WPF – Resource and Styles

Windows Programming Course

Agenda

1. Resources
2. Styles and Triggers
3. Data Template

1.
Resources

Resource in WPF

WPF allows you to define resources in code or in a variety of places in your markup (along with specific controls, in specific windows, or across the entire application).

Resources have these important benefits:

- Efficiency
- Maintainability
- Adaptability

The Resources Collection

Every Window/UserControl element includes a Resources property, which stores a dictionary collection of resources. The resources collection can hold any type of object, indexed by string. E.g.:

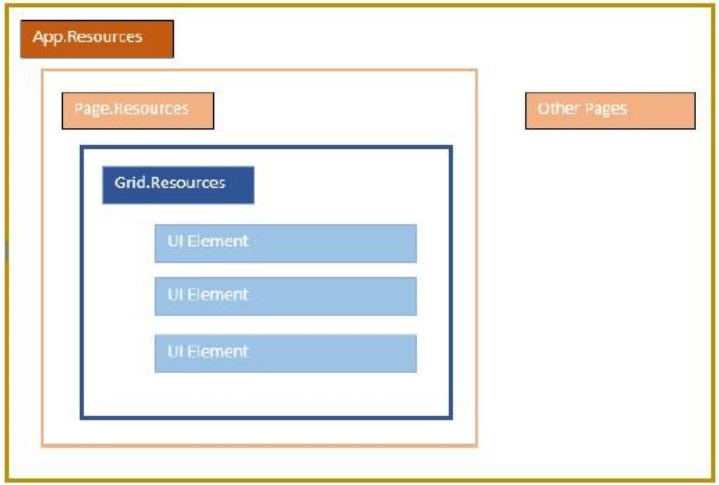
```
<Window ...>
  <Window.Resources>
    <ImageBrush x:Key="TileBrush" TileMode="Tile" ViewportUnits="Absolute"
Viewport="0 0 32 32" ImageSource="happyface.jpg" Opacity="0.3">
    </ImageBrush>
    <sys:String x:Key="strHelloWorld">Hello, world!</sys:String>
  </Window.Resources>
  <Grid>
    <Button Background="{StaticResource TileBrush}" Margin="5" Padding="5"
FontWeight="Bold" FontSize="14"> A Tiled Button</Button>
  </Grid>
</Window>
```

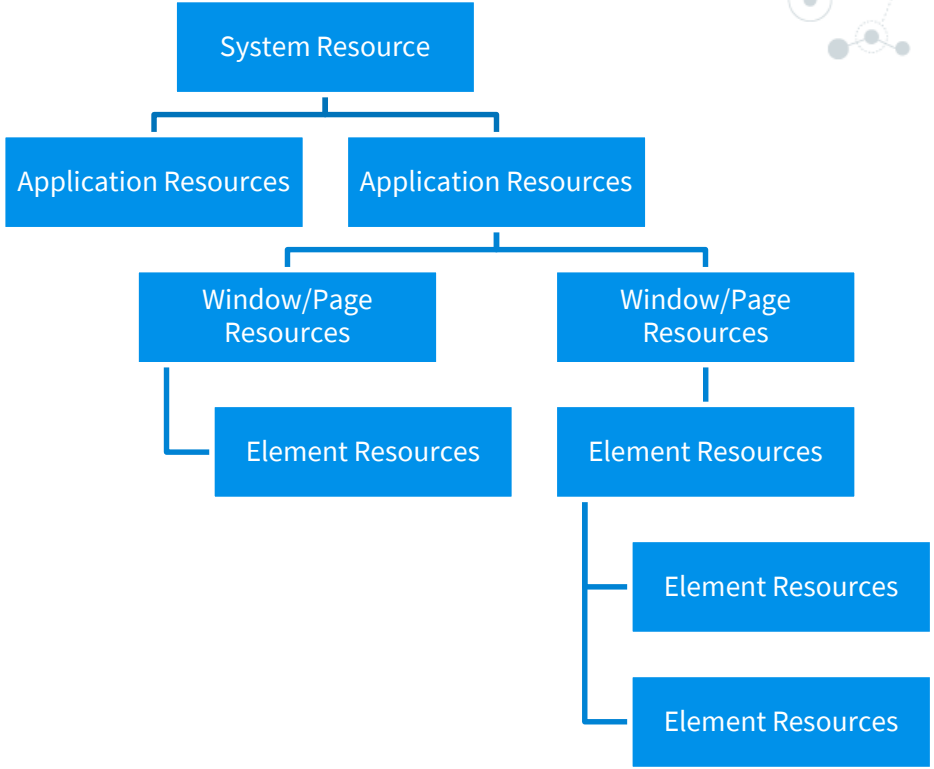
The Hierarchy of Resources

Every element has its own resource collection, and WPF performs a recursive search up your element tree to find the resource you want:

- First checks the current element's Resources collection.
- If not found, it checks the parent element, its parent, etc, until it reaches the root element.
- If all that fails, it looks in the resources collection of the Application object.
- If that fails, it looks at the system default resources collection.
- If that fails, it throws an InvalidOperationException.

The Hierarchy of Resources (cont.)





Static and Dynamic Resources

There are two ways to access a logical resource:

- Statically with **StaticResource**, meaning that the resource is applied only once when it is first needed.
- Dynamically with **DynamicResource**, meaning that the resource is reapplied every time it changes.
  - A consumer of the resource sees changes, e.g., the resource is linked.

Application Resources

The **App.xaml** file can contain resources just like the window and any kind of WPF control.

When storing resource in App.xaml, they are globally accessible in all of windows and user controls of the project.

```
<Application x:Class="WpfTutorialSamples.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:sys="clr-namespace:System;assembly=mscorlib"
  StartupUri="MainWindow.xaml">
  <Application.Resources>
    <sys:String x:Key="strHelloWorld">Hello, world!</sys:String>
  </Application.Resources>
</Application>
```

Resource Dictionaries

A resource dictionary is simply a XAML document that does nothing but store the resources you want to use.

Creating a Resource Dictionary:

Right Click on the Project -> Add -> New Item -> Select **Resource Dictionary**

```
<ResourceDictionary
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <ImageBrush x:Key="TileBrush" TileMode="Tile"
ViewportUnits="Absolute" Viewport="0 0 32 32"
ImageSource="happyface.jpg" Opacity="0.3">
  </ImageBrush>
</ResourceDictionary>
```

Resource Dictionaries – Using

Using a Resource Dictionary:

```
<Window.Resources>
  <ResourceDictionary>
    <ResourceDictionary.MergedDictionaries>
      <ResourceDictionary Source="ResourceDictionary.xaml"/>
    </ResourceDictionary.MergedDictionaries>

    <System:Double x:Key="ButtonWidth">80</System:Double>
  </ResourceDictionary>
</Window.Resources>
```

2.
Styles and Triggers

Style of WPF

A style is a collection of property values that can be applied to an element. The WPF style system plays a similar role to the Cascading Style Sheets (CSS) standard in HTML markup.

WPF styles also support **triggers**, which allow you to change the style of a control when another property is changed (as you'll see in this chapter), and they can use **templates** to redefine the built-in appearance of a control.

Style – Examples

In the example, you would like to apply a specific FontFamily, FontSize and FontWeight to all button in the application:

```
<Window.Resources>
  <FontFamily x:Key="ButtonFontFamily">Times New Roman</FontFamily>
  <sys:Double x:Key="ButtonFontSize">18</s:Double>
  <FontWeight x:Key="ButtonFontWeight">Bold</FontWeight>
</Window.Resources>

<Button Padding="5" Margin="5"
  FontFamily="{StaticResource ButtonFontFamily}"
  FontWeight="{StaticResource ButtonFontWeight}"
  FontSize="{StaticResource ButtonFontSize}">
A Customized Button
</Button>
```

Style – Examples (cont.)

Using Style to avoid code duplication:

```
<Window.Resources>
  <Style x:Key="BigFontButtonStyle" TargetType="Button">
    <Setter Property="FontFamily" Value="Times New Roman" />
    <Setter Property="FontSize" Value="18" />
    <Setter Property="FontWeight" Value="Bold" />
  </Style>
</Window.Resources>

<Button Padding="5" Margin="5" Style="{StaticResource BigFontButtonStyle}">
  A Customized Button
</Button>
```

Style – Creating

```
<Window.Resources>
  <Style TargetType="TextBlock">
    <Setter Property="HorizontalAlignment" Value="Center" />
    <Setter Property="FontFamily" Value="Comic Sans MS"/>
    <Setter Property="FontSize" Value="14"/>
  </Style>
  <!--A Style that extends the previous TextBlock Style with an x:Key of TitleText-->
  <Style BasedOn="{StaticResource {x:Type TextBlock}}"
         TargetType="TextBlock"
         x:Key="TitleText">
    <Setter Property="FontSize" Value="26"/>
  </Style>
</Window.Resources>
```

Style – Applying implicitly

```
<Window.Resources>
    <!--A Style that affects all TextBlocks-->
  <Style TargetType="TextBlock">
    <Setter Property="HorizontalAlignment" Value="Center" />
    <Setter Property="FontFamily" Value="Comic Sans MS"/>
    <Setter Property="FontSize" Value="14"/>
  </Style>
</Window.Resources>

<StackPanel>
  <TextBlock>My Pictures</TextBlock>
  <TextBlock>Check out my new pictures!</TextBlock>
</StackPanel>
```

Style – Applying explicitly

```
<Window.Resources>
    <!--A Style that affects all TextBlocks-->
  <Style x:Key="TitleText" TargetType="TextBlock">
    <Setter Property="HorizontalAlignment" Value="Center" />
    <Setter Property="FontFamily" Value="Comic Sans MS"/>
    <Setter Property="FontSize" Value="14"/>
  </Style>
</Window.Resources>

<StackPanel>
  <TextBlock Style="{StaticResource TitleText}">My Pictures</TextBlock>
  <TextBlock>Check out my new pictures!</TextBlock>
</StackPanel>
```

Triggers

Using triggers to automate simple style changes that would ordinarily require boilerplate event handling logic. For example, changing control's background when getting mouse's focus.

Trigger types:

Name	Description
Trigger	This is the simplest form of trigger. It watches for a change in a dependency property and then uses a setter to change the style.
MultiTrigger	This is similar to Trigger but combines multiple conditions.
DataTrigger	This trigger works with data binding.
MultiDataTrigger	This combines multiple data triggers.
EventTrigger	This is the most sophisticated trigger. It applies an animation when an event occurs.

A Simple Trigger

In the example below, you create mouseover and focus effects by changing Foreground:

```
<Style x:Key="BigFontButton">
  <Style.Setters>
    <Setter Property="Control.FontFamily" Value="Times New Roman" />
    <Setter Property="Control.FontSize" Value="18" />
  </Style.Setters>
  <Style.Triggers>
    <Trigger Property="Control.IsFocused" Value="True">
      <Setter Property="Control.Foreground" Value="DarkRed" />
    </Trigger>
  </Style.Triggers>
</Style>
```

3.

Data Template

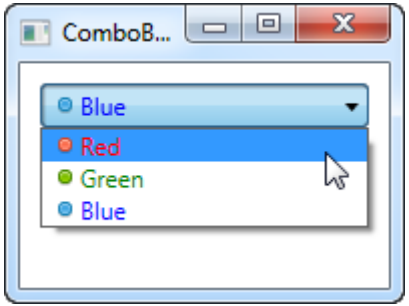
Data Template

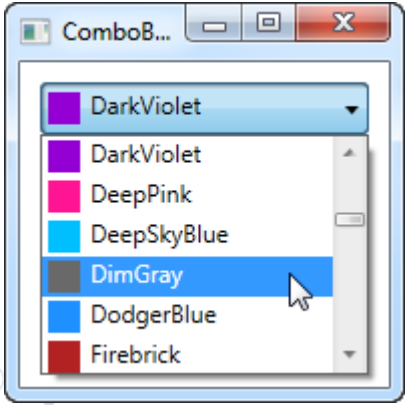
A data template is a chunk of XAML markup that defines how a bound data object should be displayed.

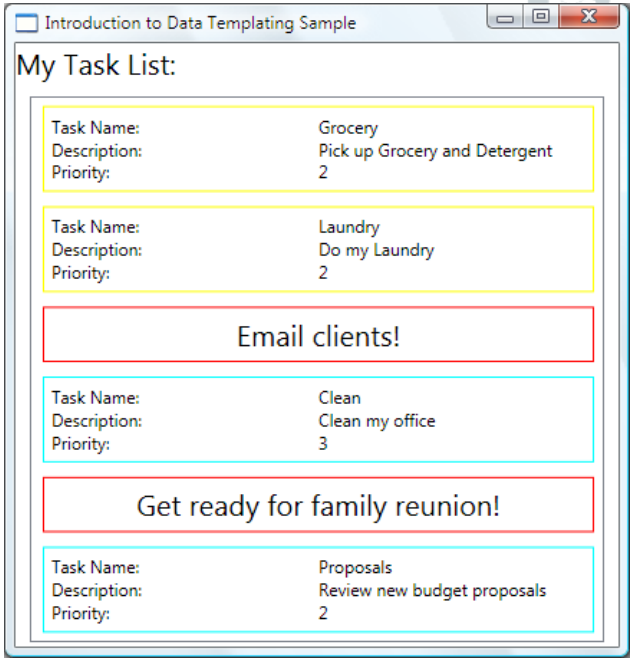
Two types of controls support data templates:

- Content controls support data templates through the **ContentTemplate** property.
- List controls (controls that derive from ItemsControl) support data templates through the **ItemTemplate** property. This template is used to display each item from the collection (or each row from a DataTable) that you've supplied as the ItemsSource

Example


ComboB...
Blue
Red
Green
Blue


ComboB...
DarkViolet
DarkViolet
DeepPink
DeepSkyBlue
DimGray
DodgerBlue
Firebrick


Introduction to Data Templating Sample
My Task List:
Task Name: Grocery
Description: Pick up Grocery and Detergent
Priority: 2
Task Name: Laundry
Description: Do my Laundry
Priority: 2
Email clients!
Task Name: Clean
Description: Clean my office
Priority: 3
Get ready for family reunion!
Task Name: Proposals
Description: Review new budget proposals
Priority: 2

Creating a Data Template

```
<ComboBox Grid.Row="2" ItemsSource="{Binding ListClass}">
  <ComboBox.ItemTemplate>
    <DataTemplate>
      <StackPanel Orientation="Horizontal">
        <TextBlock Text="{Binding ClassName}" FontWeight="Bold"/>
        <TextBlock Text=" - " />
        <TextBlock Text="{Binding Year}" />
      </StackPanel>
    </DataTemplate>
  </ComboBox.ItemTemplate>
</ComboBox>
```

Creating a Data Template (cont.)

```
<DataTemplate x:Key="classTemplate">
  <StackPanel Orientation="Horizontal">
    <TextBlock Text="{Binding ClassName}" FontWeight="Bold"/>
    <TextBlock Text=" - " />
    <TextBlock Text="{Binding Year}" />
  </StackPanel>
</DataTemplate>

<ComboBox ItemsSource="{Binding ListClass}"
  ItemTemplate="{StaticResource classTemplate}">
</ComboBox>
```

Thanks!

Any questions?

You can find me at:

tranminhphuoc@gmail.com