WPF Resources

Agenda

- Resources
- Resource Dictionaries

What Is a Resource?

- The term *resource* has a very broad meaning in **WPF**
  - Any object can be a resource
  - A brush or a color used in various parts of a user interface could be a resource
  - Snippets of graphics or text can be resources
  - An object does not have to do anything special to qualify as a resource

- The resource handling infrastructure is entirely dedicated to making it possible to get hold of the resource you require
  - It doesn't care what the resource is
  - It simply provides a mechanism for identifying and locating objects

Why Resources

- If you want to build a graphically distinctive application, the resource system provides a straightforward way to *skin* your applications with customized yet consistent visuals

- Benefits:
  - **Efficiency**
    Resources let you define an object once and use it in several places in your markup
  - **Maintainability**
    Resources let you take low-level formatting details and move them to a central place where they're easy to change (like a constant)
  - **Adaptability**.
    Once certain information is separated from the rest of your application and placed in a resource section, it becomes possible to modify it dynamically

Creating and Using Resources

- At the heart of resource management is the **ResourceDictionary** class
  - Behaves much like an ordinary Hashtable
  - Allows objects to be associated with keys (strings are used as keys)
  - Provides an indexer that lets you retrieve those objects using these keys
- Naive ResourceDictionary programming

```
// Adding resources in C#
ResourceDictionary myDictionary = new ResourceDictionary( );
myDictionary.Add("myBrush", Brushes.Green);
myDictionary.Add("HW", "Hello, world");
Console.WriteLine(myDictionary["myBrush"]);
Console.WriteLine(myDictionary["HW"]);
```

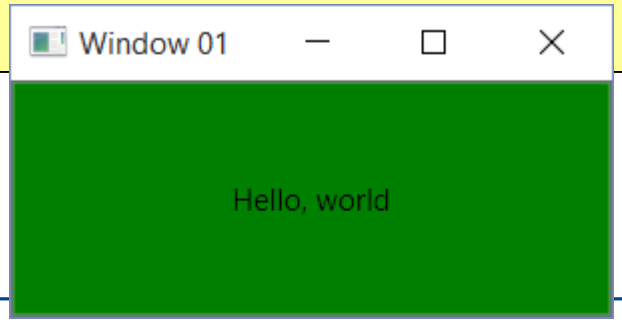
- Every WPF FrameworkElement has a Resources property which can hold a ResourceDictionary

```
Window.Resources = myDictionary;
```

Populating a ResourceDictionary from XAML

- The x:Key attribute specifies the key that identifies the resource in the dictionary
  - It is equivalent to the first parameter of the calls to Add in the previous example (C# code)

```
<Window
   xmlns:s="clr-namespace:System;assembly=mscorlib"
  …
  <Window.Resources>
    <SolidColorBrush x:Key="myBrush" Color="Green" />
    <s:String x:Key="HW">Hello, world</s:String>
  </Window.Resources>
   <Grid>
    <Button Background="{StaticResource myBrush}"
            Content="{StaticResource HW}">
  </Grid>
</window>
```



Resources ~ Objects

- You can instantiate any .NET class in the resources section
  - **including your own custom classes**
- But the class must be XAML-friendly
  - Must have a public zero-argument constructor
  - Writeable properties

```
<Window
  …
  <Window.Resources>
    <local:Agents x:Key="agents" />
    <SolidColorBrush x:Key="myBrush" Color="Green" />
  </Window.Resources>
  …
</window>
```

Resource Scope

- As well as providing a ResourceDictionary for every element, FrameworkElement also provides a **FindResource** method to retrieve resources

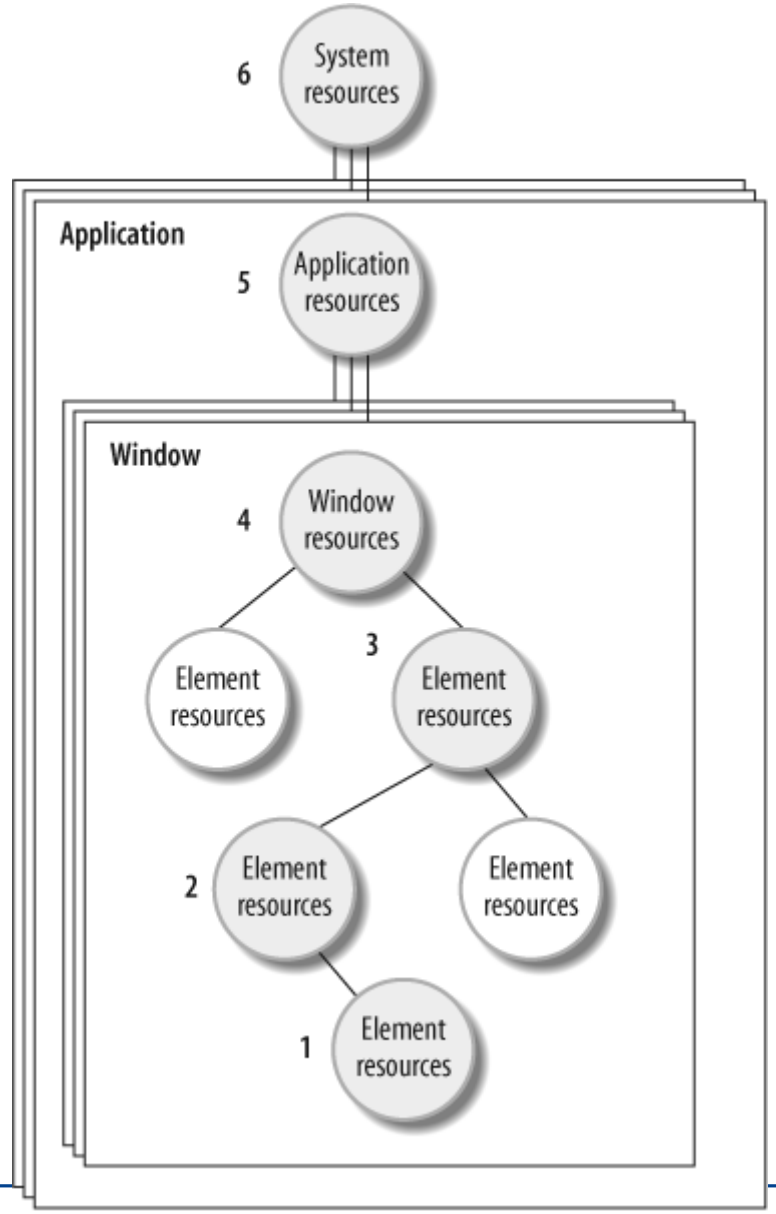
```
// Returns null
Brush b1 = (Brush) myGrid.Resources["myBrush"];

// Returns SolidColorBrush from Window.Resources
Brush b2 = (Brush) myGrid.FindResource("myBrush");
```

- The Grid doesn't have any resources, so the b1 variable will be set to null
  - because b2 is set using FindResources instead of the resource dictionary indexer, WPF considers all of the resources in scope, not just those directly set on the Grid
  - It starts at the Grid element, but then examines the parent, the parent's parent, and so on, all the way to the root element

FindResource

- If FindResource is called on the element labeled "1" in the figure,
  1. It will first look in that element's resource dictionary
     And if that fails, it will keep working its way up in the element hierarchy through the numbered items in order (the toplevel element is Window (no 4))
  2. It will then look in the Application object's resources (no. 5)
  3. If still not found it will look in the type theme resources (no. 6)
  4. And finally it will look in the system resources (no. 6)



Note:
The **`TryFindResource`**() method returns a null reference if a resource can't be found, rather than throwing an exception

Application Resources

- The application object has a Resources property that let us define resources global to the application
- This allows us to share resources among pages, windows and controls

```
<Application x:Class="ResourcesExample.App“
  xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml“
  StartupUri="Window1.xaml“
  >
  <Application.Resources>
    <LinearGradientBrush x:Key="myBrush" StartPoint="0,0" EndPoint="1,1">
      <LinearGradientBrush.GradientStops>
        <GradientStop Offset="0" Color="Red"/>
        <GradientStop Offset="1" Color="Black"/>
      </LinearGradientBrush.GradientStops>
    </LinearGradientBrush>
  </Application.Resources>
</Application>
```

The System Scope Resources

- WPF uses the system scope to define brushes, fonts, and metrics that the user can configure at a system-wide level
- All the built-in controls rely on the system scope to provide styles and templates suitable for the current OS theme
- The keys for these are provided as static properties of the **SystemColors**, **SystemFonts**, and **SystemParameters** classes, respectively.
  - These classes define more than 400 resources
    (consult the SDK documentation or MSDN for each class to see the complete set)

```
Brush toolTipBackground = (Brush)
                  myGrid.FindResource(SystemColors.InfoBrushKey);
```

- These system resource classes use objects rather than strings as resource keys.
  - This avoids the risk of naming collisions
  - System resources are always identified by a specific object, so there will never be any ambiguity between them and your own named resources

Dynamic Resources in XAML

- If you want to change the resources at runtime, then you must bind to resource using DynamicResource binding

```
<Window.Resources>
    <SolidColorBrush x:Key="myBrush" Color="Green" />
</Window.Resources>
    <Button Background="{DynamicResource myBrush}" . . . />
```

```
private void Button_Click(object sender, RoutedEventArgs e)
{
   Color color = Color.FromRgb((byte)rnd.Next(255),
                        (byte)rnd.Next(255), (byte)rnd.Next(255));
   Brush brush = new SolidColorBrush(color);
   this.Resources["myBrush"] = brush;
}
```

Dynamic Resources in C#

- In C# you use the SetResourceReference() function to bind dynamically to a resource

```
public Window1()
{
   ResourceDictionary myDictionary = new ResourceDictionary();
   myDictionary.Add("myBrush", Brushes.Green);
   this.Resources = myDictionary;
   . . .
   btn.SetResourceReference(Button.BackgroundProperty, "myBrush");
}

void btn_Click(object sender, RoutedEventArgs e)
{
   Color color = Color.FromRgb((byte)rnd.Next(255),
                 (byte)rnd.Next(255), (byte)rnd.Next(255));
   Brush brush = new SolidColorBrush(color);
   this.Resources["myBrush"] = brush;
}
```

Nonshared Resources

- When you use a resource in multiple places, you're using the same object instance → *sharing*
- This is not always what you want, so to turn off sharing, you use the Shared attribute, as shown here:

```
<Window.Resources>
    <Image x:Key="myImage" x:Shared="False" ... />
</Window.Resources>
```

RESOURCE DICTIONARIES

Creating a Resource Dictionary

- If you want to share resources between multiple projects you create a **resource dictionary**
- A resource dictionary is simply a XAML document that does nothing but store the resources you want to use in a ResourceDictionary

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

  <SolidColorBrush x:Key="myBrush" Color="Green" />
  <s:String x:Key="HW">Hello, world</s:String>
</ResourceDictionary>
```

- When you add a resource dictionary to an application, make sure the Build Action is set to Page
  - This ensures that your resource dictionary is compiled to BAML for best performance

Using a Resource Dictionary

- To use a resource dictionary, you need to merge it into a resource collection somewhere in your application
  - It's common to merge it into the resources collection for the application
  - Notice: duplicate resource keys are not allowed in a merged ResourceDictionary
    - you'll receive a XamlParseException when you compile your application!

```
<Application x:Class="Resources.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="Menu.xaml" >
  <Application.Resources>
    <ResourceDictionary>
      <ResourceDictionary.MergedDictionaries>
        <ResourceDictionary Source="AppBrushes.xaml"/>
        <ResourceDictionary Source="WizardBrushes.xaml"/>
      </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
  </Application.Resources>
</Application>
```

Sharing Resources Between Applications

- You could copy and distribute the XAML file that contains the resource dictionary
  - This is the simplest approach
- A more structured approach is to compile your resource dictionary in a separate class library assembly and distribute that component instead

How To Extract a Resource From a DLL v1

- The most straightforward solution is to use code that creates the appropriate ResourceDictionary object
- You don’t need to assign resources manually
  - Any DynamicResource references you have in your window will be automatically reevaluated when you load a new resource dictionary

```
ResourceDictionary resourceDictionary = new ResourceDictionary();
resourceDictionary.Source = new Uri(
    "ResourceLibrary;component/ReusableDictionary.xaml",
    UriKind.Relative);
```

How To Extract a Resource From a DLL v2

```
namespace ResourceLibrary
{
   public class CustomResources
   {
      public static ComponentResourceKey SadTileBrush
      {
         get
         {
            return new ComponentResourceKey(
               typeof(CustomResources), "SadTileBrush");
         }
      }
   }
}
```

```
<Button Background="{DynamicResource {x:Static
res:CustomResources.SadTileBrush}}"
        Padding="5" Margin="5"
        FontWeight="Bold" FontSize="14">
      A Resource From ResourceLibrary
    </Button>
```