```
using System;
using System.Windows;
public class App {
  [STAThread]
  public static void Main() {
    Application app = new Application();
    Window win = Window();
    win.Content = “Hello WPF Code”;
    app.Run(win);
  }
}
```

```
<Window
    xmlns="http://schemas.microsoft.com/
    winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com
    /winfx/2006/xaml"
    >
    Hello XAML
</Window>
```

Windows Vista 世代のユーザインターフェイス ～開発編～

株式会社アークウェイ
www.archway.co.jp
代表取締役社長　.NET コンサルタント
森屋英治(Hideharu Moriya)
Microsoft MVP Solutions Architect

Agenda

- 次世代UI開発WPFのコンセプト
- WPFの構成要素
- DAL　スタイル
  - 開発スタイル
  - アーキテクチャスタイル
  - 言語(XAML)スタイル
- 特徴的な機能
  - スタイル
  - リソース
  - データバインド
- まとめ
- Appendix

Innovation

or

Commodity

Enterprise App 2.0 = Experience + App

New Market

次世代UI開発のコンセプト

- 技術的な統合(これまでの様々な技術の統合）
  - 2D　　{GDI, GDI+}
  - UI　　{User32 , WinForm}
  - Media {DirectShow}
  - 3D　　{Direct3D,OpenGL}
- ベクトルグラフィック
  - すべてDirect3D経由でレンダリングされる
  - GPUのフル活用
- 宣言型プログラミングを可能に
  - XAML(zamel)　UIデザインをXMLで記述可能に
- イノベーション
  - エンタープライズアプリケーションの進化
  - 3Dアプリケーションへの挑戦

WPFの構成要素

DAL スタイル

Development Style

Architecture Style

Language Style

開発スタイル

開発スタイル

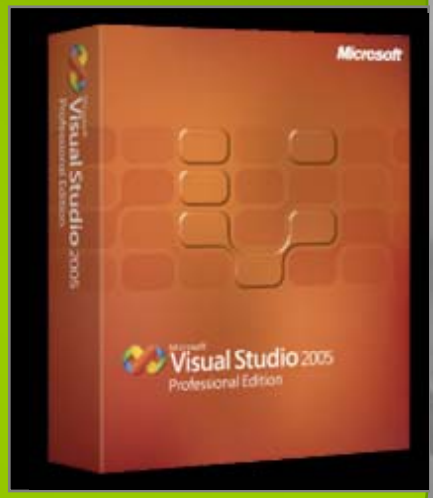
Designer

Developer

Pap
JPG /
PS
PP

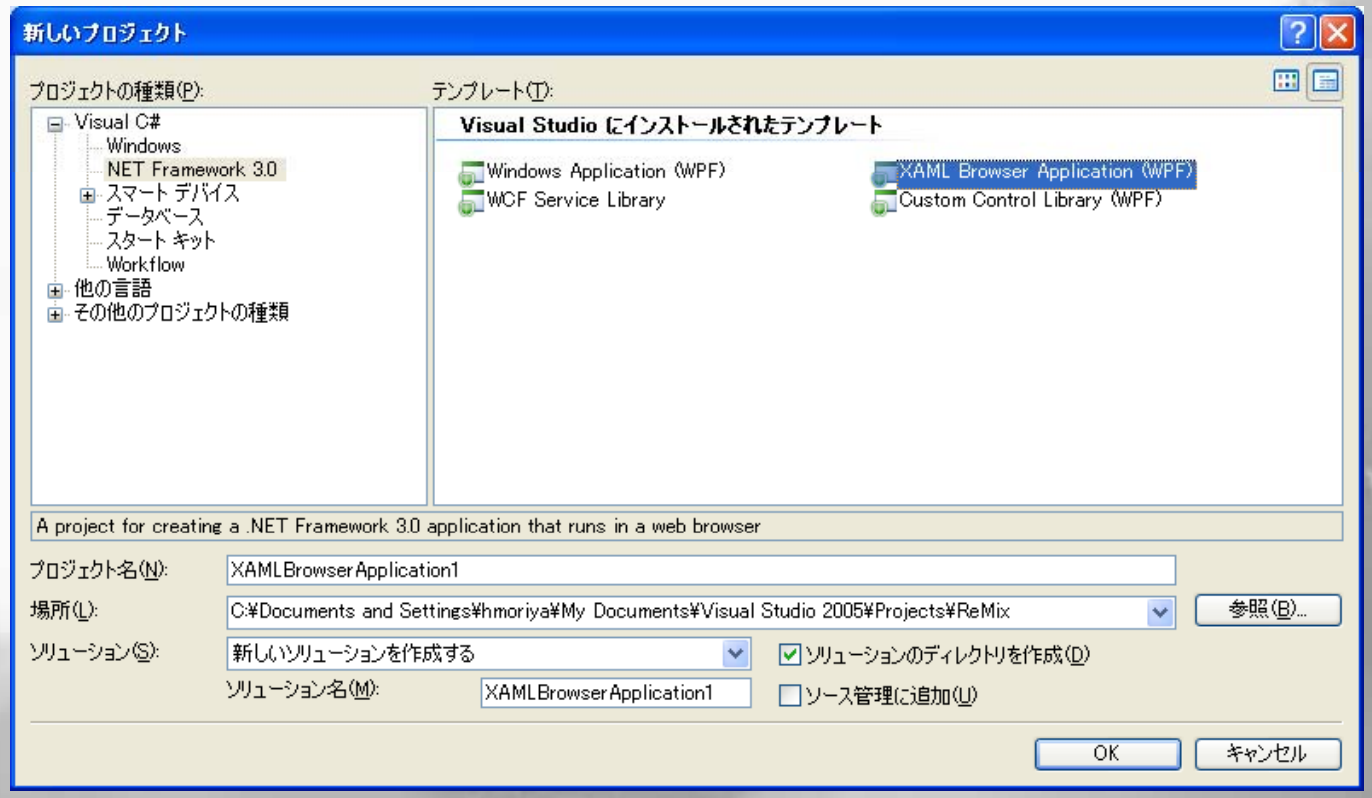
XAML

C++
C
.NET



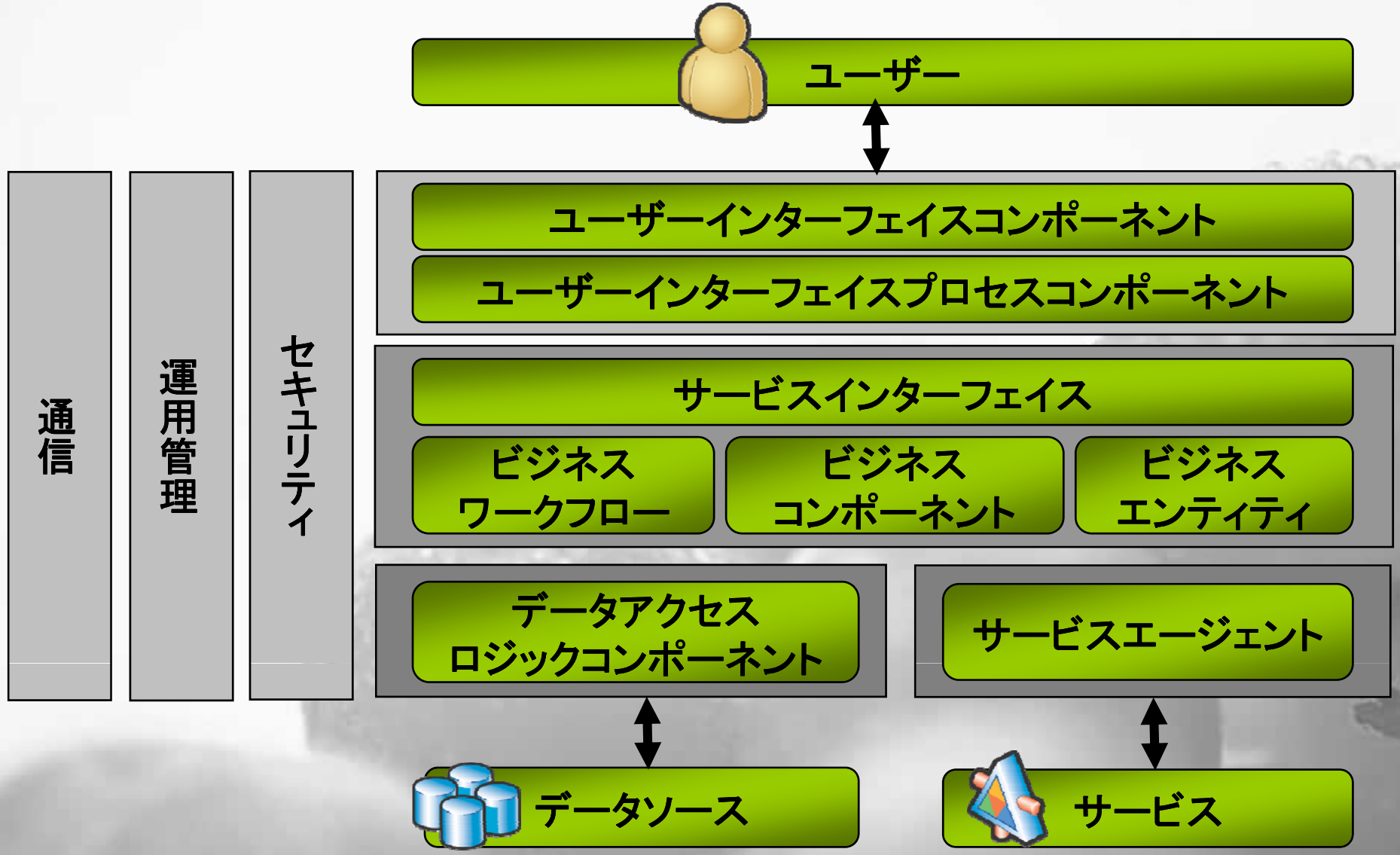
プロジェクトタイプ

- Windows Application (WPF)
- XAML Brower Application (WPF)

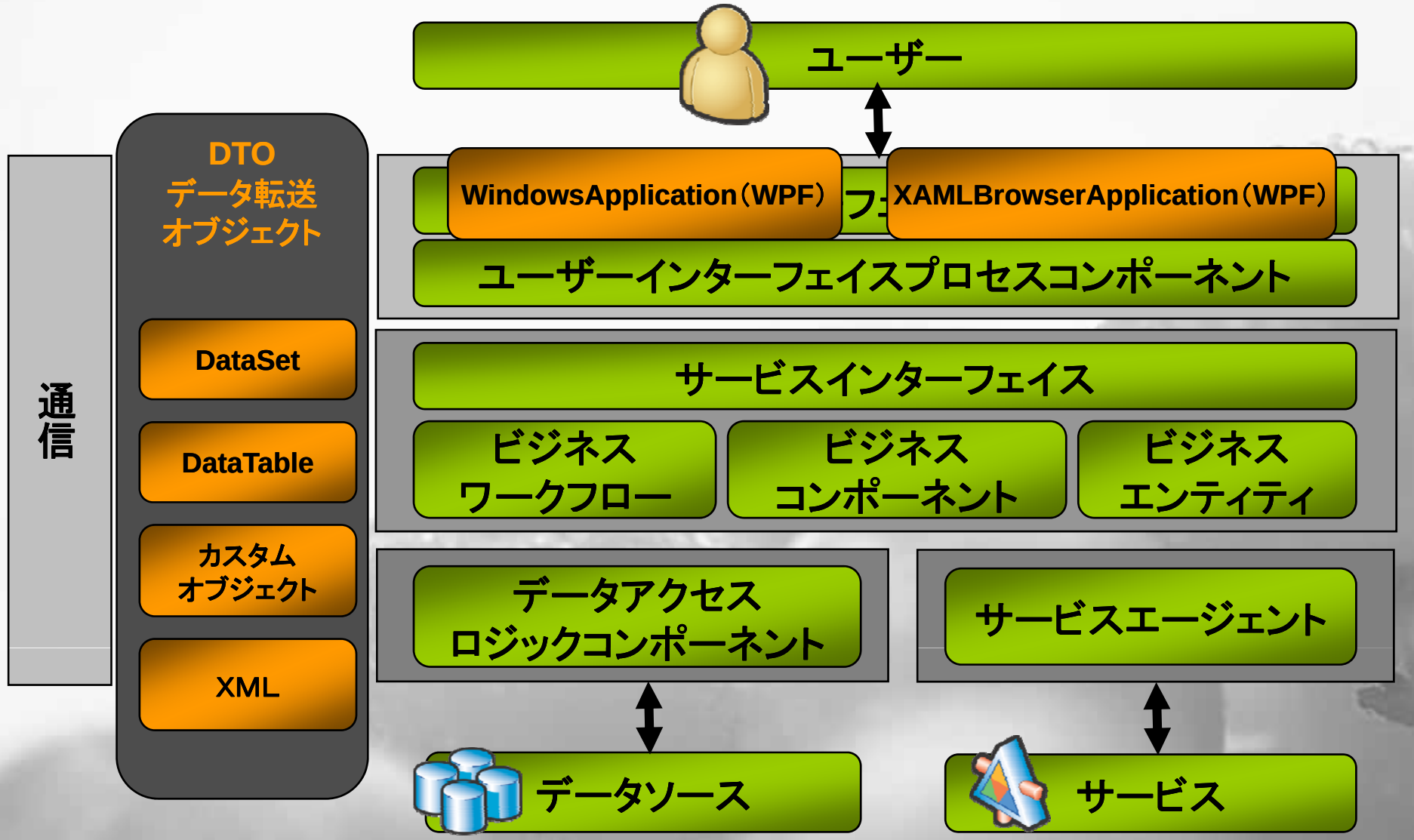


アーキテクチャスタイル

Application Architecture For .Net



Application Architecture For .Net With WPF



アーキテクチャの要素

- アーキテクチャの決定は、まず必要な要素（関心）を選択する。
- 非機能要件のトレードオフ
  - パフォーマンス、セキュリティ、配置条件
- 要素が多ければ多いほど開発コストは増大し、配置、運用コストも増大する。
- 対象となる開発で必要な分だけの要素を選択する。
  - ユーザインターフェイス
  - データアクセス
- シンプルなアーキテクチャを採用

アーキテクチャの傾向と対策

- エンタープライズアプリケーションは、ほとんどデータベースからの情報取得、更新

- データバインド可能なオブジェクトの選択
- データストアからの取り出しの容易性
  - データセット
  - データテーブル
- 注意点：なるべく分散させない
  - 分散トランザクション、分散オブジェクト
  - 本当に必要なときのみ

言語(XAML)スタイル

WPF開発言語

- .NETに対応している言語すべて
  - C#
  - VB.NET
  - IronPython
  - ...
  - そして、XAML(eXtensible Application Markup Language)

XAML

- XAML(eXtensible Application Markup Language)
  - HTMLの開発体験をWinFormに！
- 言語としてのXAML
  - コンパイルされて動作
  - 言語の範囲は、マークアップの範囲に依存し、すべて対応しているわけではない
  - パーシャルクラスの理解
  - WF（Workflow Foundation）にも適用されている

Design

<XAML>

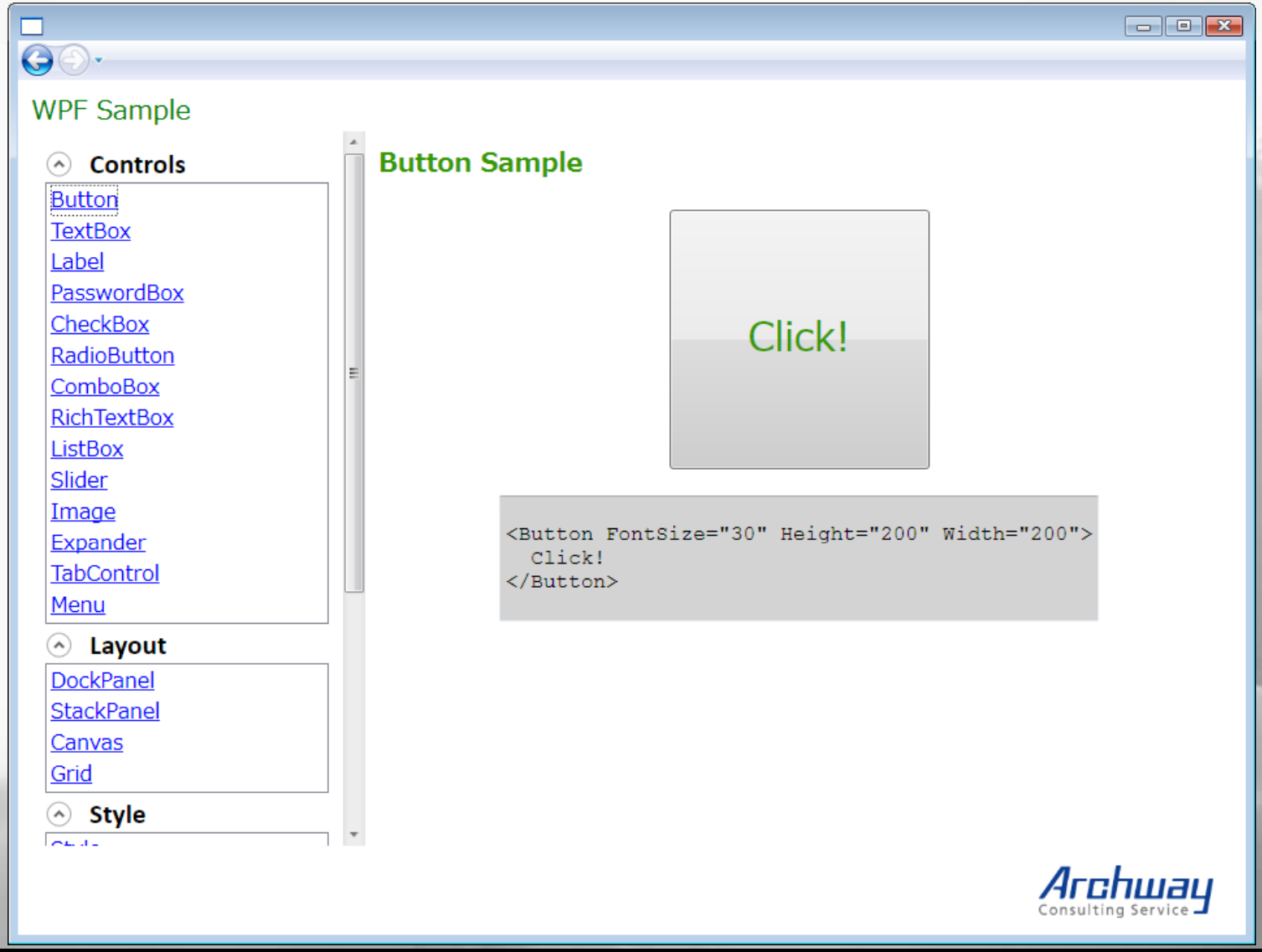
Code

XAML & C#Code

```
<Button
    Foreground=“Green”
    FontSize=“20pt”>
   XAML Button
</Button>

Button button = new Button();
Button.Foreground = Brushes.Green;
Button.FontSize = 20;
Button.Content = “Code Button”;
```

System.Windows.Control
様々なUIコントロール



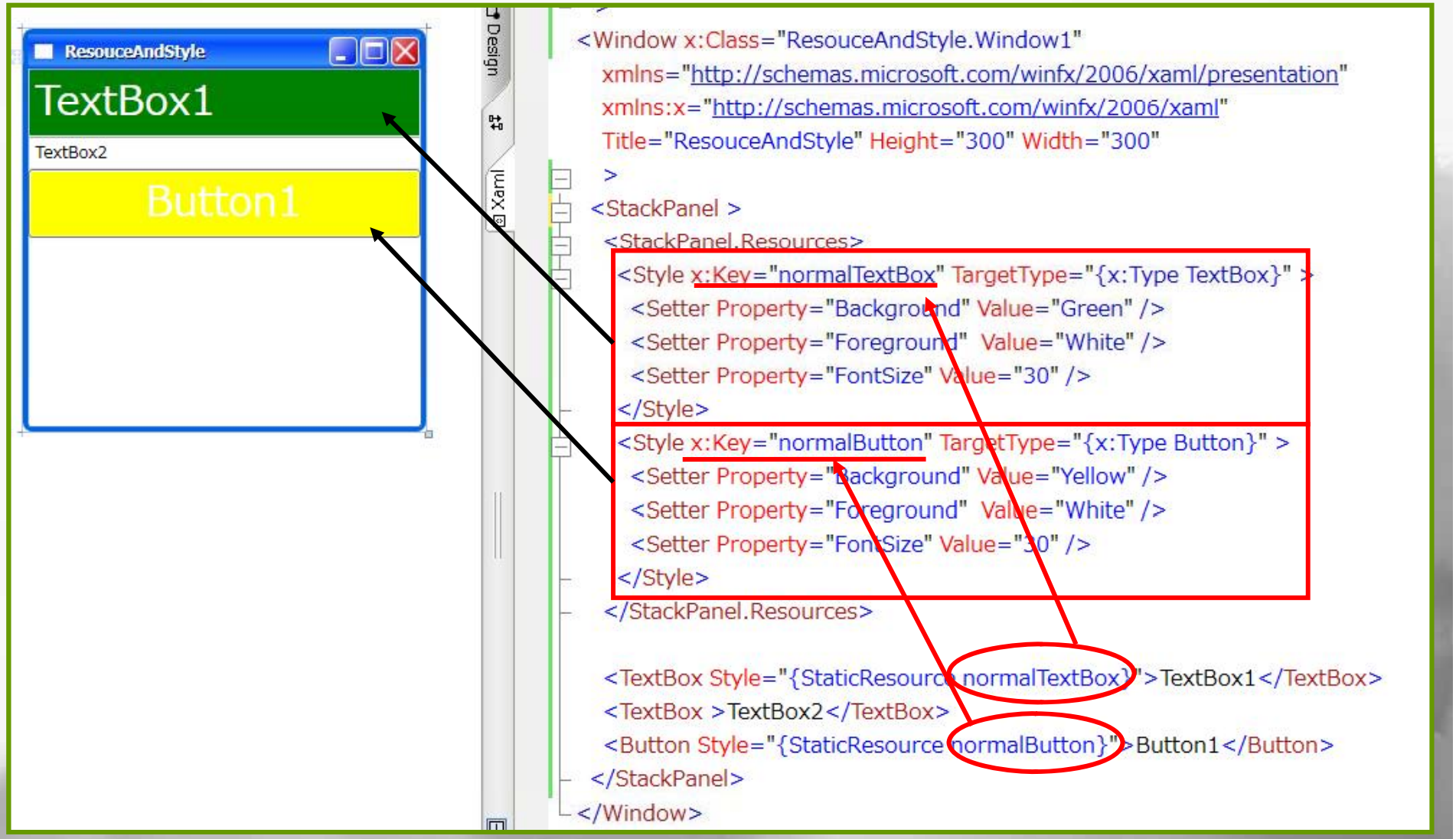
スタイル

スタイル

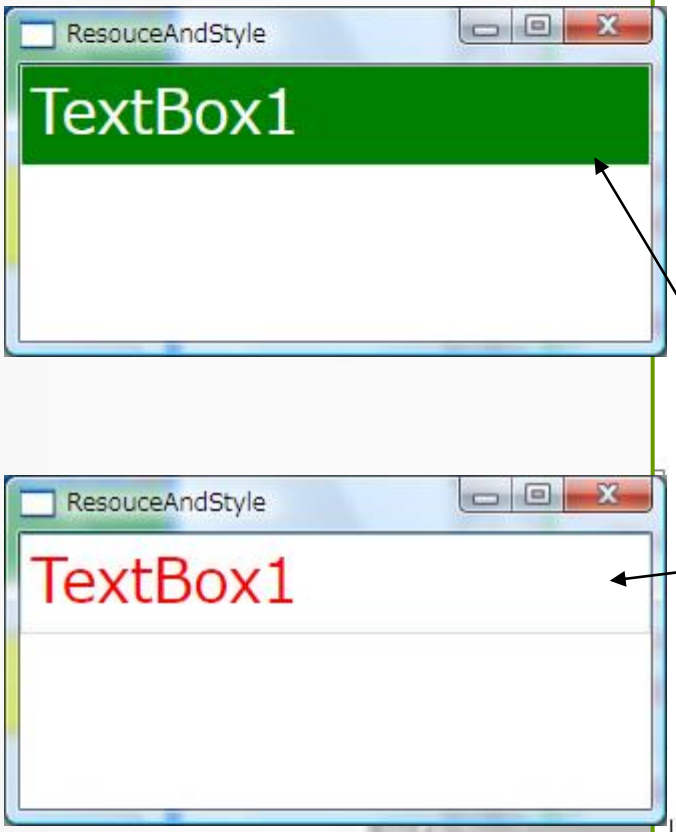
- 概念は、HTMLのスタイルシートと同様だが、WPFにおける“スタイル”は、リソース、スタイル、テンプレート、トリガ、ストーリボードを統合する広い概念の用語。
- リソースに配置して利用。

```
<Style …>
        <Setter … />
   <EventSetter … />
</Style>
```

スタイルの設定



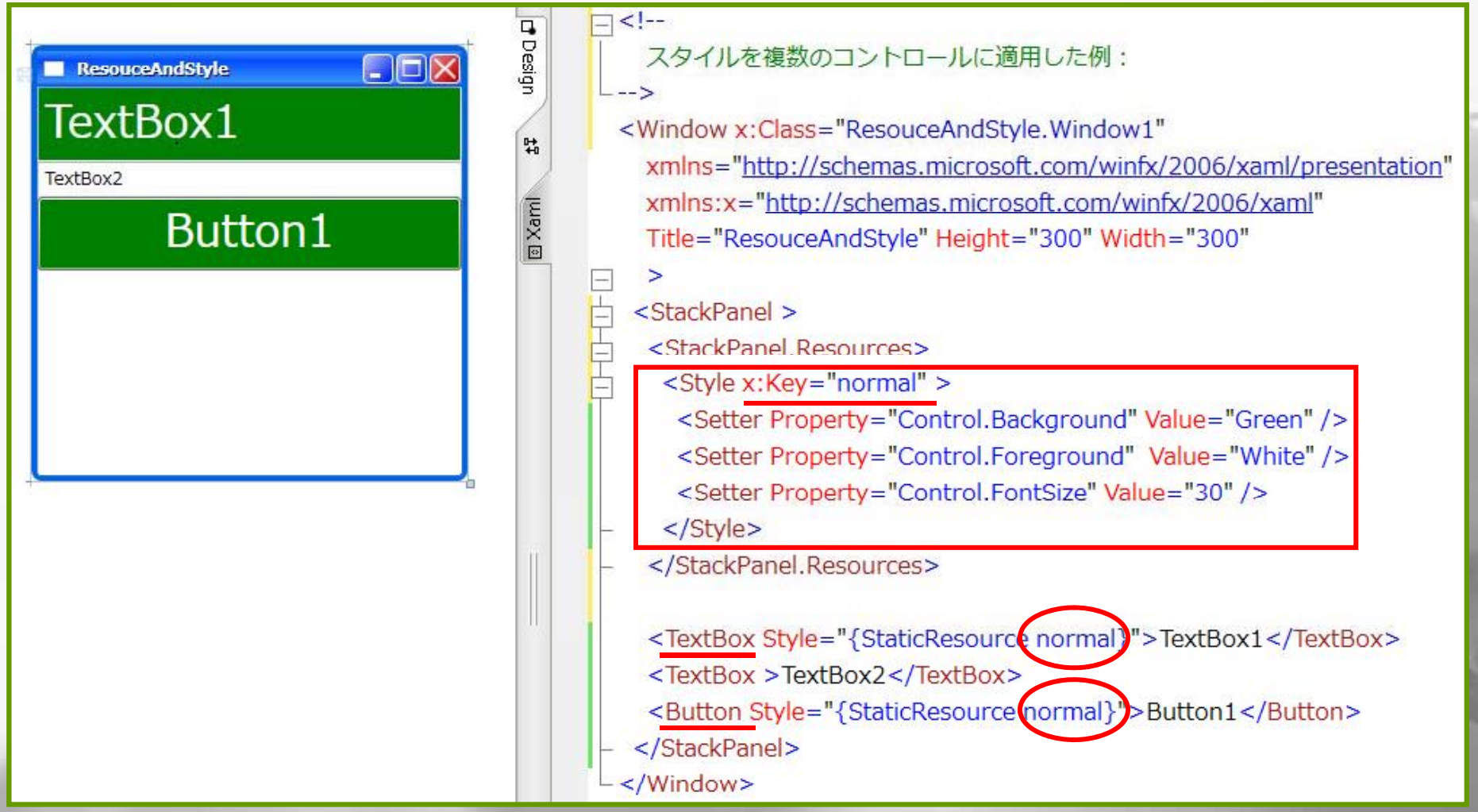
トリガーの利用




```
<Window x:Class="ResouceAndStyle.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="ResouceAndStyle" Height="300" Width="300"
  >
 <StackPanel >
  <StackPanel.Resources>
   <Style x:Key="normalTextBox" TargetType="{x:Type TextBox}" >
    <Setter Property="Background" Value="Green" />
    <Setter Property="Foreground"  Value="White" />
    <Setter Property="FontSize" Value="30" />
    <Style.Triggers>
     <Trigger Property="IsMouseOver" Value="True">
      <Setter Property="Background" Value="White" />
      <Setter Property="Foreground"  Value="Red" />
     </Trigger>
    </Style.Triggers>
   </Style>
  </StackPanel.Resources>

  <TextBox Style="{StaticResource normalTextBox}">TextBox1</TextBox>
 </StackPanel>
</Window>
```

複数コントロールへの適用



リソース

リソース

- ローカルで利用するリソース。
  - Visual Studioで一般的に使われるリソースと若干異なる
- ResourceDictionaryを型とするキーとオブジェクトからなるリソースディクショナリ
- XAMLでは、すべての要素、Applicationなどに配置可能
- ResouceDictionaryをXAMLとして別ファイルに分離することも可能

Applicationリソースの利用

```
<StackPanel >
 <StackPanel.Resources>
  <Style x:Key="normal" >
   <Setter Property="Control.Background" Value="Green" />
   <Setter Property="Control.Foreground"  Value="White" />
   <Setter Property="Control.FontSize" Value="30" />
  </Style>
 </StackPanel.Resources>

 <TextBox Style="{StaticResource normal}">TextBox1</TextBox>
 <TextBox >TextBox2</TextBox>
 <Button Style="{StaticResource appStyle}">Button1</Button>
</StackPanel>
```

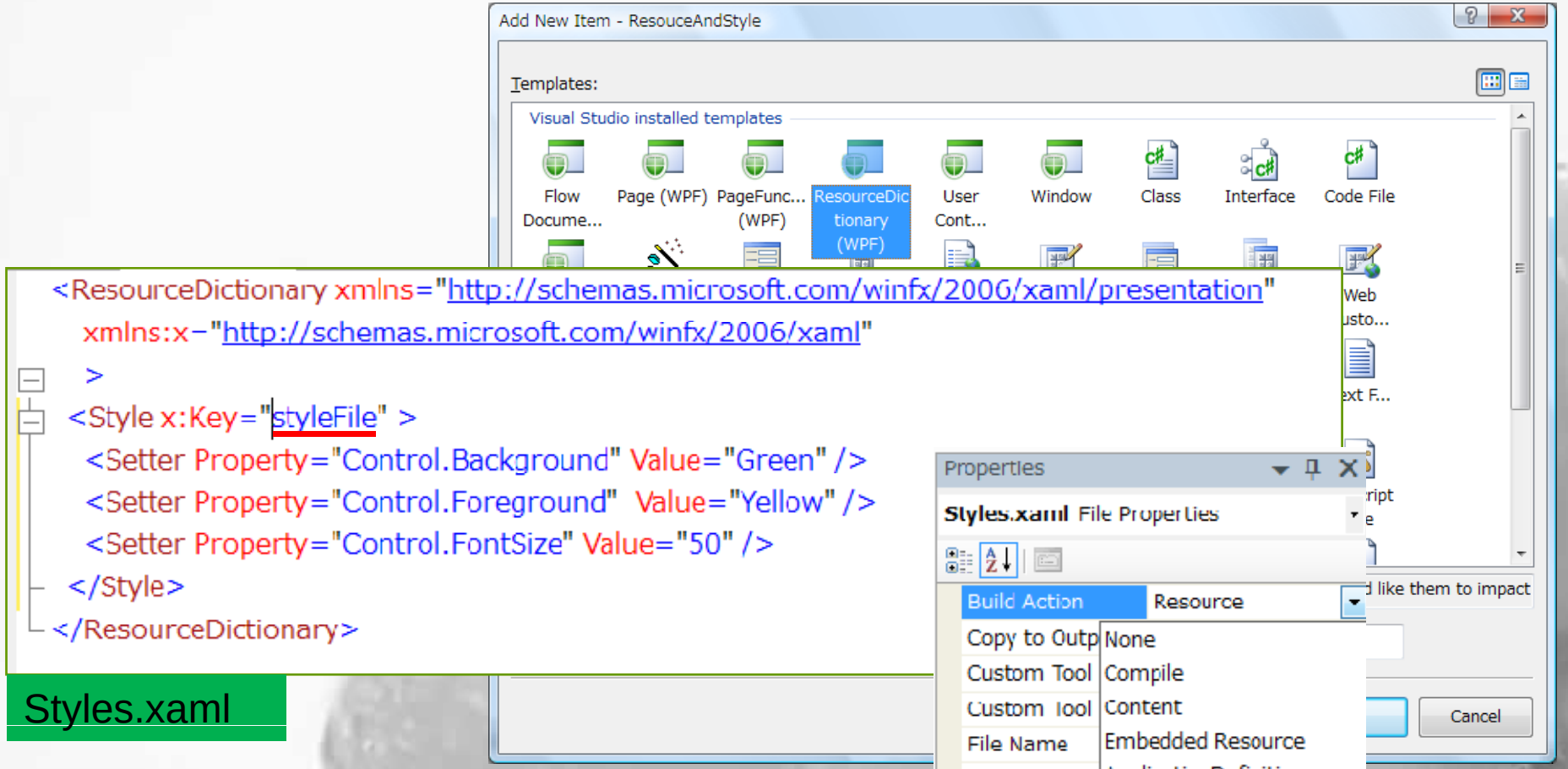
Window1.xaml

App.xaml

```
<Application x:Class="ResouceAndStyle.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  StartupUri="Window1.xaml"
  >
  <Application.Resources>
   <Style x:Key="appStyle" >
    <Setter Property="Control.Background" Value="Green" />
    <Setter Property="Control.Foreground"  Value="Yellow" />
    <Setter Property="Control.FontSize" Value="50" />
   </Style>
  </Application.Resources>
</Application>
```

様々な要素Window,Panel,Control、Applicationにリソースを配置できる

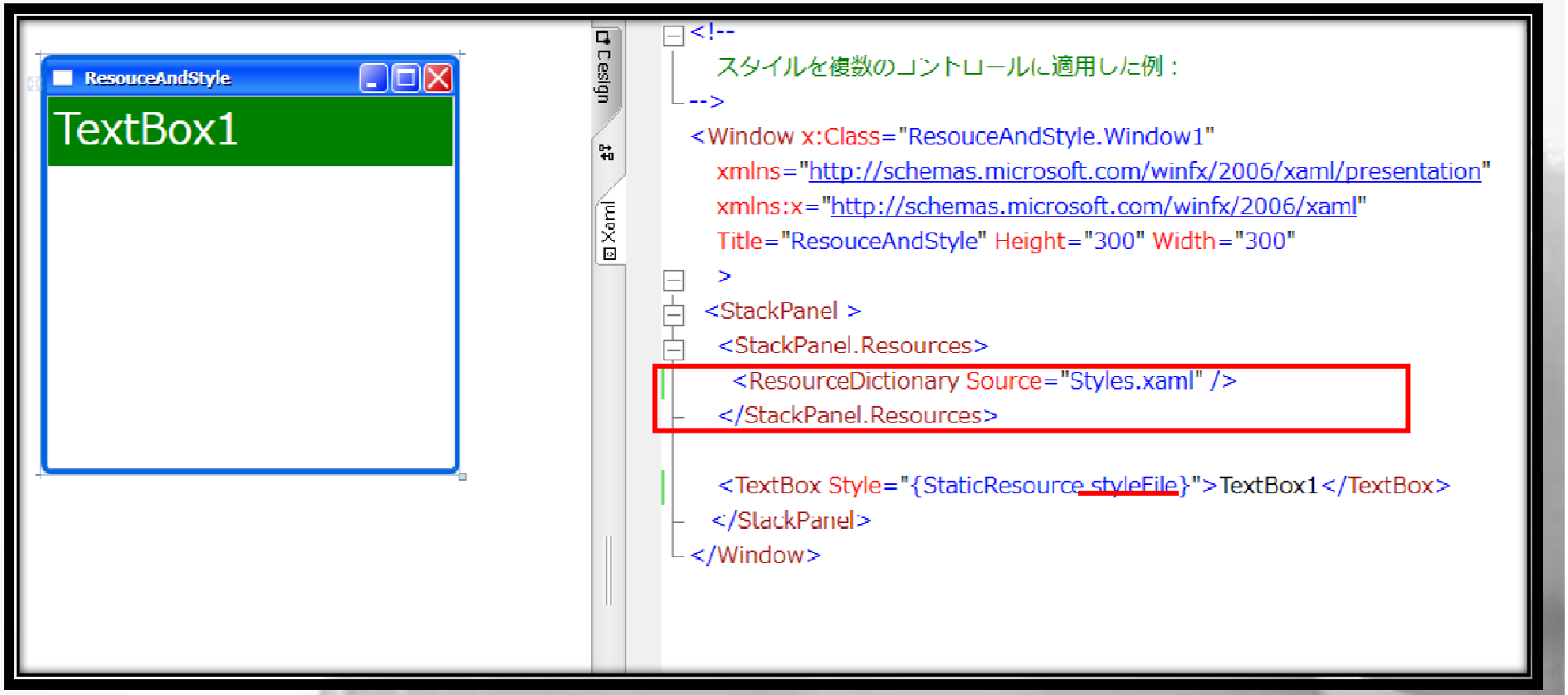
リソースディクショナリの作成




```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    >
  <Style x:Key="styleFile" >
    <Setter Property="Control.Background" Value="Green" />
    <Setter Property="Control.Foreground"  Value="Yellow" />
    <Setter Property="Control.FontSize" Value="50" />
  </Style>
</ResourceDictionary>
```

Styles.xaml

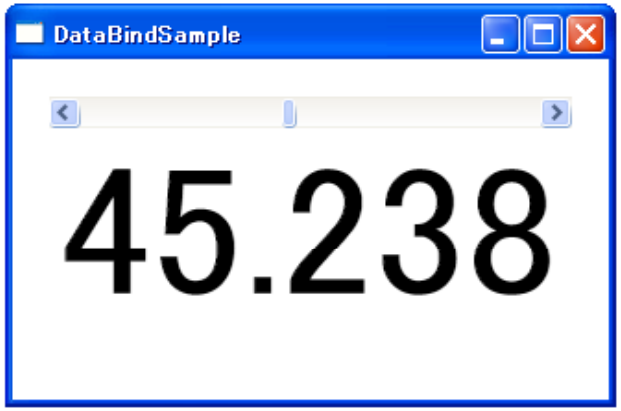
リソースディクショナリの参照



データバインド

単純なバインド

```
<Window x:Class="DataBindSample.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="DataBindSample" Height="300" Width="300"
   FontSize="100"
   >
   <StackPanel Margin="20">
     <ScrollBar Name="bar"
                Orientation="Horizontal"
                Maximum="100" />
     <Label Content="{Binding ElementName=bar , Path=Value}" />
   </StackPanel>
</Window>
```



XAML

Code

```
Binding bind = new Binding();
bind.Source = bar;
bind.Path = new PropertyPath(ScrollBar.ValueProperty);
label.SetBinding(Label.ContentProperty, bind);
```

Binding Mode

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation">
<Canvas>
  <TextBox Name="theTextBox" Text="Hello" />

  <TextBox Canvas.Top="25"
  Text="{Binding ElementName=theTextBox, Path=Text, Mode=TwoWay}" />
</Canvas>
</Window>
```

バインディングモード	説明
TwoWay	バインド先のコントロールまたはバインドのソースからの変更を、双方向でもう一方に移動する (これが既定モード)。
OneWay	ソースからの変更だけを、コントロールに移動します。ソースで変更が発生すると、バインド先のコントロールのデータが変更される。
OneTime	起動時にのみデータがバインドされ、コントロールに初めてデータが書き込まれた後は、ソースの変更は無視される。

バインドのタイミングの制御

- バインドによって変更がプッシュされるタイミングも指定できる。UpdateSourceTrigger タイプを指定することにより、バインドが特定のタイミングでしか変更を行わないように指定できる
- UpdateSourceTrigger プロパティは、ソースを変更によって更新するタイミングを指定する。
- Mode=TwoWayのバインド (既定) と併用する場合にのみ有効

```
{Binding ElementName=theTextBox, Path=Text, UpdateSourceTrigger=LostFocus}
```

UpdateSourceTrigger	説明
Explicit	BindingExpression.UpdateSource メソッドを明示的に呼び出した場合のみ、ソースが更新される。
LostFocus	バインドされたコントロールがフォーカスを失ったとき、ソースが更新される。
PropertyChanged	プロパティが変更されるたびに、変更がソースに更新されます。これが既定の動作。

XAMLのデータバインド

- ほとんどのアプリケーションでは、XMLやオブジェクト、データーセットなどにバインドしている
- これまでの.NETのデータバインドでは、バインディングの形態が限定されていて完全なオブジェクトとの連携や拡張が困難だった
- WPFでは上記の問題にチャレンジしている
- 現在対応しているデータプロバイダ
  - XmlDataProvider
    - XMLとのバインド
  - ObjectDataProvider
    - .NETオブジェクトとのバインド

XMLDataProvider

```
<StackPanel>
<StackPanel.Resources>
 <XmlDataProvider x:Key="FavoriteColors">
  <x:XData>
   <Colors xmlns="">
    <Color>Blue</Color>
    <Color>Black</Color>
   </Colors>
  </x:XData>
 </XmlDataProvider>
</StackPanel.Resources>
<TextBlock HorizontalAlignment="Center“ FontWeight="Bold">
XML Example </TextBlock>
<ListBox Width="200" Height="300"
      ItemsSource="{Binding Source={StaticResource
    FavoriteColors},
      XPath=/Colors/Color}"></ListBox>
</StackPanel>
```

XMLDataProvider
RSSとのバインド

```
<Window x:Class="DataBindSample.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="DataBindSample" Height="300" Width="300"
   FontSize="20"

   >
  <StackPanel Margin="20">
   <StackPanel.Resources>
    <XmlDataProvider x:Key="MyRSS"
     Source="http://hmoriya.spaces.live.com/feed.rss"/>
   </StackPanel.Resources>
   <TextBlock HorizontalAlignment="Center" FontWeight="Bold">
    hmoriya .NET Style
   </TextBlock>
   <ListBox
    ItemsSource=
      "{Binding Source={StaticResource MyRSS}, XPath=//item/title}">
   </ListBox>
  </StackPanel>
</Window>
```

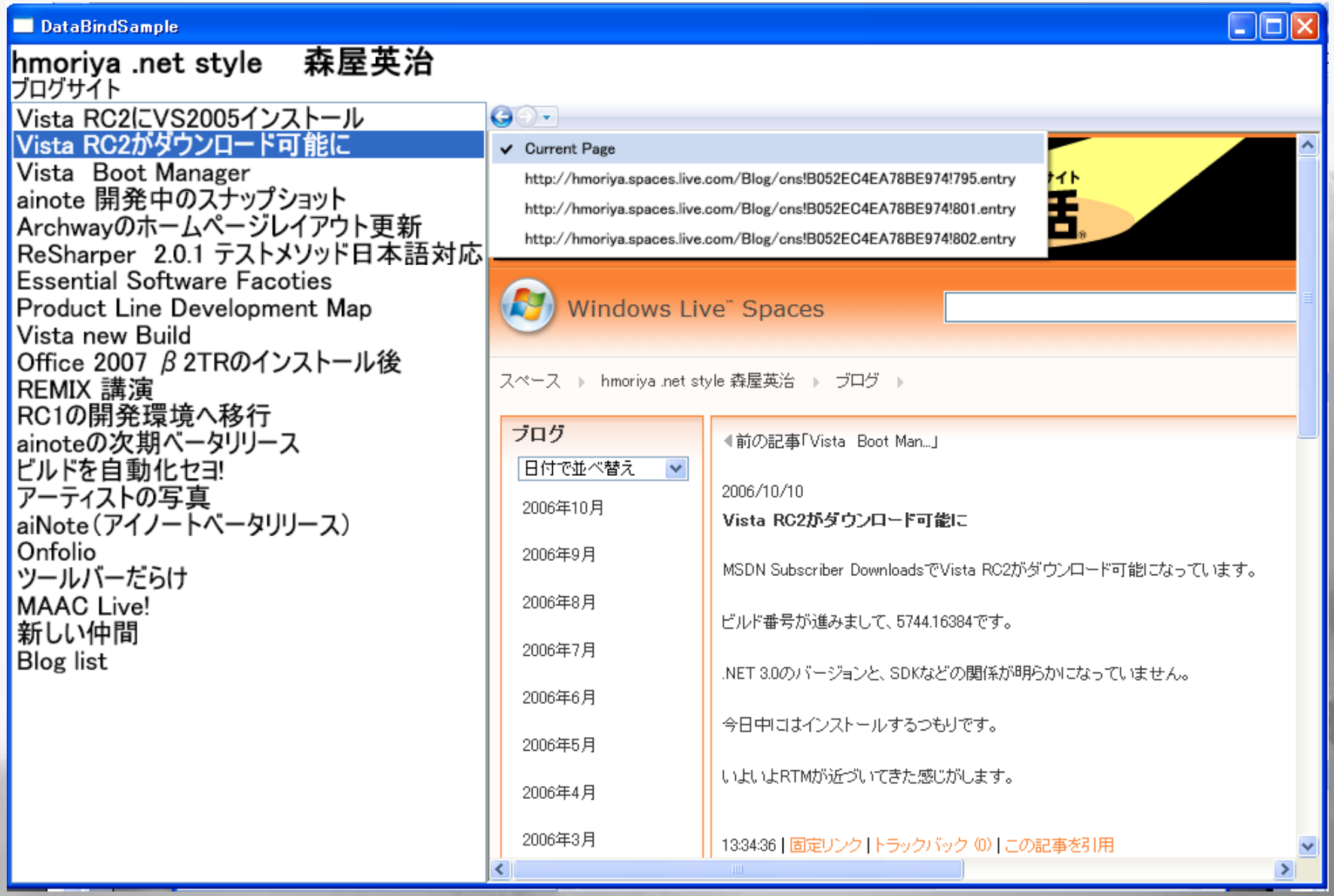
URLあるいは、ファイルパスも可

DetaContext/DataTemplate

```
<Window x:Class="DataBindSample.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="DataBindSample" Height="
   FontSize="20">
  <DockPanel>
   <DockPanel.Resources>
    <DataTemplate x:Key="TitleTemplate">
     <TextBlock Text="{Binding XPath=title}"/>
    </DataTemplate>
    <XmlDataProvider x:Key="MyRSS"
     Source="http://hmoriya.spaces.live.com/feed.rss"/>
   </DockPanel.Resources>
   <DockPanel
    DataContext="{Binding Source={StaticResource MyRSS},
          XPath=/rss/channel/item}">
    <ListBox ItemsSource= "{Binding}"
          ItemTemplate="{StaticResource TitleTemplate}"
          DockPanel.Dock="Left"
          IsSynchronizedWithCurrentItem="True"/>
    <Frame Source="{Binding XPath=link}" Width="Auto" />
   </DockPanel>
  </DockPanel>
</Window>
```

DetaContextを利用するとSourceを設定する必要がなくなる

RSSリーダ



ObjectDataProvider

```
<Window x:Class="ObjectDataBindingSample.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="ObjectDataBindingSample" Height="300" Width="300"
   Loaded="OnLoadWindow"
   FontSize="20"
   >
  <Window.Resources>
   <ObjectDataProvider x:Key="personData" />
  </Window.Resources>
  <StackPanel DataContext="{StaticResource personData}">
   <TextBlock>
    <TextBlock Width="100" Text="First Name:" />
    <TextBox Width="200" Text="{Binding Path=FirstName,Mode=OneWay}" />
   </TextBlock>
   <TextBlock>
    <TextBlock Width="100" Text="Last Name" />
    <TextBox Width="200" Text="{Binding Path=LastName,Mode=OneWay}" />
   </TextBlock>
  </StackPanel>
</Window>
```

ObjectDataProvider

Window.xaml.cs

```
public void OnLoadWindow(object  s,RoutedEventArgs arg)
{
   Person person =new Person("Hideharu","Moriya");
   ObjectDataProvider provider
      = (ObjectDataProvider)FindResource("personData");
   provider.ObjectInstance = person;
}
```

```
public class Person
{
   private string _firstName;
   private string _lastName;
   public  Person(string firstName,string lastName)
   {
      _firstName = firstName;
      _lastName = lastName;
   }
   public string FirstName { get { return _firstName;} }
   public string LastName { get { return _lastName; } }
   public override string ToString()
   {
      return string.Format("Person Name:{0},{1}", _firstName, _lastName);
   }
}
```

まとめ

- コンセプト
- WPF構成要素
- DALスタイル
  - 開発スタイル
  - アーキテクチャ
  - 言語(XAML)
- 特徴的な機能
  - スタイル
  - リソース
  - データバインド

Innovation

or

Commodity

Enjoy "WPF"

ご静聴
ありがとう
ございました

株式会社アークウェイ

森屋英治

Appendix

XAML省略した表記

```
<Canvas>
   <TextBox Name="theTextBox" Text="This is TextBox" />
   <TextBlock Canvas.Top="25"
            Text="{Binding ElementName=theTextBox,Path=Text}" />
</Canvas>
```

Elementを{}の中に属性、値を,でつなげていく省略した表記。

先ほどの表記

```
<Canvas>
   <TextBox Name="theTextBox" Text="This is TextBox" />
   <TextBlock Canvas.Top="25">
     <TextBlock.Text>
       <Binding ElementName="theTextBox" Path="Text" />
     </TextBlock.Text>
   </TextBlock>
 </Canvas>
```