

# Organizing your XAML

# Objectives

- **What are the options in splitting my application up?**
  - separating code + XAML + resources
- **How can I share colors, brushes, fonts, etc. in my app?**
  - what is a resource dictionary?
  - combining resource dictionaries at runtime
- **How can I share resources across applications?**



# What makes up my application?

- Most WPF applications are composed of

**UI + Code**

**Top level windows and layout**

**Resources**

**Brushes, drawings, bitmaps, strings, etc.**

**Templates**

**Data Templates, Control Templates, Styles**

**Controls**

**User Controls, Custom Controls, Themes**

**Most of this is described in XAML –  
knowing how to structure it all is important**



# XAML and refactoring

- **XAML and code behind describe UI and behavior**
  - keeping them segregated is a primary goal of WPF
  - push everything "visual" into the XAML (more on this later)
- **Size of XAML files tends to grow excessive as project matures**
  - tendency to "lump" everything into `window1.xaml`
  - reduces readability
- **Refactoring the XAML files will help in maintenance**
  - put only layout and control structures into main files
  - everything else can move to secondary XAML files
  - can also help performance!



# Refactoring XAML: thinking about resources

- **WPF extends resource sharing to include *any* object**
  - anything creatable in XAML can be classified as a "resource"
  - allows a single object to be easily shared throughout the UI
- **Should place reusable assets into resources**
  - shared visual elements such as brushes, colors, fonts, etc.
  - styles, templates and theme information
  - animation storyboards

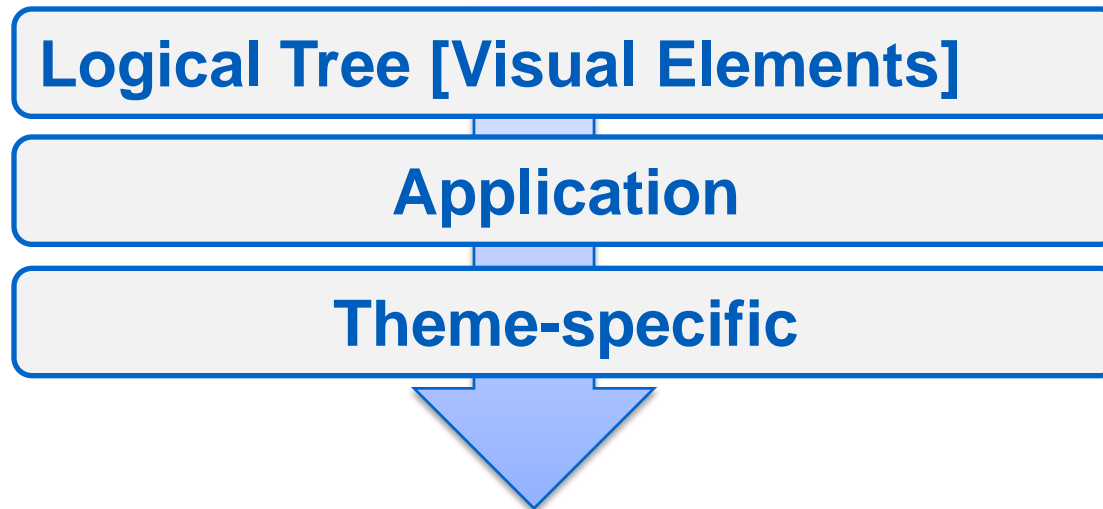
```
<StackPanel>  
  <TextBox Background="AliceBlue" Foreground="DarkBlue" />  
  <Button Background="DarkBlue" Foreground="AliceBlue" />  
</StackPanel>
```

**The brushes used could be in resources instead of directly specified – that would allow them to be centrally managed**



# Defining Resources

- **Resources can be defined at multiple levels**
  - allows scoping of resources to most appropriate level



**Each has a Resources collection that holds assets  
WPF will search each collection looking for resources**

# How to add resources

- **Resources** collection used to hold application assets
  - associates an **object** with a unique **key**
  - accessible from XAML or code behind

```
<Window>
  <Window.Resources>
    <SolidColorBrush
      x:Key="redBrush"
      Color="#DD0000" />
  </Window.Resources>
</Window>
```

**x:Key** identifies the key

```
public class Window1
{
    Window1 ()
    {
        InitializeComponent ();
        this.Resources.Add (
            "redBrush",
            new SolidColorBrush (
                Color.FromRgb (0xdd, 0, 0) )
        );
    }
}
```



# Using Resources in XAML

- `{StaticResource}` extension used to locate resources
  - parameter identifies the `x:Key`
  - resource value applied once, during element creation

```
<Window xmlns:s="clr-namespace:System;assembly=microsoft.windows.common-core-1.0.0">
  <Window.Resources>
    <SolidColorBrush x:Key="redBrush" Color="#DD0000" />
    <s:String x:Key="caption">Text Caption</s:String>
  </Window.Resources>
  <StackPanel>
    <Button Background="{StaticResource redBrush}" ... />
    <TextBlock Foreground="{StaticResource redBrush}"
      Text="{StaticResource caption}" />
  </StackPanel>
</Window>
```



# Using Resources in XAML [2]

- `{DynamicResource}` extension defers binding until usage
  - allows forward references
  - **dynamically** updates UI when value stored in dictionary changes

```
<Window Background="{DynamicResource background}">
  <Window.Resources>
    <SolidColorBrush x:Key="background" Color="White" />
  </Window.Resources>

  <TextBlock
    Foreground="{DynamicResource {x:Static
                                   SystemColors.GrayTextBrushKey}}"
    Text="Some Gray Text" />
  ...
</Window>
```

Text color changes when system changes the "GrayText" color



# How WPF locates resources

- Framework searches logical tree looking for resource with key
  - starts at current level and moves outward to system resources
  - first resource with proper key is used

```
<Window>
  <Window.Resources>
    <SolidColorBrush x:Key="redBrush" Color="#DD0000" />
  </Window.Resources>
  <StackPanel>
    <Button Background="{StaticResource redBrush}" .../>
    <StackPanel>
      <StackPanel.Resources>
        <SolidColorBrush x:Key="redBrush" Color="Pink" />
      </StackPanel.Resources>
      <Button Background="{StaticResource redBrush}" .../>
    </StackPanel>
  </StackPanel>
</Window>
```

# Locating Resources in Code

- **Resources property provides indexer access to dictionary**
  - useful for adding/removing resources dynamically
- **FindResource is the preferable way to locate resources**
  - performs same bubble search as XAML bindings
  - use **TryFindResource** if resource might not exist

```
...
window1.Resources.Add("redBrush", Brushes.Red);
Brush redBrush = (Brush) window1.Resources["redBrush"];
...
button1.Background = (Brush)
    button1.FindResource("redBrush");
...
textBlock1.Foreground = (Brush)
    textBlock1.FindResource(
        SystemColors.GrayTextBrush);
```



# Be careful of what you store!

- **Any object may be placed in resources dictionary**
  - WPF creates object and keeps a single reference (good!)
  - can cause problems when visual object is reused

```
<Window>
  <Window.Resources>
    <Image x:Key="theImage" Source="img1.jpg" />
  </Window.Resources>

  <StackPanel>
    <StaticResource ResourceKey="theImage" />
    <StaticResource ResourceKey="theImage" />
  </StackPanel>
</Window>
```



**Fails compilation because image is already part of visual tree**

# Re-using visuals

- **Solution is to tell WPF to create a new instance each time!**
  - **x:Shared** property controls instancing
  - not supported for dynamic XAML (requires compilation)

```
<Window>
  <Window.Resources>
    <Image x:Shared="false"
           x:Key="theImage" Source="img1.jpg" />
  </Window.Resources>

  <StackPanel>
    <StaticResource ResourceKey="theImage" />
    <StaticResource ResourceKey="theImage" />
  </StackPanel>
</Window>
```



# Utilizing system resources

- **Most applications rely on system-defined resources**
  - system colors
  - system fonts
  - standard widths, spacing, timings, etc.
- **Accessible through properties on SystemXXX classes**
  - `System.Windows.SystemParameters`
  - `System.Windows.SystemFonts`
  - `System.Windows.SystemColors`

```
<StackPanel>  
  <Button Width="{x:Static SystemParameters.IconHeight}"  
    Background="{x:Static SystemColors.ActiveCaptionBrush}"  
    Content="Some Button" ... />  
</StackPanel>
```



## Utilizing system resources [2]

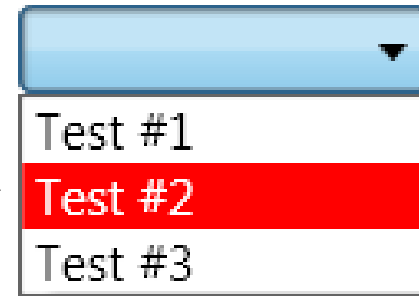
- **Applications can also access system resources using keys**
  - each property has a **xxxKey** property also defined on class
  - utilizes **ResourceKey** which identifies assembly and resource
  - prefer to use **DynamicResource** to locate in case of changes
- **Most controls use this technique to wire up colors and fonts**
  - perform lookup in dictionary to get appropriate resource

```
<StackPanel>
  <Button
    Width="{DynamicResource
      {x:Static SystemParameters.IconHeightKey}}"
    Background="{DynamicResource
      {x:Static SystemColors.ActiveCaptionBrushKey}}"
    Content="Some Button" />
</StackPanel>
```



# Replacing system resources

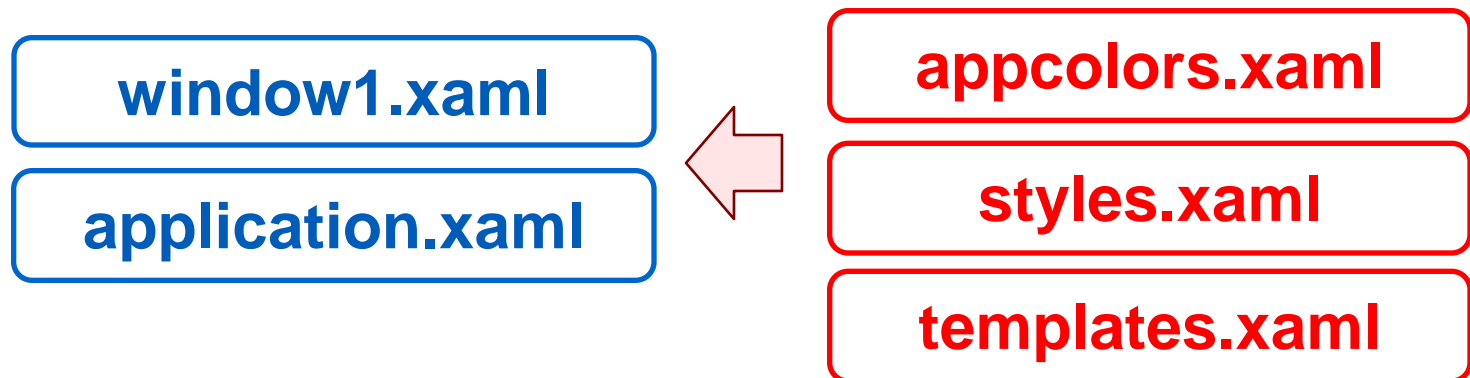
- Application can **replace** system values by adding new value to local dictionary using system key
  - WPF performs normal resource search to locate resource
  - can override underlying colors such as selection
  - does not work for all controls



```
<Application>
  <Application.Resources>
    <SolidColorBrush Color="Red"
      x:Key="{x:Static SystemColors.HighlightBrushKey}" />
  </Application.Resources>
</Application>
```

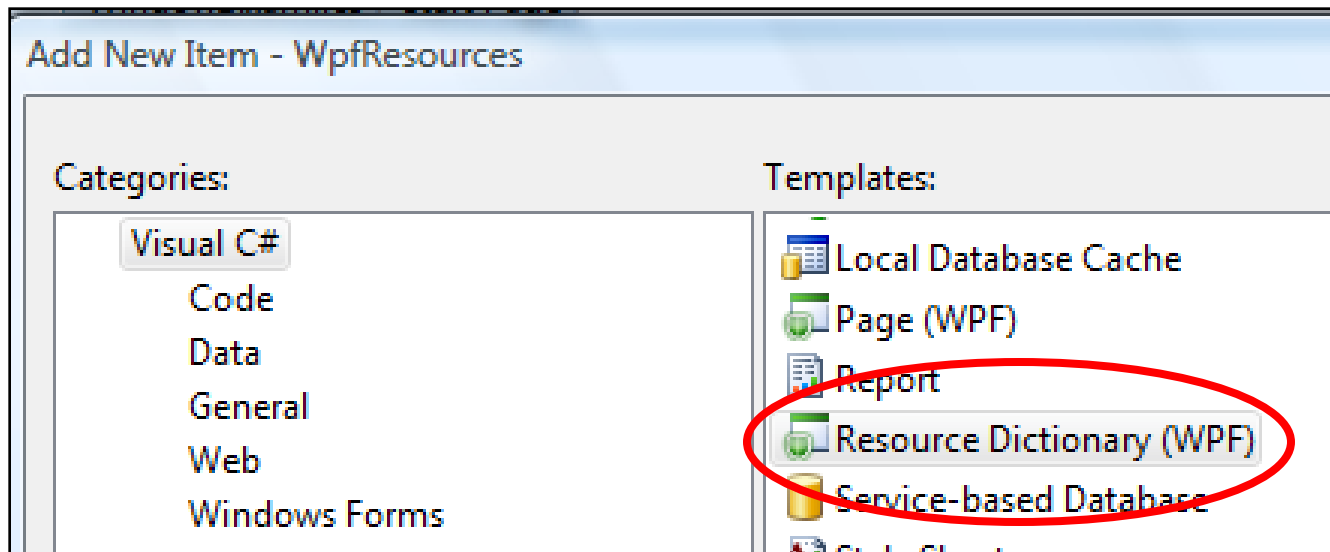
# Creating separate resource XAML files

- **Resources can also be placed into dedicated XAML files**
  - generally grouped by asset type in larger applications
  - visual designers can bundle up visual styles this way
- **Resource files must be *merged* into application resource tree**
  - can be merged in at any point (application, window, element)
  - allows WPF to locate resources through normal search



# Step 1: Create a ResourceDictionary file

- **Always contained in standalone ResourceDictionary**
  - add "Resource Dictionary (WPF)" to project



## Step 2: Add resources to dictionary

- **Add each resource to root ResourceDictionary**
  - can also name dictionary for easier access

```
colors.xaml
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <SolidColorBrush x:Key="redBrush" Color="#DD0000" />
    <SolidColorBrush x:Key="border" Color="Gold" />
    <SolidColorBrush x:Key="background" Color="Blue" />
    <Image x:Key="logo" Source="images/logo.jpg" x:Shared="false" />
    <s:String x:Key="copyright">2008 SomeCompany</s:String>
</ResourceDictionary>
```



## Step 3: merging in resource dictionaries

- Resource dictionaries must be merged in to be used
  - add each **dictionary** to **MergedDictionaries** collection
  - can also manipulate in code behind for dynamic changes

```
<Application>
  <Application.Resources>
    <ResourceDictionary>
      <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="colors.xaml" />
        <ResourceDictionary Source="templates.xaml" />
      </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
  </Application.Resources>
</Application>
```

```
<ResourceDictionary xmlns="..." xmlns:x="...">
  <SolidColorBrush x:Key="redBrush" ... />
</ResourceDictionary>
```

colors.xaml



# Sharing resources across applications

- **ResourceDictionary can be moved to secondary assembly**
  - can then be referenced by multiple applications

```
<Application>
  <Application.Resources>
    <ResourceDictionary>
      <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary
          Source="/acmecorp;Component/colors.xaml" />
      </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
  </Application.Resources>
</Application>
```

**Pack URI** syntax used to locate resources in other assemblies – takes the format: `/assembly;Component/file.xaml`



# Merging resources in code

- **Merging dictionaries in XAML creates the object each time**
  - inefficient if merged into multiple places in a single assembly

```
static class CustomResources {  
    static public ResourceDictionary Dictionary =  
        Application.LoadComponent(  
            new Uri(@"acmeCorp;Component/colors.xaml",  
                System.UriKind.Relative))  
        as ResourceDictionary  
}
```

```
public partial class CustomButton : Button {  
    public partial class CustomListBox : ListBox {  
        CustomListBox()  
        {  
            this.Resources.MergedDictionaries.Add(  
                CustomResources.Dictionary);  
        }  
    }  
}
```

**Load the resources once in code and then apply them to each specific target**



# Some final thoughts on organizing your XAML

- **Remember: resources can be placed at several levels**
  - `Window.Resources` – useful for page-specific resources
  - `Application.Resources` – for application-wide resources
  - separate `ResourceDictionary` files which are then merged into Window/Application/Control resources
- **Larger applications can benefit from a structured approach**
  - splitting resources out to separate XAML files based on usage
  - consider adopting a resource naming convention
- **Goal is keep XAML file length to small, easily managed size**
  - under 500 lines so it is easier to read



# Possible XAML organization

- **Primary assembly contains functional XAML (UI)**
  - and application specific resources for colors, brushes, data templates, etc.
- **Separate assembly contains resources**
  - styles, control themes, animations, 3D content, images, etc.
- **Group resources by type into separate XAML files**
  - helps developers quickly locate resources

## `Application.exe`

- UI + layout + code
- converters
- app colors/brushes
- data template

## `Application.UI.dll`

- control templates
- animations
- embedded images (binary)
- styles



# Styling your application with themes

- **May desired more than one "theme" for the application**
  - simple theme (traditional Windows look + feel)
  - colored theme (blue, red, etc.)
  - advanced theme (3D, animations, etc.)
- **Can supply named resources in theme assemblies and then merge at runtime through ResourceDictionary**

```
<Window
  Background="{DynamicResource
    Theme_BkBrush}" ...>
  <Button Style="{DynamicResource
    Theme_ButtonStyle}" />
  ...
</Window>
```

RedTheme.UI.dll

BlueTheme.UI.dll

AdvancedTheme.UI.dll

Theme\_BkBrush (Gradient)

Theme\_ButtonStyle

Image\_Logo (3D graphic)



# Summary

- **ResourceDictionary provides a nice way to supply common assets for elements**
  - WPF searches for appropriate resource using key
- **Consider refactoring resources into separate XAML files**
  - keep only window-specific resources in `window.xaml`
  - everything else into a separate resource dictionary for reuse
- **Proper segregation of UI and resources will provide for easier visual designer role integration!**

