http://www.blogs.intuidev.com/post/2010/02/10/TabControlStyling_PartThree.aspx Go MAR APR MAY
93 captures 13
27 Feb 2010 - 12 Feb 2019 2015 2016 2017
About this capture

Olaf Rabbachin

WPF, WinForms, SQL Server, Office et al

WPF: TabControl series - Part 3: Non-wrapping scrollable TabPanel; TabItem DropDown-Menu

By Olaf Rabbachin 10. February 2010 18:18

Introduction

The previous article left us with a TabControl featuring animated TabItems. Today, I'd like to present another couple of extensions to that TabControl. These include a custom TabPanel which will no longer wrap its TabItems when these won't fit onto a single row and a menu that'll present all TabItems' header-text, allowing users to quickly navigate to a TabItem.

Overview

This article is part of a multi-part series. Here's the four parts of the series:

- WPF: TabControl Series - Part 1: Colors and Sizes
- WPF: TabControl Series - Part 2: Animating the TabItems
- WPF: TabControl Series - Part 3: Non-wrapping TabItemPanel (this article)
- WPF: TabControl Series - Part 4: Closeable TabItems

Outcome: the result of what's covered in this article

Here's what we'll be left with at the end of this article:

Status quo (after *Part Two*)

As noted before, this article is based upon the stuff I introduced in the other parts, hence I'll simply assume that you read and understood what has been discussed there. Please see the other parts in case you find that I am assuming something you don't see discussed here.

Here's where we'll start in this part, that is, what the TabControl and its "sub-controls" looked like at the end of *Part Two*:

If you downloaded the sample solution (see the bottom for the link), click the "1. Base-style (animated, without ScrollViewer)" button to show the above window.

Before we start

In the previous parts, I always refered to the panel that contains the TabItems as the ***TabItemPanel***. Actually I don't have the slightest idea as to *why* I called it like that (I'm getting old I guess - I promise I didn't have too much beer!). Of course, the control's name is ***TabPanel***! I don't know for sure whether I'll update *Part One* and *Part Two* accordingly, but since this would require to change the solutions along the way, I'll leave everything "as is", at least

About

Hi and welcome to my blog!

I'm a developer from Germany, currently focusing on .Net and WPF.

More about me ...

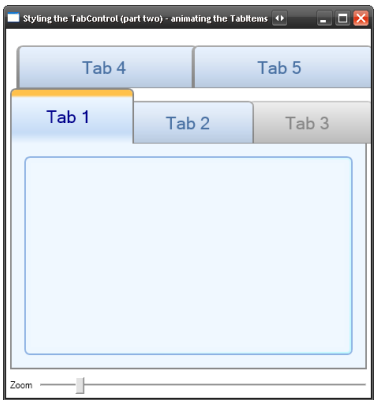
Enter search term
Search
Include comments in search

Visitors

Go MAR APR MAY
93 captures 13
27 Feb 2010 - 12 Feb 2019 2015 2016 2017 About this capture

The problem with what we have at this point

While the style provided in *Part Two* actually contains a "control" that could be used as is, it imposes quite a major drawback if the TabItems' width exceeds the width of the TabPanel. In this case, the TabPanel (which is what is being used up to this point) will wrap its items, leaving you with something more or less like this (click to enlarge):



Gee, not really what I'd consider a nifty appearance, huh? Also, the TabPanel will constantly rearrange the TabItems when the SelectedItem changes. This is what I hate very much about i.e. the Options dialogs in Office (like Word).
In the remainder of this part, I'll hence show you how to work around this by (basically) allowing users to scroll the TabPanel instead.

Enter the ScrollViewer

Whenever you're in the situation where you need like to display content that could possibly exceed the size of the hosting control (or the size that that control can take in your UI), the **ScrollViewer control** will most probably be part of your solution to the task. The ScrollViewer is what allows to actually have a "virtual area" that extends beyond the size of your control. Let's consider a simple sample:

In the above image, the black rectangle represents the control as it is being rendered in the UI. The gray rectangle, however, represents the area that would be required in order to *completeley* render all content in the control. In a scenario like this, theScrollViewer control will allow us to automatically display ScrollBar controls for the X and/or Y axis if the content exceeds the size of the control. The schema above also shows the definition for the two "areas" that exist in the ScrollViewer control:

- the area that is being rendered by the control is refered to as the ***Viewport of the control***
- the "virtual area" that makes up for the size as desired by the elements contained within is refered to as the ***Extent of the control***

Now, whenever the control's Extent gets larger than the Viewport, the ScrollViewer will display either horizontal and/or vertical scrollbars; unless you tell it not to, that is. For our specific case, we want the scrolling behavior that the ScrollViewer offers, but we sure don't want the ScrollBars, do we! More on that later.

How to enable scrolling

Now what do we need in order to allow our TabPanel to rather *scroll* when there's more TabItems than we would be able to fit onto a single row? Actually, this is as simple as wrapping your content-control into a ScrollViewer:

```
<ScrollViewer SnapsToDevicePixels="True"
              HorizontalScrollBarVisibility="Auto"
              VerticalScrollBarVisibility="Disabled">
   <TabPanel ...>
</ScrollViewer>
```

The above would already be sufficient in order to force the TabPanel to rather *scroll* than to *wrap* TabItems. Also, by setting the *VerticalScrollBarVisibility* to *Disabled*, we tell the control to *never* show the vertical scrollbar.
Well, the above would leave us with a H-ScrollBar popping up when the overall width of all TabItems would exceed the width of the TabPanel and we sure don't want that (or is it a matter of personal preference?). In order to gain more control over what happens when the Extent exceeds the Viewport, the *Horizontal-/VerticalScrollBarVisibility* properties (which actually refer to the *ScrollBarVisibility* enumeration) also allow us to set two other values: *Visible* and *Hidden*. While *Visible* will make the control show the respective ScrollBar **all** of the time, *Hidden* will **never** display it. Now what's the difference between *Hidden* and *Disabled*? If we set this to *Disabled*, scrolling will not be possible at all;

Blogroll
Beth Massi
Dr. WPF
Karl On WPF
Charles Petzold
The Worst Dental Clinic...
Download OPML file
Newsletter
Get notified when a new post is published.
Enter your e-mail
Notify me

Go MAR APR MAY
93 captures 13
27 Feb 2010 - 12 Feb 2019 2015 2016 2017
About this capture

on the far left/right resp. top/bottom) do. As a result, let's make a few changes to the XAML above.

Exit the TabPanel

While changing the XAML, let's go ahead and get rid of the TabPanel along the way, replacing it with its simpler counter part - the StackPanel. Why that? Comparing the StackPanel to the TabPanel, the latter one really only provides two things that the StackPanel doesn't:

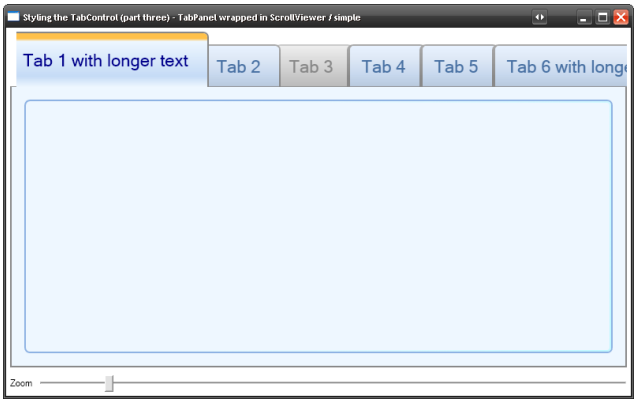
1. it wraps TabItems in rows when required
2. it rearranges TabItems when the selected TabItem changes, i.e. assures that the selected TabItem is on the bottom row (actually, that's probably the only difference between the TabPanel and the WrapPanel ...)

We don't need either of those two - all *we* need is an area that can scroll in one direction, so a StackPanel with *Orientation="Horizontal"* will provide all we need.

As a result, this would leave us with the following:

```
<ScrollViewer SnapsToDevicePixels="True"
              HorizontalScrollBarVisibility="Hidden"
              VerticalScrollBarVisibility="Disabled">
   <StackPanel ...>
</ScrollViewer>
```

After applying the above change to the XAML of what we finished with in *Part Two*, we get this (click for a larger image):



If you downloaded the sample solution (see the bottom for the link), click the "2. Standard ScrollViewer added" button to show the above window.

In the XAML for the above window, I added some more TabItems (there's now 15 of them) and, as you can see, the TabPanel no longer wraps. Kewl.
However, while you can loop through the TabItems with the arrow keys, there's no way of getting to the TabItems using the mouse.
If you watched the video in the introduction of this article, you will have seen what I really had in mind was an area to the right of the TabPanel in which the LineButtons (aka the scroll buttons) are placed.

Hosting the ScrollButtons

So where do we place the ScrollButtons? It's actually not as easy as you'd think. If you look at the last screenshot again, you'll see that the first tab is missing its leftmost part. One of the reasons for this is the negative (horizontal) margins that are applied by the triggers of the selected TabItem (-4 in the sample). That is, remember that we (err, I ) wanted the selected TabItem to overlap into the adjacent TabItems' "territory" (hey, are those Saddam-tabs? These are successful though!)? In the present situation, this forces us to do quite a substantial amount of additional work. Also, we still want to have the borders be displayed right (see *Part One*). While there is several possible approaches to all this, I opted to override the ControlTemplate of the ScrollViewer control (not least because this is also a tutorial about the power of styles!). Overriding the ControlTemplate again gives us all the flexibility we need (well, not *all* exactly, but more on that later).

Here's the part of the XAML that contains the definition/setup of the TabPanel along with the ScrollViewer in which has now been wrapped:

```
<Border Name="TabPanelBorder"
                           Height="35"
                           Background="{StaticResource TabPanel_BackgroundBrush}">
   <ScrollViewer SnapsToDevicePixels="True"
                                 Name="svTP"
                                 Grid.Row="0"
                                 HorizontalScrollBarVisibility="Hidden"
```

Go MAR APR MAY
93 captures 13
27 Feb 2010 - 12 Feb 2019 2015 2016 2017
About this capture

```
                <Setter.Value>
                    <ControlTemplate>
                        <Grid SnapsToDevicePixels="True"
                                        Grid.Row="0" Grid.Column="0">
                            <Grid.ColumnDefinitions>
                                <!--
                                    The TabItems (resp. the TabPanel)
                                    will appear here
                                -->
                                <ColumnDefinition Width="*"/>
                                <!--
                                    The following two columns will host
                                    the Scrollbuttons
                                -->
                                <ColumnDefinition Width="Auto"/>
                            </Grid.ColumnDefinitions>
                            <ScrollContentPresenter
                                x:Name="PART_ScrollContentPresenter"
                                VirtualizingStackPanel.IsVirtualizing="False"
                                SnapsToDevicePixels="True"
                                Grid.Column="0"
                                Content="{TemplateBinding ScrollViewer.Content}"/>
                            <Grid x:Name="gScrollButtons"
                                  HorizontalAlignment="Right"
                                  Grid.Column="1">
                                <Grid.RowDefinitions>
                                    <RowDefinition Height="*"/>
                                    <RowDefinition Height="Auto"/>
                                </Grid.RowDefinitions>
                                <StackPanel Grid.Row="1"
                                      Orientation="Horizontal"
                                      Margin="{StaticResource
                                          TabPanelScrollPanel_Margin}">
                                    <!--
                                        The two RepeatButtons below will actually provide
                                        the scroll-functionality for the TabItems.
                                        Here, I'm utilizing the Page[Left/Right]Command;
                                        This could as well be using the
                                        Page[Left/Right]Command instead.
                                    -->
                                    <RepeatButton
                                        Style="{StaticResource LineButtonStyle}"
                                        Command="ScrollBar.PageLeftCommand"
                                        Content="{StaticResource ArrowLeftPath}"
                                        IsEnabled="{Binding ElementName=svTP,
                                            Path=HorizontalOffset,
                                            Converter={StaticResource
                                            scrollbarOnFarLeftConverter}}"/>
                                    <RepeatButton
                                        Style="{StaticResource LineButtonStyle}"
                                        Command="ScrollBar.PageRightCommand"
                                        Content="{StaticResource ArrowRightPath}">
                                        <RepeatButton.IsEnabled>
                                            <MultiBinding Converter="{StaticResource
                                                  scrollbarOnFarRightConverter}">
                                                <Binding ElementName="svTP"
                                                    Path="HorizontalOffset"/>
                                                <Binding ElementName="svTP"
                                                    Path="ViewportWidth"/>
                                                <Binding ElementName="svTP"
                                                    Path="ExtentWidth"/>
                                            </MultiBinding>
                                        </RepeatButton.IsEnabled>
                                    </RepeatButton>
                                </StackPanel>
                            </Grid>
                        </Grid>
                        <ControlTemplate.Triggers>
                            <DataTrigger Value="false">
                                <DataTrigger.Binding>
                                    <MultiBinding Converter="{StaticResource
                                          scrollbarOnFarRightConverter}">
                                        <Binding ElementName="svTP"
                                            Path="HorizontalOffset"/>
                                        <Binding ElementName="svTP"
                                            Path="ViewportWidth"/>
                                        <Binding ElementName="svTP"
                                            Path="ExtentWidth"/>
                                    </MultiBinding>
                                </DataTrigger.Binding>
                            </DataTrigger>
                        </ControlTemplate.Triggers>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
    </ScrollViewer.Style>
    <!--
        This is the area in which TabItems (the strips)
        will be drawn.
    -->
    <StackPanel Name="TabPanel"
        Orientation="Horizontal"
```

Go MAR APR MAY
93 captures 13
27 Feb 2010 - 12 Feb 2019 2015 2016 2017
About this capture

Here's the deal: The ScrollViewer contains a Grid with two columns. Let's start with the second column. Here, another Grid (*gScrollButtons* - named only for the sake of clarity) contains two rows - the top one will remain empty and take whatever the remainder of the overall height leaves; the second will contain a StackPanel with two RepeatButtons (thus allowing for continuos scrolling while holding down the mouse-button - you'll find these in the ScrollBar control, too). Using two rows is just one way of aligning the StackPanel to the bottom so that it's right above the content area of the TabControl. In order to make the RepeatButtons actually perform scrolling, a simple CommandBinding does the trick. That is, the ScrollViewer control exposes commands for scroll-operations such as LineLeft or PageLeft (+ the adequate ones for Right/Up/Down), ScrollToLeftEnd (and various others to scroll to all edges of the Extent), and another bunch for scrolling for the MouseWheel. In the sample above, I opted to utilize the PageLeft/PageRight commands.

In the first column in the ScrollViewer's (main) grid - which will use the remainder of the horizontal extent - you'll find the *ScrollContentPresenter*. This is the control that actually represents the content to be rendered in the control, aka the ***Viewport***. To make the ScrollViewer render its content in the ScrollContentPresenter, we simply (template-) bind its content to that of the ScrollViewer (*TemplateBinding ScrollViewer.Content*).
(The reason for the above XAML looking a "bit" clunky is actually really because I wrapped it into many lines so the code-formatter that I'm using doesn't get upset with me ...)

Geometries, Paths, Converters

If you looked closely at the XAML, you probably saw that there's two converters and Path-resources for rendering the buttons' content.
Regarding Geometries - I usually have a whole bunch of those in a ResourceDictionary (or several of them) so that I can use them **a)** with themes (i.e. different paths for different themes) and **b)** simply reference them throughout applications. Another advantage is that - since we're really defining vectors - we can make them (well, paths that use their data) **scale** inside whatever content-control we use them in, without any loss or pixelation, when they need to grow larger (we Germans don't fancy six-packs, so a crate of beer to whoever came up with that!).

The Path resources are really two-fold; first, there's a Geometry that defines the shape and, second, there's a Path that utilizes them and adds i.e. the colors, etc.; for instance, for the left arrow this looks like the following:

```
<Geometry x:Key="ArrowLeft">M0,5 L10,10 10,0Z</Geometry>
<Path x:Key="ArrowLeftPath"
      Margin="4,3"
      Data="{StaticResource ArrowLeft}"
      Stroke="{StaticResource LineButtonBrush}"
      Fill="{StaticResource LineButtonBrush}"
      Stretch="Fill"
      VerticalAlignment="Center"
      HorizontalAlignment="Center"/>
```

If you're not familiar with Geometries and the "geometry mini language", I suggest to bing or google that - it's so much more convenient compared to the long version of their more verbuous counterparts, especially if only a small count of points is required, in which case you can really learn to read them over time.

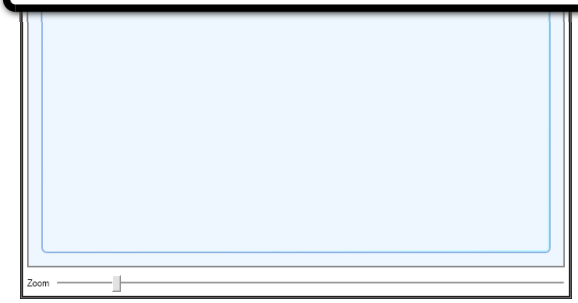
Regarding the converters: While, in the beginning, I really wanted a XAML-only solution, this was no longer possible since I really really wanted the scroll buttons to be disabled when scrolling isn't possible, i.e., when the viewport is either on the far left or far right. The far left isn't much of a problem - we could simply compare the HorizontalOffset property to zero, in which case scrolling to the left wouldn't be possible. However, in order to find out whether the Viewport is on the far right, we have to compare the Extent's width against the sum of the Viewport's width plus the HorizontalOffset, IOW, scrolling to the right not possible if [HorizontalOffset + Viewport.Width] = Extent.Width. Sadly, this is not possible without a converter due to the necessity of a MultiBinding, hence this part will require some code. However, we're not talking about the need for code-behind for every TabControl we use, but rather about a loosely coupled class. The TabControl's style can thus still be used throughout your project as long as the Converter is part of that project, too. A drawback for sure, but a minor one, if you ask me. Since we need one converter for the right-button, I thought it'd make sense to also provide one for the left-button.

You'll find both converters in the ScrollBarConverters.cs file (resp. ScrollBarConverters.vb).

Ready to (sc)roll ..?

If you downloaded the sample solution (see the bottom for the link), click the *3. ScrollViewer with Scroll-Buttons* button to show the window with the result of the above. You'll get something this (click for a larger image):

Go MAR APR MAY
93 captures 13
27 Feb 2010 - 12 Feb 2019 2015 2016 2017
About this capture



That's better - we can now scroll the TabPanel in order to get at the TabItems that are invisible/inaccessible. Let's add some functionality that I personally learned to value.

The TabControl in SAP's WebGUI

Actually, the functionality I wanted is is more or less equal to that of the TabControl in SAP's WebGUI. In early 2008 SAP tasked us to build a WinForms companion to SAP's WebGUI, that is, to implement a WinForms counterpart for the controls contained in their library. This set of WinForms controls was then used for implementing an offline client for SAP's cProjects (which is a part of SAP PS). Here's a sample screenshot of our test-client (click to enlarge):

In the above screenshot, you can see the bottom-most TabControl "in action", featuring three buttons on the right extent of the TabPanel - one for each scrolling to the left and right and another one. Another one? Yup, this one opens up a popup-menu in which all TabItem's headers are listed, allowing users to quickly select an item from the list, activating the selected TabItem, even if it's out of view when selected. *(This TabControl was one of the most complicated and non-amusing controls I've ever had to build; if you ever need to build your own TabControl with WinForms, tell ya - it's not what I consider "fun". With WPF OTOH, this is just so much easier, less complicated, way more flexible, faster, fun, ... you name it!)*

So, what do we have to do in order to create such a menu? Again, we can settle with a no-code / XAML-only solution!

Enter the Menu and MenuItem controls

First thing you probably thought of was ... the ContextMenu? Well, I did. But instead of convincing a ContextMenu to popup, I resembled to applying a custom style to the Menu and the MenuItem controls. First, let's have a look at the XAML that we'll need to add to the StackPanel (which already contains the scroll buttons) in order to get the menu in the right place:

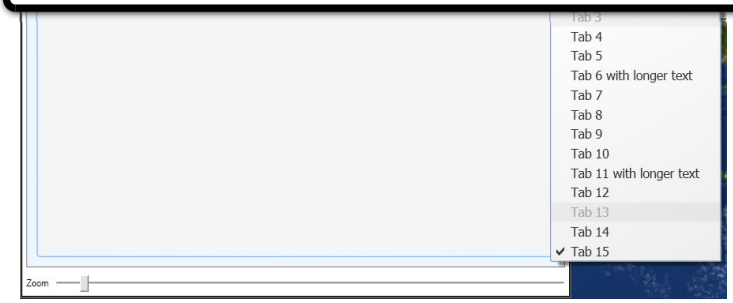
```
<Menu Background="Transparent">
   <MenuItem Style="{StaticResource TabMenuButtonStyle}"
             ItemsSource="{Binding RelativeSource=
               {RelativeSource FindAncestor,
                  AncestorType={x:Type TabControl}},
                  Path=Items}"
             ItemContainerStyle="{StaticResource TabMenuItem}">
   </MenuItem>
</Menu>
```

Pretty short really, right? Of course, the *style* for the control isn't part of the XAML, but assuming that, in a real world solution, the style will rather be dropped into a separate ResourceDictionary, this is all you need in your TabControl's style (no no, I wouldn't call that cheating!). Thus, the only really interesting part about the XAML above really is the binding that is applied to make the menu (well, the *MenuItem*, really) show all TabItems' Header texts. But it's probably easier than you might have thought, because all we have to do is to point the MenuItem to the TabControl and then bind to its TabItems. I love it - with WinForms that was so much more code!

I won't fancy discussing the style for the MenuItem in depth here. If you inspect the XAML in the sample solution, you'll find it documented - just look for *TargetType="{x:Type MenuItem}"* to find the two styles (one for the MenuItem that makes up for the Button in the StackPanel and one for the popup menu with the items themselves). Two side notes here: First, I actually failed to provide a hover-effect for **disabled** TabItems, the reason being the fact that disabled items will never receive any HitTest information. As a result, you won't see any indication when you hover over disabled items found in the menu; oh well. Second, I thought that it'd be fun to again use the geometry mini language in order to create the button's image which should be pretty close to SAP's original icon, only that this one's scalable. 😎

Here's a screenshot of what we have now (click for a larger image):

Go MAR APR MAY
93 captures 13
27 Feb 2010 - 12 Feb 2019 2015 2016 2017
About this capture



(If you downloaded the sample solution (see the bottom for the link), click the "4. TabItem-menu added" button to show the window above.)

Are we done yet?

At this point, you might want to sit back and determine whether the above already gives you what you need for your own TabControl. There's really only a couple of things that are worth dealing with the rest of the article - one minor and two major things:

1. Selecting a TabItem from the menu will not bring the first and last TabItems into view completely (major)
2. Clicking the scroll buttons will scroll by whole pages (major)
3. The TabItems on the left and right of the ScrollViewer's Viewport will be cut off abruptly (minor)

Why am I saying this? The first item might not apply to you - if you don't use negative margins, this would fade away silently. The second and third items might not be important to you. To me, however, all of these three are inacceptable. So ...

Enter IScrollInfo

Dealing with the aforementioned drawbacks turned out to be impossible by means of XAML only (I tried real hard!). So I opted to create my own panel instead. The "wanted" features that made it onto my list:

- more control over the scrolling position when moving to the beginning resp. end of the Viewport (allowing negative margins of contained controls at the edges of the Extent)
- more control over the offset that's being applied during scrolling
- animated scrolling
- a "fading" effect for TabItems that are only partially visible
- get rid of the converters required for binding the scroll buttons' IsEnabled property

To create your own panel, you can simply inherit from *Panel*. However, to provide your own scrolling logic, we'll need to implement ***IScrollInfo***.
IScrollInfo really is a beast! If you have VisualStudio create the methods required for implementing this interface for you, you'll be left with as much as **9 properties and 15 methods**! Here's the list (in the order that VS creates them):

```
public bool CanHorizontallyScroll
public bool CanVerticallyScroll
public double ExtentHeight
public double ExtentWidth
public double HorizontalOffset
public void LineDown()
public void LineLeft()
public void LineRight()
public void LineUp()
public Rect MakeVisible(Visual visual, Rect rectangle)
public void MouseWheelDown()
public void MouseWheelLeft()
public void MouseWheelRight()
public void MouseWheelUp()
public void PageDown()
public void PageLeft()
public void PageRight()
public void PageUp()
public ScrollViewer ScrollOwner
public void SetHorizontalOffset(double offset)
public void SetVerticalOffset(double offset)
public double VerticalOffset
public double ViewportHeight
public double ViewportWidth
```

Most of the above methods are either pretty easy to implement (such as LineLeft/LineRight) or do not need to be covered (LineDown/Up, PageDown/Up, MouseWheel*) at all. However, the MakeVisible method needs some more intense care-taking, as does the SetHorizontalOffset method.

BTW - note that the sample class will simply skip anything related to mouse wheel actions and any actions targetting the vertical axis. If you plan to use the TabControl with its TabItems drawn on the left or right, you'll have to add

Go MAR APR MAY
93 captures 13
27 Feb 2010 - 12 Feb 2019 2015 2016 2017
About this capture

interface, we also need to ***override*** a couple of methods, the most important being ***MeasureOverride*** and ***ArrangeOverride***. I won't discuss the whole class here as that could ***a)*** get quite boring (with respect to this article being geared at the TabControl) and ***b)***, considering that there's a substantial amount of code involved. FWIW - you'll find the code well documented in the sample solution and if you encounter any problems or want to know more about any specifics, leave a comment.

A couple of things can not be set aside though. For instance, the two aforementioned methods deserve some explanation which is critical for understanding the concept behind this, so here goes.

IScrollInfo: MeasureOverride and ArrangeOverride

When elements are being added to (or removed from) your control or one of the elements is re-rendered (i.e. after its size has changed), the whole layout of the control (that is, the Extent and Viewport) needs to be rearranged. This requires a two-fold process to which the .Net framework refers to as the ***two pass layout updating process***. This means that, whenever the layout needs to be updated, the compiler will call both methods. In MeasureOverride we need to determine the overall size of the control (i.e., the Extent) that is required to host all contained elements; in the sample class, only the width is relevant, so the class will iterate over all elements, sum up their desired width and return the result; the height will remain constant at all times.

Once that is done, the elements need to be *arranged* within the Extent, hence the second pass - ArrangeOverride. Here, we again iterate over all children and define the (horizontal) position for each of them. In some situations, the arranging of the children may result in the need to perform both first and second pass again, so this sequence may be called several times. For the ScrollableTabPanel (being the sample class) however, this is not the case.

Again, IScrollInfo would really deserve (require!) its own article, hence I'll skip everything else related to this interface at this point. A little hint though: if you want to know more about those two methods, I suggest you check the MSDN docs on **UIElement.Measure** and **UIElement.Arrange** - while you'll find control-specific topics in the docs about MeasureOverride and ArrangeOverride, the detail covered there doesn't compare to what you'll find in the respective ones behind the aforementioned links! Also, there should be plenty of tutorials on IScrollInfo basics throughout the web.

Animating the Panel

One of the two other things that I think are worth mentioning is the fact that, whenever the ScrollableTabPanel scrolls, the process will be animated (as opposed to instantly switching to the final position). Two simple reasons for that - when the user scrolls the panel, there is no real visual indication of what happens; by animating the process the user has (IMHO) a far better chance to see what's going on behind the scenes. Second, the animation is stupidly easy to implement - it's *basically* not more than a single line of code. In the solution's class, you'll actually see a *couple of lines*, but that's rather because I wanted the animation to ***a)*** accelerate and decelerate and ***b)*** because an update of the TabItems is required - **after** the animation has ended (the OpacityMasks need to be updated - more on that below).

Having the TabItems at the edges of the Viewport fade into nothingness

Last but not least, I wanted to give the user a visual indication in the case a TabItem (strip) was only partially visible, indicating that there is more items to the left or right. Achieving that was way trickier than I originally thought really (but hey, we all like digging into stuff like that, don't we ...). When I was thinking about the fade-effect, I thought of applying an Opacity Mask right away. My first attempt (call me dumb) was to apply a mask to the left and right edge of the ScrollViewer (that was actually before I implemented the ScrollableTabPanel), which is as simple as creating a (horizontal) LinearGradientBrush that fades into *Colors.Transparent* at its edges and then applying the resulting Brush to the OpacityMask property of whatever control in question. *(BTW - it doesn't matter at all what other color(s) you place into such a brush - for an OpacityMask, only the alpha channel is important, thus the color itself (meaning the R, G and B channels) is irrelevant.)* This way, only those portions (colors) of the control/content itself with an alpha-value >0 will be affected by the "fader brush".

Well, it of course wasn't that easy - the OpacityMask will be applied to the *Extent* of the control, rather than the *Viewport* - I hence did not succeed in defining an Opacity mask that would stick to the Viewport's bounds; if you know a way, make sure you leave a comment!
I thus opted to apply the mask to the TabItems themselves. In this case though, the whole task gets a little more complicated because the brush itself needs to consider the width of each TabItem, if the fade effect is to remain *as constant as possible* (which is not all too much, depending on how narrow the visible portion gets, but you'll see that yourself). In the ScrollableTabPanel class, you'll see that I simply calculate exactly *how much* of each TabItem is visible (i.e. ranging from 0 = completely invisible to 1 = completely visible). The factor or ratio gained will then be applied to the StartPoint or EndPoint respectively.
A minor quirk with this is the fact that all OpacityMasks need to be removed prior to performing scrolling as, otherwise, the faded edges would remain visible until the Viewport has reached its final position. But oh well, you can't have it all, can you.

Go MAR APR MAY
93 captures 13
27 Feb 2010 - 12 Feb 2019 2015 2016 2017
About this capture

even off limits at times. In the last window of the sample solution, I have therefore added another style for that (look for the key "*TabItemFocusVisual*"). This, again, imposes another minor issue: this can't be dealt with by means of an adorner (Dr. WPF has recently published a nice article on this), so I had to stick with calling *InvalidateArrange()* instead. However, this again can be called (reliably) only *after* any scrolling has taken place resp. finished. You will thus see the dashed border "move" when you loop through the TabItems with the keyboard (to do so, focus the slider and then hit the left/right arrow keys). However, IMNSHO the effect is too minor to deserve some decent appreciation (especially if you don't zoom in), so I'll leave that as it is now and rather go and have my beer.

The last word

This concludes *Part Three* of the TabControl series. It shouldn't really take too long to assemble the last part (the last part currently *planned*, that is 😎) as I've dealt with the close-button (and images) in another solution already. Maybe next week - we'll see. As for me, I sure learned a bunch of new things and details about the ScrollViewer and IScrollInfo - I hope you enjoyed it a bit, too.

As always, I'd appreciate you leaving a comment - whether you liked it or not, or in case you need further clarification on any of the topics discussed here - I'll do my very best to answer them.

The sample solution

I've created a sample solution that contains everything discussed here. Other than with the previous parts, the solution now again contains one project for each the C# and the VB versions.

Download: TabControlStyle - Part Three.zip (105.68 kb)

Location: SinglePost

Currently rated 4.9 by 16 people

Tags: wpf, tabcontrol, tabitem, tabpanel, styles, animation, storyboards, scrollviewer, iscrollinfo, menu, menuitem, opacitymask

E-mail | Kick it! | DZone it! | del.icio.us Permalink | Comments (31) | Post RSS

TabControl | WPF (.net)

Related posts

WPF: TabControl Series - Part 1: Colors and Sizes
How to style the WPF TabControl and the TabItems resp. the TabPanel (part one - colors and sizes)

WPF: TabControl series - Part 2: Animating TabItems
How to style the WPF TabControl and the TabItems resp. TabItemPanel (part two - animating TabItems)

WPF: TabControl series - Part 4: Closeable TabItems
How to style the WPF TabControl and the TabItems resp. the TabPanel (part four - closeable TabItems...

Comments

Community Blogs (website link) February 22. 2010 07:12

trackback Windows Client Developer Roundup for 2/22/2010

This is Windows Client Developer roundup #12. I've just returned from the MVP summit and some meetings

Microsoft Weblogs (website link) March 11. 2010 22:13

websnapr 2.0 update your websnapr code: websnapr.com/code

Windows Client Developer Roundup for 2/22/2010

This is Windows Client Developer roundup #12. I've just returned from the MVP summit and some meetings

vikas dangwal January 20. 2011 23:58

I am trying to add xceedDataGrid into the datatemplate and then assign that data template to ContentTemplate of TabControl. Everything works fine except one thing. When i rearrange , resize the columns of dataGrid inside one tab item, it reflects other as well. Because it is using same datagrid control for all the TabItems content. Is there any

Go MAR APR MAY
93 captures 13
27 Feb 2010 - 12 Feb 2019 2015 2016 2017
About this capture

Muhammad K. Shehzad January 31. 2011 15:35

Hi,
Very helpful and explanatory article. I used this tab control to extend my custom tab control.

I am stuck, when I dynamically add TabItems mean in C# code behind, Menu do not show those dynamically added TabItems.

If I add TabItems in xaml, only then they are being displayed in Menu.

Thanks once again for all of this helpful article.

Olaf Rabbachin (website link) February 1. 2011 11:33

Hi Muhammad,

hmm. Don't have the time to dig deeper into this at present, but this definitely is a problem with notifications. Have you tried to bind the TabControl's ItemsSource to an ObservableCollection and add new tabs by adding to the collection?

Cheers,
Olaf

Muhammad K. Shehzad February 1. 2011 12:25

Thanks a lot man!

Actually I was adding a string as TabItem which becomes header content, like:

tc.Items.Add("My New Tab");

Now I created TabItem object and added to TabControl which works...

Once again thanks!

Olaf Rabbachin (website link) February 1. 2011 12:37

Hi Muhammad,

good to know - thanks for letting me know (stuff like this otherwise keeps nagging in the back of my head).

Cheers,
Olaf

Olaf Rabbachin (website link) February 1. 2011 12:39

Vikas,

sorry - I must've overlooked your comment (just saw it right now)! But FWIW, I don't have any experience with the xceedDataGrid, so I guess I couldn't provide any help anyway ... :-(

Cheers,
Olaf

SR258 February 20. 2011 20:32

Hi,

thanks for your very useful tutorial. However, I have a problem with the non-wrapping TabPanel:

My application has a button that changes the selection of the TabControl by setting the SelectedIndex /

Go MAR APR MAY
93 captures 13
27 Feb 2010 - 12 Feb 2019 2015 2016 2017
About this capture

tc.SelectedIndex++ in its Click handler. Click on the Button a few times and you will see the problem.

Thank you very much for your efforts!

Olaf Rabbachin (website link) February 21. 2011 09:32

Hi there,

you're right - I must've overlooked something as I guess this should really be handled internally. FWIW, you can use the TabItem's BringIntoView-method. That is, after you change the SelectedIndex (or SelectedItem),
do a *((TabItem)tc.Items(tc.SelectedIndex)).BringIntoView();*.

Cheers,
Olaf

Italy Gae March 1. 2011 16:56

Hi Olaf,
I downloaded your example (part 3) and I included the TabControl_5 in my WPF project. It works well if I work with windows. If I copy the code in a page, it works a little bit different. How can I modify the code so it works well both in the page as in the window?

Thank you very much
Gae

Olaf Rabbachin (website link) March 1. 2011 17:50

Hi Gae,

that sure does sound odd, assuming that the template should work the same in either a Window or Page.
Now I actually haven't ever used Pages nor see the need for them in the nearer future. Considering the fact that I'm also extremely busy at present, I'm afraid I won't be able to provide any help with this regard.
I'd suggest you try and post this to the WPF-forums. See the Links section / underneath the tag-cloud for the link to the US-forum (best chances of getting answers), or you post your Q to the Italian WPF-forum: social.msdn.microsoft.com/.../threads

Buona fortuna, compagno!

Cheers,
Olaf

Phil Johnson March 23. 2011 23:07

Hi there! Your control is fantastic, and worked great for me when i first implemeted it. I have since then changed the itemsource of the tab control, and now the menuitem for the tabs no longer seem to work. I am using an ItemsSource={Binding } , and setting the DataContext in the codebehind to an observable collection of UserControls (NOT TabItems). I see the tab items fine, and the headers and everything work great, but it wasn't until recently that i noticed the menuitems were broken. I had to set a converter to your

ItemsSource="{Binding RelativeSource={RelativeSource FindAncestor,AncestorType={x:Type local:CustomTabCtl}}, Path=Items ...

in the MenuItem, so that it would convert the source from the TabControl to an observable collection of TabItems (basically by making a new collection, stepping through the collection of each usercontrol, and adding it's parent (which is a TabItem) to the new collection)

Now i see the items in the Menu fine, but I still have a problem. If i select a tab through the Menu, the previously selected tabitem still shows as selected. If i add a new tab, it will immediately go back and select the item i had selected through the Menu.

This might be pretty convoluted, I'm hoping that there is just some binding issue that i'm missing or not knowledgeable enough about. Thanks for any help!

SR258 March 24. 2011 06:47

Hi Phil,

Go MAR APR MAY
93 captures 13
27 Feb 2010 - 12 Feb 2019 2015 2016 2017
About this capture

```
        ...
        <Setter Property="IsEnabled" Value="true" />
        <Setter Property="IsCheckable" Value="true" />
        <Setter Property="IsChecked" Value="{Binding Path=IsSelected, Mode=TwoWay}" />
        ....
</Style>
```

Phil Johnson March 25. 2011 13:47

I added IsSelected to my business object like you selected, and pointed the menuitem to ItemsSource rather than Items. The IsSelected change to my object didn't seem to fix it though, a checkmark doesn't appear at all now by default, and checking one of the tabs just adds a checkmark to it, it doesn't actually select my tab. Are their other properties that your ViewModel contains that might affect how the menu works? Or something that it inherits? Thanks for the help!

Hans-Martin Häberlein July 27. 2011 11:40

Gratulation - Ganz hervorragendes Control !!!
Habe ziemlich lange gebraucht, um mit meinen WPF Schmalspurkenntnissen dahinter zu steigen.
Einem Phänomen stehe ich allerdings immer noch ratlos gegenüber:
Ich möchte meinen TabItems einen ToolTip mitgeben, der aber nur sichtbar sein soll, wenn die Maus sich überm TabItem befindet und nicht im Content (was der Fall ist, wenn man <TabItem Header="TheHeader" ToolTip="TheToolTip"/> deklariert).

Also deklariere ich:

```
<TabItem>
  <TabItem.Header>
    <Label ToolTip="TheToolTip">TheHeader</Label>
  </TabItem.Header>
</TabItem>
```

Funktioniert wie's soll. Auch links-rechts scrollen kein Problem.
Wenn ich allerdings das Popup Menu öffne, stehen dort alle Header korrekt als MenuItem (sogar mit ToolTip), das betreffende TabItem verliert jedoch jegliche Beschriftung ??!!

An welchem Binding muß ich denn da was drehen ?

Vielen Dank für die Hilfe, Grüße aus Germany

Olaf Rabbachin (website link) July 27. 2011 13:05

Hi Hans-Martin,

I'll answer in English so others following this can understand what we're talking about; hope that's OK.

For other fellowers: the question was how to work around the fact that the TabItem's header gets lost if it actually contains another control, like a Label or a TextBlock which is added in order to provide a ToolTip that only shows up when hovering over the TabItem's header, which again is done in order to have the ToolTip show up only over the TabItem's header rather than its complete content.

The thing is that the whole control is meant to be used with text-only headers. That is, I never tested it with anything else (i.e. an Image + text, etc.). I would assume that the reason for the effect you've noticed (and which I can confirm) lies in the fact that the content of the header is shared between the TabItem itself and the MenuItem that displays it, but that's more of a guess really.

I'm pretty swamped presently which is why I just can't put much time into resolving this (albeit the fact that I think this is rather interesting plus annoying).

However, as a matter of fact, if your TabItem's header is a Label at all times, you could work around the effect by changing the binding of the TabMenuItem's Header-setter. That is, replace this:

```
<!-- This will help us bind to the Header of a TabItem -->
<Setter Property="Header" Value="{Binding Path=Header}" />
```

with this:

```
<!-- This will help us bind to the Header of a TabItem -->
<Setter Property="Header" Value="{Binding Path=Header.Content}" />
```

Note that this will have another side-effect: when hovering over the menu's items, the TabItems themselves will be displayed as if the mouse was hovering over them respective item. Not all too bad, I guess.

Another (more generic / better) approach to resolving this ***could*** be to simply move the TabItem's ToolTip to its

Go MAR APR MAY
93 captures 13
27 Feb 2010 - 12 Feb 2019 2015 2016 2017
About this capture

<Setter Property="ToolTipService.IsEnabled" Value="False"/>

Then, change the ContentPresenter found underneath to the following (the added line in bold text):

```
<!-- This is where the Content of the TabItem will be rendered. -->
<ContentPresenter x:Name="ContentSite"
  VerticalAlignment="Center"
  HorizontalAlignment="Center"
  ContentSource="Header"
  Margin="7,2,12,2"
  ToolTip="{TemplateBinding ToolTip}"
  RecognizesAccessKey="True"/>
```

From here, you simply go back to creating TabItems like this:

```
<TabItem Header="Test-Tab with TT #1"
      ToolTip="This is a Tab with a ToolTip that only shows in the header.">
  <Canvas Background="AliceBlue"/>
</TabItem>
```

Not nice/flexible, but it seems to do the trick.

Cheers,
Olaf

Hans-Martin Häberlein July 28. 2011 10:56

Hi Olaf,

thanks for the quick response and your recommendations.

I found out that it's not bad to also display some tooltip in the ContentPresenter area so I left it as is.

In order to also display the items tooltip in the menu, I added a Setter Property for the TabMenuItem Style declaration.

```
<!-- This will help us bind to the Header of a TabItem -->
<Setter Property="Header" Value="{Binding Path=Header}" />
<!-- This will help us bind to the ToolTip of a TabItem -->
<Setter Property="ToolTip" Value="{Binding Path=ToolTip}" />
```

Thanks again,
Martin

Russia alexandra August 2. 2011 09:48

Hi, Olaf!
thanks, it's really good and useful article!
but if you use e.g. stackpanel or smth else (not simple text) as headers for tabitems, headers will disappear, when you press menubutton. do you know what i can do with it?

alexandra

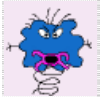
Olaf Rabbachin (website link) August 2. 2011 19:34

Hi Alexandra,

your request essentially is the same as what Martin already wrote - please read his and my comments; this should make things clearer.

Cheers,
Olaf

Netherlands Ronald Schaap (website link) September 15. 2011 11:10



This is a great control. However if you put another tabcontrol in one of the tabpages and you select that tabpage then the ArrowRightButton and the MenuButton of the parent tab become empty.

What can i do to solve that

Go MAR APR MAY
93 captures 13
27 Feb 2010 - 12 Feb 2019 2015 2016 2017
About this capture

Olaf Rabbachin (website link) September 15. 2011 11:27

Hi Ronald,

this issue is being discussed in the comments on part IV. I suggest you read those as I personally haven't ever had to use nested TCs with my style.

Cheers,
Olaf

India sri November 24. 2011 08:53

HI,
I am looking for a tabcontrol with tabitems placed vertically and a repeat button at the top and bottom of the tab control so that when no.of items is more scroll using the buttons like how it is done horizontally. Is there any way to do vertical scrolling using repeat buttons??

Olaf Rabbachin (website link) November 24. 2011 09:00

Hi Sri,

that would definitely be feasible. However, the tutorial focuses only on horizontal TabItems simply since I never needed/wanted them anywhere else, so you'd have to basically rewrite the whole thing.

Cheers,
Olaf

Suman (website link) December 9. 2011 03:24

After going through these series I have to say thank you for detail explanation on Tabcontrol.

India sri February 6. 2012 10:43

Hey how to control the width of tabitem to be scrolled when repeat button is clicked?? i need only one tab item to be crolled per click.

Also if i select this tab item from other control (i am binding the tab control which is under scrollable horizontal panel to other control which is vertical) how would i make sure that the panel is scolled to appropriate selected tab item ??

thanks in advance

Olaf Rabbachin (website link) February 24. 2012 22:03

trackback WPF: TabControl series - Part 4: Closeable TabItems

WPF: TabControl series - Part 4: Closeable TabItems

India Balaji Gupta February 27. 2012 18:43

Hi Olaf,
I would like to display this dropdownlist with the Tab items those are having Visibility as Visible. At runtime if i change any of the tab Visibility, that should update this dropdownlist. Other than runtime case, i am able to do my job with itemcollection filter in the converter.

Go MAR APR MAY
93 captures 13
27 Feb 2010 - 12 Feb 2019 2015 2016 2017
About this capture

```
        ItemsSource="{Binding RelativeSource=
        {RelativeSource FindAncestor,
        AncestorType={x:Type TabControl}},
        Path=Items,
        Converter={StaticResource VisibleItems}}"
        ItemContainerStyle="{StaticResource
        TabMenuItem}">
    </MenuItem>
</Menu>
```

And In my converter i am using following code:

```
public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo
culture)
{
   ItemCollection collection = value as ItemCollection;
   if (collection.CanFilter)
   {
      collection.Filter = delegate(object obj)
      {
         return (Visibility)(obj as TabItem).Visibility == Visibility.Visible;
      };
   }
   return collection;
}
```

Give me some idea that how the dropdownlist will appear with Visible TabItems even if Visibility updates at runtime?

Olaf Rabbachin (website link) February 27. 2012 20:19

Hi Balaji,

actually, the first thing I'd consider would be to implement my own data-binding where it'd be easier to setup the proper notifications.

This could either be your own Visibility-property that implements INPC or, depending on exactly what changes the TabItems' visibility, your own collection that is restriced to visible TabItems. In the latter case, you could use i.e. an ObservableCollection where you implicitly get INCC.

Hope that helps a tiny bit ...

Cheers,
Olaf

INPC: msdn.microsoft.com/.../...tifypropertychanged.aspx
INCCmsdn.microsoft.com/.../...fycollectionchanged.aspx
OC: msdn.microsoft.com/en-us/library/ms668604.aspx

Netherlands Ronald Schaap (website link) March 7. 2012 11:19

TabItem header becomes invisible when bound and one clicks the TabControl menu.

```
<TabItem.Header>
    <StackPanel Orientation="Horizontal">
<TextBlock Text="ClientData" />
<Image Source="/pm;component/Images/Cross.png"
      Height="12"
      Width="12" Margin="4 0 0 0"
      Visibility="{Binding ElementName=MyCheckBox,
            Path=IsChecked,
            Converter={StaticResource
                  BooleanToVisibilityConverter}}" />
    </StackPanel>
</TabItem.Header>
```

What can i do to solve that?

Greet Ronald

XAML Sorpresa (website link) March 29. 2012 01:15

trackback [WPF] Desnudando el TabControl: ajustar las pestañas en una fila

Comments are closed

Powered by: BlogEngine.NET 1.6.0.0 | Theme: stablestart | Design by: Thomas A. Bosscher