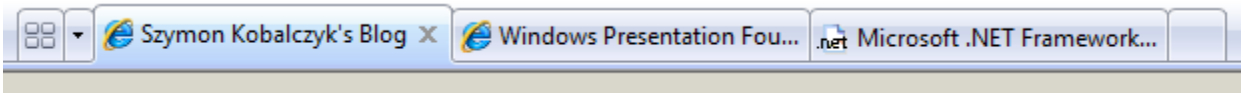
WPF TabItems With Close Button

One of the most common questions I've seen regarding the TabControl in Windows Forms was how to add a close button to each tab (similar to seen on tabs in Internet Explorer 7).



Although there were some solutions available the results weren't quite satisfactory and often requiring to rewrite the whole control from scratch. Recently I faced the same challenge working on the TSRI project. It turned out that in WPF this pretty straightforward task and in this article I'm going to show all the steps required to complete it.

To follow the discussion you can download the demo code first:

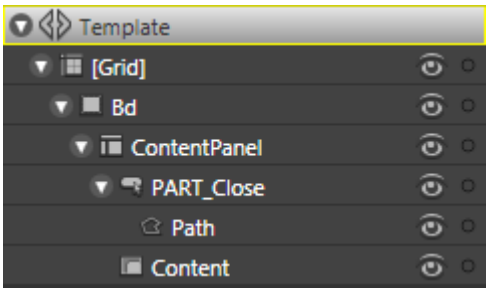
Download the source code

We start by creating a new custom control deriving from TabItem that implements this behavior. I'll name it `CloseableTabItem`.

```
public class CloseableTabItem : TabItem
{
    static CloseableTabItem()
    {
        DefaultStyleKeyProperty.OverrideMetadata(typeof(CloseableTabItem),
            new FrameworkPropertyMetadata(typeof(CloseableTabItem)));
    }
}
```

The instruction in static constructor informs the system that this element wants to use different style than it's parent. Since we are creating the default theme for the custom control it would be defined in `generic\themes.xaml`.

Before we add more code let's create the control template first. With the aid of Expression Blend we can easily create a copy of the default template for TabItem control and start from there. The default template consists only of `Grid` and `Border` that wrap the `ContentPresenter`. We need to place additional DockPanel inside this `Border` to host both the Content and our close `Button`. Because we will reference this button from code later it's named `PART_Close` following the WPF naming convention. The button contains the "x" icon defined as `Path` element. You can see final hierarchy of elements on the image below:

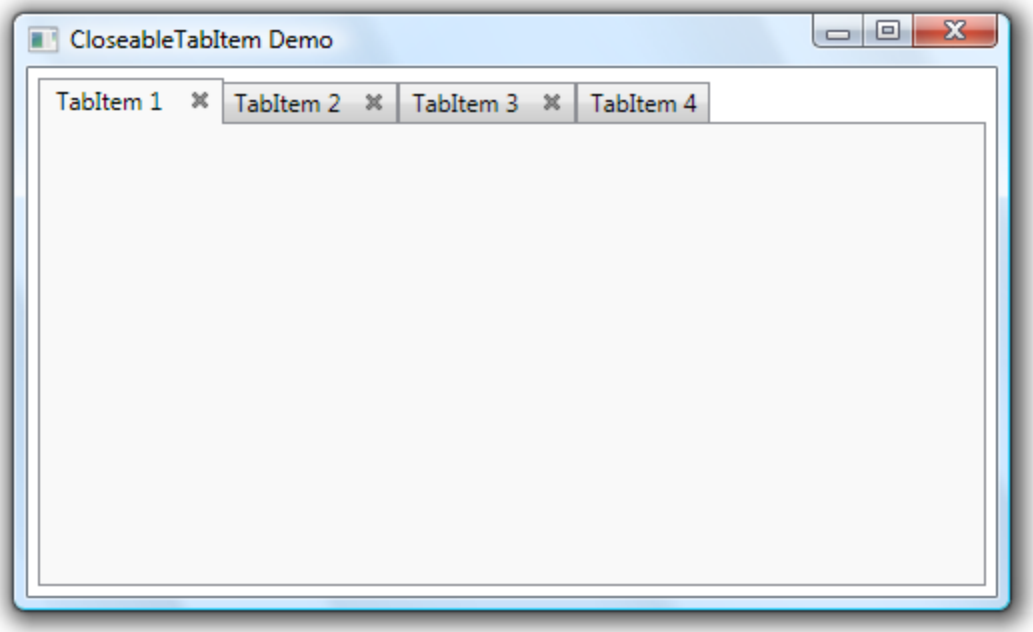


The close button should only show it's border when mouse is over and it shouldn't accept keyboard focus, hence there is additional style called `CloseableTabItemButtonStyle`. This template consist of Border and ContentPresenter inside of a Grid. The Border is hidden by default and shows only when mouse is over the button. To be consistent with IE7 behavior I've also added triggers to change the "x" icon fill color to red when mouse is over the button or when it's pressed.

We are now ready to test these templates. To use our new control first the containing namespace must be mapped to a XML namespace. We don't need to reference the resource dictionary because it is declared as default theme for the control. Here is example markup for the main window of the application:

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:CloseableTabItemDemo"
    x:Class="CloseableTabItemDemo.MainWindow"
    Title="CloseableTabItem Demo" Height="300" Width="500"
    >
    <Grid>
      <TabControl Margin="5">
        <local:CloseableTabItem Header="TabItem 1"/>
        <local:CloseableTabItem Header="TabItem 2" />
        <local:CloseableTabItem Header="TabItem 3" />
        <TabItem Header="TabItem 4" />
      </TabControl>
    </Grid>
</Window>
```

Note that we can freely mix the regular TabItems with our custom controls. Here is the result when we run this application:



Having all the visuals in place lets switch back to Visual Studio and finish the remining code. I choose to publish the Close button click event as the a routed event on the control. Of course I could handle it directly in the code of the control but this way I have more control on how to handle it in the application (for example I could display a confirmation dialog before closing the tab). Alternatively I could also create a custom Command and bind it directly to the close button. This would allow me to declare everything in XAML markup but I think event would be easier to use in this case. So below is the declaration for the CloseTab event:

```
public static readonly RoutedEvent CloseTabEvent =
    EventManager.RegisterRoutedEvent("CloseTab", RoutingStrategy.Bubble,
        typeof(RoutedEventHandler), typeof(CloseableTabItem));

public event RoutedEventHandler CloseTab
{
    add { AddHandler(CloseTabEvent, value); }
    remove { RemoveHandler(CloseTabEvent, value); }
}
```

To raise the event I need first to attach a handler to the Button's Click event. I can do it easily by overriding the ... method. This is where I use the `PART_Close` name mentioned earlier to find the button declared in template by using the `GetTemplateChild` method. The event handler simply raises the new event.

```
public override void OnApplyTemplate()
{
    base.OnApplyTemplate();

    Button closeButton = base.GetTemplateChild("PART_Close") as Button;
    if (closeButton != null)
        closeButton.Click += new
System.Windows.RoutedEventHandler(closeButton_Click);
}

void closeButton_Click(object sender, System.Windows.RoutedEventArgs e)
{
    this.RaiseEvent(new RoutedEventArgs(CloseTabEvent, this));
}
```

So the only thing left is to actually handle this event. This will be done in the MainWindow's code behind. Normally I would have to attach handlers for this event to each tab I created but since this is a routed event (with bubble strategy) I can also attach it once on any of it's parents (up to the Window itself). Closing the tab is done by finding the parent TabControl removing the source tab from it's Items collection.

```
public partial class MainWindow : System.Windows.Window
{
    public MainWindow()
    {
        InitializeComponent();

        this.AddHandler(CloseableTabItem.CloseTabEvent, new
RoutedEventHandler(this.CloseTab));
    }

    private void CloseTab(object source, RoutedEventArgs args)
```

```
    {
        TabItem tabItem = args.Source as TabItem;
        if (tabItem != null)
        {
            TabControl tabControl = tabItem.Parent as TabControl;
            if (tabControl != null)
                tabControl.Items.Remove(tabItem);
        }
    }
}
```

That's all we need to do. Now we have customized tab items with fully working tab control. And as you can see this is relatively easy to implement so nothing prevents adding more buttons or other elements on the tabs. For example in the TSRI project we have additional button that opens the tab contents as floating window.