Organizing XAML

Estimated time for completion: **45 minutes**

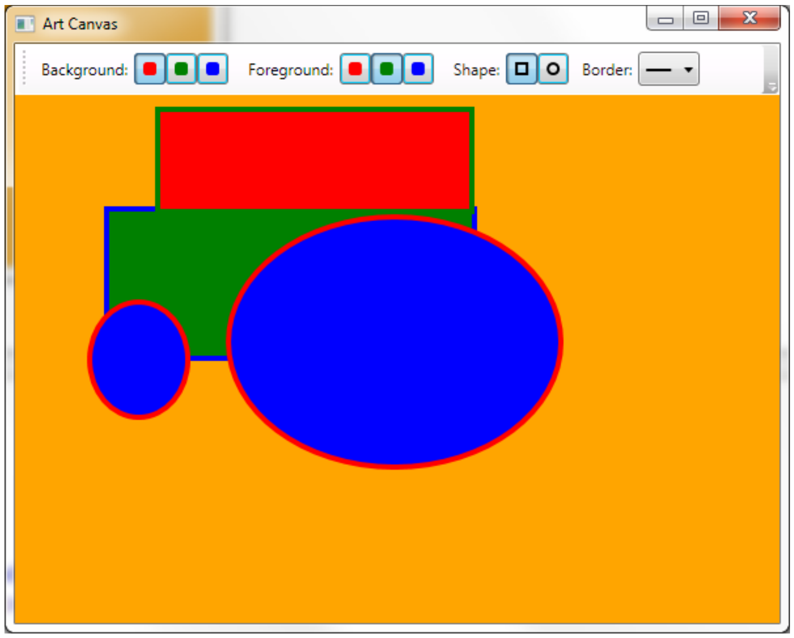
Goals:

- Learn how to define reusable resources in XAML
- Supply application-wide reusable resources
- Separate resources into individual XAML files

Overview:

In this lab you will learn the various options of organizing XAML and the resources that make up the visual aspects of your application. During this lab, we will also reinforce some of the things you've already learned such as using controls and shapes and handling mouse events.

We'll work with a simple art application which looks like the following:



Part 1 – Setup the application

In this part, we are going to examine the initial shell of the application and begin refactoring out duplicate settings to resources.

1. Open the starter application located in the **before** directory – it's called **ArtCanvas.sln**.
2. Build and run the application – it should be functional, allowing you to draw and move elements around on the window.
3. Open the **MainWindow.xaml** file – this is where all the XAML for the application is located. The code is fairly simple – it has a root `DockPanel` that owns a `ToolBar` and a Canvas where all the drawing actually happens.
4. Notice that everything is duplicated in several places, the same colors, sizes, margins, etc. are all applied on each element directly.
5. The first thing we are going to move out is brush colors. Notice that we are using the **Black** brush in quite a few places – if we wanted to change that color, we'd have to do it in every spot we used it. This is exactly what resources are for.
6. Create a `<Window.Resources>` section in your `<Window>` element. Inside that, create a `SolidColorBrush` with the color black and assign it a key of "blackBrush".
7. Next, replace all instances of the black brush within your current XAML with this new resource. Remember that to use an item in resources from XAML, we use either `{StaticResource}` or `{DynamicResource}`. In general, you should prefer `DynamicResource` because it allows resources to shift around more fluidly (i.e. be declared *after* usage). There are a few places where it cannot be used as you will see in the next module. For now, use `DynamicResource`.
8. Run the application and make sure that it has not changed visually. Your XAML should look something like this:

```
<Window ...>
   <Window.Resources>
      <SolidColorBrush x:Key="blackBrush" Color="Black" />
   </Window.Resources>
   ...
   <StackPanel>
      <ToggleButton ...>
         <Rectangle Stroke="{DynamicResource blackBrush}" ... />
      </ToggleButton>
   </StackPanel>
   ...
</Window>
```

9. Next, create three brush resources and replace the colors for the foreground and background toggle buttons. You can name them color1, color2 and color3. Once this is

done, you can change the colors in one place and affect both sets of buttons (Go ahead and try it!).

10. Next, let's move some of the numeric constants into resources. Look at each of the rectangles and you'll see that they are almost all identical – with the exception of the `Height` property. Move the `RadiusX/Y` and `Width/Height` properties into resources.
    a. **Hint**: you will need the **System** namespace from **mscorlib** declared `xmlns:System="clr-namespace:System;assembly=mscorlib"`

11. For example, here is the first set of `ToggleButtons`:

```
<Window.Resources>
   <SolidColorBrush x:Key="blackBrush" Color="Black" />
   <SolidColorBrush x:Key="color1" Color="Red" />
   <SolidColorBrush x:Key="color2" Color="Green" />
   <SolidColorBrush x:Key="color3" Color="Blue" />

   <System:Double x:Key="Radius">2.5</System:Double>
   <System:Double x:Key="RectSize">10</System:Double>
</Window.Resources>

<DockPanel>
   <ToolBar DockPanel.Dock="Top">
      <StackPanel Margin="5" Orientation="Horizontal"
                  Button.Click="ChangeBackground">
         <TextBlock VerticalAlignment="Center"
                    Margin="5,0">Background:</TextBlock>
         <ToggleButton x:Name="currentBackground" Padding="5"
                Tag="{DynamicResource color1}">
            <Rectangle Fill="{DynamicResource color1}"
                       RadiusX="{DynamicResource Radius}"
                       RadiusY="{DynamicResource Radius}"
                       Width="{DynamicResource RectSize}"
                       Height="{DynamicResource RectSize}" />
          </ToggleButton>
...
```

12. Run the application – it should still visually look the same, but we've consolidated a lot of the properties off to the resources.

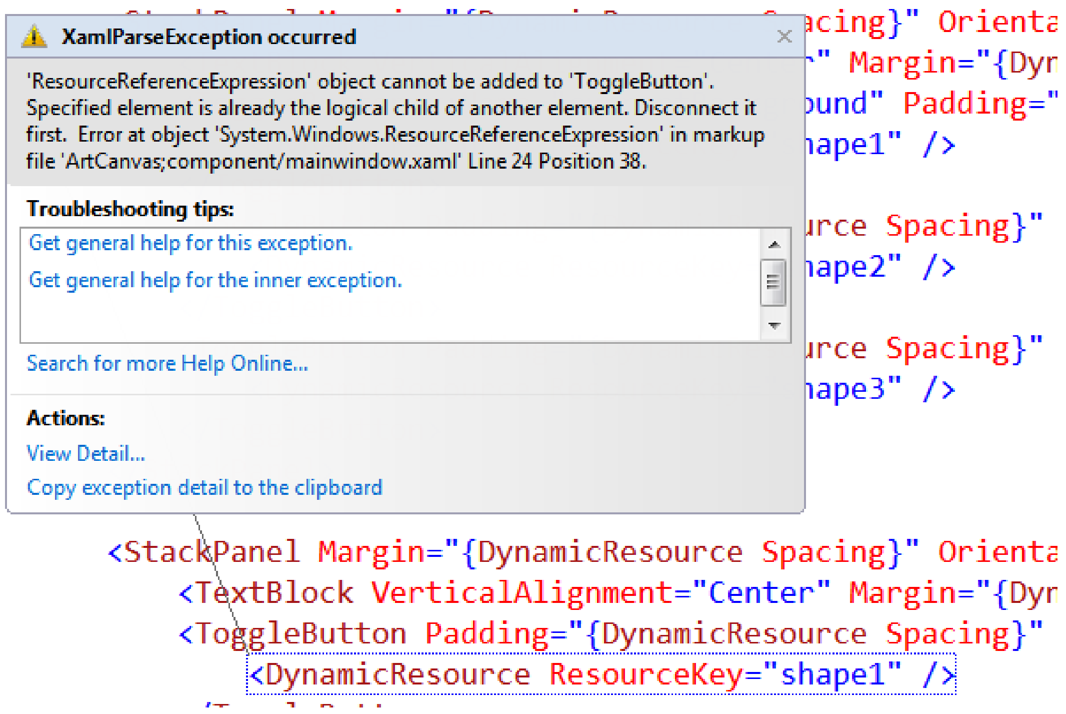
13. Move some of the other common properties off into resource keys until you have 5-6 properties off in resources. Try changing the resources to see the global impact it has on the application.

14. As a last test, let's move the `Rectangle` shapes we are using with the `ToggleButtons` into resources. We have two copies of each rectangle being used – one for background selection and one for foreground selection. Move the Rectangle and

assign it a key and they use a `DynamicResource` element tag to associate it with the `ToggleButton`:

```
<Rectangle x:Key="shape1" Fill="{DynamicResource color1}"
     RadiusX="{DynamicResource Radius}"
     RadiusY="{DynamicResource Radius}"
     Width="{DynamicResource RectSize}"
     Height="{DynamicResource RectSize}" />
...
<ToggleButton x:Name="currentBackground"
       Padding="{DynamicResource Spacing}" Tag="{DynamicResource color1}">
   <DynamicResource ResourceKey="shape1" />
</ToggleButton>
```

15. Run the application – what happened? Can you remember how to fix this?



XamlParseException occurred

'ResourceReferenceExpression' object cannot be added to 'ToggleButton'. Specified element is already the logical child of another element. Disconnect it first. Error at object 'System.Windows.ResourceReferenceExpression' in markup file 'ArtCanvas;component/mainwindow.xaml' Line 24 Position 38.

Troubleshooting tips:

Get general help for this exception.

Get general help for the inner exception.

Search for more Help Online...

Actions:

View Detail...

Copy exception detail to the clipboard

```
<StackPanel Margin="{DynamicResource Spacing}" Orienta
    <TextBlock VerticalAlignment="Center" Margin="{Dyr
    <ToggleButton Padding="{DynamicResource Spacing}"
        <DynamicResource ResourceKey="shape1" />
```

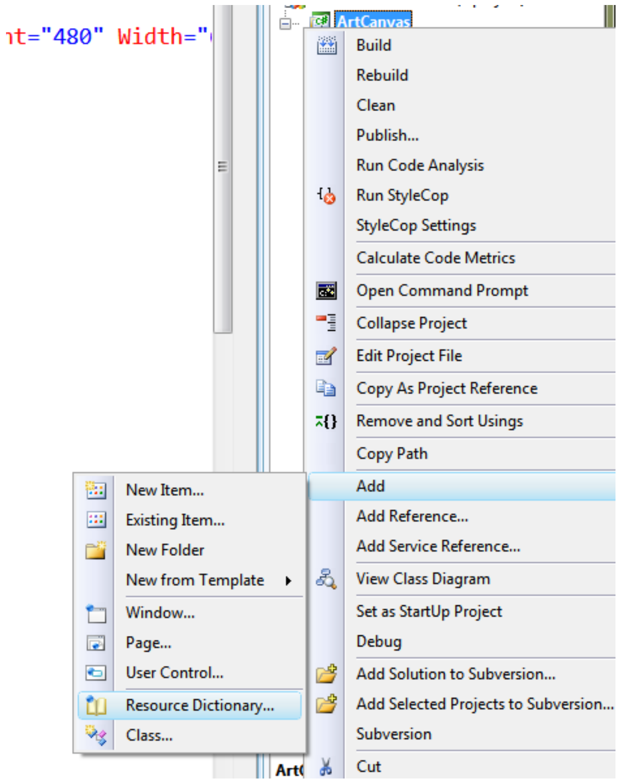
16. Add the **x:Shared=”false”** tag to each of the `Rectangle` shapes and try it again – it should work now, but we aren’t sharing the shapes – just defining them in one place.

Part 3 – Refactor resource into separate XAML files

Now we’ve got a lot of resources contained within the Window XAML file. It works and is reusing resources but has bloated our XAML considerably and makes it hard to read through. In this step, we are going to separate out different pieces into separate files.

1. The first thing we are going to move out is brush colors. We’ll place colors off in their own resource file – **Colors.xaml** .

a. Create a new Resource Dictionary named “Colors.xaml” in your project – use the “Add | Resource Dictionary” context menu option available on the project:



b. Move all the brushes into this dictionary. Since they already have keys you will only need to move the location of the brush itself. It should look something like this:

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

    <SolidColorBrush x:Key="blackBrush" Color="Black" />
    <SolidColorBrush x:Key="color1" Color="Red" />
    <SolidColorBrush x:Key="color2" Color="Green" />
    <SolidColorBrush x:Key="color3" Color="Blue" />

</ResourceDictionary>
```

2. Next, move the numeric constants into a XAML file called “Layout.xaml”. Use the same steps you used for colors. Remember to move the **xmlns** definition as well!
3. Finally, move the rectangles into a resource dictionary called “Shapes.xaml”.
4. Now we need to integrate these new dictionaries into the application. Open the App.xaml file and merge each new dictionary into the application resources.

   a. Create a `<ResourceDictionary>` in the application resources section

b. Assign the `ResourceDictionary.MergedDictionaries` to a list of `ResourceDictionary` tags where the `Source` is set to each of your named files.

c. You can define them in any order as long as you used `DynamicResource` everywhere. If you used `StaticResource` then they will need to be in the usage order (**Colors**, **Layout**, **Shapes**) because Shapes relies on the other two.

d. Once you are finished, it will look something like:

```
<Application x:Class="ArtCanvas.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="MainWindow.xaml">
    <Application.Resources>

      <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
          <ResourceDictionary Source="Colors.xaml" />
          <ResourceDictionary Source="Layout.xaml" />
          <ResourceDictionary Source="Shapes.xaml" />
        </ResourceDictionary.MergedDictionaries>
      </ResourceDictionary>

    </Application.Resources>
</Application>
```

5. Examine the structure of the application now – do you think this is easier to find the composite elements that make up the application?

6. Run the application and note that it still looks the same. We've simply reorganized where things are placed and provided some structure to the source code.

Part 4 – Using Dynamic Resources

In this final section, we are going to change the background color of our canvas to be the same color as the desktop.

1. Set the Background property of the Canvas in the main window to be assigned the static property `SystemColors.DesktopBrushKey`

   a. Use the `StaticResource` markup extension.

2. Run the application. Note the new background color.

3. Open the desktop properties and change the background color. Does the application background change too?

4. Stop the application and change the code to use a `DynamicResource` static extension.

5. Run the application.

6. Open the desktop properties and change the background color – notice how it changes in the running application and is changed dynamically.

Solutions

The full solution to the lab is located in the **after** folder associated with this lab. There are versions for Visual Studio 2008 SP1 and Visual Studio 2010.