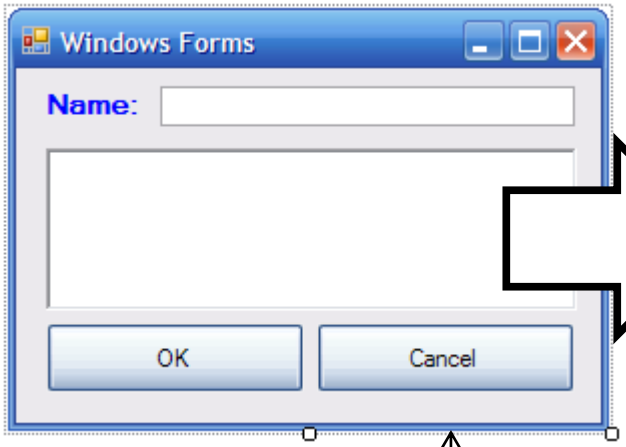
Using XAML

DEVELOPMENTOR

DEVELOPING PEOPLE WHO DEVELOP SOFTWARE

Reminder: How do we create UI in Windows Forms?

- 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()
{
```

One or more errors encountered while loading the designer. The errors are listed below. S
your project, while others may require code changes.

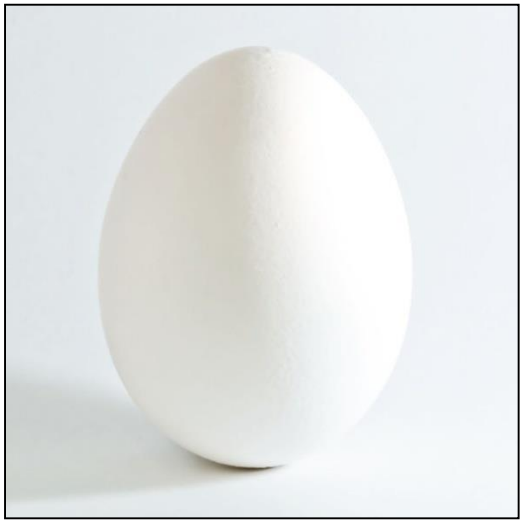
The variable 'richTextBox1' is either undeclared or was never assigned.

Hide Edit

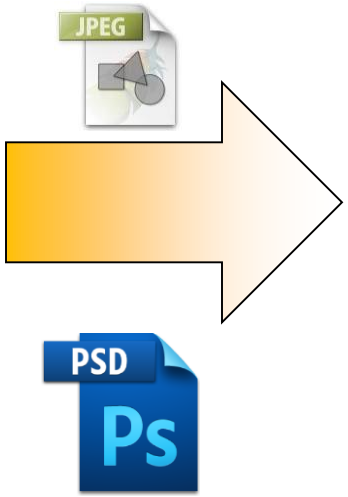
at System.ComponentModel.Design.Serialization.CodeDomSerializerBase.Error(IDesignerSerializationManage
helpLink)
at System.ComponentModel.Design.Serialization.CodeDomSerializerBase.DeserializeExpression(IDesignerSer

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;
  ...
}
```

Hmm, that button
would look nicer
if it were green...

Extensible Application Markup Language (XAML)

- Allows WPF to separate logic and visual design
  - can be developed independently by the appropriate roles

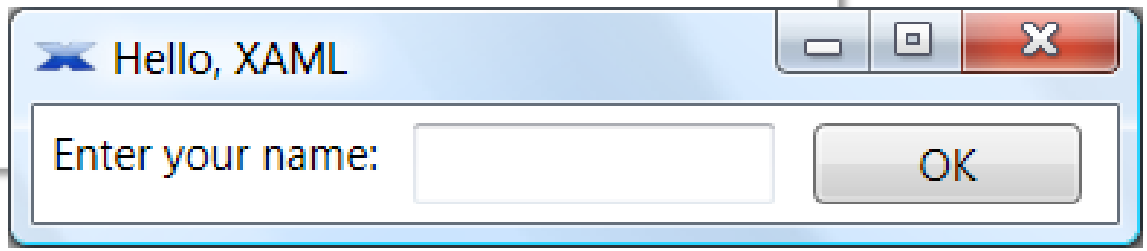
XAML Basics

- XAML is a markup language that creates and initializes objects
  - expressive: can create almost anything (controls, graphics, etc.)
  - compact: easily readable (it is just XML after all)
  - extensible: can be used with a variety of technologies

```
<Window Title="Hello, XAML">

    <StackPanel Orientation="Horizontal">
      <TextBlock>Enter your name:</TextBlock>
      <TextBox />
      <Button>OK</Button>
    </StackPanel>

</Window>
```



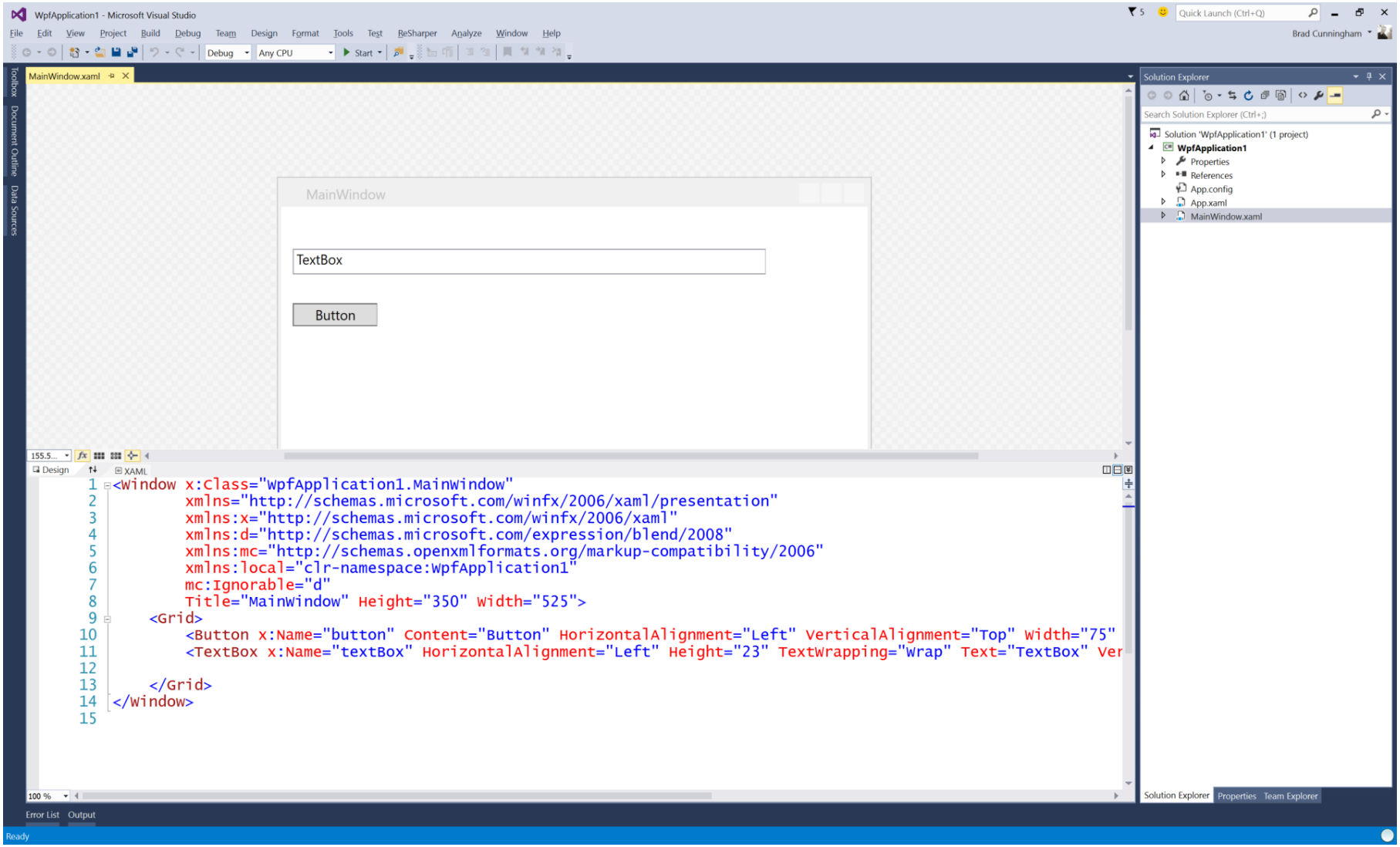
Creating XAML based applications

- 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
  - 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!

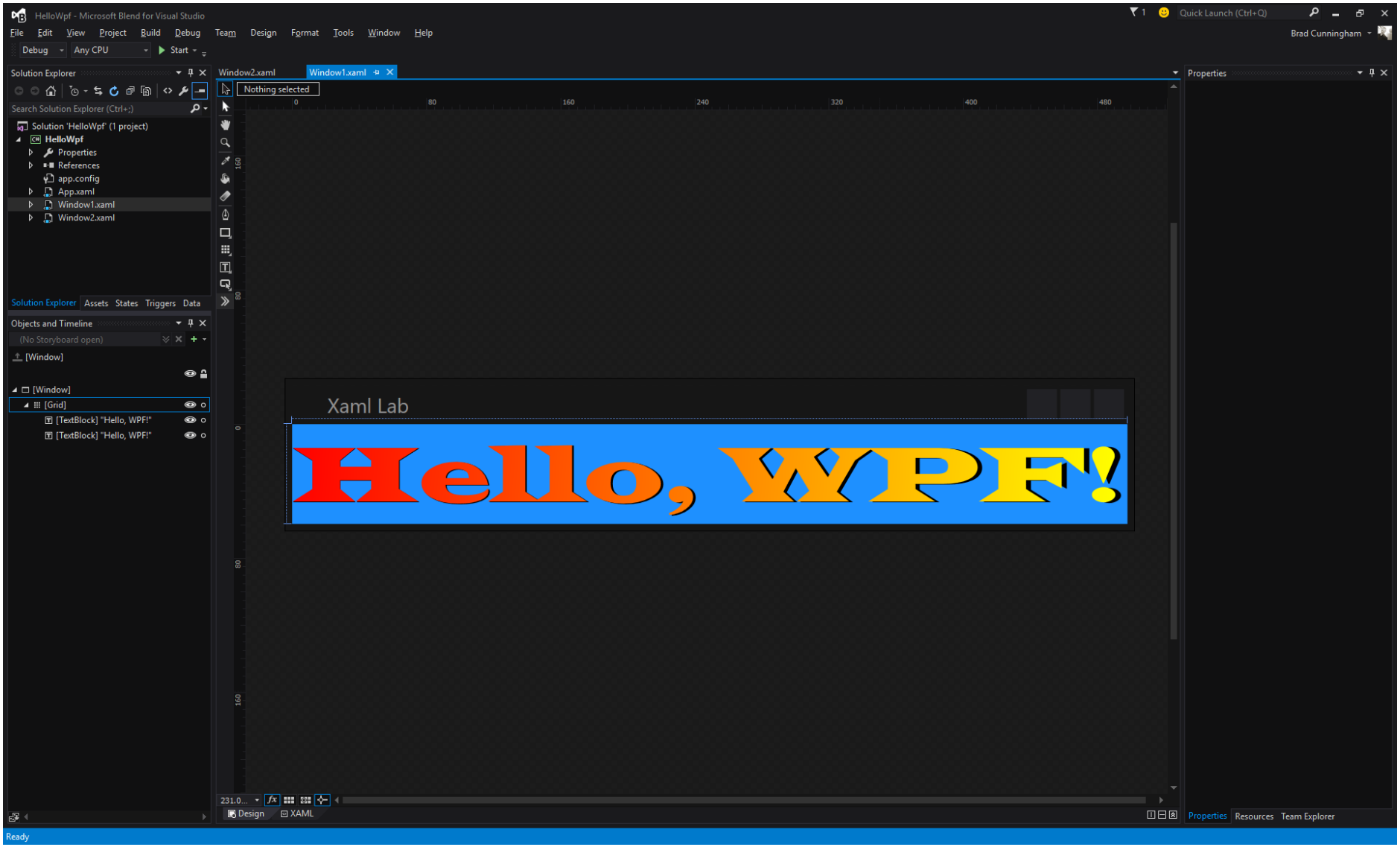
Expression Design for designers

- Version 4 is the last version (free to download)

Visual Studio



Microsoft Blend for Visual Studio 2013 / 2015



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

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

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

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);
```

Problem: assigning complex objects

- Not every object can be created from a string
  - could create **`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
- Primary WPF types require two known `xmlns` statements
  - generally included in every WPF-based XAML file
  - makes all major WPF 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
  - not necessary when type contained in same assembly[1]

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

App.exe

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

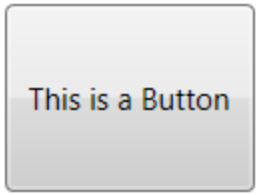
customctls.dll

Assigning content in XAML

- Content can be assigned directly to the property
  - use either attribute or property element syntax

```
<Button Content="This is a Button"/>
```

strings can be assigned directly



```
<Button>
   <Button.Content>
      <Ellipse Width="50"
               Height="50"
               Fill="Gold" />
   </Button.Content>
</Button>
```

complex content requires property element syntax

Default Content Property

- Child XML element(s) are assigned to the *default* property
  - allows short-hand syntax
  - identified by **`ContentPropertyAttribute`** on the class

```
<Button>
   <Button.Content>
      <Ellipse Width="50"
               Height="50"
               Fill="Gold" />
   </Button.Content>
</Button>
```

```
<Button>
   <Ellipse Width="50"
            Height="50"
            Fill="Gold" />
</Button>
```

```
[ContentProperty("Content")]
public class Button
{
  public object Content { get; set; }
}
```

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

MainWindow.xaml

```
1  <Window x:Class="ExampleApp.MainWindow"
2          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3          xmlns:x="http://sc
4          Title="MainWindow"
5      <Grid>
6
7      </Grid>
8  </Window>
9
```

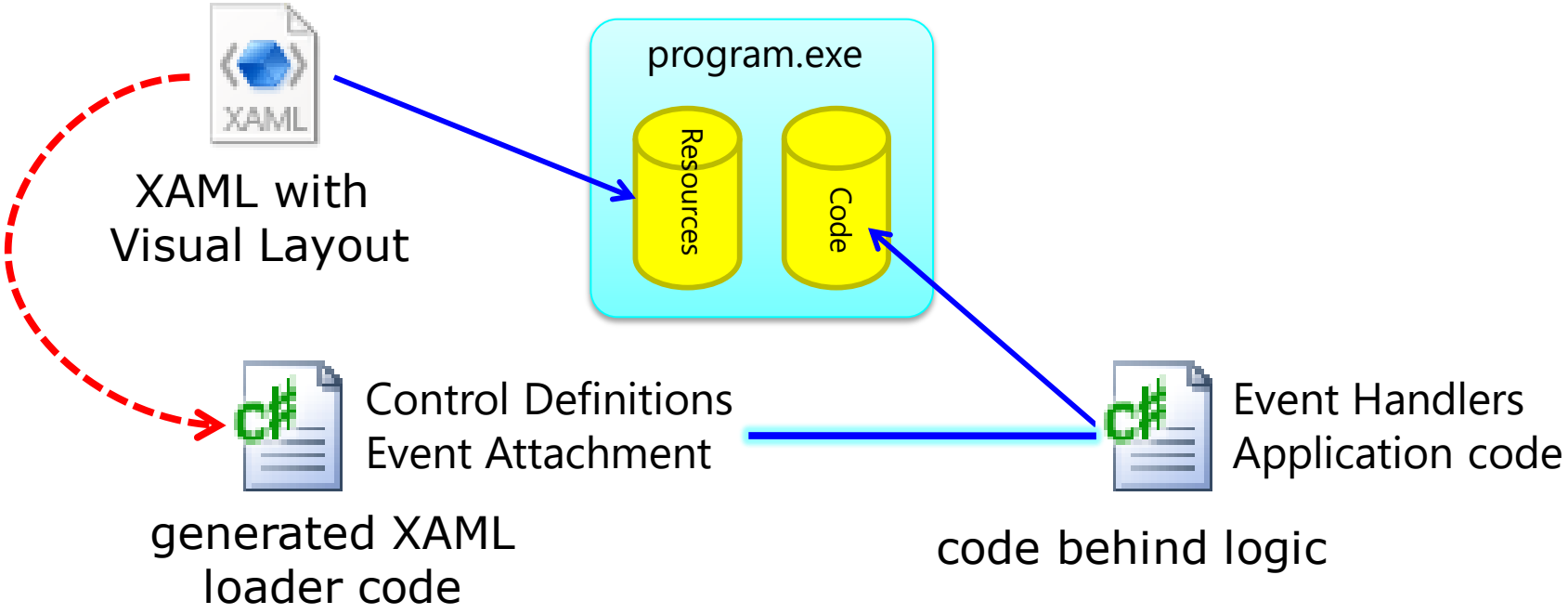
MainWindow.xaml.cs

ExampleApp.MainWindow

```
1  using System.Windows;
2
3  namespace ExampleApp
4  {
5      /// <summary>
6      /// Interaction logic for MainWindow.xaml
7      /// </summary>
8      public partial class MainWindow : Window
9      {
10         public MainWindow()
11         {
12             InitializeComponent();
13         }
14     }
15 }
16
```

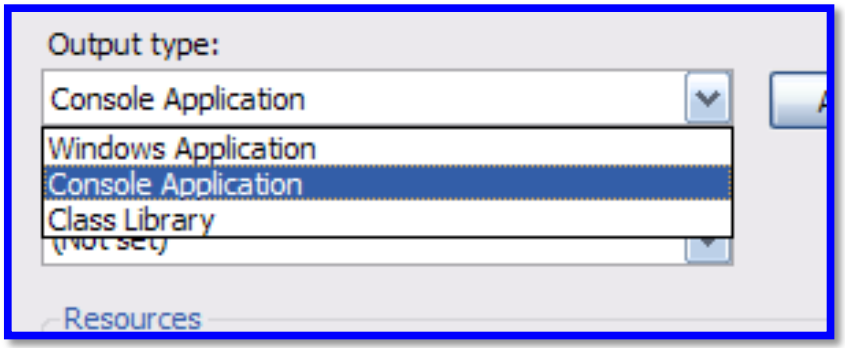
XAML compilation process

- Most applications are a mix of markup and procedural code
  - procedural code (behavior) goes in **Code Behind** file
  - XAML compiled into binary form and stored in resources
  - partial class generated to load XAML and bind event handlers



Debugging XAML

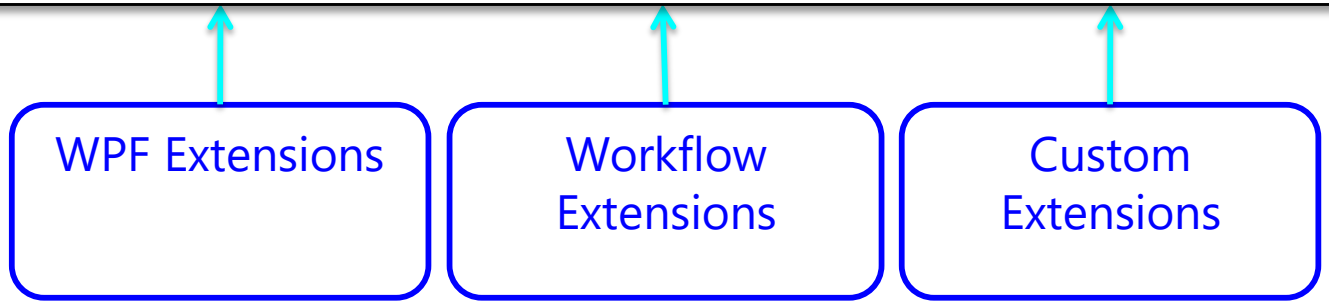
- Errors in compiled XAML often result in a runtime failure
  - retrieve error text through the `InnerException`
  - "innermost" exception normally has real problem details
  - no debugger step-through capabilities for XAML
- Helpful to change project to Console application
  - exception is then printed on console window



Extending XAML: Markup Extensions

- Markup Extensions provide additional keywords for XAML
  - allows specific technology extensions to be created
  - is the preferred extension mechanism for XAML
  - instance is created and evaluated at runtime

```
public abstract class MarkupExtension
{
   protected MarkupExtension();
   public abstract object ProvideValue(IServiceProvider sp);
}
```



Using Markup Extensions

- Values enclosed in curly braces "{}" treated as an extension
  - must escape values if "{" used as first character
  - compiler assumes "Extension" suffix on class
  - properties are set as strings or type-converted values

```
<Student x:Name="p1" ID="{}{N/A}"
         Gpa="{Binding Path=CurrentGpa}" />
```

```
Student p1 = new Student();
BindingExtension be = new BindingExtension();
be.Path = "CurrentGpa";
p1.Gpa = be.ProvideValue(...);
```

Some built-in markup extensions

```
<Button x:Name="button1" Content=""
    Background="        " />
```

set properties to null

```
<Style TargetType="                    " />
<Style TargetType="{x:Type local:SomeClass+NestedClass}" />
```

Identify a specific Type (C# typeof operator)

read static property value

```
<Label x:Name="osVersion"
     xmlns:s="clr-namespace:System;assembly=mscorlib"
     Content="                                    " />
```

data bind properties

```
<TextBox x:Name="tb1" Text="Red"
         Foreground="                                    " />
```

What's the future of XAML?

- XAML is a first class choice in Windows App development
  - Can build Windows Universal Apps using XAML
  - WPF still the defacto choice for thick client windows development
  - New tooling support in VS 2015 for better XAML debugging

  - New WPF Features in 4.5
  - Data binding to static properties
  - Improved perf for large data sets
  - Data shaping
  - Ribbon Control
  - Etc..

  - New WPF specific features added to .NET 4.6
  - Better HDPI support
  - Better touch support

Summary

- XAML provides an easy, concise way to represent static UI
  - intended for tool generation vs. human generation
  - anything in XAML can also be done in code, but not vice-versa
- The tool story is continuing to improve as time goes on
  - Blend + Visual Studio is a great combination for development
  - might also consider 3rd party XAML generation tools