WPF: TabControl Series - Part 1: Colors and Sizes

By Olaf Rabbachin 25. January 2010 14:42

Introduction

The out-of-the-box TabControl is pretty ugly (is that just my personal opinion?). Attempting to KISS, meaning i.e. simply changing the colors used for rendering the TabControl itself and/or the TabItems however will not give you all too much of a chance to "remedy" its appearance. Instead, you'll have to completely replace the control's Style (that is, its default template). I'll try to cover some aspects in the scope of this article.

Overview

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

- WPF: TabControl Series - Part 1: Colors and Sizes (this article)
- WPF: TabControl Series - Part 2: Animating the TabItems
- WPF: TabControl Series - Part 3: Non-wrapping scrollable TabPanel; TabItem DropDown-Menu
- WPF: TabControl Series - Part 4: Closeable TabItems

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

First, let's see what we'll be left with at the end of this article (in other words "lessening the ugly screenshot coming up next"):

Before we start, please note that I wrapped up everything provided in the scope of this article as a sample solution. See the end of the article for the download-link.

Also, you'll notice that, in the screenshots, the TabControl appears larger than it would be in a real-world application. For the sake of visibility (and of course for debugging the tons of minor tweaks I encountered) I added a Slider control to the forms. This slider will allow you to zoom in on the TabControl, with a factor between 1 and 10. All the screenshots here were taken with a factor of 2.

Status quo: the default appearance

Alright, let's get started. Here's a little screenshot of an "unstyled" TabControl in which all I did was to change a couple of colors, in all its shy uglyness (click to enlarge):

And here's the markup that produced the above TabControl (I skipped everything but the TabControl itself):

```
<TabControl x:Name="tc" Margin="5" SelectedIndex="0"
            Background="CadetBlue">
    <TabControl.LayoutTransform>
    <!-- Allows to zoom the control's content using the slider -->
    <ScaleTransform CenterX="0"
                    CenterY="0"
                    ScaleX="{Binding ElementName=uiScaleSlider,Path=Value}"
                    ScaleY="{Binding ElementName=uiScaleSlider,Path=Value}"/>
    </TabControl.LayoutTransform>
    <TabItem Header="Tab 1" Background="CadetBlue">
        <Canvas Background="AliceBlue"/>
    </TabItem>
    <TabItem Header="Tab 2" Background="CadetBlue">
        <Canvas Background="Lavender"/>
    </TabItem>
    <TabItem Header="Tab 3" IsEnabled="False"
          ToolTip="I'm disabled.">
        <Canvas Background="PaleGreen"/>
    </TabItem>
    <TabItem Header="Tab 4" Background="CadetBlue">
        <Canvas Background="Cornsilk"/>
    </TabItem>
    <TabItem Header="Tab 5" Background="CadetBlue">
        <Canvas Background="WhiteSmoke"/>
    </TabItem>
</TabControl>
```

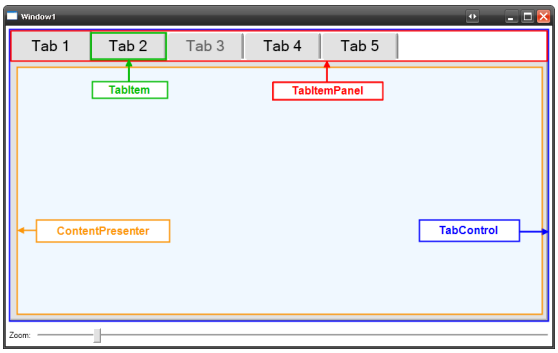
Zooming into the control (which is what the slider in the screenshot is for) even better reveals that it's far from looking good (and the colors don't matter much either ... or did I intend to make it look as ugly as possible, after all ..?).

Fundamentals: the TabControl's sections/panels

TabControl, TabItemPanel, TabItem, TabPage ... huh!?

There's a bunch of sections that the TabControl really consists of. Knowing that we'll have to override the default template of the control, we should make sure that we're on the same page regarding a couple of basics.

Here's the (fundamental) sections that the TabControl is made of (click to enlarge):



The colored rectangles in the above screenshots and their meanings:

- **Blue**: the TabControl itself - all other panels are placed inside this rectangle.
- **Orange**: the ContentPresenter - this is what will host the content of the selected TabItem (the active TabPage).
- **Red**: the TabItemPanel - this is the panel that hosts the TabItems (strips).
- **Green**: the TabItem - this is a single TabItem (strip), i.e. the portion of the control that allows users to change the currently selected TabPage.

The terms ***TabItem*** and ***TabStrip*** are really interchangeable - both refer to the portion of the control that allows users to select the element that is to be shown resp. rendered. The same actually applies to ***ContentPresenter*** and ***TabPage*** - both mean the same thing.

That being said, what you can learn from the above structure is that, in order to create a homogeneous look, we'll actually have to style **two** controls rather than one: the ***TabControl*** and the ***TabItem*** (-control).

being used as resource-references which IMHO makes up for better readable XAML:

```
<Style TargetType="{x:Type TabControl}">
    <Setter Property="SnapsToDevicePixels" Value="true"/>
    <Setter Property="Template">
    <Setter.Value>
       <ControlTemplate TargetType="TabControl">
          <Grid>
             <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="*"/>
             </Grid.RowDefinitions>

             <Border Padding="{StaticResource TabItemPanel_Padding}">
                <!-- This is the area in which TabItems (the strips) will be drawn. -->
                <TabPanel IsItemsHost="True"/>
             </Border>

             <Border BorderThickness="1,0,1,1"
                     Grid.Row="1"
                     BorderBrush="{StaticResource TabItem_BorderBrush_Selected}"
                     Background="{StaticResource TabControl_BackgroundBrush_Base}">
                <!--
                     This is where the Content of the selected TabPage
                     will be rendered.
                -->
                <ContentPresenter ContentSource="SelectedContent" Margin="0"/>
             </Border>
          </Grid>
       </ControlTemplate>
    </Setter.Value>
    </Setter>
</Style>
```

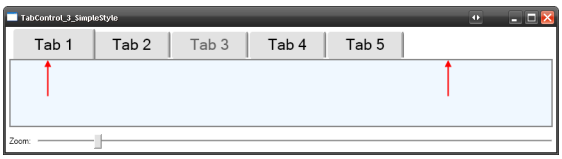
Using the above will produce the following (click to enlarge):

I admit this isn't much of a change to the better really (actually it's worse!) - it's just a different color for the inner portion of the TabControl. However, a couple of things are worth noting here (and we'll need that knowledge later on):

1. The template consists of a Grid control that separates the area for the TabItemPanel and the ContentPresenter; if you wanted to render the TabItems at the bottom of the control, all you would need to do would be to exchange the *Grid.Row* assignments for the two Border controls (i.e move the TabItemPanel to the bottom row and the ContentPresenter to the top row of the Grid). Likewise, if you wanted the TabItems on the left, you would replace the RowDefinitions with ColumnDefinitions and change the *Grid.Row* assignments to *Grid.Column* assignments.
2. Note the ***IsItemsHost*** assignment in the style - this is where we tell the TabControl where to render the TabItemPanel.
3. Also note the ***ContentPresenter*** assignment - this is is where we tell the TabControl where to render the content of a TabPage.
4. The TabControl has **no top border**; actually, looking from the bottom of the TabControl to its top, the left and right borders stop when they reach the TabItemPanel.

The first three points deserve no further discussion, but let me clarify the fourth: In the XAML, you can see that I chose to not display the top border of the TabControl. However, where would you think the top border would be drawn? Since we're targetting the TabControl itself, one could assume that it would be drawn above the TabItemPanel - that's the top border of the control, after all. However, it would really be drawn at the top of the *ContentPresenter* (i.e. at the top of the blue area in the screenshot), separating the ContentPresenter and the TabItemPanel. So, from a Style-perspective, the TabControl is really targetting the area of the ContentPresenter rather than the control itself (including the TabItemPanel).

That being said, if we were drawing the top border, the control would look like this (click to enlarge):



Here you'll notice that, while we now have a line underneath the TabItems (the strips, that is) as well as underneath the TabItemPanel's empty area (above the right arrow), this is not really what we want because this line is also drawn underneath the selected TabItem (Tab 1, in this case, above the left arrow). Since we'll cover a workaround for this later on, let's just keep in mind that we won't draw a top border.

Styling the TabItem

```
    <Setter Property="SnapsToDevicePixels" Value="true"/>
    <Setter Property="Template">
       <Setter.Value>
          <ControlTemplate TargetType="TabControl">
             <Grid>
                <Grid.RowDefinitions>
                   <RowDefinition Height="Auto"/>
                   <RowDefinition Height="*"/>
                </Grid.RowDefinitions>

                <!--
                         The Border around each TabItem will allow us to draw the line
                         between the TabItemPanel and the TabControl (resp. the
                         TabPage-container) when a TabItem is NOT selected, which
                         replaces the bottom line of the TabItemPanel's border.
                         Thus, we'll avoid drawing the bottom line for the selected
                         TabItem. Also, since the TabItem, when selected, applies a left
                         Margin of 4px, we need to add these here as Padding.
                      -->
                <Border Background="{StaticResource TabItemPanel_BackgroundBrush}"
                        Padding="{StaticResource TabItemPanel_Padding}">
                   <!-- This is the area in which TabItems (the strips) will be drawn. -->
                   <TabPanel IsItemsHost="True"/>
                </Border>

                <!--
                         This is the outer border of the TabControl itself, actually meaning
                         the Panel that will host a TabItem's content.
                         The top-border here will not be drawn as, otherwise, the TabItemPanel
                         would always show a thin line for the selected Tab (which we want
                         to avoid).
                      -->
                <Border BorderThickness="1,0,1,1"
                        Grid.Row="1"
                        BorderBrush="{StaticResource TabItem_BorderBrush_Selected}"
                        Background="{StaticResource TabControl_BackgroundBrush_Base}">
                   <!-- This is the first/outer Border drawn on the TabPage -->
                   <Border BorderThickness="1"
                           BorderBrush="{StaticResource TabPage_InnerBorderBrushDark}"
                           CornerRadius="3"
                           Margin="8">
                      <!--
                               This is the second/inner Border drawn on the TabPage.
                               This Border is drawn with a horizontal Gradient that is transparent
                               on the left which produces the fading effect.
                            -->
                      <Border BorderThickness="1"
                              BorderBrush="{StaticResource TabPage_InnerBorderBrushBright}"
                              CornerRadius="2"
                              Margin="0"
                              Padding="2,2,3,3"
                              >
                         <!--
                                  This is where the Content of the selected TabPage
                                  will be rendered.
                               -->
                         <ContentPresenter ContentSource="SelectedContent" Margin="0"/>
                      </Border>
                   </Border>
                </Border>
             </Grid>
          </ControlTemplate>
       </Setter.Value>
    </Setter>
 </Style>

 <!-- The Style for TabItems (strips). -->
 <Style TargetType="{x:Type TabItem}">
    <Setter Property="Template">
       <Setter.Value>
          <ControlTemplate TargetType="{x:Type TabItem}">
             <!-- The Grid helps defining the general height of TabItems. -->
             <Grid Height="35" VerticalAlignment="Bottom">
                <!--
                         The important aspect here is that the bottom of the Border is 0px thick,
                         helping the TabItem/strip to blend into the TabPage.

                      -->
                <Border Name="Border"
                        Background="{StaticResource TabItem_BackgroundBrush_Unselected}"
                        BorderBrush="{StaticResource TabItem_BorderBrush_Selected}"
                        Margin="{StaticResource TabItemMargin_Selected}"
                        BorderThickness="2,1,1,0"
                        CornerRadius="3,3,0,0"
                        >
                   <!-- 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"
                                     RecognizesAccessKey="True"/>
                </Border>
```

Compared to the previously defined TabControl Style, there now is a border inside the TabControl's ContentPresenter (actually it's two - an outer blue one and a [fading] inner one).
Regarding the TabItems' Style, all I did was to determine the fundamental appearance, such as the Grid that hosts a TabItem, the Border inside and the Content presenter which is actually being refered to as the ***ContentSite***. Also, some brushes make up for the (default) appearance of each TabItem.

The above would look like this (click to enlarge):

This is still kept very simple and doesn't actually give us what we want. If you run this sample you'll notice that no change whatsoever is applied to the appearance of a TabItem when, for instance, it is being selected. Yikes.

Triggers to the rescue

In order to provide a different appearance for different ***states*** of a TabItem, Triggers will help us solve the task. From my perspective, there's the following states that a TabItem can be in:

1. Unselected (i.e., the default)
2. Selected
3. Disabled
4. Hover (i.e., the mouse is over the TabItem)

Generally, all states should be easy to differentiate which, IMHO, is best done with size and color. That is, each state should have its own (set of) color(s) and size; I won't pay much attention to the colors here (you'll find them all in the XAML of the sample solution), so let's just note that these will be part of the Triggers being applied below. The different sizes of the TabItems deserve a little more attention though. So, let's concentrate on the height (changes) applied for the above states:

1. Unselected TabItems should have the lowest height; this includes TabItems that are presently disabled (*IsEnabled = "False"*).
2. Selected TabItems should have the largest height.
3. When the mouse hovers over a TabItem, the height should be somewhere in between the height of selected and unselected TabItems.

Also, I'd like the ZIndex to change with respect to the TabItem's state. That is, the ZIndex should be applied as follows, from back to front:

1. Disabled (lowest)
2. Unselected
3. Hover
4. Selected (highest)

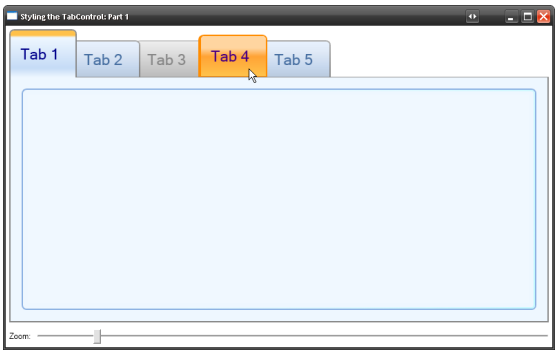
All the above can be achieved with the help of triggers. Here's the four triggers which would need to be added before the end of the TabItem's Style:

```
<ControlTemplate.Triggers>
   <!-- The appearance of a TabItem when it's inactive/unselected -->
   <Trigger Property="IsSelected" Value="False">
      <Setter Property="Panel.ZIndex" Value="90" />
      <Setter TargetName="Border" Property="BorderBrush"
                              Value="{StaticResource TabItem_Border_Unselected}" />
      <Setter Property="Foreground"
                              Value="{StaticResource TabItem_TextBrush_Unselected}" />
      <!-- Except for the selected TabItem, tabs are to appear smaller in height. -->
      <Setter TargetName="Border" Property="Margin"
                              Value="{StaticResource TabItemMargin_Base}"/>
   </Trigger>

   <!--
                        The appearance of a TabItem when it's disabled
                        (in addition to Selected=False)
                     -->
   <Trigger Property="IsEnabled" Value="False">
      <Setter Property="Panel.ZIndex" Value="80" />
      <Setter TargetName="Border" Property="BorderBrush"
                              Value="{StaticResource TabItem_DisabledBorderBrush}" />
      <Setter TargetName="Border" Property="Background"
                              Value="{StaticResource TabItem_BackgroundBrush_Disabled}" />
      <Setter Property="Foreground"
                              Value="{StaticResource TabItem_TextBrush_Disabled}" />
   </Trigger>

   <!-- The appearance of a TabItem when the mouse hovers over it -->
   <MultiTrigger>
      <MultiTrigger.Conditions>
```

```
      <Setter Property="BorderBrush"
                              TargetName="Border"
                              Value="{StaticResource TabItem_HoverBorderBrush}" />
      <Setter TargetName="Border" Property="BorderThickness" Value="2,1,1,1" />
      <Setter Property="Background" TargetName="Border"
                              Value="{StaticResource TabItem_HoverBackgroundBrush}"/>
      <!--
                         To further increase the hover-effect, extend the TabItem's height a little
                         more compared to unselected TabItems.
                      -->
      <Setter TargetName="Border" Property="Margin"
                              Value="{StaticResource TabItemMargin_Hover}"/>
   </MultiTrigger>

   <!-- The appearance of a TabItem when it's active/selected -->
   <Trigger Property="IsSelected" Value="True">
      <!-- We want the selected TabItem to always be on top. -->
      <Setter Property="Panel.ZIndex" Value="100" />
      <Setter TargetName="Border" Property="BorderBrush"
                              Value="{StaticResource TabItem_BorderBrush_Selected}" />
      <Setter TargetName="Border" Property="Background"
                              Value="{StaticResource TabItem_BackgroundBrush_Selected}" />
      <Setter TargetName="Border" Property="BorderThickness" Value="1,1,1,0" />
      <Setter Property="Foreground"
                              Value="{StaticResource TabItem_TextBrush_Selected}"/>
      <Setter TargetName="Border" Property="Margin"
                              Value="{StaticResource TabItemMargin_Selected}"/>
   </Trigger>
 </ControlTemplate.Triggers>
```

In the above XAML you can see that one Trigger is actually a ***MultiTrigger*** - why is that? The MultiTrigger is required due to the fact that, in order to create a *Hover*-Style, we need to pay attention to more than a single property. While, for the *Selected*-Trigger, we only need to pay attention to the *IsSelected* state of the TabItem, we need to also watch out for the position of the mouse for the *Hover*-Trigger to work correctly.

Here's what the Window will look like with the Triggers added (see the start of the article for a video that shows the control in action):



That pretty much concludes part one.

But wait!

What happened to the line between the TabItemPanel and the TabControl's ContentPresenter? Maybe someone else has an easier approach to working around this, but - FWIW - here's what I did.

Actually it's pretty simple - in the above screenshot, the lines you see between the TabItemPanel, the TabItems and the TabControl resp. ContentPresenter aren't lines, but rather the result of some stupid **gradients** being drawn. Let me explain that; instead of drawing a line (or border), I'm painting the "line" along with the background. Let's take the TabItemPanel as an example. When the TabItemPanel and its content is rendered, the TabItemPanel itself will be rendered **before** the TabItems themselves. That is, if you drew the TabItemPanel with a red background, the TabItems would overlay the red background. However (*sadly!*), this doesn't apply to the Border of the TabItemsPanel - this Border would be drawn **above** the TabItems. Thus, we cannot use that Border to render our line in the area that is not covered by TabItems. Pretty much the same concept applies to the Border of the TabControl - we cannot draw its top Border and selectively hide it.

So, back to taking the TabItemPanel as an example, we want the TabItemPanel to be generally transparent, so how do we insert a line at the bottom? The simple trick is to define a (vertical) LinearGradientBrush that is transparent for 99% of its height, with the bottom 1% being drawn with the same color that is being used to draw the the TabControl's Border-lines. Since the *SnapsToDevicePixels* property has been set to *True*, this will result in the last 1% to make up for a 1px line. Not pretty, but it works.

So, here's the Brush that is being used to draw the background of the TabItemPanel:

```
<LinearGradientBrush x:Key="TabItemPanel_BackgroundBrush"
                     StartPoint="0,0" EndPoint="0,1">
   <LinearGradientBrush.GradientStops>
      <GradientStop Offset="0.98" Color="Transparent"/>
      <GradientStop Offset="0.99"
```

leaves me, however, if you know of a simpler way of dealing with this, I'm all ears!

A note regarding the definition of Thickness-resources in VS2008

When you load the sample solution, VS2008 (I haven't tried VS2010 so I can't tell whether the issue has disappeared in .Net4) may or may not give you a compile error that is geared at the definition of the Margin-resources. These are defined like:

```
<Thickness x:Key="TabItemMargin_Base">0,8,-4,0</Thickness>
```

The Visual Studio 2008 compiler seems to dislike this format, even though it is perfectly valid. If you see an error, just hit *Ctrl-B* to do a build and the error will go away. However, it might re-appear at some point, namely when you change a portion of the XAML close to one of those definitions. If that disturbes you, use the alternative syntax, i.e., the equivalent for the above excerpt:

```
<Thickness x:Key="TabItemMargin_Base" Left="0" Top="8" Right="-4" Bottom="0"/>
```

I opted to use the first format because I simply consider it to be much better readable compared to the second.

The last word

As you probably noticed at this point, there's a bunch of resources being used throughout the XAML. The concept behind that is the attempt to define stuff only once and reuse it wherever required. In the previous XAML, I referenced the resource *BorderColor_Base*. This is a color that is being used in many places, hence I defined the color at the very beginning of the XAML and reference it wherever appropriate.

Also, the XAML in the sample solution contains a lot of comments (and another trick or two) that I skipped here in order to keep the XAML-sections short and to allow to better concentrate on the points I wanted to discuss. I thus encourage you to download the sample solution and check the XAML yourself.

And, last but not least, I'm always happy to receive some feedback for the stuff I'm publishing. So, please leave a comment, regardless of wether you liked it or not.

The sample solution

I've created a sample solution that contains everything discussed here. This time, the solution is C# only, but there is no code behind involved whatsoever (not taking the main form into account) which is why you won't find a VB counterpart. However, if you want to use this in a VB-project, simply paste the XAML into your VB-window and remove the ***TabControlStyle.*** that is preluding each window's ***x:Class*** attribute (and indicating the namespace that is required for C#).

Download: TabControlStyle - Part One.zip (27.32 kb)

Location: SinglePost

Currently rated 4.9 by 27 people

Tags: wpf, tabcontrol, tabitem, tabpanel, styles TabControl | WPF (.net)

E-mail | Kick it! | DZone it! | del.icio.us Permalink | Comments (10) | Post RSS

Related posts

WPF: TabControl series - Part 3: Non-wrapping scrollable TabPanel; TabItem DropDown-Menu
How to style the WPF TabControl and the TabItems resp. the TabPanel (part three - non-wrapping scro...

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

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

Comments

Ayman June 22. 2011 15:16

V.nice article ,
but the controls size inside the tabs is very big , which property can fix this ??

thanks

Olaf Rabbachin (website link) June 22. 2011 16:03

Hi Ayman,

would it be possible that you didn't notice the slider at the bottom which controls the zoom-level? By default, it's set to zoom at factor 2. Drag it to the left to see the original/regular size.

Cheers,
Olaf

Ayman June 23. 2011 10:08

Thanks for your reply ,

and yes i notice the slide at the bottom ,

but i'd like to the set the controls in the tab content to it's normal size but leave the tab header at this size ?

Olaf Rabbachin (website link) June 23. 2011 10:19

Hi Ayman,

then you'll have to apply a scale-transform to the panel (the ContentPresenter) that hosts the TabItems. IOW, look for this line:
<!-- This is where the Content of the TabItem will be rendered. -->
Replacing the ContentPresenter underneath the above line with the following might do what you're after:

```
<ContentPresenter x:Name="ContentSite"
                  VerticalAlignment="Center"
                  HorizontalAlignment="Center"
                  ContentSource="Header"
                  Margin="7,2,12,2"
                  RecognizesAccessKey="True">
  <ContentPresenter.LayoutTransform>
    <ScaleTransform CenterX="0"
                    CenterY="0"
                    ScaleX="2"
                    ScaleY="2"/>
  </ContentPresenter.LayoutTransform>
</ContentPresenter>
```

Remember that the zoom-slider will still be active, doubling the effect as long as it remains on its default setting (zoom=2x).

Cheers,
Olaf

Ayman June 23. 2011 11:29

Hi Olaf

Thanks , this works great

I'm new in WPF programming - just used WinForms with VB .net in the past.

I was playing around with your Tab Templates and the results are amazing
I'm just wondering how to adapt the Template in order to get TabStripItems placed at the bottom instead of the top (or maybe left, right side as well).

I was seeking the template for this information, but didn't find it
The TabStripPlacement property is not working anymore as soon as I have applied the template.

Many thanks for a hint and thanks a lot again for this tutorial

Cheers
Stefan

Olaf Rabbachin (website link) August 25. 2011 13:02

Hi Stefan,

the whole series is actually really only supporting a TC with its TabItems rendered at the top. However (and regarding part 1 of the series), if you want them at the bottom, you would basically (!) get away with simply exchanging the RowDefinitions and control-positions in the TI-Template:
In the article, look for the section "*Styling the TabControl*". There, exchange the two RowDefinitions; then make the first border go into the second and the second border go into the first row of the Grid.
Then, however, you'll have to also change everything that affects the layout. For instance, the coloring that makes the TabItems look as if they fade into the TabItem itself. Here, you'd also have to change the LinearGradientBrush (look for "*TabItemPanel_BackgroundBrush*") accordingly. Plus the margins, animations and all that other fancy stuff; and that's all only off the top of my head. That is, there might be more/other objects that you'd have to change.

Having TabItems appear on the left or right would require even more work, such as the re-arrangement of the TC itself (i.e. ColumnDefinitions instead of RowDefinitions)
If this was done right, there'd be one template for each left/right/top/bottom plus a Trigger that would apply the right one depending on what TabItem-alignment was set on the TC itself. However, I was too lazy to do that, especially with respect to the fact that the other parts of this tutorial would add a **whole lot** of work to support the other alignments.

So, in the end, I'm sorry to say that moving the *TabPanel* (which erroneously is called the *TabItemPanel* in this article) is not as straight-forward/easy as it might seem ...

Cheers & Gruß,
Olaf

Stefan August 27. 2011 11:28

Hi Olaf,

many thanks for your swift reply.
I will try to build a updated template supporting TabPanel at the bottom as well. Will let you know my results - hopefully I will have some
Thanks again for the hints

Cheers
Stefan

John (website link) December 5. 2011 13:05

About the border between the TabItemPanel, as you write "this Border would be drawn above the TabItems", that's just what it looks like. What actually happens, is that the TabItems are clipped to fit inside the border. Give the TabItems a negative bottom margin, and they will be drawn on top of the TabItemPanel bottom border.

Comments are closed