Introduction to Building XAML UIs

DEVELOPMENTOR

DEVELOPING PEOPLE WHO DEVELOP SOFTWARE

Introduction

- **WPF/SL genesis was to build the next generation UI technology**
  - dynamic and interactive user interfaces

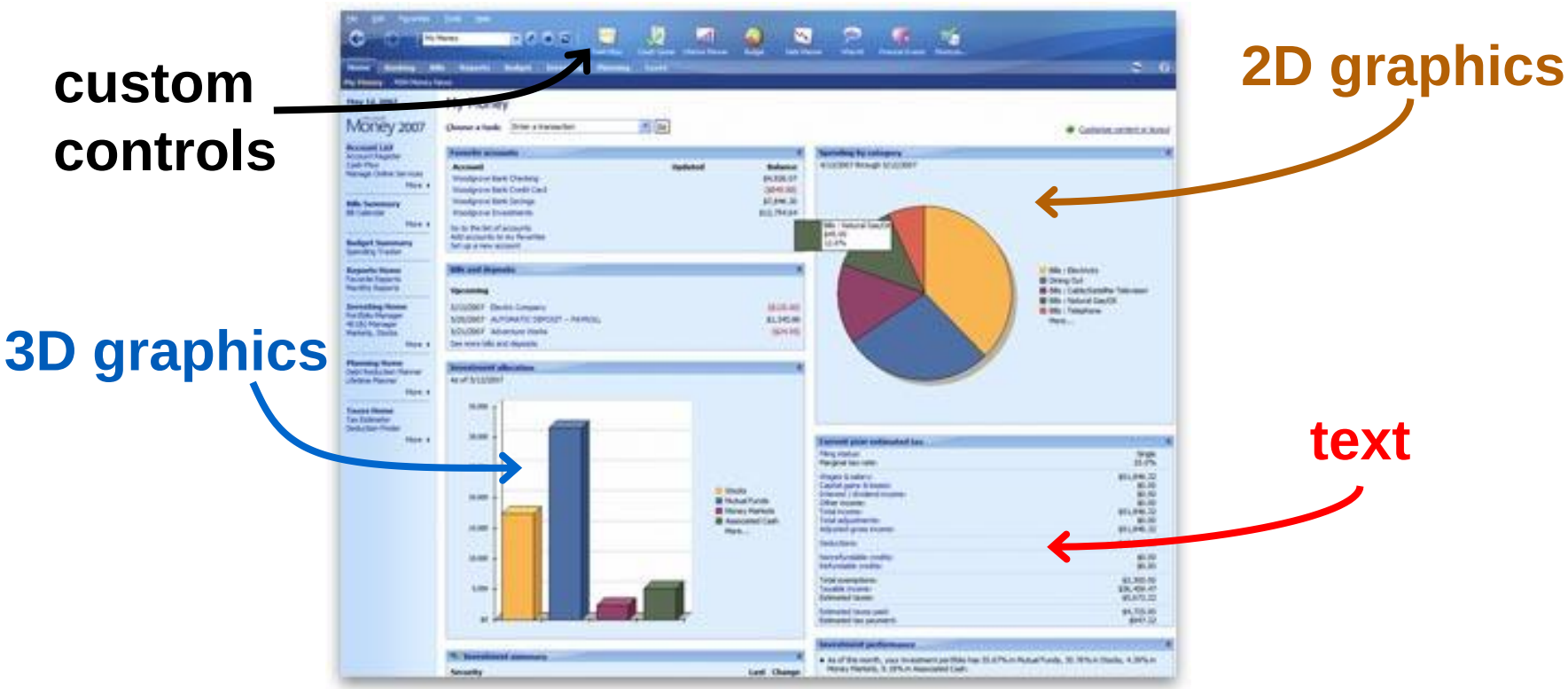
"In 2001 a new team was formed in Microsoft with a simple sounding mission – build a unified presentation platform that could eventually replace User32/GDI32, VB6, MSHTML, and Windows Forms.

The goal being to produce a best of breed platform that could really be a quantum leap forward."

-- Chris Anderson, WPF Architect

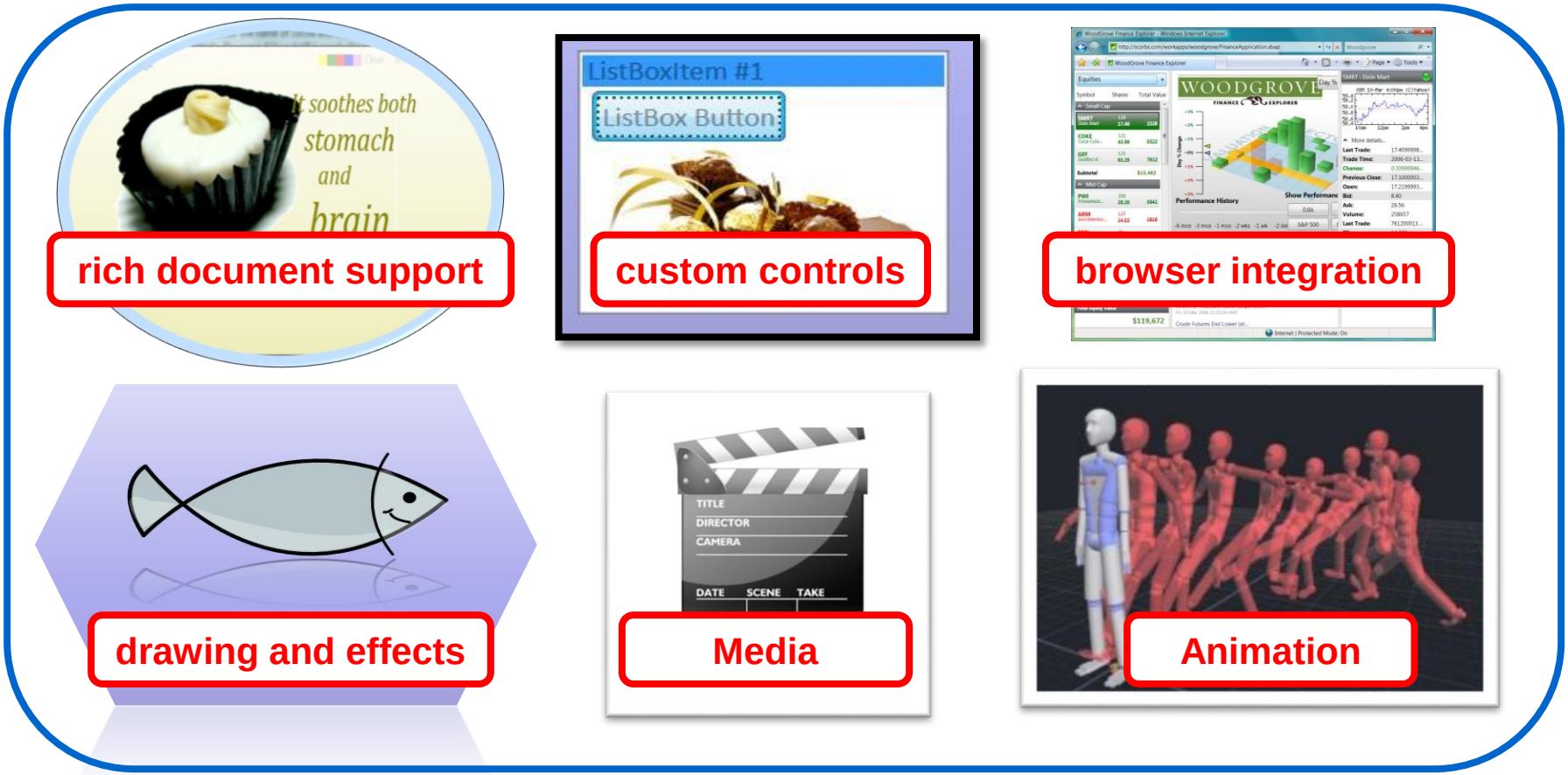
Motivation [multiple skillsets]

- **Modern applications use a mix of technologies**
  - each requires extensive time to master



Features of XAML based applications

- **WPF/SL provide broad integration of technologies**
  - single, consistent framework exposes all features

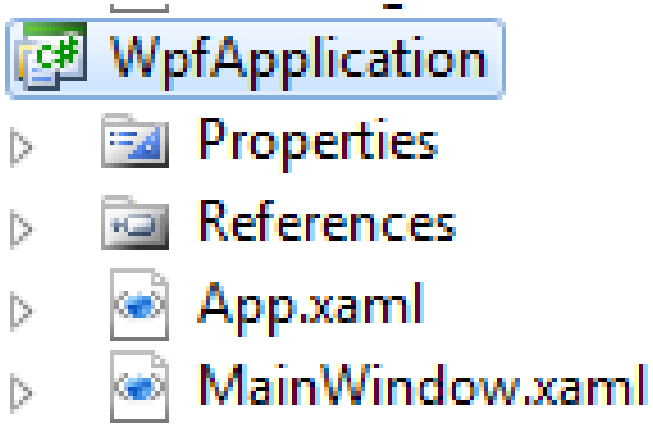


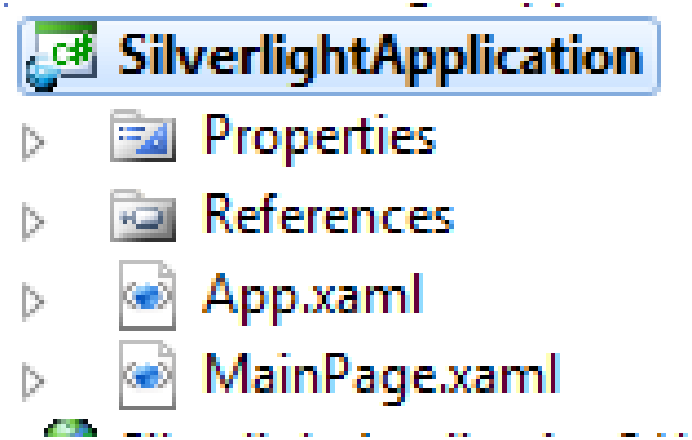
Pick your favorite flavor

- **Technology forms the core of every new Microsoft UI initiative today**

Project structure

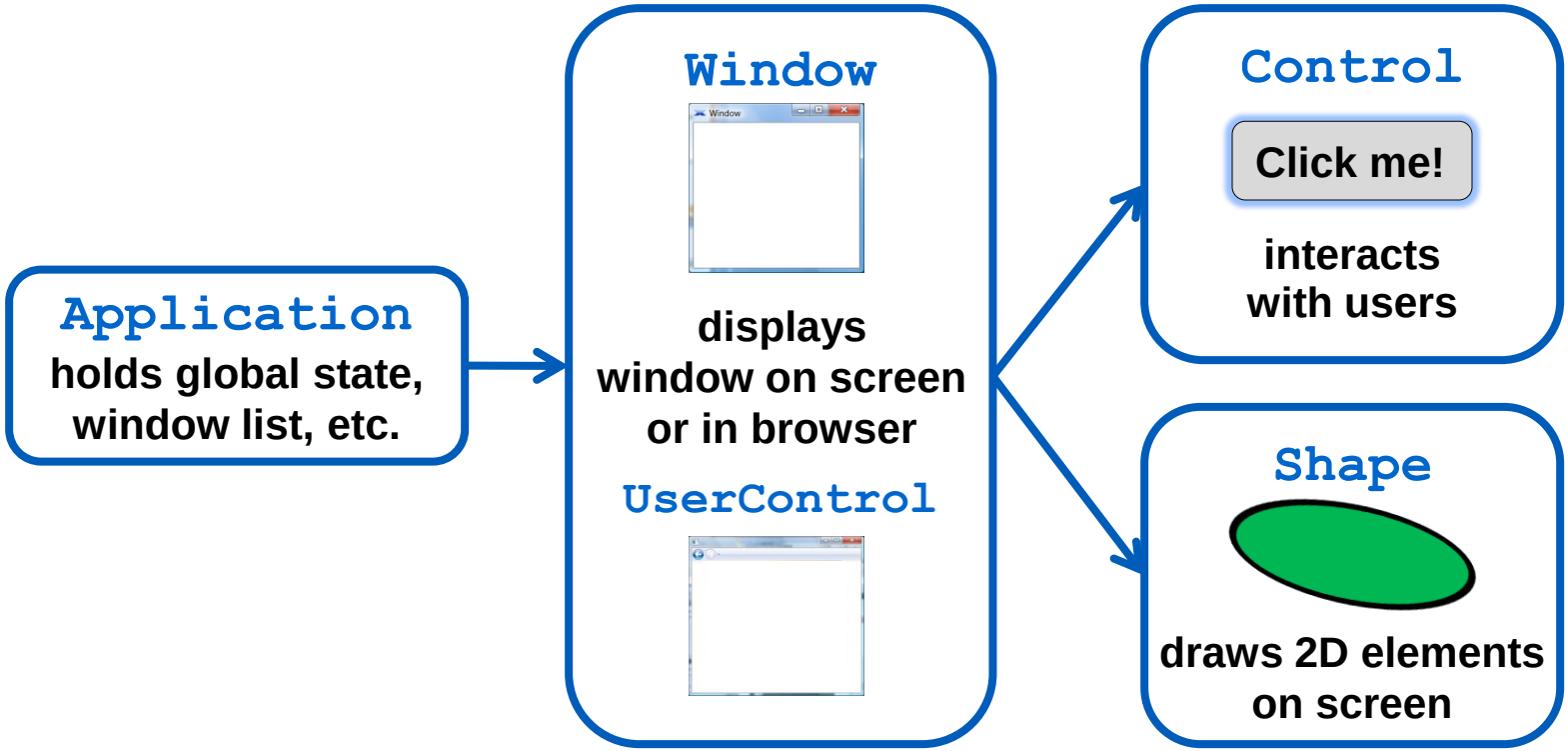
- **WPF / SL applications start with two main files**
  - **`App.xaml`** provides initial starting point for application
  - **`MainPage.xaml`** or **`MainWindow.xaml`** provides initial UI





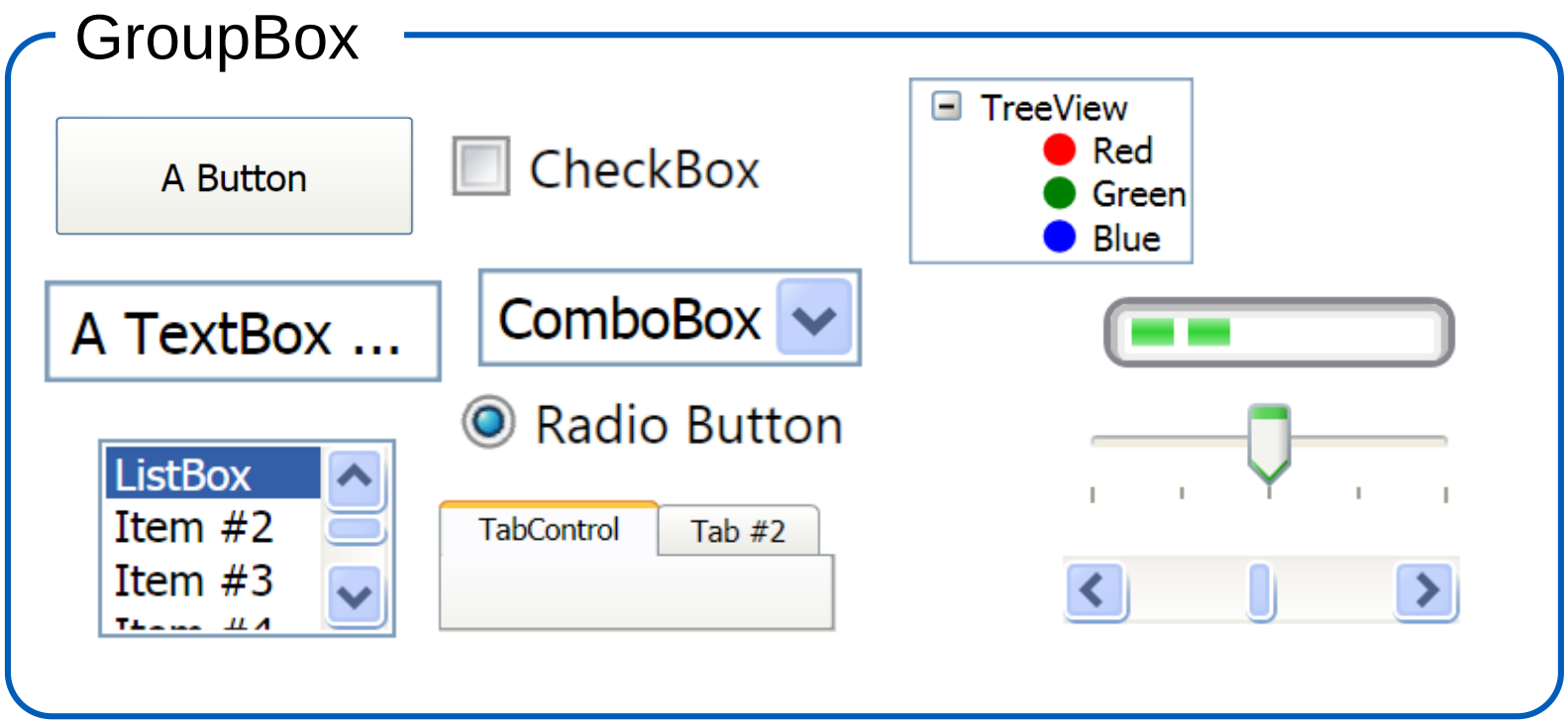
Building Applications

- Several **classes** work together to build an application



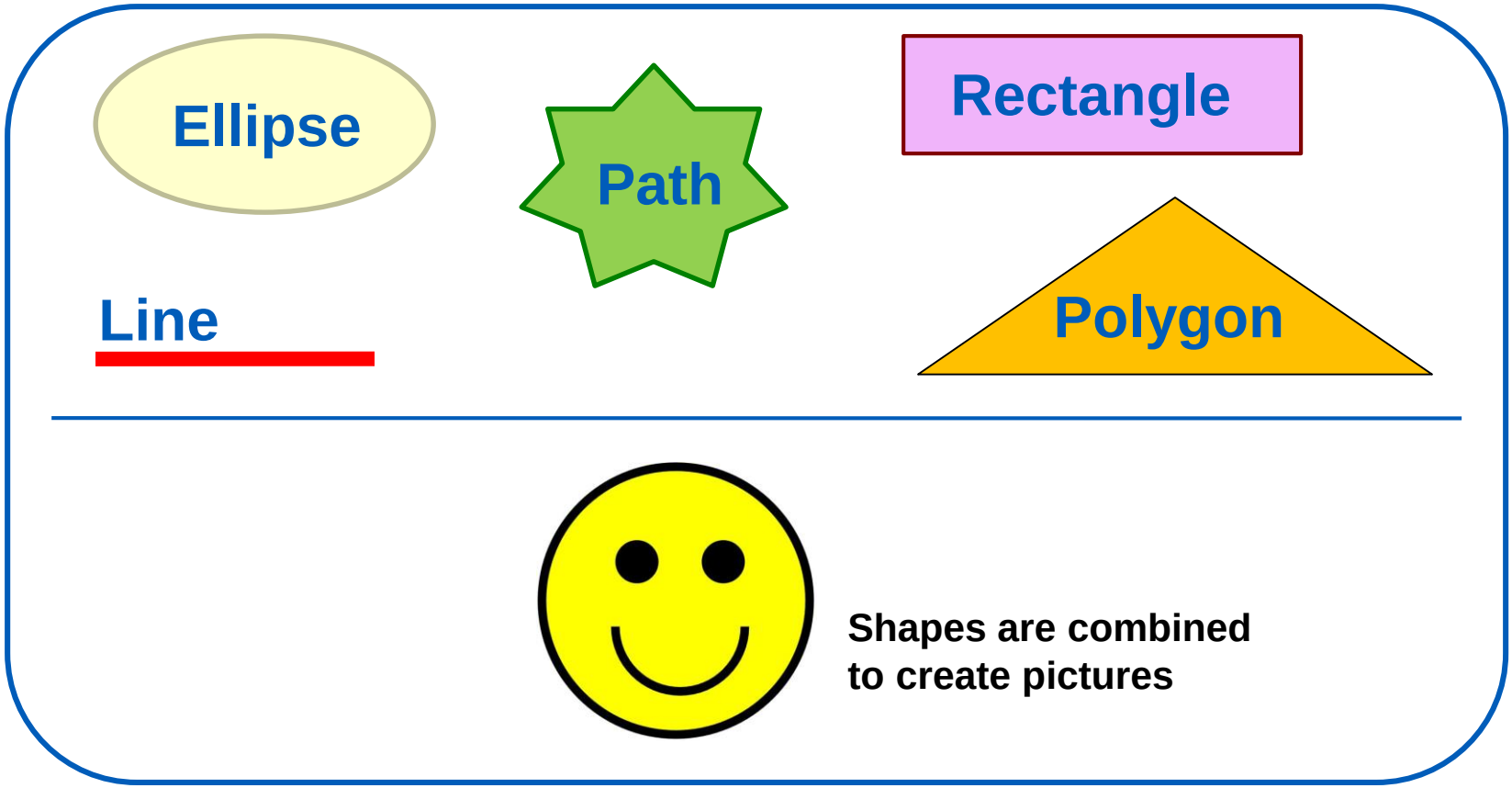
System.Windows.Controls

- **Controls allow interaction with the user**
  - display feedback, receive input and focus



System.Windows.Shapes

- **Shape objects can be used to add 2D elements to window**
  - can react to input but cannot have focus or children



Content limitations

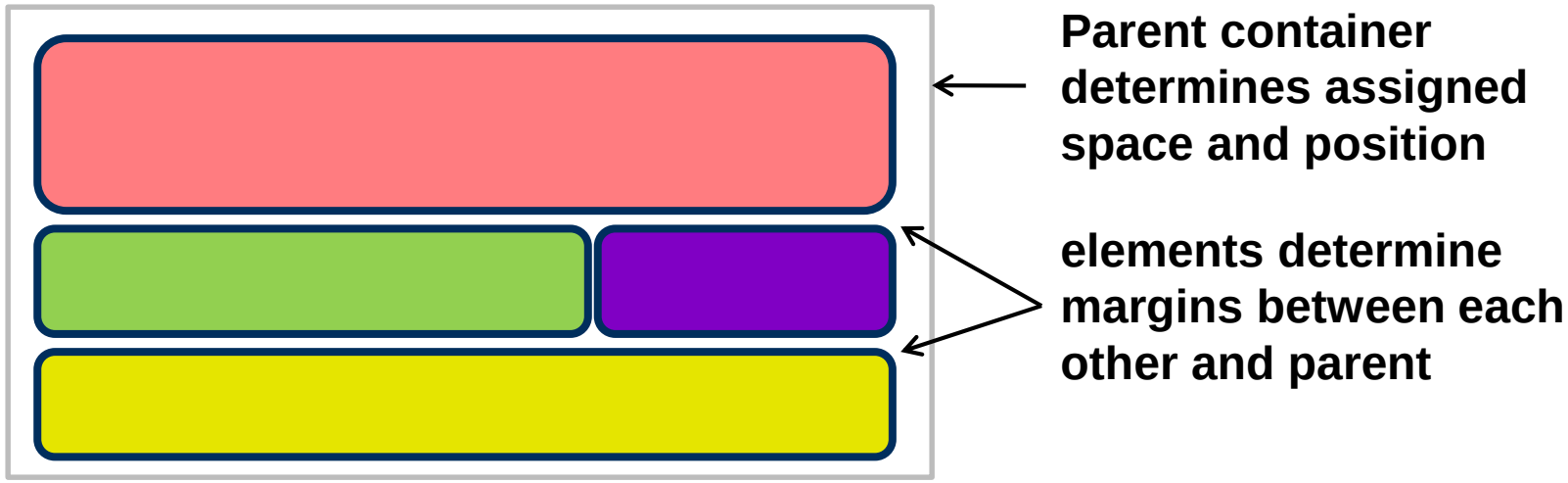
- **Most elements are limited to a single piece of Content**

```
partial class TheButtonDisplay
{
   public TheButtonDisplay()
   {
     Button btn = new Button();
     btn.Content = "Click Me";
     Ellipse el = new Ellipse();
     el.Fill = Brushes.Gold;

     this.Content = ???;
   }
}
```

… anything more complex requires layout

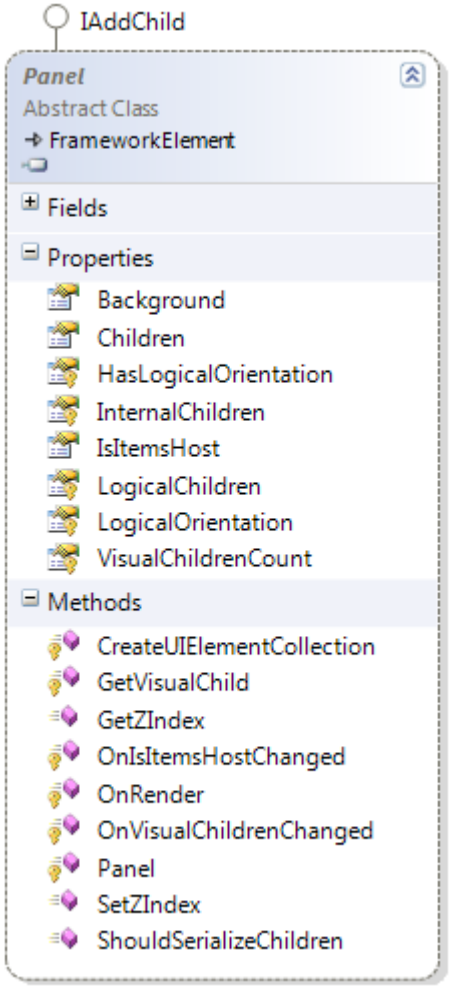
Layout in XAML based applications

- **Layout is performed through two mechanisms**
  - visual elements request spacing and margins individually
  - containers enforce specific positions for child elements
- **May seem cumbersome at first compared to other methods**
  - but provides for more flexibility than just pixel-oriented layout



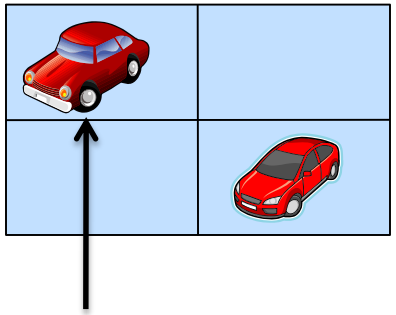
Organizing Layout

- **Panels organize content**
  - each child is added to **`Children`** collection
  - children laid out based on the type of panel
  - panel decides size and position of each child
  - drawing order determined by position of child within **`Panel.Children`** collection
- **Panels can add background color but in general should not affect visualization**
  - they are intended to be a positioning container



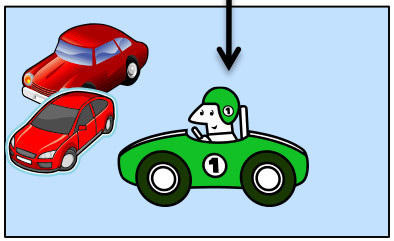
Layout mechanics

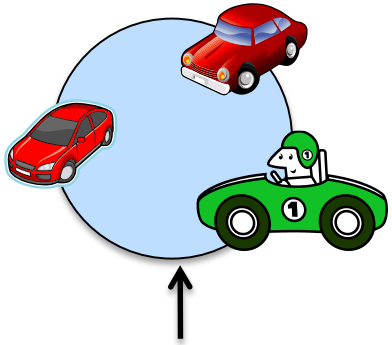
- **Children must supply layout desires to panel**
  - each panel likely has different layout rules and requirements



`Grid` needs to know **`Column`** and **`Row`**

`Canvas` needs to know **`Top`** and **`Left`** position





custom **`RadialPanel`** needs to know angle position

Supplying panel-specific layout information

- **Panels define specific properties for layout management**
  - values are "attached" to each child to indicate preferences
  - referred to as "attached properties"
  - **`Panel`** then reads current value from each child at runtime

```
Canvas panel = new Canvas();
Image sportsCar = LoadCarImage();
panel.Children.Add(sportsCar);

Canvas.SetLeft(sportsCar, 100.5);
Canvas.SetTop(sportsCar, 50.75);
```

static methods used to store value on **`Image`** in code

`Type.Property` syntax used in XAML

```
<Canvas>
<Image Source="sportsCar.jpg"
       Canvas.Top="50.75"
       Canvas.Left="100.5" />
</Canvas>
```

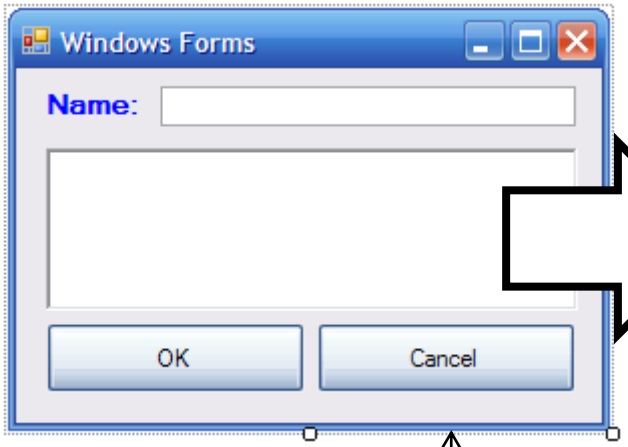
Built-in panels

- **Most common layout scenarios supported directly**
  - very common to compose panels together for complex layout

`StackPanel`	Organizes content horizontally or vertically in a stack
`WrapPanel`	Flows content around based on the size of the container
`Canvas`	Provides for pixel-based placement of children
`DockPanel`	Docks content to the edges of the parent
`UniformGrid`	WPF grid where all columns/rows are the same size
`Grid`	Grid where columns/rows can have different sizes

Reminder: How do we create desktop UI today?

- **Windows Forms designer writes code to create UI**
  - code stored in `InitializeComponent` method
  - solves problems of dedicated resource format in Win32




```
private void InitializeComponent()
{
   ...
   this.button2.Text = "Cancel";
   ...
}
```

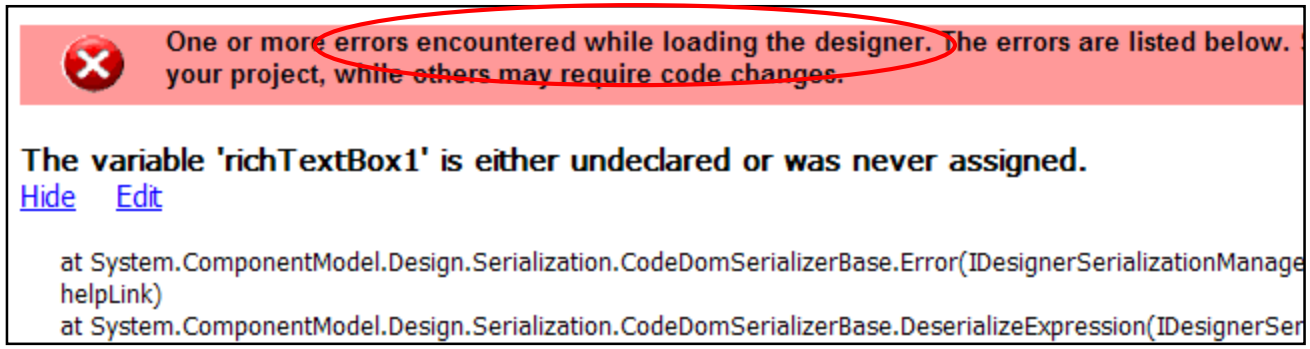
you make a change using the designer

the IDE designer updates the code

Motivation [fragility of WinForms designer]

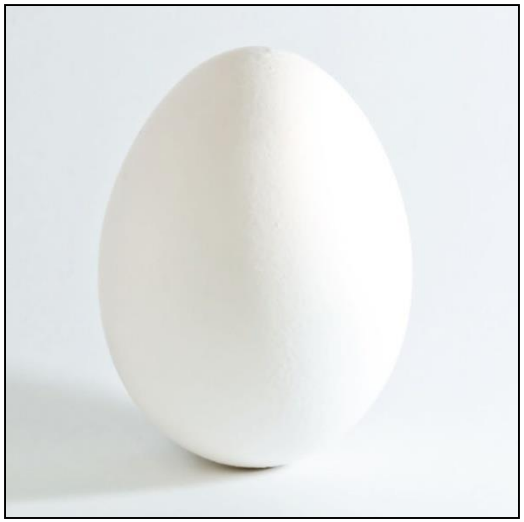
- **Direct changes to `InitializeComponent` can break designer**
  - uses partial classes to hide the method from you

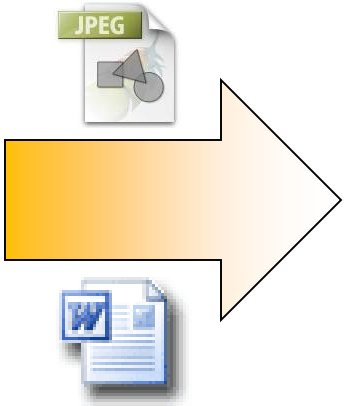
```
#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
```



Motivation [translating design to implementation]

- **Designer does not provide *developer-usable* visual assets**
  - e.g. limited ability to impact final application





what the designer *envisioned*

what the developer *built*

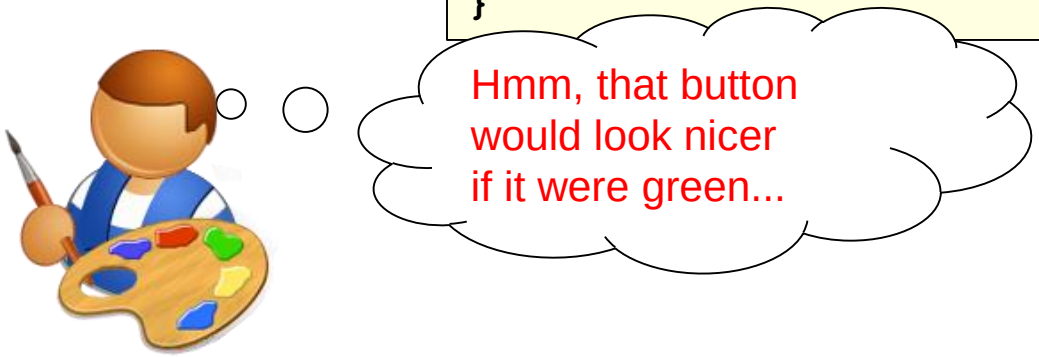
Motivation [mixing of UI and behavior]

- **Windows Forms mixes together visual design and behavior**
  - violates programming principle of separation of concerns
  - even simple visual changes require code change and retest

visual design →

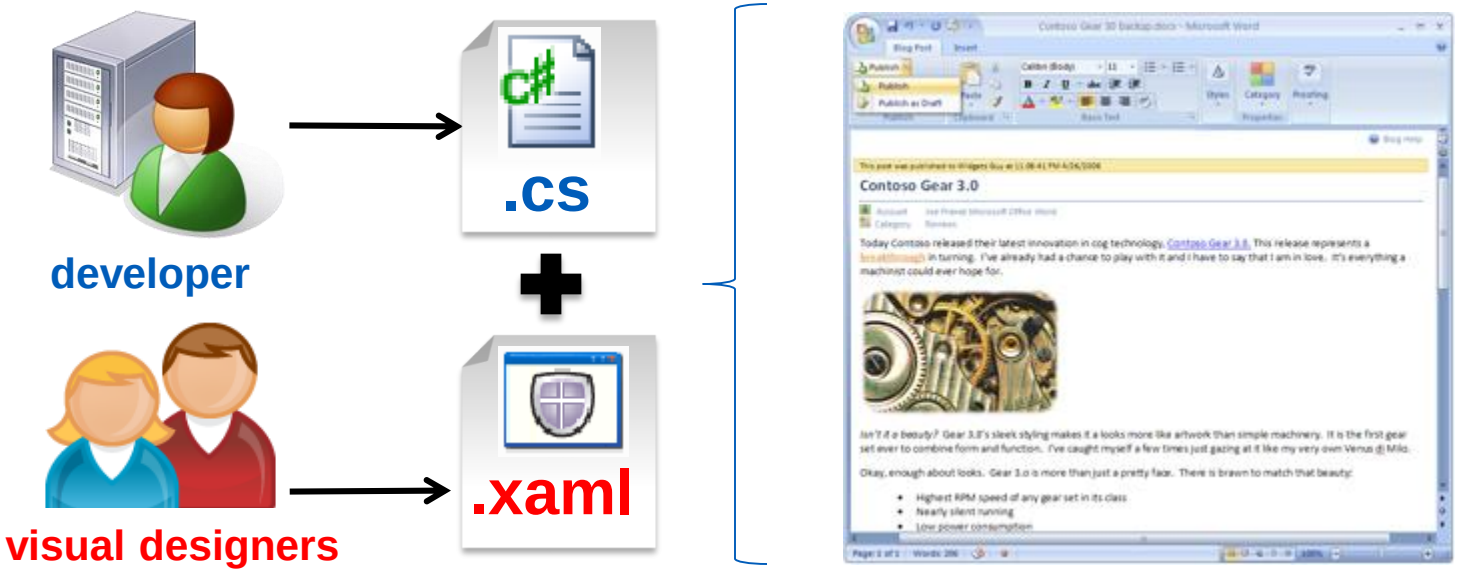
behavior →

```
private void InitializeComponent()
{
   ...
   this.button2.BackColor = Color.Blue;

   this.button2.Click += OnClick;
   ...
}
```



Extensible Application Markup Language (XAML)

- **Application can separate logic and visual design**
  - can be developed independently by the appropriate roles



Creating XAML based applications

- **Visual Studio includes full featured XAML designer**
  - this is for the developer role
- **Microsoft has several XAML tools for designers + developers**
  - XamlPad is a free tool included in the SDK[1]
  - Expression Blend is Microsoft's professional XAML designer
  - Expression Design is a 2D illustrator tool which emits XAML
- **XAML specific tools**
  - ZAM3D for generating 3D models for WPF
  - Kaxaml (similar to XamlPad but a bit nicer)
- **XAML can also be generated from other common formats**
  - Adobe Flash (SWF) and Illustrator
  - Visio diagrams
  - even VB6 forms!

Visual Studio 2010

WpfApplication1 - Microsoft Visual Studio

File Edit View Project Build Debug Team Data Format Tools Architecture Test ReSharper Analyze Window Help

Debug x86

Toolbox
- TextBlock
- TextBox
- All WPF Controls
- Pointer
- Border
- Button
- Calendar
- Canvas
- CheckBox
- ComboBox
- ContentControl
- DataGrid
- DatePicker
- DockPanel
- DocumentViewer

Server Explorer | Toolbox

Document Outline
- Window
- Grid
- TextBlock
- Label (label1)
- TextBox (textBox1)
- Label (label2)
- TextBox (textBox2)
- Button (button1)

Document Outline | Data Sources

MainWindow.xaml

MainWindow

User Name

Password

Login

Design | XAML

```
<Window x:Class="WpfApplication1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentati
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="300" Width="525">
    <Grid Background="#B4B48250" Width="484">

        <TextBlock Foreground="SaddleBrown" Text="Login" FontSize="72p
                   FontFamily="Broadway"
                   HorizontalAlignment="Center" VerticalAlignment="Cer
            <TextBlock.RenderTransform>
```

100 %

Window Window

Solution Explorer
- Solution 'WpfApplication1' (1 project)
- WpfApplication1
- Properties
- References
- App.xaml
- MainWindow.xaml
- MainWindow.xaml.cs

Solution Explo... | Team Explorer | Class View

Properties

Window <no name>

Properties | Events

Search

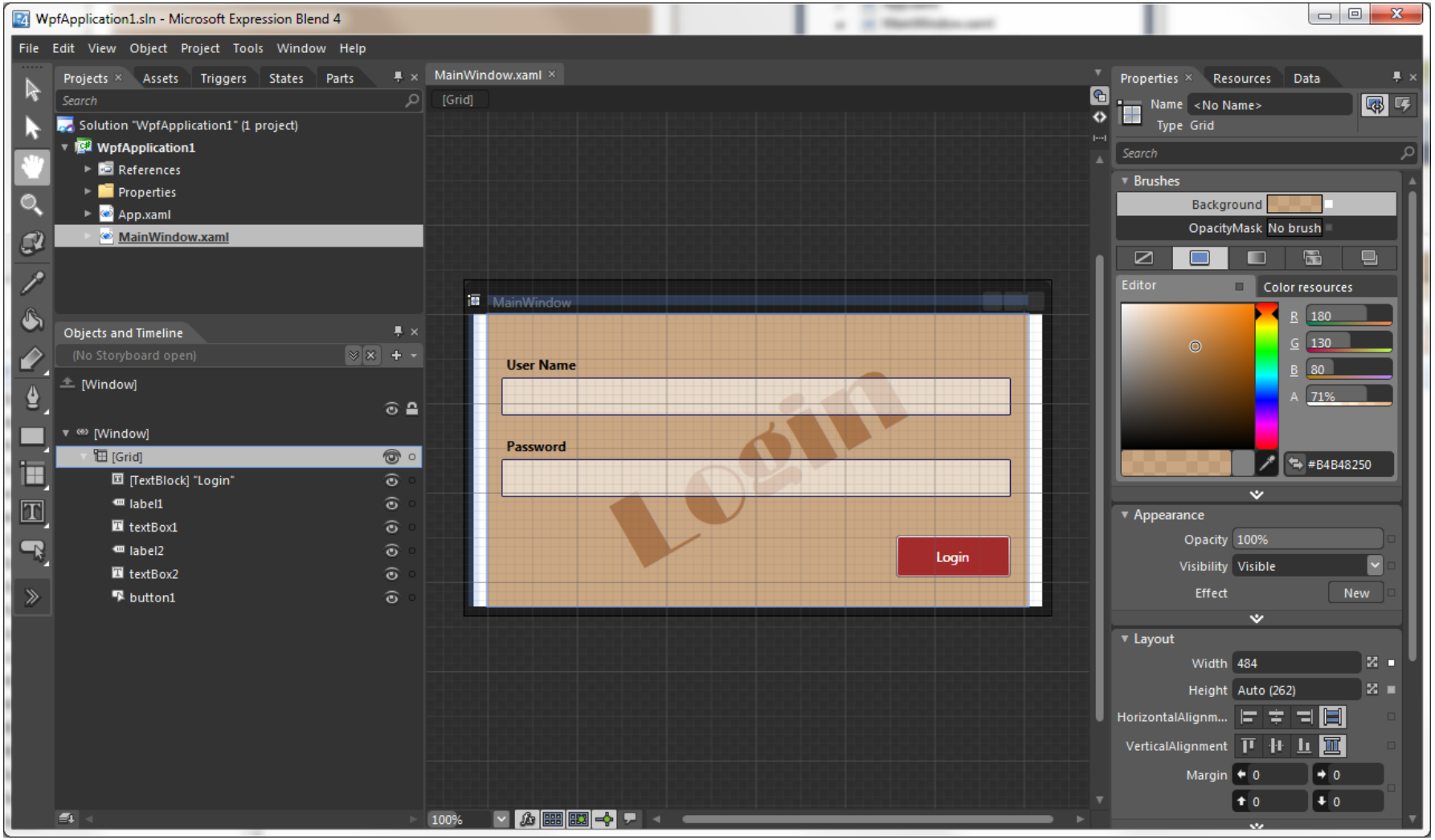
Common

DataContext	Binding...
Style	Resource...
Title	MainWindow
ContentTemplate	Resource...
Icon	
IsEnabled	✓
ResizeMode	CanResize
ShowInTaskbar	✓
SizeToContent	Manual
Topmost	
WindowStartupLocati...	Manual
WindowState	Normal

Pending Changes | Error List | Output | Test Results | Find Results 1 | Find Symbol Results

Ready

Expression Blend for Developers / Designers / Integrators



XAML 101: objects and properties

- **Elements create objects at runtime**
  - must have a default constructor[1] and cannot be a nested type
- **Attributes assign property values**
  - must have public setter
- **Content set as child of element**

```
<Window Background="Red"
        Width="300"
        Height="300"
        Title="Hello, XAML">

   <Button>Hello</Button>

</Window>
```

```
Window window = new Window();
window.Background = Brushes.Red;
window.Width  = 300;
window.Height = 300;
window.Title  = "Hello, XAML";

Button btn = new Button();
btn.Content = "Hello";
window.Content = btn;
```

***Equivalent* C# code**

Converting strings to property values

- **Attribute strings are coerced to types with Type Converters**
  - runtime failure occurs if conversion fails
  - applied to the type definitions with **`[TypeConverter]`**

```
<Button x:Name="b"
  Background="Blue"
  FontFamily="Arial"
  FontSize="24pt"
  Padding="5" />
```

Type Converters

- BrushConverter
- FontFamilyConverter
- FontSizeConverter
- ThicknessConverter

```
Button b = new Button();
b.Background = new SolidColorBrush(Colors.Blue);
b.FontFamily = new FontFamily("Arial");
b.FontSize = 32;
b.Padding = new Thickness(5);
```

Note: Silverlight does not use **`FontSizeConverter`**.

Problem: assigning complex objects

- **Not every object can be created from a string**
  - can associate **`TypeConverter`** if you can modify the class

```
<Button Content="Click Me"
    Foreground="White"
    FontSize="36pt"
    Background="ImageBrush Stretch="Fill"
    ImageSource=bliss.jpg" />

</Button>
```

Cannot define `ImageBrush` through a simple string…

Assigning complex property values

- **Property Element syntax used to assign complex objects**
  - takes the form `TypeName.PropertyName`

define the **ImageBrush** in normal XAML element way

```
<Button Content="Click Me"
        Foreground="White"
        FontSize="36pt">
   <Button.Background>
      <ImageBrush
         ImageSource="bliss.jpg" />
   </Button.Background>

</Button>
```


```
Button b = new Button();
...
b.Background = new ImageBrush(..);
...
```

Locating CLR types

- How can XAML determine the proper CLR type to create?

```
<Window>
   ...
    <Button>Click Me</Button>
   ...
</Window>
```

?

System.Windows.Forms

```
public class Button
{
   ...
}
```

System.Web.UI.WebControls

```
public class Button
{
   ...
}
```

System.Windows.Controls

```
public class Button
```

Solution: XML namespaces

- **XAML locates CLR types using XML namespace declarations**
  - defined using xmlns attribute on root element in XAML file
- **Most XAML files require two known `xmlns` statements**
  - makes all major UI namespaces visible to XAML

Controls, Shapes, Data Binding is default namespace

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

XAML keywords (x:Name) is mapped to 'x'

Locating custom CLR types in same assembly

- **XAML must know proper CLR namespace**
  - defined using `xmlns` attribute on element or ancestor element

```
namespace Controls
{
  public class MaskedEdit
  {
    ...
  }
}
```

```
<MaskedEdit x:Name="edit1"
xmlns="clr-namespace:Controls">
```

App.exe

```
using Controls;
...
MaskedEdit edit1 = new MaskedEdit();
```

Locating custom CLR types in other assemblies

- **XAML must also know proper .NET assembly**
  - necessary when type contained in different assembly

```
<UserControl xmlns="..." xmlns:x="..."
   xmlns:ctls="clr-namespace:Controls;assembly=maskededitctl">
  ...

   <ctls:MaskedEdit x:Name="edit1" ... />

</UserControl>
```

MainApp.xap

```
namespace Controls
{
   public class MaskedEdit
   { ... }
}
```

MaskedEditCtl.dll

Working with runtime values

- **Markup extensions instruct the XAML parser to perform custom handling of an attribute value**
  - performs runtime lookup of value
  - custom extensions can be created to extend XAML syntax

surrounded by braces

```
<TextBox Text="{Binding Path=FirstName}" />
```

```
<Grid Background="{x:Null}">
  ...
</Grid>
```

```
<TextBlock Text="{}{Not an extension}" />
```

string literals that start with `{` must be escaped with `{}`, this text will be `"{Not an extension}"`

Providing behavior for XAML objects

- **Event Handlers can be wired up through XAML attributes**
  - handler must exist in code-behind associated with XAML file

```
public class Button
{
    public event RoutedEventHandler Click;
}
```

```
<Button Click="OnOK" />
```

```
...
Button button = new Button();
button.Click += OnOK;
```

```
void OnOK(object sender, RoutedEventArgs e) {...}
```

Accessing XAML objects in code behind

- **XAML created objects can be accessed in code behind**

`x:Name` creates field in code behind file [1]

```
<Window>
   <StackPanel>
      <TextBlock/>
      <TextBox x:Name="tbEntry" x:FieldModifier="private"/>
      <Button x:Name="btnOK" x:FieldModifier="public"/>
   </StackPanel>
</Window>
```

`x:FieldModifier` changes visibility of created field

```
void CheckUserInterfaceControls()
{
   btnOk.IsEnabled = (tbEntry.Text.Length > 0);
}
```

can then access object by name in code-behind

Visual Studio and Code Behind

- **VS.NET creates associated code-behind files for logic**
  - matched to XAML files through `x:Class` tag

Window1.xaml* | Window1.xaml.cs | Start Page

```
<Window x:Class="MainApp.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/x
    Title="MainApp" Height="300" Width="300">
    <StackPanel>
      <Button Name="butt
        Press Me</Button
    </StackPanel>
</Window>
```

Window1.xaml* | Window1.xaml.cs* | Start Page

MainApp.Window1

```
namespace MainApp
{
    public partial class Window1 : System.Windows.Window
    {
        public Window1()
        {
            InitializeComponent();
        }

        void Button_Click(object sender, RoutedEventArgs e) {
            button1.Content = "Button has been pressed";
        }
    }
}
```

Summary

- **Architecture was redesigned from the group up**
  - do not assume it works in the traditional Win32 fashion
- **Flexible design provides for almost any style of application**
  - learn a single technology
- **Controls and Shapes provide simple building blocks for UI**
  - anything not present is easily composed
- **Layout is performed with Panels**
  - goal is to provide flexibility + simplicity
  - enables automatic resize and UI scaling
- **UI Is created declaratively using XAML**
  - enables tooling and designer / developer roles