|  |  |
| --- | --- |
|  | [**WPF Tutorial - Creating A Custom Panel Control**](http://www.switchonthecode.com/tutorials/wpf-tutorial-creating-a-custom-panel-control) |

Posted in:

* [Tutorials](http://www.switchonthecode.com/tutorial)
* [.NET](http://www.switchonthecode.com/frameworks/dotnet)
* [.NET 3.0](http://www.switchonthecode.com/frameworks/dotnet/dotnet-30)
* [C#](http://www.switchonthecode.com/languages/csharp)
* [WPF](http://www.switchonthecode.com/frameworks/wpf)
* [XAML](http://www.switchonthecode.com/languages/xaml)

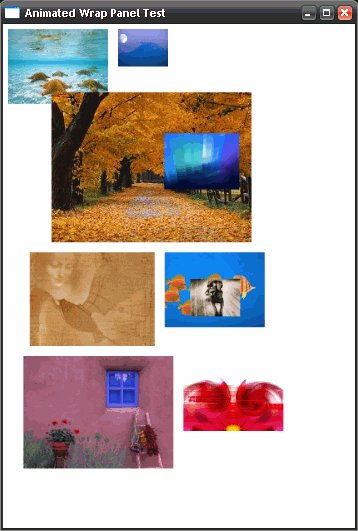
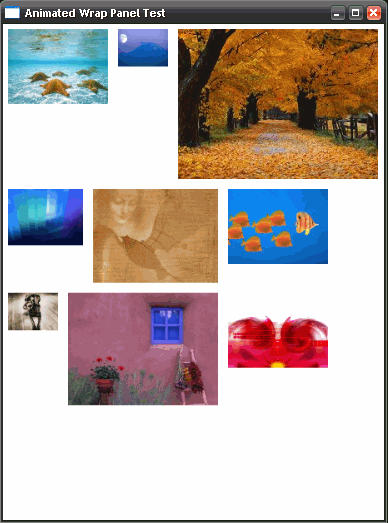
**Related Posts**

* [WPF Tutorial - Implementing IScrollInfo](http://www.switchonthecode.com/tutorials/wpf-tutorial-implementing-iscrollinfo)
* [WPF Tutorial - Using A Visual Collection](http://www.switchonthecode.com/tutorials/wpf-tutorial-using-a-visual-collection)
* [WPF Tutorial - Using An ItemsPanel](http://www.switchonthecode.com/tutorials/wpf-tutorial-using-an-itemspanel)
* [WPF Tutorial - How To Use Custom Cursors](http://www.switchonthecode.com/tutorials/wpf-tutorial-how-to-use-custom-cursors)

The world of WPF gives you an extreme amount of flexibility right off the bat - with things like styles, control templates, and the composability of almost anything, at first it seems like everything is right there at your fingertips. And while there is a lot immediately accessible, there are still cases where you have to get down into the nitty-gritty. Today we are going to take a look at how to create a WPF custom control - more specifically, a custom panel. The available panels in WPF are great (perhaps we will have a tutorial on how to use them all at some point in the future). But maybe you have a very specific need, and none of the panels quite work the way you want?

Well, in that case, it is custom controls to the rescue! The custom panel we are going to write today is going to be extremely similar to the wrap panel, except that we are going to add animation into the picture. When items in the panel need to wrap to the next line because the size of the panel changed, they won't just switch positions - they will animate to the new position. Useful? Not really. But it is a cool example and fun to play with.

Here's a couple screenshots of the panel in action (the second one is in the middle of an animated move):



Ok, so the first thing we need to do is create the skeleton of the class for the new custom control. In my case, I called it AnimatedWrapPanel, and it will extend the base Panel class:

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Windows;  
using System.Windows.Controls;  
using System.Windows.Data;  
using System.Windows.Documents;  
using System.Windows.Input;  
using System.Windows.Media;  
using System.Windows.Media.Imaging;  
using System.Windows.Navigation;  
using System.Windows.Shapes;  
using System.Windows.Media.Animation;  
  
namespace AnimatedWrapPanel  
{  
  public class AnimatedWrapPanel : Panel  
  {  
  
  }  
}

Ok, so there is our base structure. Technically, this is already a control, but it wouldn't actually do anything yet. To actually get it to do something useful, there are two very important methods that we need to override:

protected override Size MeasureOverride(Size availableSize)  
{  
  
}  
  
protected override Size ArrangeOverride(Size finalSize)  
{  
  
  
}

WPF uses a two pass layout system to determine the positions and sizes of parents and children. Each of these methods corresponds to one of those steps. In the first step, the Measure step, parents are supposed to ask their children what their desired size is, given an available size. In the second step, the Arrange step, parents position their children and tell their children how much size they are actually getting. There are a couple things to note about each of these steps, so lets take a look at them one at a time.

The simplest MeasureOverride method (that actually works) for a panel would be the following:

protected override Size MeasureOverride(Size availableSize)  
{  
  Size resultSize = new Size(0,0);  
  
  foreach (UIElement child in Children)  
  {  
    child.Measure(availableSize);  
    resultSize.Width = Math.Max(resultSize.Width, child.DesiredSize.Width);  
    resultSize.Height = Math.Max(resultSize.Height, child.DesiredSize.Height);  
  }  
  
  resultSize.Width = double.IsPositiveInfinity(availableSize.Width) ?  
      resultSize.Width : availableSize.Width;  
  
  resultSize.Height = double.IsPositiveInfinity(availableSize.Height) ?   
      resultSize.Height : availableSize.Height;  
  
  return resultSize;  
}

So what are we doing here? Well, first, in the MeasureOverride function, you are required to call Measure on every child. If you don't, then the child's desired size is never determined - and this will break the Arrange pass of the layout system. The other thing to note is that you can't just return the passed in availableSize as your desired size. This is because availableSize could potentially be infinite (think about inside of a scroll view) - but your desired size can never be infinite. So what we are doing here is using the size of our largest child to determine our desired size, but only when the availableSize is infinite.

Thats the bare minimum you can do in MeasureOverride - but for our custom panel, we need to do a little bit more than that. Heres what the MeasureOverride looks like in the AnimatedWrapPanel code:

protected override Size MeasureOverride(Size availableSize)  
{  
  Size infiniteSize = new Size(double.PositiveInfinity, double.PositiveInfinity);  
  double curX = 0, curY = 0, curLineHeight = 0;  
  foreach (UIElement child in Children)  
  {  
    child.Measure(infiniteSize);  
  
    if (curX + child.DesiredSize.Width > availableSize.Width)  
    { *//Wrap to next line*  
      curY += curLineHeight;  
      curX = 0;  
      curLineHeight = 0;  
    }  
  
    curX += child.DesiredSize.Width;  
    if(child.DesiredSize.Height > curLineHeight)  
      curLineHeight = child.DesiredSize.Height;  
  }  
  
  curY += curLineHeight;  
  
  Size resultSize = new Size();  
  
  resultSize.Width = double.IsPositiveInfinity(availableSize.Width) ?   
      curX : availableSize.Width;  
  
  resultSize.Height = double.IsPositiveInfinity(availableSize.Height) ?   
      curY : availableSize.Height;  
  
  return resultSize;  
}

Essentially, we are doing more work here to come back with a better desired size. We do the wrap layout, keeping track of how big we end up being. To every child, we hand 'infinite' as the available size, so the children can be as big as they want to be. And in the end, if our available space is infinite, we use the size we calculated through the layout, but otherwise we take up all the room.

Ok, so thats measure. How about arrange? Here is the simplest possible arrange method:

protected override Size ArrangeOverride(Size finalSize)  
{  
  foreach (UIElement child in Children)  
  {  
    child.Arrange(new Rect(0, 0, child.DesiredSize.Width, child.DesiredSize.Height));  
  }  
  
  return finalSize;  
}

Again, you will want to call Arrange on every child, otherwise that child will not get placed/rendered. What we are doing here is placing every child at (0,0) and letting them be their desired size. The finalSize that gets passed into the ArrangeOverride can never be infinite, so we are allowed to just return it if we want to take all the room our parent has given us.

Thats a simple ArrangeOverride method, but it doesn't actually do any real layout of the children. Lets take a look at some code that actually acts like a wrap panel:

protected override Size ArrangeOverride(Size finalSize)  
{  
  if (this.Children == null || this.Children.Count == 0)  
    return finalSize;  
  
  double curX = 0, curY = 0, curLineHeight = 0;  
  
  foreach (UIElement child in Children)  
  {  
    if (curX + child.DesiredSize.Width > finalSize.Width)  
    { *//Wrap to next line*  
      curY += curLineHeight;  
      curX = 0;  
      curLineHeight = 0;  
    }  
  
    child.Arrange(new Rect(curX, curY, child.DesiredSize.Width,  
        child.DesiredSize.Height));  
  
    curX += child.DesiredSize.Width;  
    if (child.DesiredSize.Height > curLineHeight)  
      curLineHeight = child.DesiredSize.Height;         
  }  
  
  return finalSize;  
}

So here we are actually doing some layout - very similar to the MeasureOverride shown above. By calling Arrange with the new position and size of each child, the children are laid out in wrap-panel-like fashion.

Ok, ok, I can hear you complaining already "But where is my animation? You said this would be animated!" Well, actually, all that we need to do is modify this ArrangeOverride function a bit. Instead of just placing the child at its new position, we will use a TranslateTransform and animate it to the new position:

protected override Size ArrangeOverride(Size finalSize)  
{  
  if (this.Children == null || this.Children.Count == 0)  
    return finalSize;  
  
  TranslateTransform trans = null;  
  double curX = 0, curY = 0, curLineHeight = 0;  
  
  foreach (UIElement child in Children)  
  {  
    trans = child.RenderTransform as TranslateTransform;  
    if (trans == null)  
    {  
      child.RenderTransformOrigin = new Point(0, 0);  
      trans = new TranslateTransform();  
      child.RenderTransform = trans;  
    }  
  
    if (curX + child.DesiredSize.Width > finalSize.Width)  
    { *//Wrap to next line*  
      curY += curLineHeight;  
      curX = 0;  
      curLineHeight = 0;  
    }  
  
    child.Arrange(new Rect(0, 0, child.DesiredSize.Width, child.DesiredSize.Height));  
  
    trans.BeginAnimation(TranslateTransform.XProperty,   
        new DoubleAnimation(curX, TimeSpan.FromMilliseconds(200)),   
          HandoffBehavior.Compose);  
  
    trans.BeginAnimation(TranslateTransform.YProperty,   
        new DoubleAnimation(curY, TimeSpan.FromMilliseconds(200)),   
          HandoffBehavior.Compose);  
  
    curX += child.DesiredSize.Width;  
    if (child.DesiredSize.Height > curLineHeight)  
      curLineHeight = child.DesiredSize.Height;         
  }  
  
  return finalSize;  
}

So what are we doing here? Well, we are adding a TranslateTransform to each child. Then, when the time comes to move the child to a new position, we tell the transform to animate to the new position (using a DoubleAnimation that takes 200 milliseconds). And thats it! Man, animation is so easy in WPF.

Now, if we throw all that code together, we get all we need for a custom animated wrap panel:

using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Windows;  
using System.Windows.Controls;  
using System.Windows.Data;  
using System.Windows.Documents;  
using System.Windows.Input;  
using System.Windows.Media;  
using System.Windows.Media.Imaging;  
using System.Windows.Navigation;  
using System.Windows.Shapes;  
using System.Windows.Media.Animation;  
  
namespace AnimatedWrapPanel  
{  
  public class AnimatedWrapPanel : Panel  
  {  
    private TimeSpan \_AnimationLength = TimeSpan.FromMilliseconds(200);  
  
    protected override Size MeasureOverride(Size availableSize)  
    {  
      Size infiniteSize = new Size(double.PositiveInfinity, double.PositiveInfinity);  
      double curX = 0, curY = 0, curLineHeight = 0;  
      foreach (UIElement child in Children)  
      {  
        child.Measure(infiniteSize);  
  
        if (curX + child.DesiredSize.Width > availableSize.Width)  
        { *//Wrap to next line*  
          curY += curLineHeight;  
          curX = 0;  
          curLineHeight = 0;  
        }  
  
        curX += child.DesiredSize.Width;  
        if(child.DesiredSize.Height > curLineHeight)  
          curLineHeight = child.DesiredSize.Height;  
      }  
  
      curY += curLineHeight;  
  
      Size resultSize = new Size();  
      resultSize.Width = double.IsPositiveInfinity(availableSize.Width)   
          ? curX : availableSize.Width;  
      resultSize.Height = double.IsPositiveInfinity(availableSize.Height)   
          ? curY : availableSize.Height;  
  
      return resultSize;  
    }  
  
    protected override Size ArrangeOverride(Size finalSize)  
    {  
      if (this.Children == null || this.Children.Count == 0)  
        return finalSize;  
  
      TranslateTransform trans = null;  
      double curX = 0, curY = 0, curLineHeight = 0;  
  
      foreach (UIElement child in Children)  
      {  
        trans = child.RenderTransform as TranslateTransform;  
        if (trans == null)  
        {  
          child.RenderTransformOrigin = new Point(0, 0);  
          trans = new TranslateTransform();  
          child.RenderTransform = trans;  
        }  
  
        if (curX + child.DesiredSize.Width > finalSize.Width)  
        { *//Wrap to next line*  
          curY += curLineHeight;  
          curX = 0;  
          curLineHeight = 0;  
        }  
  
        child.Arrange(new Rect(0, 0, child.DesiredSize.Width,   
                      child.DesiredSize.Height));  
  
        trans.BeginAnimation(TranslateTransform.XProperty,   
            new DoubleAnimation(curX, \_AnimationLength), HandoffBehavior.Compose);  
  
        trans.BeginAnimation(TranslateTransform.YProperty,   
            new DoubleAnimation(curY, \_AnimationLength), HandoffBehavior.Compose);  
  
        curX += child.DesiredSize.Width;  
        if (child.DesiredSize.Height > curLineHeight)  
          curLineHeight = child.DesiredSize.Height;         
      }  
  
      return finalSize;  
    }   
  }  
}

But now that we have this awesome custom panel, how do we use it? Well, it is actually not that hard at all - just a couple lines of XAML code:

<Window x:Class="AnimatedWrapPanel.Window1"  
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
   xmlns:ARP="clr-namespace:AnimatedWrapPanel"  
   Title="Animated Wrap Panel Test" Height="300" Width="300">  
  <ScrollViewer HorizontalScrollBarVisibility="Disabled"   
     VerticalScrollBarVisibility="Auto">  
    <ARP:AnimatedWrapPanel>  
      <Image Source="Images\Aquarium.jpg" Stretch="Uniform" Width="100" Margin="5"/>  
      <Image Source="Images\Ascent.jpg" Stretch="Uniform" Width="50" Margin="5" />  
      <Image Source="Images\Autumn.jpg" Stretch="Uniform" Width="200" Margin="5"/>  
      <Image Source="Images\Crystal.jpg" Stretch="Uniform" Width="75" Margin="5"/>  
      <Image Source="Images\DaVinci.jpg" Stretch="Uniform" Width="125" Margin="5"/>  
      <Image Source="Images\Follow.jpg" Stretch="Uniform" Width="100" Margin="5"/>  
      <Image Source="Images\Friend.jpg" Stretch="Uniform" Width="50" Margin="5"/>  
      <Image Source="Images\Home.jpg" Stretch="Uniform" Width="150" Margin="5"/>  
      <Image Source="Images\Moon flower.jpg" Stretch="Uniform" Width="100" Margin="5"/>  
    </ARP:AnimatedWrapPanel>  
  </ScrollViewer>  
</Window>

First, in your Window tag (or page, or UserControl) you need to define the namespace that this custom control exists in. In this case the namespace is "AnimatedWrapPanel" - and so I added the attribute xmlns:ARP="clr-namespace:AnimatedWrapPanel". Now, to use it, you just add the tag:

<ARP:AnimatedWrapPanel>  
  
</ARP:AnimatedWrapPanel>

And to give it children, you use it like any other WPF control that can have children - you throw them between the opening and closing tags!

<ARP:AnimatedWrapPanel>  
  <Image Source="Images\Home.jpg" Stretch="Uniform" Width="150" Margin="5"/>  
  <Image Source="Images\Moon flower.jpg" Stretch="Uniform" Width="100" Margin="5"/>  
</ARP:AnimatedWrapPanel>

Thats all you need to do to use a custom control. Pretty simple, eh?

Well, thats it for this tutorial on how to create a custom panel control in WPF. If you want to play around with the control we made here, the animated wrap panel, you can download the visual studio project [here](http://www.switchonthecode.com/sites/default/files/147/source/SOTC-AnimatedWrapPanel.zip). And, as always, if you ahve any questions or comments, feel free to leave them below.