオブジェクトの構造を端的に記述できているのはどちらになるでしょうか？個人的な主観になりますが私はXAMLに軍配があがると思います。もし、画面構築に複雑な計算ロジックが含まれるようなケース（この場合も大体はWPFに用意されているレイアウトの仕組みでカバーできることが多いです）の場合はC#で記述したほうが有利かもしれません。

### WPFを構成するもののまとめ

ということで、WPFは以下のものから構成されていることがわかりました。

* 膨大な数のクラス群
* 画面を構築するためのXAML

XAMLは、オブジェクトを組み立てるためのものなので、どのようなクラスがWPFにあるのかを理解し、それをXAMLで組み立てるという考えで理解していくといいと思います。

.NET Frameworkのクラス群

WPFのクラス群  
Button, Window, Grid, etc…

記述言語

XAML  
宣言的（オブジェクトのインスタンス化に特化）

C#  
汎用的

相互補完

## XAML

## WPFを構成するクラス群

XAML、分離コード、Applicationクラス、Windowクラス、XBAP、ナビゲーション  
http://msdn.microsoft.com/ja-jp/library/aa970268(v=vs.110).aspx

# WPF入門

## DispatcherObject

## DependencyObject

## プロパティシステム

## データバインド

### 単純なデータバインド

### コレクションのデータバインド

INotifyCollectionChanged

### 入力値の検証

## イベント

## レイアウトシステム

## コンテンツモデル

http://msdn.microsoft.com/ja-jp/library/bb613548.aspx

## 代表的なコントロール

## リソース

## スタイル

## テンプレート

### DataTemplate

### 階層構造を扱うテンプレート

### ControlTemplate

### DataTemplateSelector

## Visual State Manager

# 応用

## データバインディングを前提としたプログラミングモデル