Framework Architecture

Estimated time for completion: 45 minutes

Overview:

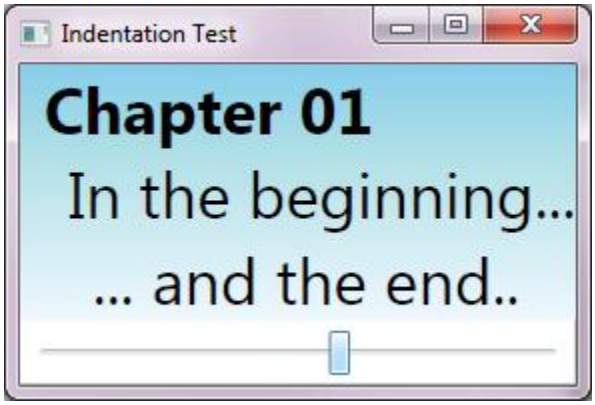
In this lab you will look at dependency properties and the visual elements of WPF.

Goals:

- Understand how to create and use Dependency Properties
- Examine the Visual/Logical tree

Part 1 – Creating and using Dependency Properties

In this part of the lab you will create a custom panel which does an outline format:



The XAML that creates this visual should look like:

```
<me:OutlinePanel x:Name="panel"
    HorizontalAlignment="Left" VerticalAlignment="Top"
    TextElement.FontSize="24pt" IndentSize="10">
   <me:OutlinePanel.Background>
      <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
         <GradientStop Offset="0" Color="SkyBlue" />
         <GradientStop Offset=".35" Color="PowderBlue" />
         <GradientStop Offset="1" Color="AliceBlue" />
      </LinearGradientBrush>
```

```
    </me:OutlinePanel.Background>
    <TextBlock Text="Chapter 01"
       me:OutlinePanel.Indentation="1" FontWeight="Bold"  />
    <TextBlock Text="In the beginning..."
       me:OutlinePanel.Indentation="2" />
    <TextBlock Text="... and the end.."
       me:OutlinePanel.Indentation="3" />
 </me:OutlinePanel>
```

1. Create a new WPF application using Visual Studio.
2. Create a new class in the project called "OutlinePanel". Derive this new class from the **System.Windows.Controls.Panel** base class.
3. Add the following code to the class:

```
protected override Size MeasureOverride(Size availableSize)
{
   Size totalSize = new Size();

   foreach (UIElement child in InternalChildren)
   {
      child.Measure(availableSize);
      Size childSize = child.DesiredSize;
      totalSize.Height += childSize.Height;
      totalSize.Width = Math.Max(totalSize.Width,
                  childSize.Width + CalculateIndentation(child));
   }
   return totalSize;
}

protected override Size ArrangeOverride(Size finalSize)
{
   double yPos = 0.0;
   Size usedSize = new Size();

   foreach (UIElement child in InternalChildren)
   {
      Size childSize = child.DesiredSize;
      Point startPos = new Point(CalculateIndentation(child), yPos);
      yPos += childSize.Height;

      child.Arrange(new Rect(startPos, childSize));
      usedSize.Height += childSize.Height;
      usedSize.Width = Math.Max(startPos.X + childSize.Width,
                                  usedSize.Width);
   }
   return usedSize;
}
```

```
private double CalculateIndentation(UIElement child)
{
   return 0.0;
}
```

Notice that both methods use the **Panel.InternalChildren** property? We could also use **Children**, internally it calls this property. This is specifically for panel-derivatives.

4. Now, your job is to add two dependency properties to the panel.
    a. The first property will be attached to child elements so register it with the **DependencyProperty.RegisterAttached** method. Name the property "Indentation" and make it an integer. Go ahead and write the static getter and setter to go with it.
    b. The second property will be an instance property of the panel – it will change the pixel sizing used for indentations. Name it "IndentSize" and make it a double. Go ahead and write the instance property wrapper for it.
    c. When you are finished, the code should look something like:

```
public static readonly DependencyProperty IndentSizeProperty =
      DependencyProperty.Register("IndentSize",
               typeof(double), typeof(OutlinePanel));

public readonly static DependencyProperty IndentationProperty =
          DependencyProperty.RegisterAttached("Indentation",
                typeof(int), typeof(OutlinePanel));

public double IndentSize
{
   get { return (double)GetValue(IndentSizeProperty); }
   set { SetValue(IndentSizeProperty, value); }
}

public static int GetIndentation(DependencyObject e)
{
   return (int)e.GetValue(IndentationProperty);
}

public static void SetIndentation(DependencyObject e, int value)
{
  e.SetValue(IndentationProperty, value);
}
```

    d. Finally, fill in the implementation of the **CalculateIndentation** method. It should multiply the IndentSize with the indentation of the UIElement:

```
private double CalculateIndentation(UIElement child)
{
```

```
    return Convert.ToDouble(GetIndentation(child) * IndentSize);
}
```

5. Now, switch to the window1.xaml file and add the XAML from above into the root Grid to test your panel. Run the application and make sure it works.
6. Add a new row into the Grid and place a slider into the row. It should have a range of "1" to "10" with a default value of "3". Add a ValueChanged handler to it.
7. Put a name onto the OutlinePanel in your XAML (the solution uses "panel"). In the ValueChanged handler, change the IndentSize property of the panel.
    a. Run the application and slide the slider. What happens? Can you explain the result?
8. To fix the issue, we need to change the dependency property backing the IndentSize property. It should affect the Measure and Arrange phases of layout when it is changed. Add a new parameter to the registration using a new **FrameworkPropertyMetadata** object and use a default value of "5" with the appropriate metadata flags to invalidate layout when the value is changed.
9. The final result should look like:

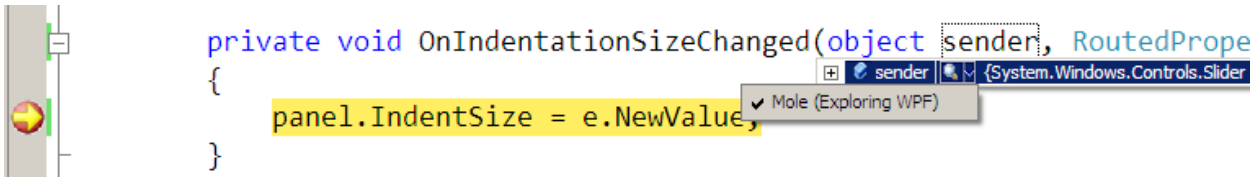
```
public static readonly DependencyProperty IndentSizeProperty =
            DependencyProperty.Register("IndentSize",
                    typeof(double), typeof(OutlinePanel),
                    new FrameworkPropertyMetadata(5.0,
                            FrameworkPropertyMetadataOptions.AffectsArrange |
FrameworkPropertyMetadataOptions.AffectsMeasure));
```

10. Run the application again and slide the slider. It should now properly change the indentation of all the text.

Part 2 – Examining the Visual and Logical Tree (Optional)

In this part of the lab you will use the Mole visualizer to examine the logical and visual tree for your application.

1. Download Mole from http://www.codeproject.com/KB/macros/MoleForVisualStudioEdit.aspx or, if you are using Visual Studio 2010, you can use the built-in debug visualizer which works pretty much like Mole.
2. Copy the .DLL into the "My Documents\Visual Studio 2008\Visualizers" directory.
3. Debug the application.
4. Set a breakpoint on the slider value change method and change the slider so it hits.
5. Hover over the "sender" parameter



6. Click on the magnifying glass to start Mole.
7. Compare the visual tree to the logical tree in the Mole tool – look especially at the difference in the Slider definition.

Solution

The completed lab solution is available in the **after** folder supplied with the lab. It includes solutions for both Visual Studio 2008 and 2010.