

http://www.blogs.intuidev.com/post/2010/01/26/TabControlStyling_PartTwo.aspx

Go

JAN APR MAY

06

2015 2016 2017

About this capture

61 captures

1 Mar 2010 - 6 Aug 2017

Olaf Rabbachin

WPF, WinForms, SQL Server, Office et al

HomeArchiveContactAboutLog in

SubscribeFilter by APML

<< WPF: TabControl Series - Part 1: Colors and Sizes | WPF: ColorHelper - how to retrieve the name of a given color or to retrieve a color by its name >>

WPF: TabControl series - Part 2: Animating TabItems

By [Olaf Rabbachin](#)26. January 2010 17:36

Introduction

In my previous article, I started the TabControl series, demonstrating how to define a new Style for the TabControl, its TabItemPanel and the TabItems. Today I will extend the sample introduced there, adding transition effects that are being applied to the TabItems when they change state (for instance, from *Selected* to *Unselected*).

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 (this article)
- 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

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

Status quo (after *Part One*)

As noted before, this article is based upon the stuff I introduced in *Part One*. If you haven't read it and find that you need to find out more about the basics, I suggest that you start there.

Here's where we'll start here, i.e. what the TabControl and its "sub-controls" looked like at the end of *Part One*:

Animating ... what?

Basically, the above style IMHO makes up for a much better appearance compared to the original template. However, when the user hovers over a TabItem or selects a different TabItem, the change will be instant. The goal of this article is to change that so that there's more of a smooth transition between states.

Let's do a quick recap on the different states that a TabItem can take:

- Unselected (i.e., the default)
- Selected
- Disabled
- Hover (i.e., the mouse is over the TabItem)

I'll leave out the *Disabled* state with respect to animations as I presume that, for most of all times, that state will have

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

2355 visitors
Mar. 01st - Mar. 31st



Click to see details

Category list

Access/VBA	
General	
SQL Server	
TabControl	
Utilities	
WPF (.Net)	
XML	

Month List

2011
2010
2009
2006
2005
2004
2003
2002

Tag Cloud

Access Animation Converter
Forms Helpers Multibinding Paths
Query Reflection Scrollviewer
Storyboards Styles Tabcontrol
Tabitem Tabpanel Treeview Utils
Windowbyname Wpf

Links

[Company Web \(English\)](#)

JAN APR MAY
06
2015 2016 2017

61 captures
1 Mar 2010 - 6 Aug 2017

▼ About this capture

- Unselected » Selected
- Unselected » Hover
- Selected » Unselected
- Hover » Unselected
- Hover » Selected

So, where's the *Selected* » *Hover* state change? I'm simply not considering it due to the fact that I don't want any effect when the mouse hovers over the selected TabItem. As a matter of fact, the style introduced in *Part One* doesn't consider this either.

States of the TabItems

Back when I created *Part One*'s style I thought that adding some animations to the TabItem would be a matter of a couple of minutes. However, as it turned out, the whole thing got me pretty much stuck for quite a while. To be precise, I spent an **absolutely ridiculous** amount of time trying to find out why it wouldn't work as desired. Up until now I still think that there's a bug somewhere in the part of the framework that's responsible for the animations resp. Storyboards. But let's start with the fundamentals first. In order to provide the transition between TabItem-states, we first need to determine what type of animation is the most suitable for what we're planning to achieve. In *Part One*, I introduced a couple of Thickness resources that were targetting the TabItems' Margin property. These were defined as follows:

```
<Thickness x:Key="TabItemMargin_Base">0,8,-4,0</Thickness>
<Thickness x:Key="TabItemMargin_Selected">-4,0,-4,0</Thickness>
<Thickness x:Key="TabItemMargin_Hover">0,2,0,0</Thickness>
<Thickness x:Key="TabItemPanel_Padding">4,0,0,0</Thickness>
```

The last one (*TabItemPanel_Padding*) isn't of any interest with respect to animations (see the comments in the XAML of the sample solution) - it's only part of the group in order to rather have all the Margin assignments in a central place (allowing for easier changing them when required). The resources' names should be pretty self-explanatory for all but the first: *TabItemMargin_Base* actually refers to what I would call the *default state* of TabItems - the *Unselected* state. That being said, whenever a TabItem changes into that state, its size will be determined by the available height minus the 8px top-margin (making it a little smaller in height). Also, adding -4px as the right margin will effectively apply a negative amount, allowing the TabItem to the right (of the one to which this is being applied) to overlay it, which in turn "removes" (covers) the rounded corner on the TabItem's top right, resulting in the overlay-effect (can I call that 2.9D?). The *TabItemMargin_Hover* resource will make two changes compared to *TabItemMargin_Base*: the height-reduction is two pixels less, which will render the hovered TabItem 2px higher in comparison. Also, the negative right margin is removed which results in the hovered TabItem no longer being covered by the TabItem to the right (if there is one anyway). Finally, the *TabItemMargin_Selected* resource further increases the height resp. allows the TabItem to extend to the full height that the TabItemPanel allows for; since this is also the state at which TabItems are to receive the user's strongest attention, negative left and right margins are being applied so that the selected TabItem will always cover the TabItems to the left and right respectively. What you can't see in the XAML above is the fact that the ZIndex property is part of the gameplay here, too. That is, the ZIndex is changed with respect to the TabItem's state - from back to front:

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

ThicknessAnimation

With the above Margin-settings we actually have all we need to apply state-transitions resp. create/define our animations. The framework provides a wealth of animation-types; that being said, there is more than one way to create the transitions. However, there also is the **ThicknessAnimation** which is just what we need, since we really want to animate the TabItems' Margin-property, and the Margin actually is a ... Thickness! Also, utilizing a ThicknessAnimation allows us to simply neglect any starting parameters or values. Instead, we can define all we need only by defining the Margin as it is to be applied when the animation ends - the ThicknessAnimation will thus run from whatever source Margin it uses to the value we specify (there is other combinations such as using *From* or *By* or a combination of them - see the [MSDN docs](#) for more information). To **a)** keep things in a central place and **b)** allow for not having to define something more than once, we'll again make use of resources instead. Here's a sample of the Storyboard-resource that is to run when a TabItem enters the *Selected* state:

```
<Storyboard x:Key="TabItemStoryboard_Selected">
  <ThicknessAnimation Storyboard.TargetName="Border"
    Storyboard.TargetProperty="Margin"
    To="{StaticResource TabItemMargin_Selected}"
    FillBehavior="HoldEnd"
    Duration="0:0:0.1"/>
</Storyboard>
```

A couple of points deserve attention in the above XAML: We want to apply the animation to the Margin-property of the TabItem, hence the *TargetProperty*-assignment. The *TargetName* property is set to *Border* simply because that's the **name** of the the Border-control that we defined in the TabItem's Template and which is the parent-control of its ContentPresenter (see *Part One* for more info). By setting the *FillBehavior* property to *HoldEnd*, we determine that the

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

JAN

APR

MAY

06

2015

2016

2017

About this capture

61 captures

1 Mar 2010 - 6 Aug 2017

name to the *To* property, making up for the state that we'd like to have reached when the animation ends.

The above principle can actually be applied to all other animations in the exact same way, only replacing the Margin-resource in the *To*-property with the respective resource's name.

A (better) alternative?

However, I'd like to introduce another type of animation that allows us to perform the same task with an alternative approach - the **ThicknessAnimationUsingKeyFrames**. If we wanted to instead use the ThicknessAnimationUsingKeyFrames to achieve the same effect as in the prior XAML, the equivalent would look like this:

```
<Storyboard x:Key="TabItemStoryboard_Selected">
  <ThicknessAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                   Storyboard.TargetProperty="Margin"
                                   FillBehavior="HoldEnd">
    <SplineThicknessKeyFrame KeyTime="0:0:0.1"
                           Value="{StaticResource TabItemMargin_Selected}"/>
  </ThicknessAnimationUsingKeyFrames>
</Storyboard>
```

A little bit more complex compared to the ThicknessAnimation, huh, so why the hell bother with it? The reason is that the ThicknessAnimationUsingKeyFrames gives us a chance to define **several states resp. changes** during the scope of a single Storyboard/animation. To better illustrate this (ah well, maybe that's more about finding a valid reason), I thought it'd be fun to utilize this type of animation when animating the change of height that is to be applied when a TabItem enters the *Hover* state. That is, instead of just having the TabItem extend the height by 4px (well, actually we only reduce the reduction, luv'it 😊) when the mouse hovers over it, we'll apply two KeyFrames, the first with a 2px and the second with the final value of a 4px reduction. As a result, the TabItem will first extend above its final height and then swing back to the final value (being 4px). Since we have defined the target margins as resources, we'll simply split up the original TabItemMargin resource into two new ones, dumping the original one:

```
<!--<Thickness x:Key="TabItemMargin_Hover">0,4,-4,0</Thickness>-->
<Thickness x:Key="TabItemMargin_Hover_Start">0,2,0,0</Thickness>
<Thickness x:Key="TabItemMargin_Hover_Final">0,4,0,0</Thickness>
```

Note: If you fell for the right Margin of -4 having been replaced by 0 - don't bother, we'll get to that later on.

Now the ThicknessAnimationUsingKeyFrames comes back into play, which is where all we have to do is to provide the two KeyFrames that will utilize the above two resources:

```
<Storyboard x:Key="TabItemStoryboard_Hover">
  <ThicknessAnimationUsingKeyFrames Storyboard.TargetName="Border"
                                   Storyboard.TargetProperty="Margin"
                                   FillBehavior="HoldEnd">
    <SplineThicknessKeyFrame KeyTime="0:0:0.1"
                           Value="{StaticResource TabItemMargin_Hover_Start}"/>
    <SplineThicknessKeyFrame KeyTime="0:0:0.2"
                           Value="{StaticResource TabItemMargin_Hover_Final}"/>
  </ThicknessAnimationUsingKeyFrames>
</Storyboard>
```

Et voilà - that's all we have to do. And, by the way, both the ThicknessAnimation and the ThicknessAnimationUsingKeyFrames are not restricted to a single dimension such as either height, width, left, top, etc., but to the Thickness definition itself. That is, this would work just as well if we wanted a transition between a Margin of 10,5,0,2 and 5,0,2,10.

At a glance: the Storyboards

Here's all three Storyboards that we need:

```
<!-- This will run when a TabItem enters the "Unselected" state -->
<Storyboard x:Key="TabItemStoryboard_Unselected">
  <ThicknessAnimation Storyboard.TargetName="Border"
                     Storyboard.TargetProperty="Margin"
                     To="{StaticResource TabItemMargin_Base}"
                     FillBehavior="HoldEnd"
                     Duration="0:0:0.1"/>
</Storyboard>
<!-- This will run when a TabItem enters the "Selected" state -->
<Storyboard x:Key="TabItemStoryboard_Selected">
  <ThicknessAnimation Storyboard.TargetName="Border"
                     Storyboard.TargetProperty="Margin"
                     To="{StaticResource TabItemMargin_Selected}"
                     FillBehavior="HoldEnd"
                     Duration="0:0:0.1"/>
</Storyboard>
<!-- This will run when a TabItem enters the "Hover" state -->
<Storyboard x:Key="TabItemStoryboard_Hover">
  <ThicknessAnimationUsingKeyFrames Storyboard.TargetName="Border"
```

61 captures

1 Mar 2010 - 6 Aug 2017

Go

JAN APR MAY

06

2015 2016 2017



About this capture

```
</ThicknessAnimationUsingKeyFrames>
</Storyboard>
```

Alright, now all we have to do is to go ahead and apply the Storyboards we just defined. Since the Triggers that come into play for state-changes are already in place, that would (should) mean that this would be a matter of adding *EnterActions* for the three Triggers we already have in our XAML.

Ain't that easy after all, it seems ...

However, this is where what I thought to be overly simply became more like a nightmare. Since this is a rather long story, you can read this [thread on the MSDN WPF forums](#) (German!) for the **very** long story or to [this thread](#) (in English) for the long story. I have posted a simplified and ready-to-run sample in both threads which illustrates the issues I was encountering. Since I don't want to rap so much about what *doesn't* work but rather demonstrate what *does*, let's just cut this short - here's what I *really* had to do and what you might stumble over in the XAML ahead:

1. While the documentation states that the default *RepeatBehavior* of a Storyboard means that the animation will only run once, there seems to be a bug of some sort that forced me to manually **stop** them at certain occasions.
2. The above in turn leads to the necessity of naming the Storyboards when they are started from the Triggers. Since the names have to be unique, I suffixed them with an indication of the respective Trigger it was created in.
3. There seems to be some sort of interference that must be happening due to the fact that the Storyboard that runs when a TabItem enters the *Unselected* state is being used multiple times (i.e. Hover » Unselected and Selected » Unselected). This forced me to replace the *Unselected* Trigger to a MultiTrigger (analogous to the Hover-MultiTrigger).

Ah well ...

At a glance: the Triggers with the Storyboards applied

... here's the complete XAML section in which all Triggers are defined and which is where the Storyboards are put into action:

```
<ControlTemplate.Triggers>
  <!-- The appearance of a TabItem when it's inactive/unselected -->
  <MultiTrigger>
    <MultiTrigger.Conditions>
      <Condition Property="Border.IsMouseOver" Value="False"/>
      <Condition Property="IsSelected" Value="False"/>
    </MultiTrigger.Conditions>
    <!-- The Triggers required to animate the TabItem when it enters/leaves the "Unselected" state (added
in part two) -->
    <MultiTrigger.EnterActions>
      <BeginStoryboard x:Name="sbUnselected"
                      Storyboard="{StaticResource TabItemStoryboard_Unselected}"/>
    </MultiTrigger.EnterActions>
    <MultiTrigger.ExitActions>
      <StopStoryboard BeginStoryboardName="sbUnselected"/>
    </MultiTrigger.ExitActions>
    <Setter Property="Panel.ZIndex" Value="90" />
    <Setter TargetName="Border" Property="Background"
            Value="{StaticResource TabItem_BackgroundBrush_Unselected}" />
    <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}"/>
  </MultiTrigger>

  <!--
      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}" />
    <Setter TargetName="Border" Property="Margin"
            Value="{StaticResource TabItemMargin_Base}"/>
  </Trigger>

  <!-- The appearance of a TabItem when the mouse hovers over it -->
  <MultiTrigger>
    <MultiTrigger.Conditions>
      <Condition Property="Border.IsMouseOver" Value="True"/>
      <Condition Property="IsSelected" Value="False"/>
    </MultiTrigger.Conditions>
```

JAN APR MAY
06
2015 2016 2017

61 captures
1 Mar 2010 - 6 Aug 2017

About this capture

```

<BeginStoryboard x:Name="sbHover" Storyboard="{StaticResource TabItemStoryboard_Hover}"/>
</MultiTrigger.EnterActions>
<MultiTrigger.ExitActions>
  <BeginStoryboard x:Name="sbUnselected_Hover_Exit" Storyboard="{StaticResource
TabItemStoryboard_Unselected}"/>
</MultiTrigger.ExitActions>
<Setter Property="Panel.ZIndex" Value="99" />
<Setter Property="Foreground" Value="{StaticResource TabItem_TextBrush_Hover}" />
<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_Final}"/>
<!--
    At runtime, we want a transition when changing between the regular/hover/regular
    states.
-->
</MultiTrigger>

<!-- The appearance of a TabItem when it's active/selected -->
<Trigger Property="IsSelected" Value="True">
  <!-- The Triggers required to animate the TabItem when it enters/leaves the "Selected" state (added
in part two) -->
  <Trigger.EnterActions>
    <StopStoryboard BeginStoryboardName="sbUnselected_Selected_Exit"/>
    <BeginStoryboard x:Name="sbSelected"
      Storyboard="{StaticResource TabItemStoryboard_Selected}"/>
  </Trigger.EnterActions>
  <Trigger.ExitActions>
    <BeginStoryboard x:Name="sbUnselected_Selected_Exit" Storyboard="{StaticResource
TabItemStoryboard_Unselected}"/>
  </Trigger.ExitActions>
  <!-- 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>

```

But wait!

If you stumbled over the fact that I got rid of the right Margin of 4px for the hover-effect when I replaced the *TabItemMargin_Hover_Start* Margin resource with the split up ones - here's the reason. If you run the sample solution, comparing the two Windows it comes with (one for what we started with and one for what we have now), you'll notice that, for the non-animated sample, the hover-effect will include the hovered TabItem to come up to the front. The reason is simple - each state-Trigger comes with its own ZIndex-setting; the Hover-Trigger's ZIndex places it only behind the selected TabItem, but in front of all others. Now, by removing the right Margin in the animated TabItem's Style, we actually animate the width of the item along the way, resulting in the hovered TabItem seeming to "push" any other TabItem to its right out of the way. Hey, I like that. We could as well have defined the right margin of the *TabItemMargin_Hover_Start* resource (i.e., the first of the animation's two frames) to be 1px which would've smoothed the effect a little more, but I'll leave that up to you.

The last word

This concludes Part Two of the TabControl series. As always, I'm very happy to receive any kind of feedback for the stuff I'm publishing.

The sample solution

I've created a sample solution that contains everything discussed here. As with Part One, 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 precluding each window's **x:Class** attribute (and indicating the namespace that is required for C#).

Go

JAN APR MAY

06

2015 2016 2017

61 captures

1 Mar 2010 - 6 Aug 2017

▼ About this capture

Currently rated 4.8 by 11 people

Tags: [wpf](#), [tabcontrol](#), [tabitem](#), [tabpanel](#), [styles](#), [animation](#), [storyboards](#)

TabControl | WPF (.net)

E-mail | [Kick it!](#) | [DZone it!](#) | [del.icio.us](#)

[Permalink](#) | [Comments \(7\)](#) | [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 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 4: Closeable TabItems](#)
How to style the WPF TabControl and the TabItems resp. the TabPanel (part four - closeable TabItems...

Comments




Günter Schwaiger

January 27, 2010 12:07




Cool blog post! That looks like a lot of work.




Olaf Rabbachin ([website link](#))

January 27, 2010 13:23




Hi Günter,
thanks for the comment. Yes, it sure was more work than I thought it would've been. You should know because you were part of the forums discussion after all ... 😊
So, thanks for your help there!




Pedro

February 8, 2010 12:15




beautiful job!
congratulations.




Olaf Rabbachin ([website link](#))

February 8, 2010 12:23




Hi Pedro,
thanks for the comment!




Scott

June 30, 2010 07:14




Thank you for the tutorial series. It is very helpful. What would it take to style the tab to look slanted like the SAP's WebGUI that you have in Part 3?



Olaf Rabbachin ([website link](#))

June 30, 2010 09:29



Hi Scott,
funny that you're asking - I haven't yet done that myself, but - now that cProjects 5.0 is in ramp up - we might have to create a new version of our control library as well. If we do, it will most likely be WPF that we'll be using ...

Go

JAN APR MAY

06

2015 2016 2017

About this capture

61 captures

1 Mar 2010 - 6 Aug 2017

Olaf

 Umang Patel

September 28, 2010 20:15



Thanks Olaf.

Its very helpful to me in understanding details of Tab Controls.

Comments are closed

Powered by: [BlogEngine.NET 1.6.0.0](#) | Theme: [stablestart](#) | Design by: [Thomas A. Bosscher](#)