<http://stackoverflow.com/questions/2282662/wpf-textblock-font-resize-to-fill-available-space-in-a-grid>

<Viewbox StretchDirection="DownOnly" Stretch="Uniform">

<TextBlock Text="{Binding Path=Title}" HorizontalAlignment="Center"/>

</Viewbox>

<https://social.msdn.microsoft.com/Forums/vstudio/en-US/2062f6ab-b4b5-487c-890a-9c3ffcdc1d2e/karaoke-text-effect-in-wpf?forum=wpf>

<TextBlock FontSize="20" Text="the text will hightlight base on a synced time">

<TextBlock.Foreground>

<LinearGradientBrush StartPoint="0,0.5" EndPoint="1,0.5">

<GradientStop Color="Red" Offset="0"/>

<GradientStop x:Name="GradientStop2" Color="Red" Offset="0"/>

<GradientStop Color="Black" Offset="{Binding ElementName=GradientStop2, Path=Offset}"/>

<GradientStop Color="Black" Offset="1"/>

</LinearGradientBrush>

</TextBlock.Foreground>

<TextBlock.Triggers>

<EventTrigger RoutedEvent="MouseLeftButtonDown">

<BeginStoryboard Name="karaokeTextEffect">

<Storyboard>

<DoubleAnimation From="0.0" To="1.0" Duration="0:0:1" Storyboard.TargetName="GradientStop2" Storyboard.TargetProperty="(GradientStop.Offset)"></DoubleAnimation>

</Storyboard>

</BeginStoryboard>

</EventTrigger>

<EventTrigger RoutedEvent="MouseLeftButtonUp">

<RemoveStoryboard BeginStoryboardName="karaokeTextEffect"/>

</EventTrigger>

</TextBlock.Triggers>

</TextBlock>

<http://www.wpfsharp.com/2012/08/24/a-wpf-searchable-textblock-control-with-highlighting/>

using System;

using System.Text.RegularExpressions;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using System.Windows.Media;

using System.Collections.Generic;

namespace HighlightText

{

    public class SearchableTextBlock : TextBlock

    {

        #region Constructors

        // Summary:

        //     Initializes a new instance of the System.Windows.Controls.TextBlock class.

        public SearchableTextBlock()

        {

            //Binding binding = new Binding("HighlightableText");

            //binding.Source = this;

            //binding.Mode = BindingMode.TwoWay;

            //SetBinding(TextProperty, binding);

        }

        public SearchableTextBlock(Inline inline)

            : base(inline)

        {

        }

        #endregion

        #region Properties

        new private string Text

        {

            set

            {

                if (string.IsNullOrWhiteSpace(RegularExpression) || !IsValidRegex(RegularExpression))

                {

                    base.Text = value;

                    return;

                }

                Inlines.Clear();

                string[] split = Regex.Split(value, RegularExpression, RegexOptions.IgnoreCase);

                foreach (var str in split)

                {

                    Run run = new Run(str);

                    if (Regex.IsMatch(str, RegularExpression, RegexOptions.IgnoreCase))

                    {

                        run.Background = HighlightBackground;

                        run.Foreground = HighlightForeground;

                    }

                    Inlines.Add(run);

                }

            }

        }

        public string RegularExpression

        {

            get { return \_RegularExpression; }

            set

            {

                \_RegularExpression = value;

                Text = base.Text;

            }

        } private string \_RegularExpression;

        #endregion

        #region Dependency Properties

        #region Search Words

        public List SearchWords

        {

            get

            {

                if (null == (List)GetValue(SearchWordsProperty))

                    SetValue(SearchWordsProperty, new List());

                return (List)GetValue(SearchWordsProperty);

            }

            set

            {

                SetValue(SearchWordsProperty, value);

                UpdateRegex();

            }

        }

        // Using a DependencyProperty as the backing store for SearchStringList.  This enables animation, styling, binding, etc...

        public static readonly DependencyProperty SearchWordsProperty =

            DependencyProperty.Register("SearchWords", typeof(List), typeof(SearchableTextBlock), new PropertyMetadata(new PropertyChangedCallback(SearchWordsPropertyChanged)));

        public static void SearchWordsPropertyChanged(DependencyObject inDO, DependencyPropertyChangedEventArgs inArgs)

        {

            SearchableTextBlock stb = inDO as SearchableTextBlock;

            if (stb == null)

                return;

            stb.UpdateRegex();

        }

        #endregion

        #region HighlightableText

        public event EventHandler OnHighlightableTextChanged;

        public string HighlightableText

        {

            get { return (string)GetValue(HighlightableTextProperty); }

            set { SetValue(HighlightableTextProperty, value); }

        }

        // Using a DependencyProperty as the backing store for HighlightableText.  This enables animation, styling, binding, etc...

        public static readonly DependencyProperty HighlightableTextProperty =

            DependencyProperty.Register("HighlightableText", typeof(string), typeof(SearchableTextBlock), new PropertyMetadata(new PropertyChangedCallback(HighlightableTextChanged)));

        public static void HighlightableTextChanged(DependencyObject inDO, DependencyPropertyChangedEventArgs inArgs)

        {

            SearchableTextBlock stb = inDO as SearchableTextBlock;

            stb.Text = stb.HighlightableText;

            // Raise the event by using the () operator.

            if (stb.OnHighlightableTextChanged != null)

                stb.OnHighlightableTextChanged(stb, null);

        }

        #endregion

        #region HighlightForeground

        public event EventHandler OnHighlightForegroundChanged;

        public Brush HighlightForeground

        {

            get

            {

                if ((Brush)GetValue(HighlightForegroundProperty) == null)

                    SetValue(HighlightForegroundProperty, Brushes.Black);

                return (Brush)GetValue(HighlightForegroundProperty);

            }

            set { SetValue(HighlightForegroundProperty, value); }

        }

        // Using a DependencyProperty as the backing store for HighlightForeground.  This enables animation, styling, binding, etc...

        public static readonly DependencyProperty HighlightForegroundProperty =

            DependencyProperty.Register("HighlightForeground", typeof(Brush), typeof(SearchableTextBlock), new PropertyMetadata(new PropertyChangedCallback(HighlightableForegroundChanged)));

        public static void HighlightableForegroundChanged(DependencyObject inDO, DependencyPropertyChangedEventArgs inArgs)

        {

            SearchableTextBlock stb = inDO as SearchableTextBlock;

            // Raise the event by using the () operator.

            if (stb.OnHighlightForegroundChanged != null)

                stb.OnHighlightForegroundChanged(stb, null);

        }

        #endregion

        #region HighlightBackground

        public event EventHandler OnHighlightBackgroundChanged;

        public Brush HighlightBackground

        {

            get

            {

                if ((Brush)GetValue(HighlightBackgroundProperty) == null)

                    SetValue(HighlightBackgroundProperty, Brushes.Yellow);

                return (Brush)GetValue(HighlightBackgroundProperty);

            }

            set { SetValue(HighlightBackgroundProperty, value); }

        }

        // Using a DependencyProperty as the backing store for HighlightBackground.  This enables animation, styling, binding, etc...

        public static readonly DependencyProperty HighlightBackgroundProperty =

            DependencyProperty.Register("HighlightBackground", typeof(Brush), typeof(SearchableTextBlock), new PropertyMetadata(new PropertyChangedCallback(HighlightableBackgroundChanged)));

        public static void HighlightableBackgroundChanged(DependencyObject inDO, DependencyPropertyChangedEventArgs inArgs)

        {

            SearchableTextBlock stb = inDO as SearchableTextBlock;

            // Raise the event by using the () operator.

            if (stb.OnHighlightBackgroundChanged != null)

                stb.OnHighlightBackgroundChanged(stb, null);

        }

        #endregion

        #endregion

        #region Methods

        public void AddSearchString(String inString)

        {

            SearchWords.Add(inString);

            Update();

        }

        public void Update()

        {

            UpdateRegex();

        }

        public void RefreshHighlightedText()

        {

            Text = base.Text;

        }

        private void UpdateRegex()

        {

            string newRegularExpression = string.Empty;

            foreach (string s in SearchWords)

            {

                if (newRegularExpression.Length > 0)

                    newRegularExpression += "|";

                newRegularExpression += RegexWrap(s);

            }

            if (RegularExpression != newRegularExpression)

                RegularExpression = newRegularExpression;

        }

        public bool IsValidRegex(string inRegex)

        {

            if (string.IsNullOrEmpty(inRegex))

                return false;

            try

            {

                Regex.Match("", inRegex);

            }

            catch (ArgumentException)

            {

                return false;

            }

            return true;

        }

        private string RegexWrap(string inString)

        {

            // Use positive look ahead and positive look behind tags

            // so the break is before and after each word, so the

            // actual word is not removed by Regex.Split()

            return String.Format("(?={0})|(?<={0})", inString);

        }

        #endregion

    }

}

<http://joshsmithonwpf.wordpress.com/2007/08/13/animating-text-in-wpf/>

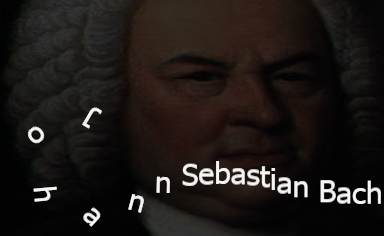
## Animating text in WPF

One of the many treasures in WPF is the animation system.  There are many examples on the Web showing how to animate various element properties such as height, color, etc., but not many people are aware that text can be animated too.  This blog post shows how to animate text in a TextBlock.

The demo application associated with this post performs a fairly useless animation on the name of my favorite composer; Johann Sebastian Bach.  In a realistic setting you probably would not need to make Bach’s name wave up and down, and rotate in circles.  However it is conceivable that a user interface could benefit from employing the basic technique shown here.  I got the basic code sample for this demo app from Chris Anderson’s [Essential Windows Presentation Foundation](http://joshsmithonwpf.wordpress.com/2007/05/27/review-of-essential-wpf/).

Here is what the demo application looks like:

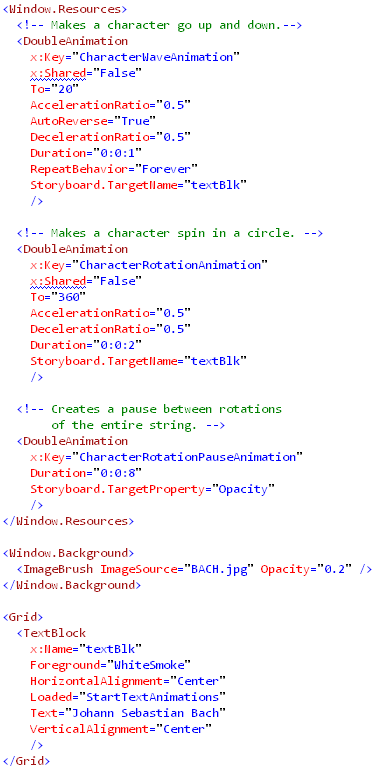






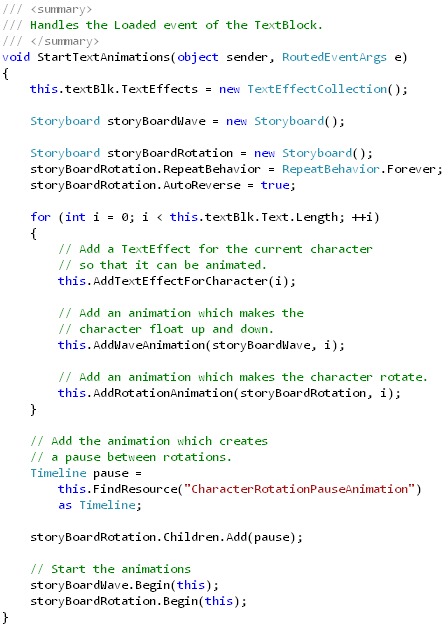
It is difficult to describe exactly what these animations look like.  I recommend that you download and run the demo app at the bottom of this post to get a much clearer understanding of what the following code snippets accomplish.

Here is the XAML file for the demo app’s main Window:

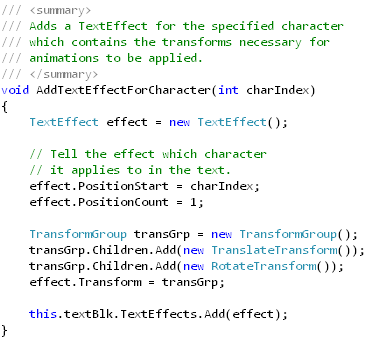


Two of the animation resources are marked as x:Shared=”False” because each character in the TextBlock has its own copy of them.  The “CharacterRotationPauseAnimation” is applied to the entire Timeline in which the various animations run. Since it is only used once there is no need to mark it as unshared.

Here is the method which handles the TextBlock’s Loaded event.  It configures the animations on each character in the display text.

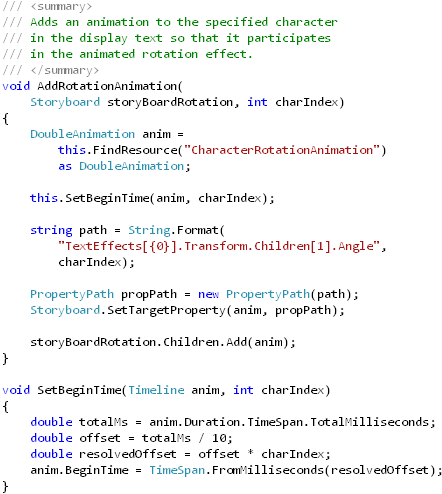


The heart of that method is the loop which configures each character in the TextBlock’s text.  It first creates a TextEffect object for a character.  The TextEffect class allows us to apply transformations to a contiguous set of characters in a text run.  The AddTextEffectForCharacter method is seen below:



The TranslateTransform and RotateTransform objects added to the TextEffect’s Transform property are the targets of the animations.  We use a TransformGroup to associate more than one transformation with the TextEffect.

Lastly let’s take a look at how one of the animations is configured.  The following code is used to make each character rotate in a circle, and is called from the StartTextAnimations method seen previously.



Download the demo project here: [Animating Text (demo project)](http://joshsmithonwpf.files.wordpress.com/2007/08/animatedtextdemozip.doc)  Be sure to change the file extension from .DOC to .ZIP and then decompress it.

[NOTE: This blog post was written while listening to [Glenn Gould play Bach’s keyboard partitas](http://www.amazon.com/Partitas-2-Anniversary-Glenn-Gould/dp/B00006FIAO/ref=sr_1_1/104-8552467-3627901?ie=UTF8&s=music&qid=1187061569&sr=1-1).]