# WebAqua.NET™ 2.0 White Paper

## Table of Contents

## Overview

WebAqua.NET™ is Intersoft's latest innovation on Web User Interface delivering breakthrough concepts and revolutionary products. WebAqua is designed with two key objectives – bringing **aqua-fresh** and **crystal-clear** experience to your Web application.

In its second generation, WebAqua.NET™ is rewritten from the scratch using managed language to target Silverlight 2.0 platform entirely. The result is high performance UI components that extensively take advantage of powerful Silverlight 2.0 architectures such as databinding, templating, and theming.

WebAqua.NET™ 2.0 can be developed using Microsoft Expression Blend 2.5 and Microsoft Visual Studio 2008 with Silverlight Tools installed.

Enter the next generation of Web UI. Our revolutionary innovations WebAqua.NET™ is ready to shine up your Web with amazing User Experiences that are never possible before.

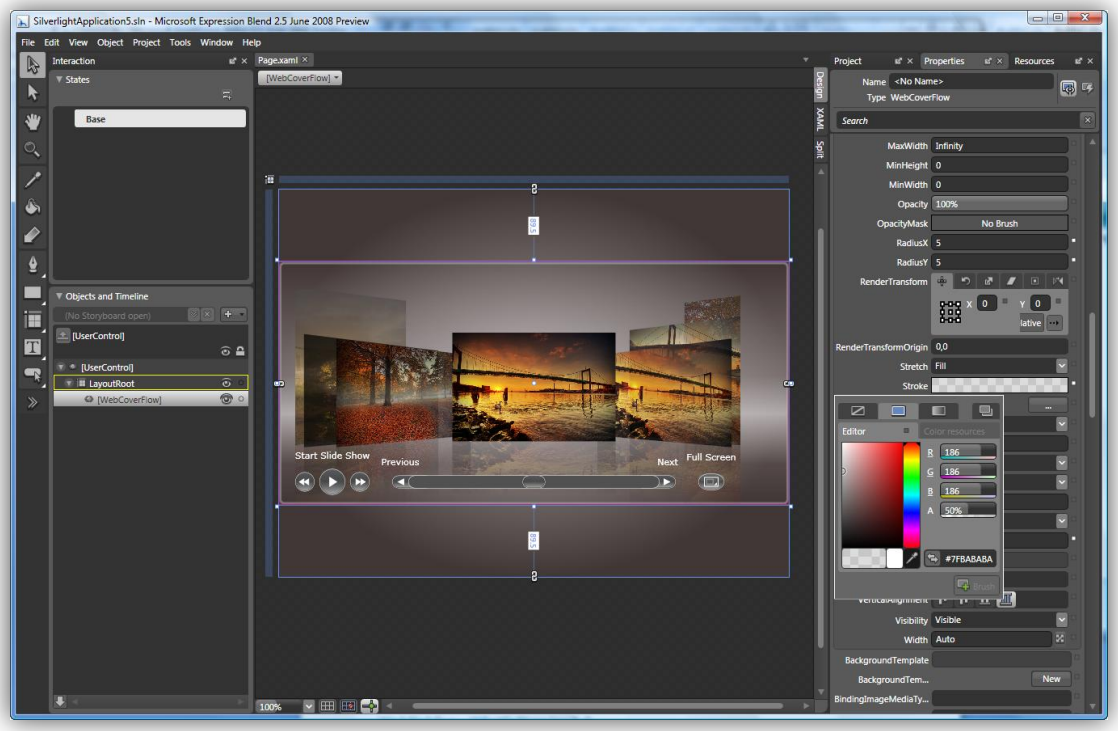
# Technology

## Silverlight 2.0

WebAqua.NET™ 2.0 is built entirely on Silverlight 2.0 technology. This means that WebAqua components are now delivered as .NET managed assemblies (.dll extension) rather than javascript files. This enables developers to build Silverlight application using WebAqua components in managed language such as C#, VB, as well as other DRM languages supported by Silverlight 2.0.

WebAqua.NET 2.0 can be developed using the following development environment:

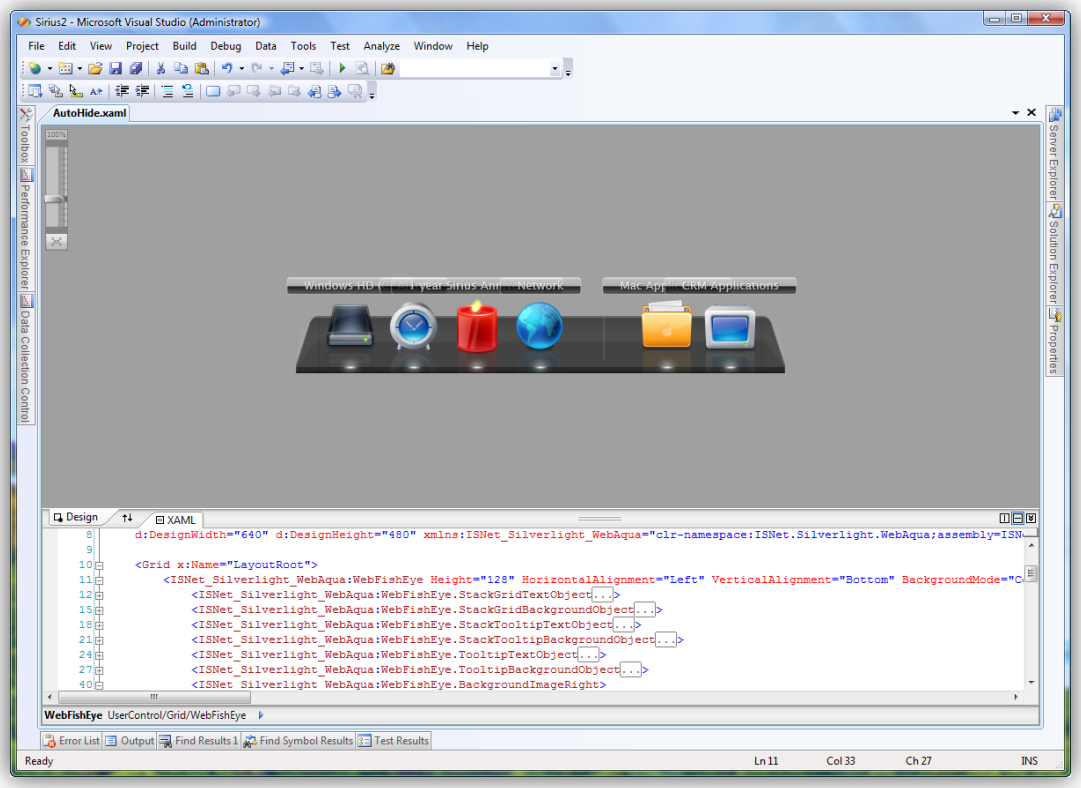
- **Microsoft Expression Blend 2.5 (June CTP or later)**




WebAqua 2.0 components can be designed directly from Microsoft Expression Blend as shown in the above image. You can author the control in much like the same fashion as in Visual Studio, such as drag the control from toolbox and drop it the designer surface. You can then customize the appearances, styles and behaviors through property sets.

Design tasks are suitable to be performed using Expression Blend, as it provides variety of tools and design-time features. For instances, you can control every aspect of the control's background such as customizing the brushes, gradient stops, colors, opacity, and more – all directly inside Expression Blend design-time surface.

- **Microsoft Visual Studio 2008 (With Silverlight Tools)**




When it comes to programmatic access and logic development, Visual Studio 2008® provides all the tools and features needed for robust Silverlight development, such as Intellisense, Auto Completion and more.

WebAqua 2.0 features dazzling design-time features that allow both Expression Blend® and Visual Studio 2008® to render the control inside the development environment such as shown in the above image. This enables developers to easily customize a property inside Visual Studio and preview the result directly without have to go back-and-forth between Expression Blend and Visual Studio.

In addition to declarative-style control definition, WebAqua components can be also constructed from managed .NET code such as C# and VB.NET. Thanks to the comprehensive API and elegant object model of WebAqua.NET components, developers have total control and flexibility over the exposed styles, templates, databinding, appearances as well as behaviors through programmable object model, API and managed events.

## ASP.NET

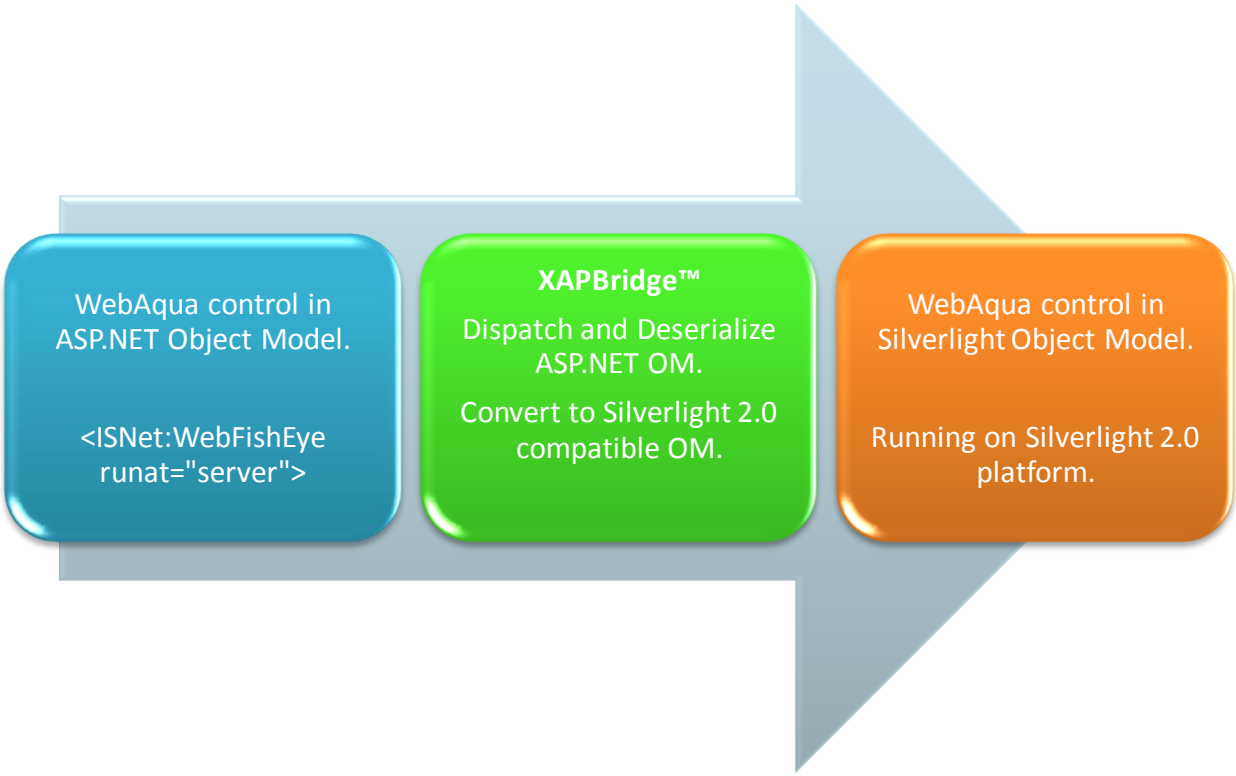
By integrating Silverlight technology seamlessly into the existing platforms and architectures such as WebUI.NET Framework and ASP.NET, developer can enjoy the same development environment and technology which they have become familiar with. This means developer does not have to learn new language such as Xaml or new tool such as Blend, and therefore significantly increasing developer's productivity while maximizing existing investments at the same time.

With WebAqua.NET™, developers can simply drag the desired component from the toolbox to the designer surface in either Visual Studio® 2005 or Visual Studio® 2008, and then set several properties to customize the behaviors and appearances. At runtime, WebAqua automatically converts the server side object into Silverlight-enabled object, including stunning animation and elegant visual effects.

WebAqua.NET™ 2.0 continues to support ASP.NET Web development as in its predecessor. Unlike the previous version, WebAqua for ASP.NET is sharing the same codebase and core runtime assemblies as those in Silverlight version of WebAqua. This translates to easier maintenance between ASP.NET and Silverlight assemblies, and providing developers a high quality and synchronized features when new features and enhancements are added to the control.

The codebase sharing between Silverlight and ASP.NET is made possible with our robust serialization and automatic mapping architecture, thanks to Intersoft's innovative XAPBridge™ technology.

The following illustration describes the concept of XAPBridge™.



# Licensing and Deployment for Silverlight 2.0

## Runtime Licensing

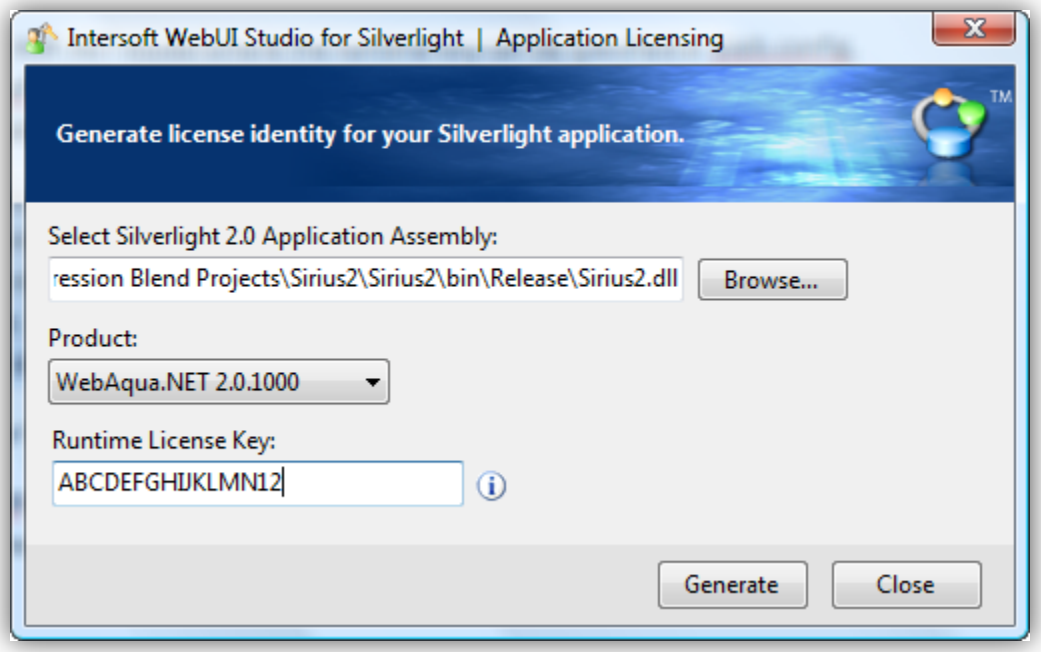
Built entirely on Silverlight 2.0 platform, WebAqua.NET™ 2.0 runtime base is purely client side control. This means, the control will not have access to ASP.NET or other server side features. As the result, the Silverlight versions of WebAqua.NET as well as upcoming Silverlight components are going to use different licensing mechanism.

Unlike the runtime licensing for ASP.NET model where the runtime key can be specified in Web.config, the runtime licensing for Intersoft's Silverlight products should be placed in special license file called "licenses.islicx" file, which should be marked as embedded resource in the Silverlight application.

The new licensing model introduced by Intersoft's Silverlight products add better security measurement and protection with advanced encryption technique, rather than specifying sensitive information as plain text. This is due to the client side nature of Silverlight, where the entire Silverlight application is delivered to end user's machine. With this added security measurement, our customers can ship their Silverlight application with confidence and peace of mind with their investments protected.

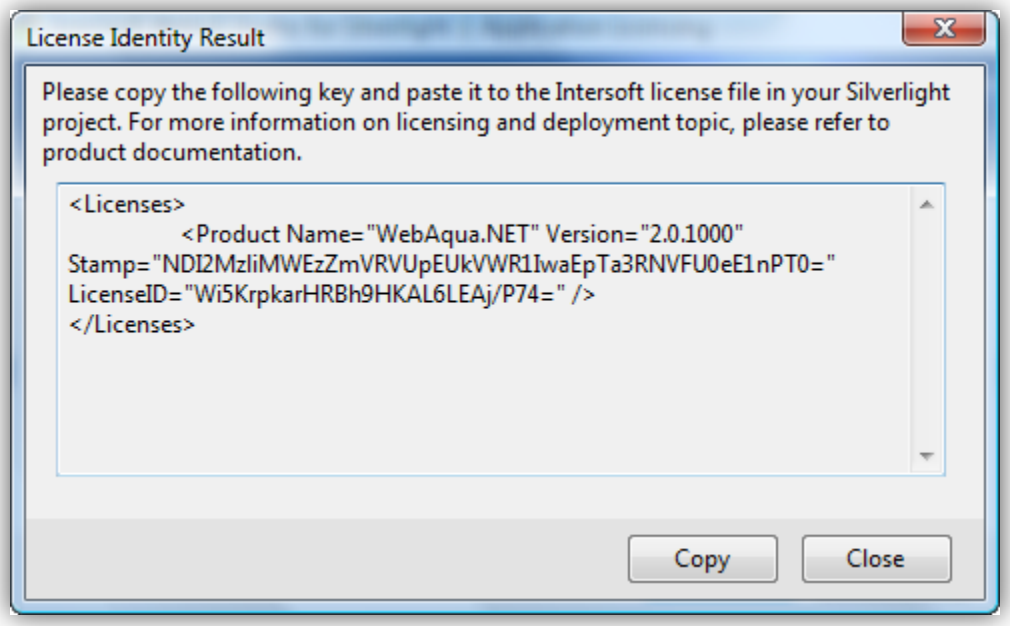
The following steps explain how to license WebAqua components for your Silverlight application:

- Launch **Intersoft WebUI Licensing for Silverlight** application from Intersoft's program group.
- The application will show immediately such as shown in the following. Then, click **Browse** button to select your main Silverlight application assembly. Next, enter the **Runtime License Key** into the provided textbox.




To obtain Runtime License Key, please logon to Developer Network 2.0 using the login credential that you received during purchase.

- Click **Generate** button.
- If all provided fields are correct, the following screen will appear as the result of the license identification for your Silverlight application.




- Copy the keys by pressing the **Copy** button.
- Add a new Xml File with file name licenses.islicx into the root of your Silverlight application project. It's recommended that you used Visual Studio 2008 to add this file to your project, as you will need to mark this file as Embedded Resource. To mark it as Embedded Resource, select the newly added file, and switch to Property Window, then set **Build Action** to Embedded Resource.
- Paste the content of the text from the License Identity Result into the **licenses.islicx** file.
- You're all set. You only need to do this licensing process once for each Silverlight application.

Note: If you change any information of the Silverlight assembly, eg, the information in AssemblyInfo.cs, you will need to regenerate the license identity and update it to licenses.islicx. It's recommended that you perform the licensing process at the time when your Silverlight application is ready to deploy.

## Deployment

The deployment process in Silverlight application is quite straightforward. When your application is built, it will include all necessary Silverlight assemblies and include it in the XAP (Silverlight Application Package) file. As long as you already perform the licensing correctly, you can simply upload the XAP to your production server and get ready to deliver exciting experience for your end users.

## Design-time Licensing

When you purchased Intersoft's products, you will normally receive two keys. One is to activate the design-time licensing, and another is for runtime. The runtime licensing is explained in above topic.

At the time of this writing, Microsoft Silverlight™ is still in beta 2 stage. Furthermore, the development environments such as Expression Blend 2.5 and Visual Studio 2008 Toolkit for Silverlight are still in CTP stage. Due to this reason, Intersoft's components for Silverlight have not enabled design-time licensing. This means you can use Intersoft's Silverlight components in design-time without a license at the time being.

However, it's highly recommended that you keep the design-time license key for Intersoft's Silverlight products, as it will be needed when the design-time licensing is activated in the future.

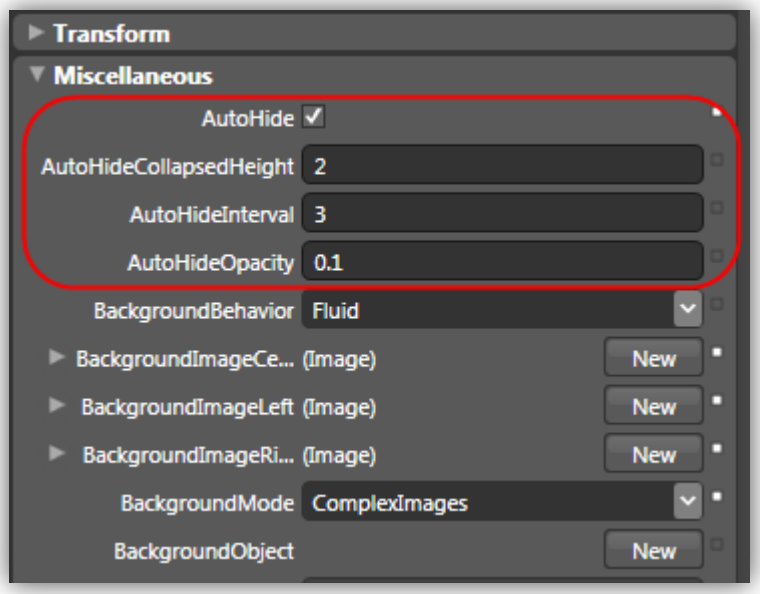
## WebFishEye – What's New in v2.0

WebAqua marks its second release with significant enhancements on WebFishEye navigation control. Some of the key new features in 2.0 are:

### Auto Hide

Auto hide is a nice feature in v2.0 that automatically hides the FishEye control after certain time intervals of being idle. When screen real estate is an important aspect in your User Interface design, consider to enable Auto Hide feature. This way, the WebFishEye navigation control will be shown only as necessary when user mouse over at the edge of the dock position.

There are several properties that related to Auto Hide feature, such as shown in the following screenshot:




- *AutoHide*. Check on this property to enable Auto Hide feature.
- *AutoHideCollapsedHeight*. The number in pixels that indicates the height of the WebFishEye after it is collapsed. In some scenarios, you might want to show some part of the WebFishEye to let your

end users aware that the control is hidden. In this case, you can set it to a number such as between 2 to 10.

- *AutoHideInterval*. The time interval in seconds when the control should be automatically hidden.
- *AutoHideOpacity*. The control's opacity when WebFishEye is hidden/collapsed. If you would like the entire control to be hidden, set it to 0. Otherwise, set it to a number between 0 and 1.

## Improved Zoom Logic

WebFishEye 2.0 is smoother and sleeker than ever before with its improved zooming logic and high performance animation.


Unlike its previous version, WebFishEye 2.0 is using new zooming logic that rewritten from ground up to take advantage of the new layout architecture in Silverlight 2.0. You can notice that the zooming process is no longer causing distortion on the other buttons when a specific area is zoomed, and is more stable and faster.

## New "Fluid" Background Behavior

WebFishEye 2.0 now provides a new **Fluid** background behavior in addition to Fixed behavior. WebFishEye 2.0 is using Fluid background behavior by default.

With Fluid mode, the Dock's background will automatically shrink and grow as the zooming process takes place. This translates to a nicer and more elegant user experience as users can interact with the navigation control with stunning visual effects.

The best of all is that Fluid background behavior supports all background modes, such as simple background, custom background as well complex-images background. This allows designers to create stunning design, and still applying Fluid behavior consistently. The following images show how the Fluid background is working.

*The dock's background in initial state, when there the control is not zoomed.*

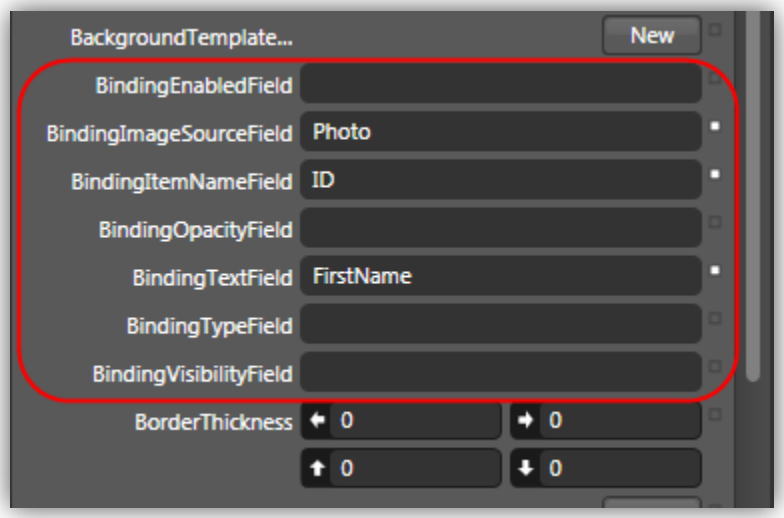

*As WebFishEye control is zoomed through mouse movement, the dock's background will follow.*

## Silverlight Databinding and Templating

Silverlight 2.0 (Beta 2 or later) introduces new databinding and templating capability. WebFishEye.NET™ 2.0 is designed to take advantage of these powerful features, enabling Silverlight developers to bind the controls to datasource in elegant fashion.

WebFishEye supports databinding to collection type of datasource such as List<T> or IEnumerable type of datasource. The datasource is assigned to *ButtonsSource* property at runtime.

When bound to the datasource properly, WebFishEye will populate the Buttons dynamically based on the given datasource. It will also automatically bind the values to the binding properties provided by WebFishEye control. In most common scenarios, you may only want to bind to *ImageSource*, *ItemName* and *Text* to sufficiently provide the required data for the Button presentation in the WebFishEye control. See following image:




Item templating is a very powerful feature in WebFishEye as it enables designers and developers to create their own containers and user interface, and embed it into the Button presentation. With item templating, you can achieve many innovative LOB scenarios.

The following is an example of data template for the Button presentation:

```
<DataTemplate x:Key="ItemTemplate">
      <Border CornerRadius="5,5,5,5" BorderThickness="2,2,2,2">
            <Border.BorderBrush>
                  <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                        <GradientStop Color="#B2FFFFFF"/>
                        <GradientStop Color="#B2FFFFFF" Offset="1"/>
                        <GradientStop Color="#FF575757" Offset="0.47"/>
                        <GradientStop Color="#FF434343" Offset="0.491"/>
                  </LinearGradientBrush>
            </Border.BorderBrush>
            <Image Source="{Binding Photo}"/>
      </Border>
</DataTemplate>
```

In the above sample, notice that data binding is defined directly in the data template. By using Silverlight's declarative *{Binding}* syntax such as used in the Image object above, the property's value will be automatically extracted from the specified datasource, which is assigned to the *ButtonsSource* property.

Item templating is commonly used together with databinding. As such, you should assign the data template to the Button's template programmatically at runtime. The best event for assigning the button template is at **ItemBind** event.

See following C# example:

```
private void WebFishEye_ItemDataBind(object sender,
      ISNet.Silverlight.WebAqua.WebFishEyeButtonEventArgs e)
{
      e.Button.ButtonTemplate = this.Resources["ItemTemplate"] as
            DataTemplate;
}
```

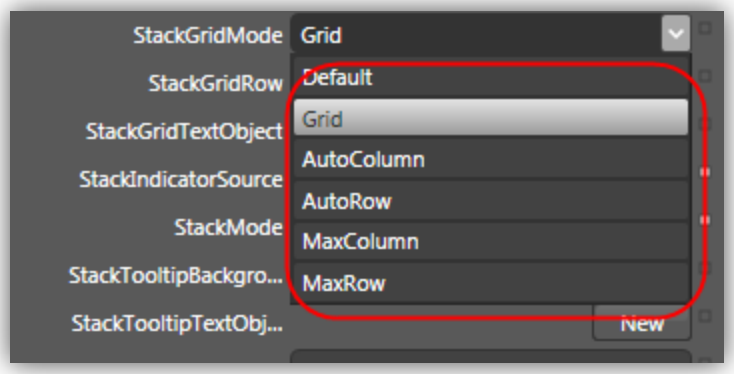
In addition to *ButtonTemplate*, WebFishEye also provides *ReflectionButtonTemplate* property which embeds the specified data template to the Button's reflection presentation.

By utilizing the new powerful architecture of binding and templating concept, you can use WebFishEye for any kind of user interface functions, in addition to its main function as navigation purpose. The following screenshot shows the result of the above sample.


## New StackGrid Modes

Stack Button is a very exceptional feature in WebFishEye control, allowing child buttons or commands to be shown in intuitive way. WebFishEye 2.0 adds significant enhancement to the Grid style of the Stack Button.

WebFishEye 2.0 introduces a new *StackGridMode* property which determines the layout behavior of the Grid style. There are several options provided by *StackGridMode* such as shown in the following screenshot:




The following is the explanation of each option of StackGridMode:

- *Grid*. This is the default option for StackGridMode. When it is set to this mode, the Grid layout will use the value specified in *StackGridColumn* and *StackGridRow*.
- *AutoColumn*. When this mode is used, the Grid layout will show the stack buttons in one row with as many columns as the number of stack buttons.
- *AutoRow*. When this mode is used, the Grid layout will show the stack buttons in one column with as many rows as the number of stack buttons.
- *MaxColumn*. When this mode is used, the Grid layout will show the stack buttons in the number of columns specified in *StackGridColumn*. The stack button will be filled from left to right direction, and automatically wrap to the next row when it has reach the maximum number of columns as specified in the *StackGridColumn* property. This mode is preferred when you have unknown length of stack buttons and is usually suitable in dynamic data scenarios.
- *MaxRow*. When this mode is used, the Grid layout will show the stack buttons from top to bottom direction, and automatically wrap to the next column when it has reach the maximum number of rows as specified in *StackGridRow* property.

## New Layout Properties

WebFishEye 2.0 comes with more layout and appearance related properties so that designers can have more granular control over the design and styles.

Several noteworthy new layout properties in WebFishEye 2.0 are:

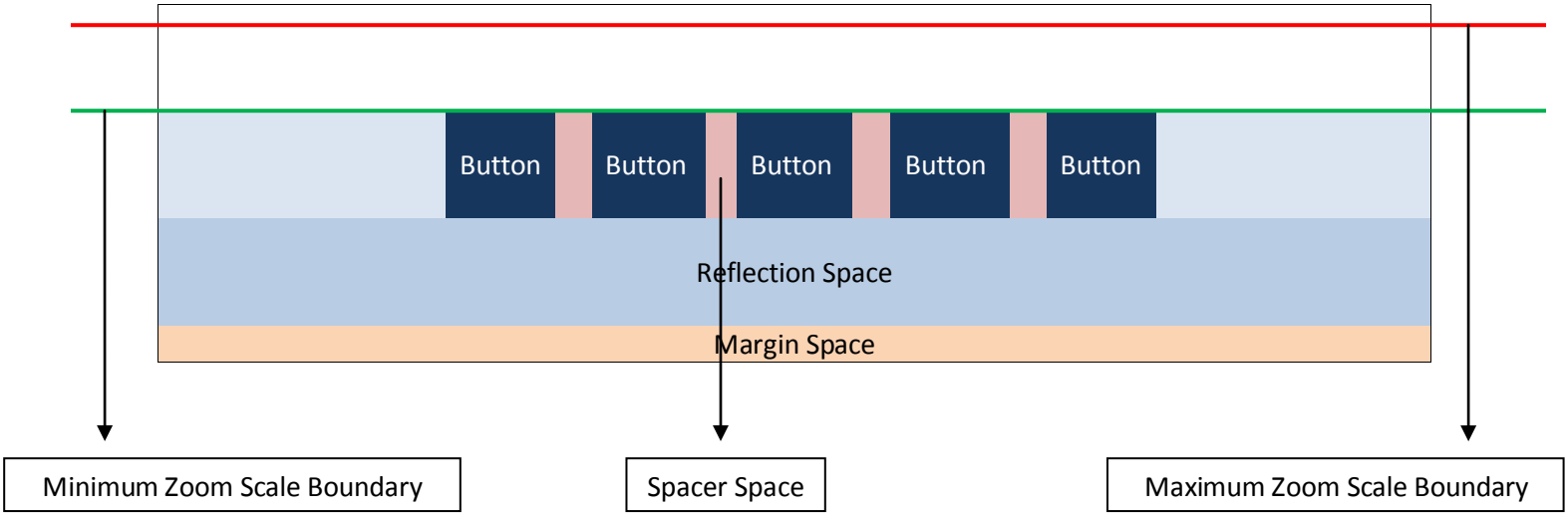
- *ButtonReflection* brush. Customizes the brush for the button's reflection effect.
- *ReflectionGap.* The gaps between image and its reflection in pixel.
- *StackGridItemSpace.* The space between stack buttons in pixel.
- *StackGridItemMargin.* The margin of each stack button in pixel.
- *StackGridMargin.* The margin of the stack grid's container in pixel.
- *TooltipMode.* Determines how the tooltip for Button should be positioned.
- *TooltipShadow.* Determines whether the shadow should be applied to the tooltip's text.

## Version 2.0 Concepts

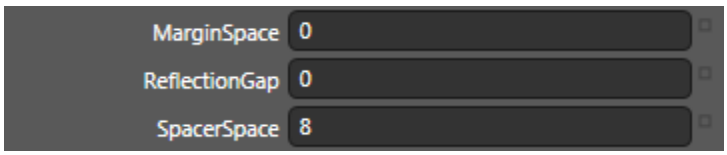
This section discusses the new WebFishEye's concept in version 2.0. Since there are new enhanced way in layouting, styling, templating  in Silverlight 2.0 we decide to revamp our internal structure of WebFishEye.NET which offers more flexibility to the users to come out with their own design.

### Concept: Layout Structure

Bellow is the illustration the structure of WebFishEye.NET




In Microsoft Expression Blend you'll see the following properties:




#### *Margin Space*

In version 1.0 Margin Space determines the distance between the buttons (with/without reflection) to the Silverlight control boundary. However in version 2.0 it works in conjunction with Reflection Space to give some space between the buttons and Silverlight control boundary.

In version 2.0 Margin space works in conjunction with Reflection Space. Therefore if you set the Reflection Space to 0 and Margin space to 32, you'll have the same effect.

This property is reserved for backward compatibility with ASP.NET control version.

#### *Spacer Space*

Spacer Space value determines the distance between one button with another button.


*Spacer Space = 8 (Default Value)*


*Spacer Space = 20*


*Spacer Space = 0*

### *Reflection Space*

Reflection Space value determines the height of reflection area when reflection effect is enabled.


*Reflection Space = 32 (Default Value)*


*Reflection Space = 64*


*Reflection Space = 0*

In version 2.0 Reflection Space will always be rendered even though Reflection Effect is disabled. Therefore if you want to entirely remove the Reflection Space you need to set it to 0.

### Concept: Background

The background container for WebFishEye is independent form the Layout structure. So you are free to use any kind of background dimension you like to use.

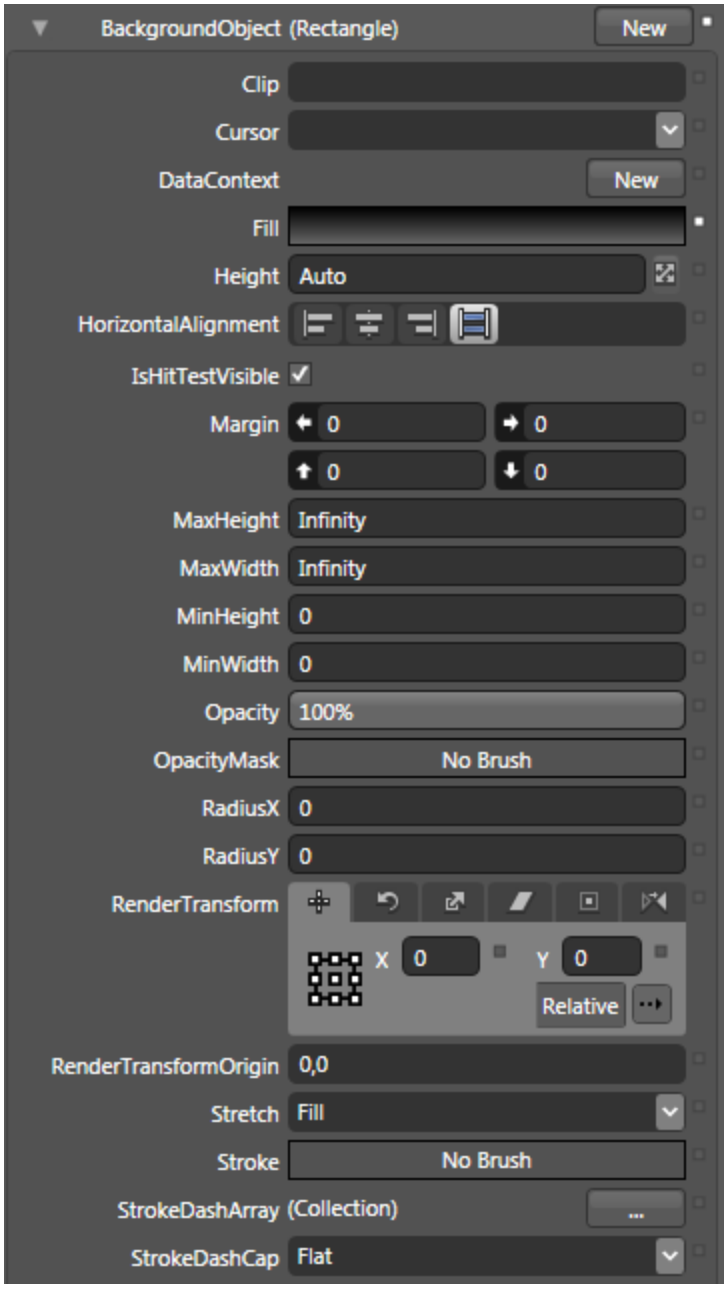
The background container is placed at the back of layout layer and its use Grid object as the layouting purpose. So you can determine the layout / position of your background object using HorizontalAlignment and / or VerticalAlignment and also Margin property.

There are three types of background settings that you can choose from:

1. Simple Mode
2. Complex Images Mode
3. Custom Mode

#### *Simple Mode*

In simple mode, the background object is a Rectangle object which you can control completely using the BackgroundObject property.




You can specify the width and/or height of the rectangle / background through height's property and width's property. If you prefer the background's dimension to follow the control's dimension you can set them to ***Auto***.This will allow the background object to fill its container.

Here's some of the sample image what you can achieve by modifying the background object.

*Using solid color brush*

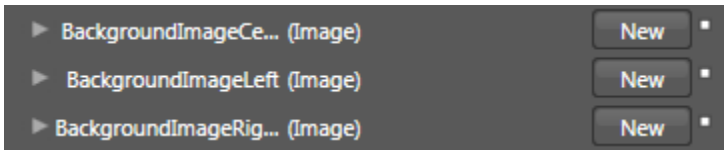
*Using linear gradient brush*

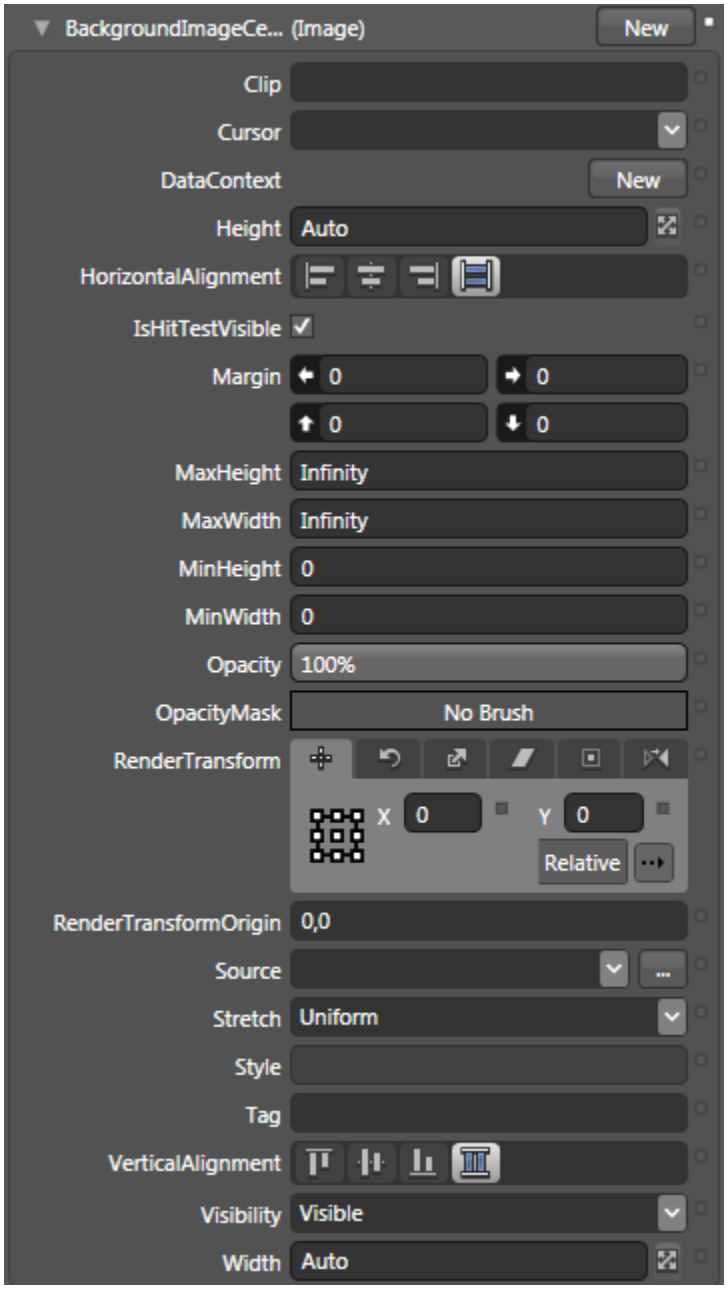
*Using radial gradient brush*

### *Complex Images Mode*

In complex image mode you can specify a set of images (three images). One for the left part of the background, one for the right part of the background and one for the center part of the background.

All the image settings can be controlled through BackgroundImageCenter / BackgroundImageLeft / BackgroundImageRight properties which give you full access to Image object.



Here's some of the sample image what you can achieve by modifying the background object.

---



*Images Sets*

*Result*

---

*Images Sets*

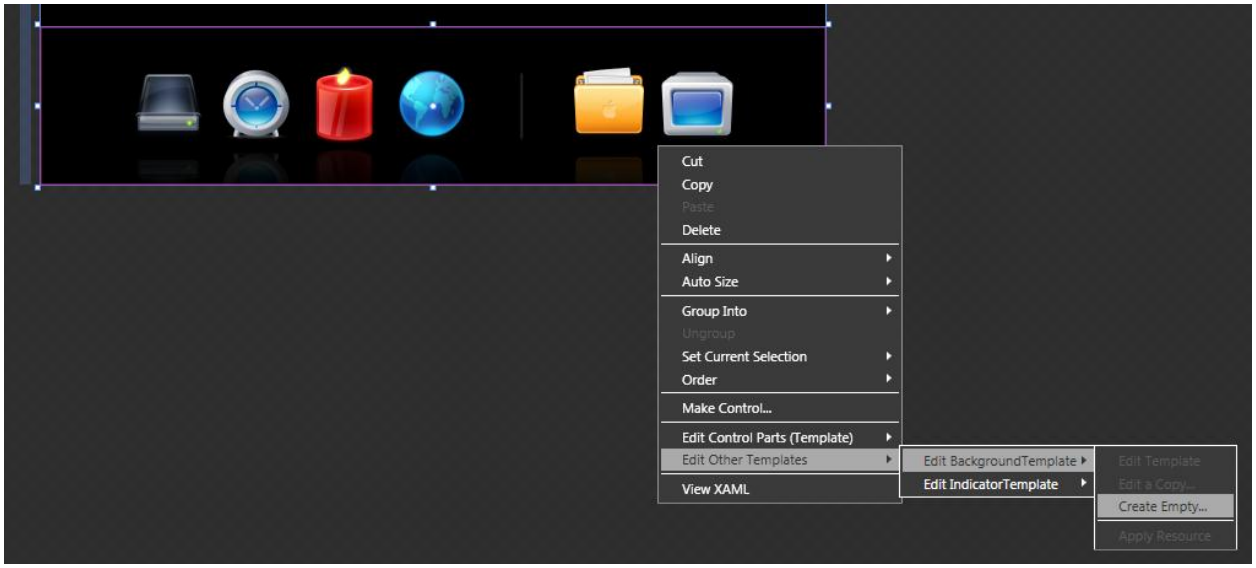
*Result*

---

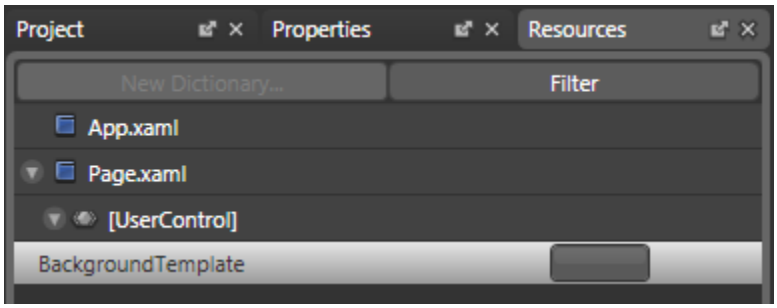
### *Custom Image Mode*

In custom image mode you can create a template object and attach it to WebFishEye's background container.

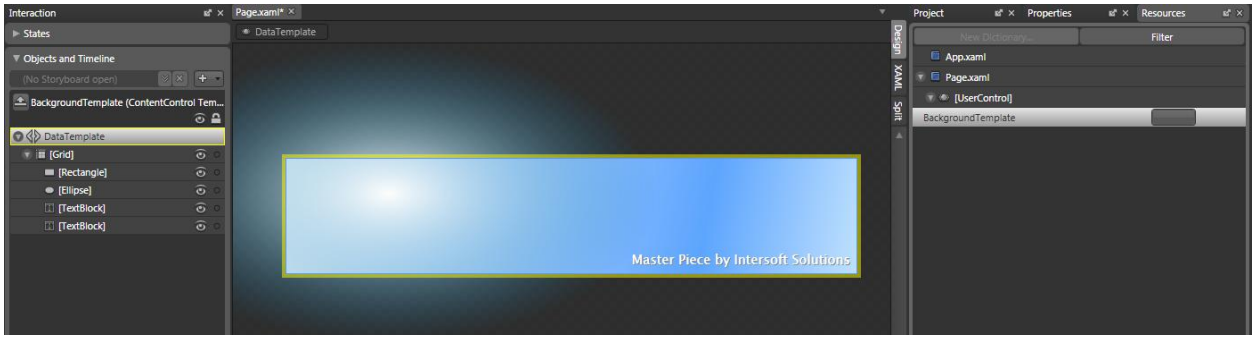
This is one way to create the template object.




You can then go to resources tab and edit your template.




Create your custom template.




When you go back you'll get something as follow.


Here's template (Xaml):

```
<UserControl.Resources>

    <DataTemplate x:Key="BackgroundTemplate">
        <Grid Height="128" Width="640">
            <Rectangle Stroke="#FF000000" StrokeThickness="0">
                <Rectangle.Fill>
                <LinearGradientBrush EndPoint="1.01400005817413,1"
                    StartPoint="-0.0299999993294477,-0.00800000037997961">
                    <GradientStop Color="#FFFFFFFF"/>
                    <GradientStop Color="#FF5DA6FF" Offset="0.728"/>
                    <GradientStop Color="#FFC6E5FF" Offset="1"/>
                </LinearGradientBrush>
                </Rectangle.Fill>
            </Rectangle>
            <Ellipse Stroke="#FF000000" StrokeThickness="0"
                Margin="-205,-138,205,-88">
                <Ellipse.Fill>
                <RadialGradientBrush>
                    <RadialGradientBrush.RelativeTransform>
                        <TransformGroup>
                            <ScaleTransform CenterX="0.5" CenterY="0.5"
                                ScaleX="0.937" ScaleY="0.937"/>
                            <SkewTransform CenterX="0.5" CenterY="0.5"/>
                            <RotateTransform CenterX="0.5" CenterY="0.5"
                                Angle="76.956"/>
                            <TranslateTransform/>
                        </TransformGroup>
                    </RadialGradientBrush.RelativeTransform>
                    <GradientStop Color="#FFFFFFFF"/>
                    <GradientStop Color="#000D81B7" Offset="1"/>
                </RadialGradientBrush>
                </Ellipse.Fill>
            </Ellipse>
            <TextBlock Text="Master Piece by Intersoft Solutions"
                VerticalAlignment="Bottom" HorizontalAlignment="Right"
                Margin="0,0,6,3" Foreground="#FF888888"/>
            <TextBlock Text="Master Piece by Intersoft Solutions"
                VerticalAlignment="Bottom" HorizontalAlignment="Right"
                Margin="0,0,7,4" Foreground="#FFFFFFFF"/>
        </Grid>
    </DataTemplate>
</UserControl.Resources>
```

### Concept: Background Behavior

There are two modes of background behaviors:

1. Fluid Mode
2. Fix Mode

#### *Fluid Mode*

In fluid mode, the background is resized proportionally when the WebFishEye is zoomed.

*Normal*

*Zoomed*

This background behavior is also applicable to background with simple mode and custom mode as long the background object has a fix width.

#### *Fix Mode*

In fix mode, the background preserves its original dimension when the WebFishEye is zoomed.

*Normal*

*Zoomed*

## Concept: Button

### *Button's Dimension*

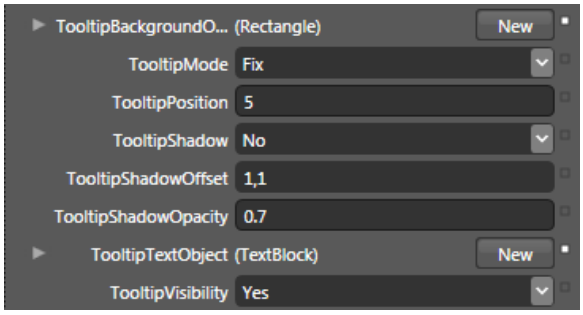
The size of the button is determined by ButtonSize and MagnifiedSize property. ButtonSize property indicates the initial size of the button (when the button isn't focused yet). On the other hand MagnifiedSize property indicates the maximum size of the button (when the button is fully focused).


*Comparison between Button Size (normal) and Magnified Size (zoomed)*

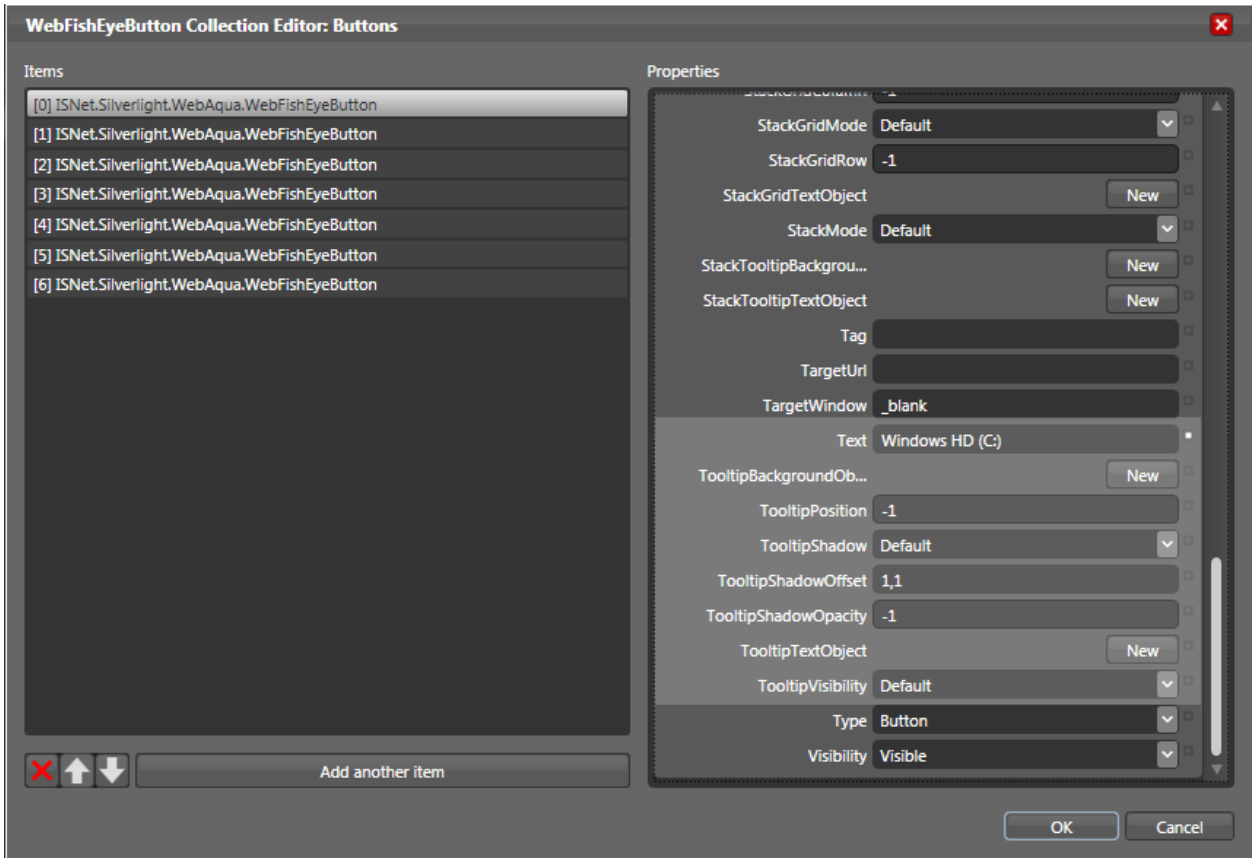
### *Button's Tooltip*

Each button has a tooltip object. This tooltip object will be shown when the button is focused. The Text value of this tooltip object is retrieved from WebFishEyeButton.Text property.

For overall tooltip's appearance it can be set from Tooltip Settings property, which can be accessible from control level and button's level.




*Tooltip Settings at Control's Level*




*Tooltip Settings at Button's Level*

To notify the control that's current button is supposed to use global settings. User need to specify the default value for specific property as follows:

Tooltip Background Object: null

Tooltip Position: -1

Tooltip Shadow: Default

Tooltip Shadow Opacity: -1

Tooltip Text Object: null

Tooltip Visibility: Default

The tooltip object consists of two elements:

1. Background Object
   The Background object is actually a Rectangle object which you can control completely from TooltipBackgroundObject property.

2. Text Object
   The Text object is actually a TextBlock object which you can control completely from TooltipTextObject property.

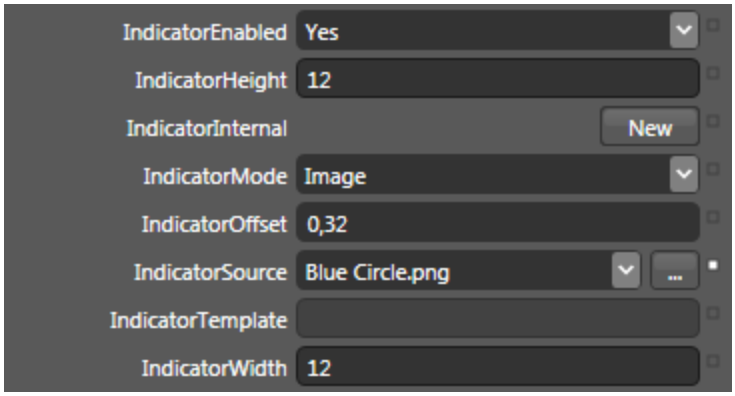

Put both settings together and you'll have something as follow:


### *Button's Indicator*

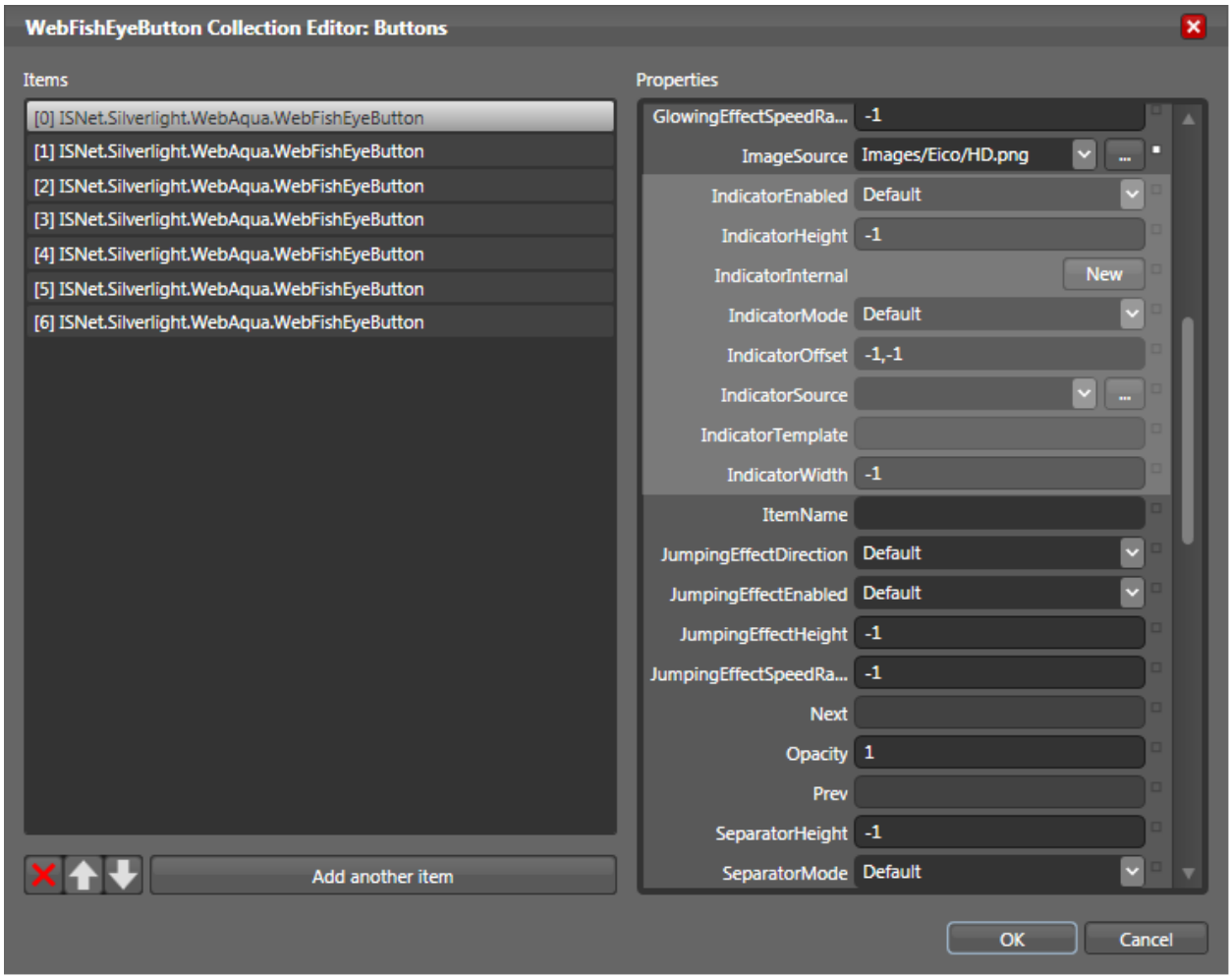
Each button also has an indicator element, which indicated the last selected button.

Important thing to be noted is that each button have an indicator that take place below the button. If reflection effect is enabled, it will take some space from the reflection area. However if you disabled the reflection effect, you might want to set some value to margin space. (E.g. margin space: 16)

This indicator appearance can be controlled from Indicator settings which accessible from control's level and button's level.




*Indicator Settings at Control's Level*




*Indicator Settings at Button's Level*

To notify the control that's current button is supposed to use global settings. User need to specify the default value for specific property as follows:

Indicator Enabled: Default

Indicator Height: -1

Indicator Mode: Default

Indicator Offset: -1, -1

Indicator Source: null

Indicator Template: null

Indicator Width: -1

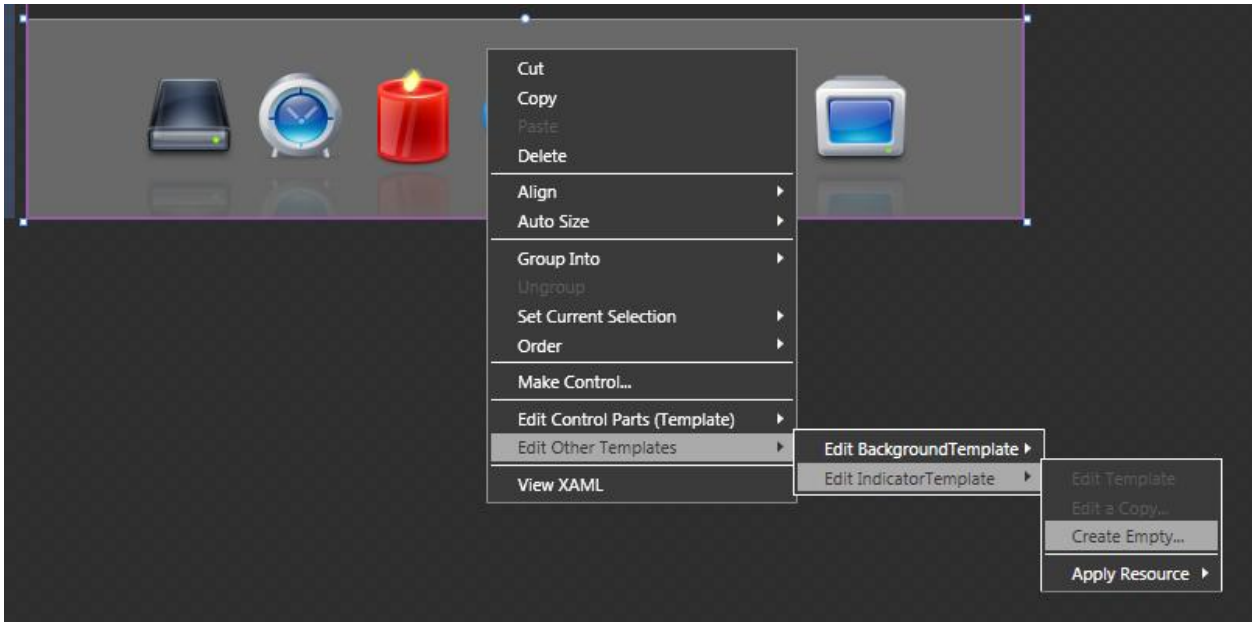
### Customizing Button's Indicator Appearance

Button's indicator object has two modes:

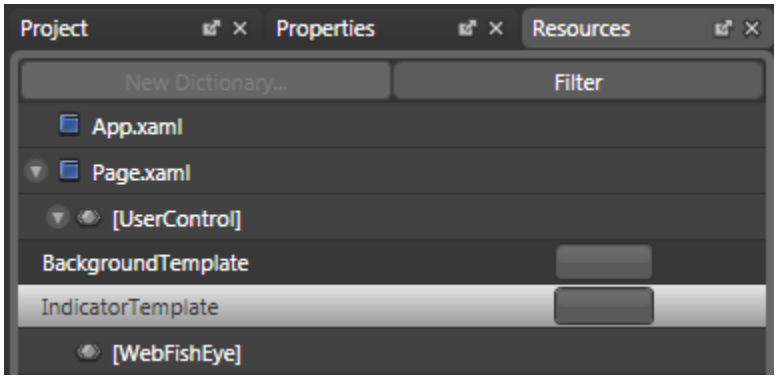
1. Image mode
2. Custom mode

In Image mode the indicator will use images specify in IndicatorSource property and the dimension is determined by IndicatorHeight and IndicatorWidth property.

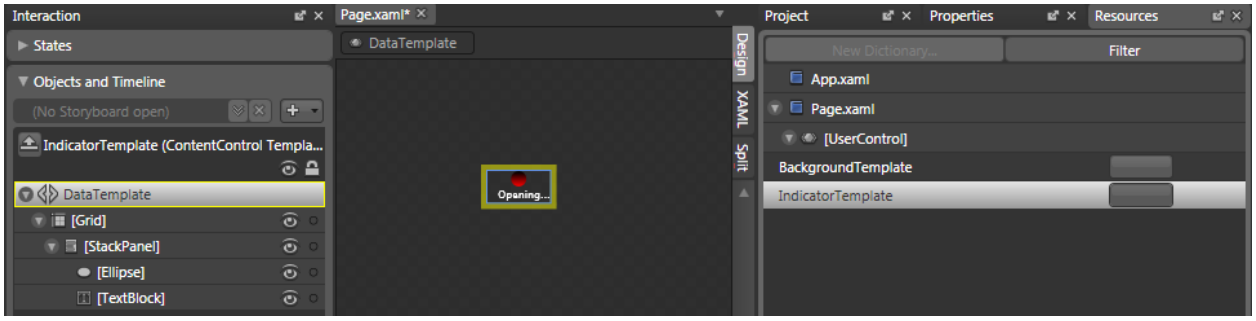
While in Custom mode the indicator will use a template object that specified in Indicator Template.




You can then go to resources tab and edit your template.




Create your custom template.




When you go back you'll get something as follow


Here's template (Xaml):

```
<UserControl.Resources>
    <DataTemplate x:Key="IndicatorTemplate">
        <Grid>
            <StackPanel Orientation="Vertical">
                <Ellipse StrokeThickness="0" Height="12" Width="12">
                    <Ellipse.Fill>
                        <LinearGradientBrush EndPoint="0.5,1"
                            StartPoint="0.5,0">
                            <GradientStop Color="#FF000000" Offset="0.0"/>
                            <GradientStop Color="#FFFF0000" Offset="1"/>
                        </LinearGradientBrush>
                    </Ellipse.Fill>
                </Ellipse>
                <TextBlock Text="Opening..."
                    Foreground="#FFFFFFFF" Margin="8,0,0,0" FontSize="8"/>
            </StackPanel>
        </Grid>
    </DataTemplate>
</UserControl.Resources>
```

In both modes the indicator is positioned at 0.0, 32.0 point, you can modify the position using Indicator Offset property. Compare the following images to have better understanding.

*Indicator Offset: 0,32*

*Indicator Offset: 0,0*

*Indicator Offset: 0,-160*

### *Stack Buttons*

Adapting the latest technology in MAC OS Leopard, WebFishEye also supports stack buttons / items which can be revealed with two different effects.

Those visual effects are:

**Stack Arc Style**

**Stack Grid Style**


The stack buttons appearance can be modified in unique ways starting with the stack button size. You can control the size of your stack button using StackButtonSize property which applicable in both StackMode.

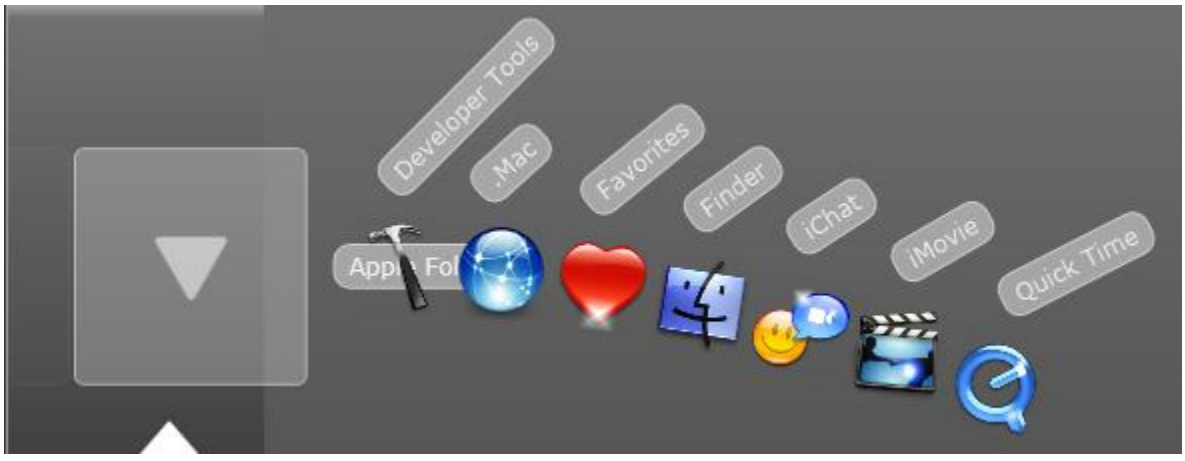
Furthermore there are settings that dependents on each type of stack mode and any of them is configurable from control level as well as in button level.

### Stack Arc Style

In arc style there are several settings that you can play around. First of all is the arc direction, you can set the direction to left or to right. Furthermore this setting is also applicable in any dock mode.


*Dock Mode: Bottom | Arc Direction: Right*


*Dock Mode: Bottom | Arc Direction: Left*



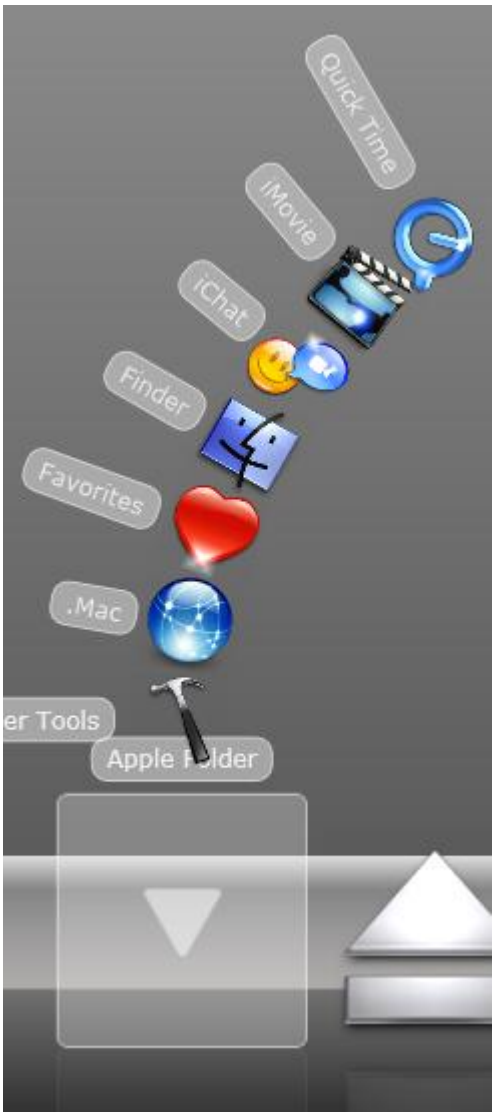

*Dock Mode: Left | Arc Direction: Right*


*Dock Mode: Left | Arc Direction: Left*

Second is the arc degree. Arc degree determines the angle of the arc style as follows:


*Arc Degree: 3.0*


*Arc Degree: 6.0*




*Arc Degree: 12.0*

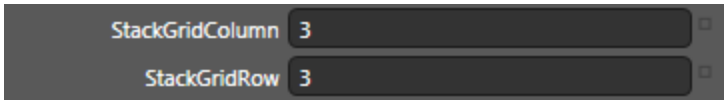

*Arc Degree: 24.0*

### Stack Grid Style

In grid style there are several settings that you can play around. First of all is stack grid mode, stack grid mode determines the layout of the grid mode. In total there are five type of stack grid mode:

1. Grid
2. Auto Row
3. Auto Column
4. Max Row
5. Max Column

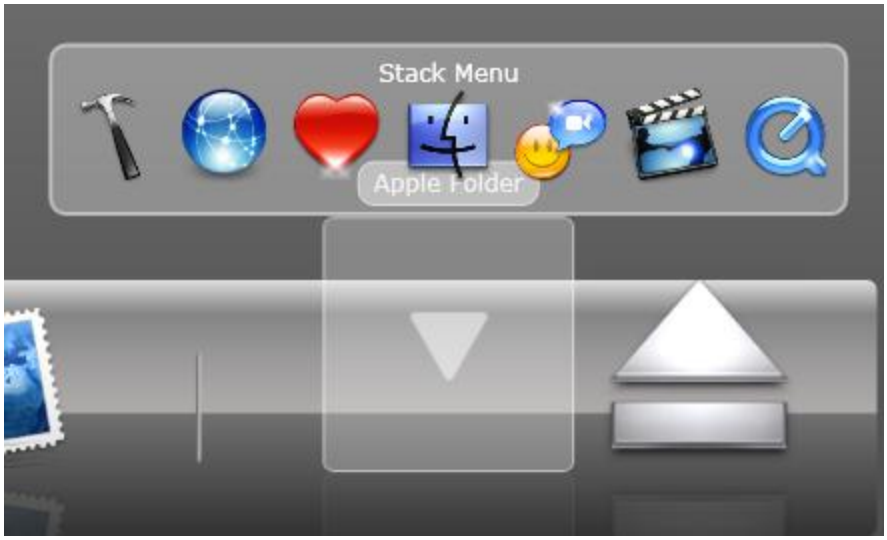
In Grid mode you'll have a grid layout for your stack buttons where you can specify the number of row and column using StackGridColumn and StackGridRow property.





In Auto Row mode, the grid layout will only have one column and the number of rows will be determined automatically based on the number of stack buttons.


In Auto Column mode, the grid layout will only have one row and the number of columns will be determined automatically based on the number of stack buttons.




In Max Row mode, the grid layout will have a maximum number of rows determined by StackGridRow and unlimited number of columns which will follow t he number of stack buttons.


In Max Column mode, the grid layout will have a maximum number of columns determined by StackGridColumn and unlimited number of rows which will follow the number of stack buttons.


Second you can also control the positioning and the layout of the grid.

To control the positioning you can use StackGridMargin property, with this property you can determine where the position of the stack grid layout relative to its original position.


*Stack Grid Margin: 150,150,0,0 (Top: 150, Left: 150)*

To control the layout you can utilizes two properties (The StackGridItemMargin, and StackGridItemSpace). StackGridItemMargin determines the margin between the outer stack layout boundaries with the inner stack grid item boundary. As shown in the image bellow.

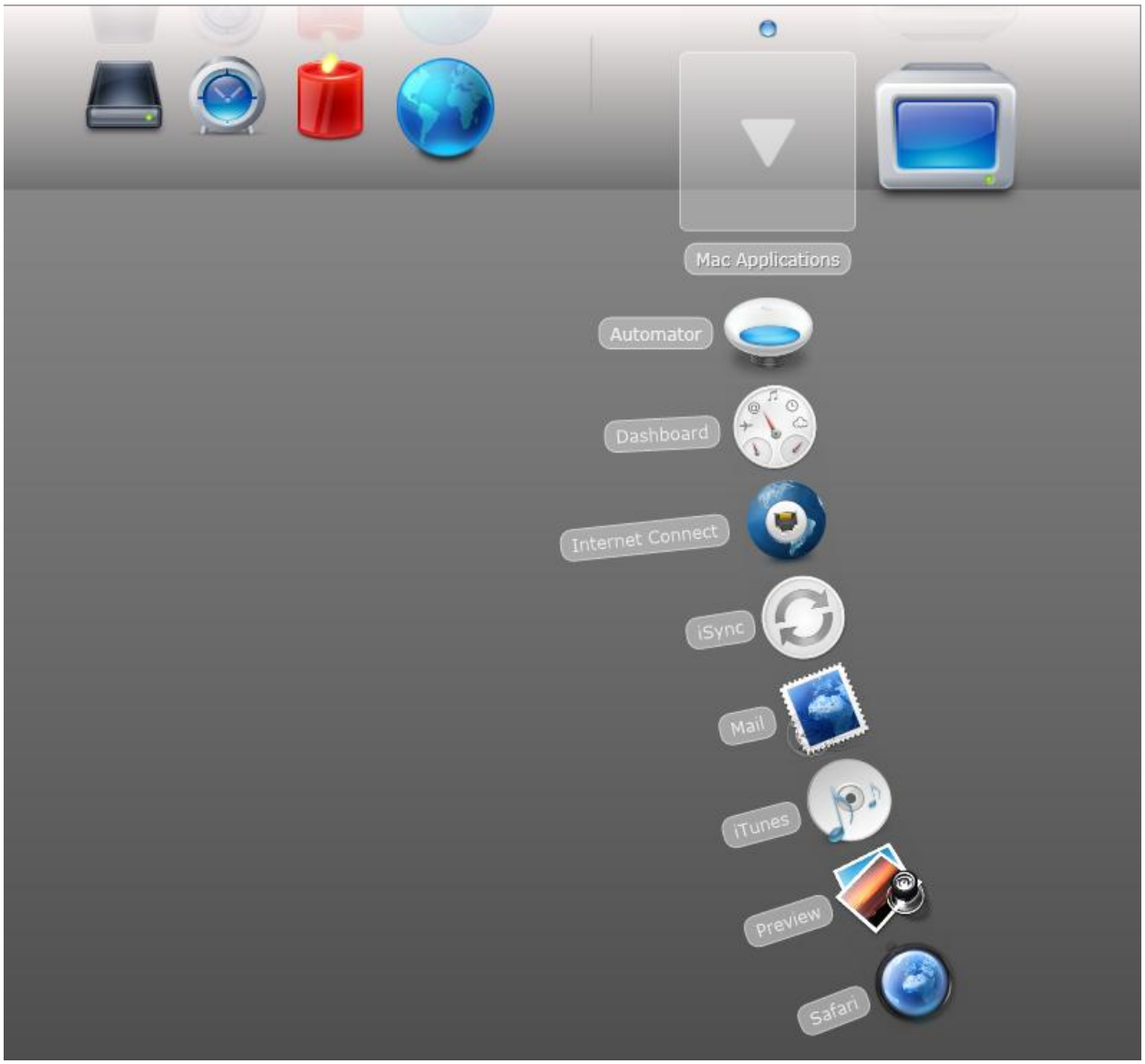

*Stack Grid Item Margin: 40, 40, 40, 40 (Top,Left,Right,Bottom: 40).*

Meanwhile the StackGridItemSpace determines the distance between items.


*Stack Grid Item Space: 40*

## Concept: Docking

WebFishEye support docking, you can dock the control to 4 different place (Top, Left, Right, Bottom). All the effect already aligned automatically when you switch between docks.



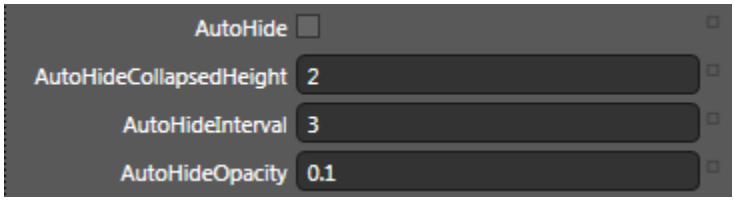

*DockMode: Top*


*DockMode: Right*

The beautiful of the Docking feature is that all WebFishEye features automatically adapt to the selected Docking position. For instance, when the Docking is set to the Left position, the zoom animation and visual effects automatically adjusts its direction toward the correct position based on the Docking state.

### Concept: Auto Hide

Auto Hide feature is introduced in version 2.0. This feature basically allows the WebFishEye to collapse to certain point after certain time.




To enable the auto hide feature you need to check the AutoHide property (set it to true). Then configure the additional properties if needed.

The AutoHideCollapsedHeight value indicates the actual height left after the auto hide is perform. This space is actually the space where you can mouse over to show the WebFishEye.

The AutoHideInterval value indicates the length to wait before the AutoHide mechanism is called.

The AutoHideOpacity value indicates the actual opacity after the auto hide is perform.

### Concept: Template Item

Template item is introduced in version 2.0. With this feature you can use your own template object as content for WebFishEyeButton.

What you need to do firstly is to create the template object.

```
<UserControl.Resources>
    <DataTemplate x:Key="ItemTemplate">
        <Border CornerRadius="5,5,5,5" BorderThickness="2,2,2,2">
            <Border.BorderBrush>
                <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                    <GradientStop Color="#B2FFFFFF"/>
                    <GradientStop Color="#B2FFFFFF" Offset="1"/>
                    <GradientStop Color="#FF575757" Offset="0.5"/>
                    <GradientStop Color="#FF434343" Offset="0.5"/>
                </LinearGradientBrush>
            </Border.BorderBrush>
            <Image Source="Images/Eico/HD.png "/>
        </Border>
    </DataTemplate>
</UserControl.Resources>
```

After that you only need to specify the ButtonTemplate property in the Button object.

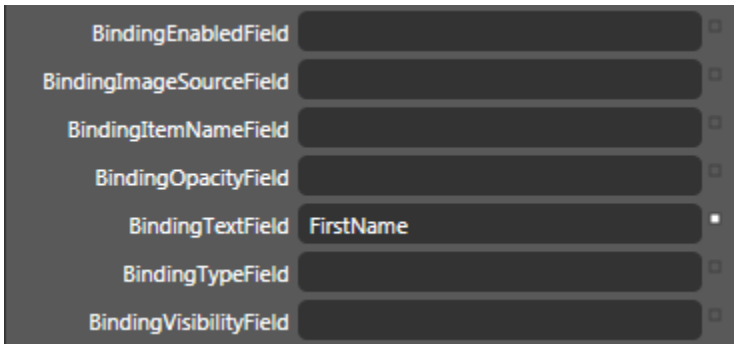
```
<ISNet_Silverlight_WebAqua:WebFishEye Height="256">
    <ISNet_Silverlight_WebAqua:WebFishEye.Buttons>
        <ISNet_Silverlight_WebAqua:WebFishEyeButton Text="Windows HD (C:)"
            ButtonTemplate="{StaticResource ItemTemplate}"/>
```

```
                .....
                .....
                .....
    </ISNet_Silverlight_WebAqua:WebFishEye.Buttons>
</ISNet_Silverlight_WebAqua:WebFishEye>
```

Then the WebFishEye will automatically use your template object for its WebFishEyeButton's content.

### Concept: Data Binding & Template Item

Instead of filling the WebFishEye's Button collection by creating a WebFishEyeButton object and filling the info to it, you can provide or reuse a generic collection of your business object to WebFishEye and set the data binding member as follows.




This will map your business object member to WebFishEyeButton member. For example as in the image you're binding your FirstName property from your business object to WebFishEyeButton Text property.

Another thing that you should pay attention is that when a data binding occurred. The item data context is pass along to the ButtonTemplate, therefore you can also retrieved the data from your business object (data context) to your button template as illustrated bellow.

Template Object

```
<UserControl.Resources>
    <DataTemplate x:Key="ItemTemplate">
        <Border CornerRadius="5,5,5,5" BorderThickness="2,2,2,2">
            <Border.BorderBrush>
                <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                    <GradientStop Color="#B2FFFFFF"/>
                    <GradientStop Color="#B2FFFFFF" Offset="1"/>
                    <GradientStop Color="#FF575757" Offset="0.5"/>
                    <GradientStop Color="#FF434343" Offset="0.5"/>
                </LinearGradientBrush>
            </Border.BorderBrush>
            <Image Source="{Binding Photo}"/>
        </Border>
    </DataTemplate>
</UserControl.Resources>
```

As you can see now we're going to bind Photo property from your business object to this template object.

Data Binding (cs Code)

```
_fishEye.ButtonsSource = BindingManager.GetEmployees();
```

To perform the data bind what you need to do is to fill the ButtonsSource property with a generic collection. BindingManager.GetEmployees() returns a generic collection of type employee.

```
    public class Employee
```

```
    {
        public int EmployeeID { get; set; }
        public string LastName { get; set; }
        public string FirstName { get; set; }
        public string Title { get; set; }
        public string TitleOfCourtesy { get; set; }
        public DateTime BirthDate { get; set; }
        public DateTime HireDate { get; set; }
        public string Address { get; set; }
        public string City { get; set; }
        public string Region { get; set; }
        public string PostalCode { get; set; }
        public string Country { get; set; }
        public string HomePhone { get; set; }
        public string Extension { get; set; }
        public string Notes { get; set; }
        public string Photo { get; set; }
    }
```

To attach a template object you can use the ItemDataBind event and attached the template object there.

```
private void WebFishEye_ItemDataBind(object sender,
ISNet.Silverlight.WebAqua.WebFishEyeButtonEventArgs e)
{
    e.Button.ButtonTemplate = this.Resources["ItemTemplate"] as DataTemplate;
}
```

Then when it's completed you'll have something like this.

## Concept: Visual Effects

### *Visual Effect: Zoom Effect*

The WebFishEye.NET's zooming effect is determined by the following attributes:

- Button Size vs Magnified Size
  This attributes affect to the scaling of focused button and its surrounding

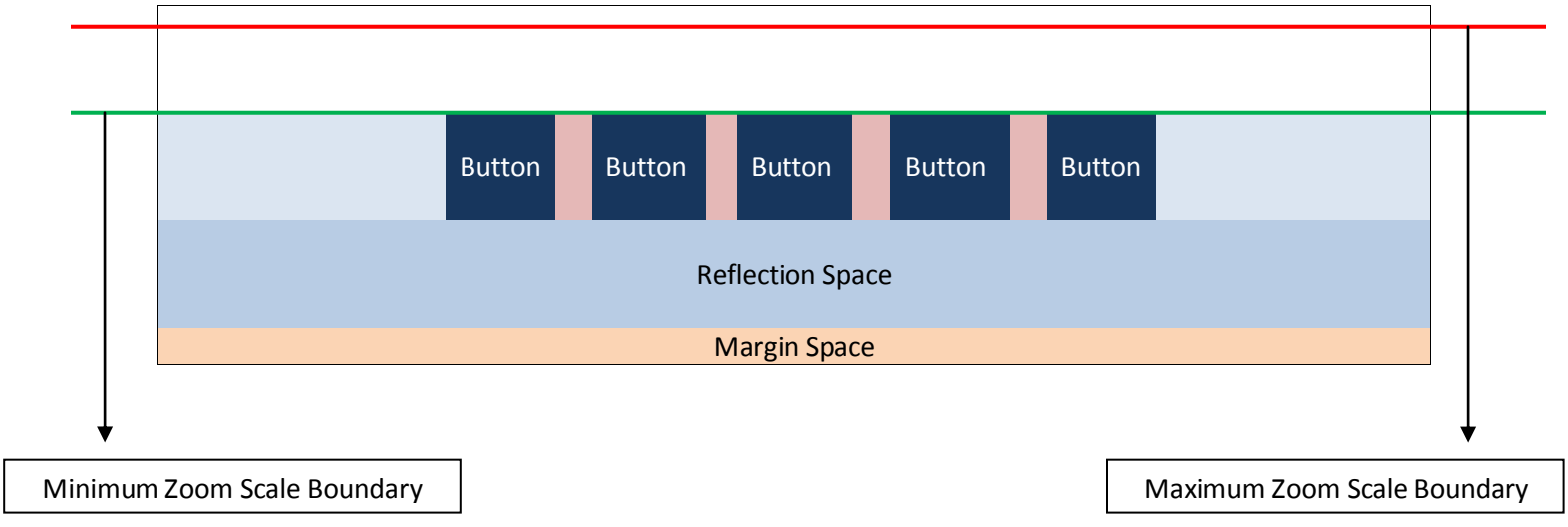

- Zoom Effect
  This attribute affect to number of buttons that will be affected by the zoom mechanism.


*Zoom Effect: 2*


*Zoom Effect: 4*


*Zoom Effect: 8*

- Zoom Boundary Mode




There are three types of zoom boundary mode.

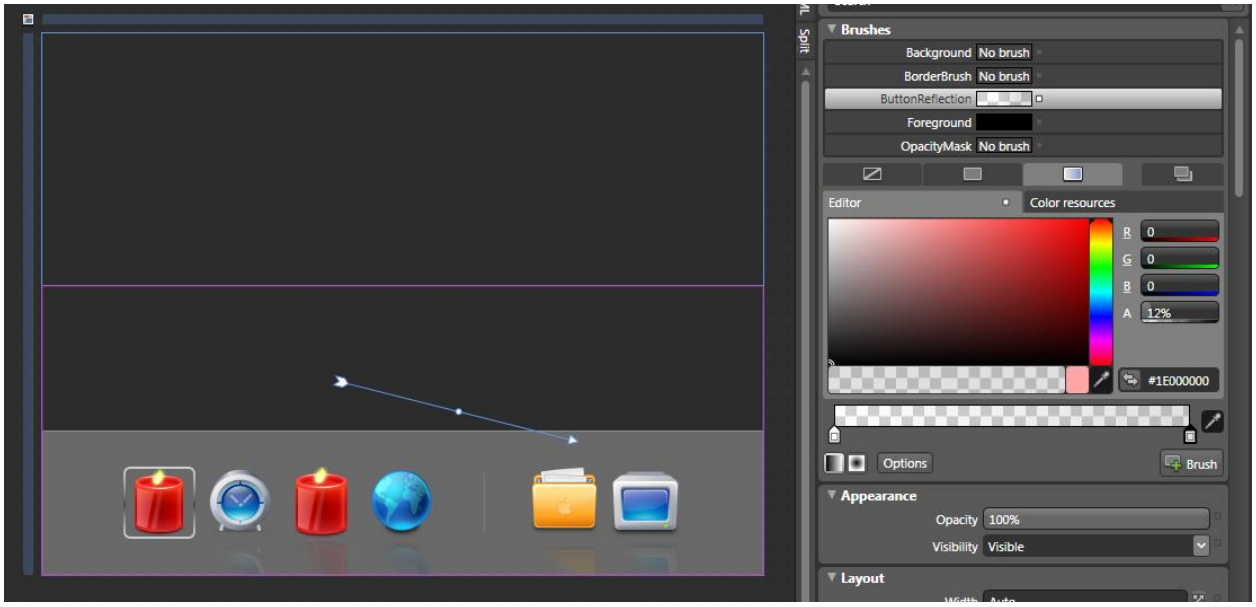
- Minimum Zoom Scale

When the zoom boundary mode is set to minimum zoom scale, the WebFishEye will start zooming when the cursor cross the minimum zoom scale boundary.

- Maximum Zoom Scale
  When the zoom boundary mode is set to maximum zoom scale, the WebFishEye will start zooming when the cursor cross the maximum zoom scale boundary.
- Custom Zoom Scale
  When the zoom boundary mode is set to custom zoom scale, the WebFishEye will use the value in custom zoom boundary property as indicator when to start zooming.

### *Visual Effect: Reflection*

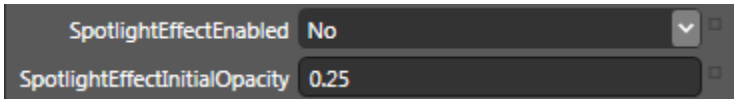
Reflection effect is one of the build-in effects embedded in WebFishEye.NET, which can be control from the following property.

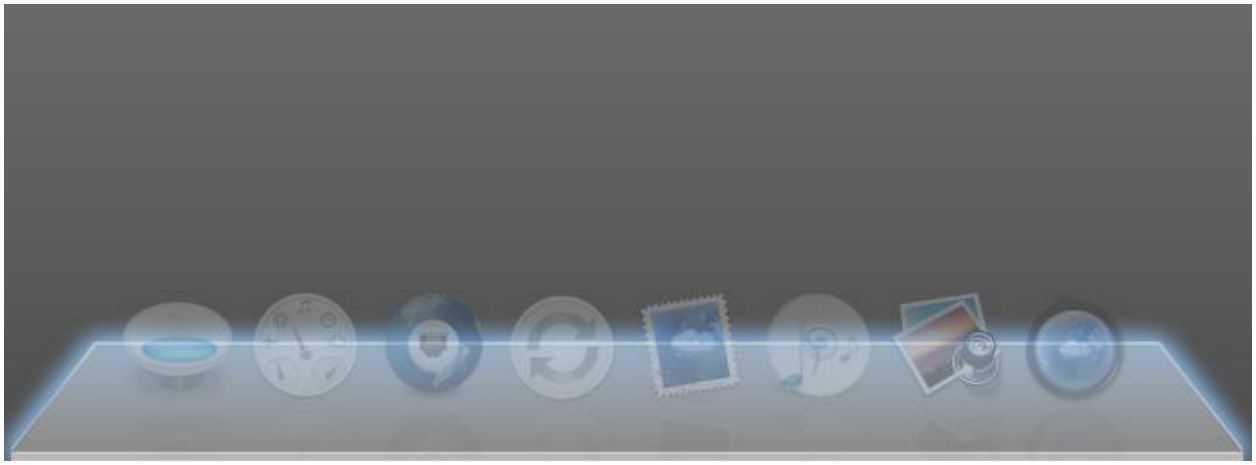

The Button Reflection property is a Linear Gradient Brush type so you can control the direction of the gradient, and the gradient stops using the designer as shown in the picture above.

### *Visual Effect: Spotlight*

Spotlight is one of the build-in effects embedded in WebFishEye.NET. When this effect is enabled, all the buttons will initially be more transparent (blur), but as soon as the button is focused it will be more solid (clear).




Initial Opacity indicates opacity rating when the buttons is out of focus.



*Out of focus (initial opacity: 25)*


*In focus*

### *Visual Effect: Jumping*


Jumping effect is one of build-in effects embedded in WebFishEye.NET that can be applied when a button is clicked. This effect can be combined with other visual effect, to produce a unique effect of your own.

Note that this effect can be applied globally to all WebFishEye's button from control's level, or individually at each button level.

Things that you can control from this effect are:

- Speed Ratio
- Jump Height
- Jump Direction
  There are 5 types of jump direction, Default, Left, Right, Top, Bottom. Top, Left, Right, Bottom is pretty straight forward, the button will jump to the specified direction. However in Default mode, the button will jump from its base.

  So if the dock is set to Top, the jump direction will be set to Bottom, and if the dock is set to Left, the jump direction will be set to Right and so on.

### *Visual Effect: Flipping*


Flipping effect is one of build-in effects embedded in WebFishEYe.NET that can be applied when a button is clicked. This effect can be combined with other visual effect, to produce a unique effect of your own.

Note that this effect can be applied globally to all WebFishEye's button from control's level, or individually at each button level.

Things that you can control from this effect are:

- Speed Ratio
- Flip Mode
  There are two flip mode, you can only apply one of them. Flip horizontally or flip vertically.

### *Visual Effect: Glowing*


Glowing effect is one of build-in effects embedded in WebFishEYe.NET that can be applied when a button is clicked. This effect can be combined with other visual effect, to produce a unique effect of your own.

Note that this effect can be applied globally to all WebFishEye's button from control's level, or individually at each button level.

Things that you can control from this effect are:

- Speed Ratio
- Radius
  Radius is the circle radius of the glow effect. The larger the radius the larger the effect will be. However it is recommend not to use a value larger than 1.5.
- Inner Glow
  Inner glow is the inner color of the glow effect.
- Outer Glow
  Outer glow is the outer color of the glow effect.

### *Visual Effect: Custom*

If the build-in effect is not enough for you, you can create your own effect that can be applied when a button is clicked. This effect can be combined with other visual effect, to produce a unique effect of your own.

Note that this effect can be applied globally to all WebFishEye's button from control's level, or individually at each button level.

What you need to do is to create the storyboard

```
<UserControl.Resources>
    <Storyboard x:Name="CustomEffect3">
        <DoubleAnimationUsingKeyFrames BeginTime="00:00:00"
            Storyboard.TargetName="rectangle"
            Storyboard.TargetProperty="(UIElement.RenderTransform).
                (TransformGroup.Children)[2].(RotateTransform.Angle)">
             <SplineDoubleKeyFrame KeyTime="00:00:00.3000000" Value="45">
                 <SplineDoubleKeyFrame.KeySpline>
                     <KeySpline ControlPoint1="1,0" ControlPoint2="1,1"/>
                 </SplineDoubleKeyFrame.KeySpline>
             </SplineDoubleKeyFrame>
             <SplineDoubleKeyFrame KeyTime="00:00:01" Value="-360">
             </SplineDoubleKeyFrame>
        </DoubleAnimationUsingKeyFrames>
    </Storyboard>
</UserControl.Resources>
```

Then assign the custom effect name along with enabling the custom effect.

```
<ISNet_Silverlight_WebAqua:WebFishEye
    CustomEffectEnabled="Yes" CustomEffectName="CustomEffect1">
```

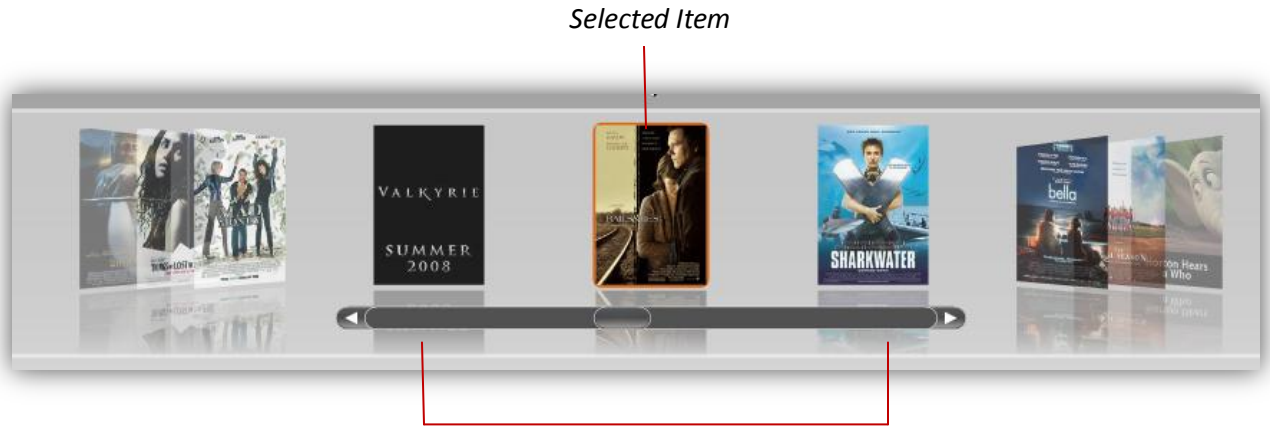
## WebCoverFlow – What's new in v2.0

WebCoverFlow includes many new properties for styles and appearance customization that are not available in its previous version. Some of the highlighted new features are:

### Multiple Focus Mode

WebCoverFlow adds a new innovative feature in its second major release, which is called Multiple Focus mode. In its original concept, WebCoverFlow consists of 1 focal image – that is the image at the center – and the rest of items will be located in the preview areas (the left and right side of the focal image).

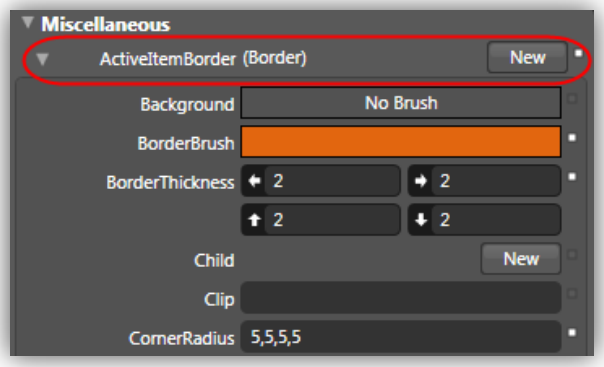
WebCoverFlow 2.0 now allows you to set more than one focus image. With this new feature, WebCoverFlow can also be used as selection and navigation user interface function, in addition to the media player function.




*WebCoverFlow is configured to use 3 focused items.*

The Multiple Focus mode can be easily configured by setting the *TotalFocusItem* property, which is 1 by default. For instance, the above illustration has the *TotalFocusItem* property set to 3.

WebCoverFlow 2.0 also introduces a new property called *ActiveItemBorder*. This property is especially useful when used in Multiple Focus mode, so that users can tell the current selection through different border style of the cover item.



## Improved Flow Logic

Flowing is the primary action and core animation engine of WebCoverFlow control. In version 2.0, the Flow logic has been totally revamped and built from the ground up to use Silverlight 2.0's new layout architecture. The result is obvious – more superior performance, sleeker motion, and more elegant than ever before.


## Comprehensive API

As WebCoverFlow 2.0 is rewritten using managed language targeting Silverlight 2.0 runtime, WebCoverFlow has exposed comprehensive API to allow developers to perform programmatic access and control over WebCoverFlow functions. For instances, you can get the selected coverflow item by using *GetSelectedItem* method, play selected video using *PlayVideo* method, or stop it using *StopVideo* method, and many more.

The API for direct element access have also been made available, such as *GetCoverFlowElement*, *GetControlPanelElement*, *GetLayoutElement*, and much more. This ensures great scalability and high extensibility for Silverlight developers to take granular control over WebCoverFlow behaviors and functions.

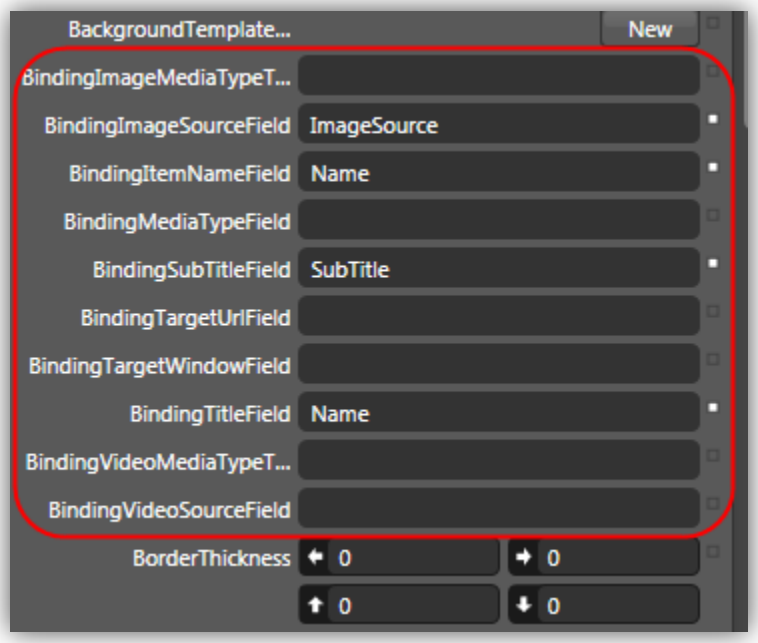
For more information about WebCoverFlow object model and API for Silverlight 2.0, please refer to *WebAqua Reference for Silverlight* node in the product documentation.

## Silverlight Databinding and Templating

Silverlight 2.0 (Beta 2 or later) introduces new databinding and templating capability. WebCoverFlow.NET™ 2.0 is designed to take advantage of these powerful features, enabling Silverlight developers to bind the controls to datasource in elegant fashion.

WebCoverFlow supports databinding to collection type of datasource such as List<T> or IEnumerable type of datasource. The datasource is assigned to *ItemsSource* property at runtime.

When bound to the datasource properly, WebCoverFlow will populate the cover items dynamically based on the given datasource. It will also automatically bind the values to the binding properties provided by WebCoverFlow control. In most common scenarios, you may only want to bind to *ImageSource*, *ItemName* and *Title* to sufficiently provide the required data for the Coverflow presentation in the WebCoverFlow control. See following image:




Item templating is a very powerful feature in WebCoverFlow as it enables designers and developers to create their own containers and user interface, and embed it into the Coverflow presentation. With item templating, you can achieve many innovative LOB scenarios that can't be done using built-in functions.

The following is an example of data template for the Coverflow presentation:

```
<DataTemplate x:Key="ItemTemplate">
      <Border BorderThickness="3,3,3,3" CornerRadius="5,5,5,5"
            HorizontalAlignment="Stretch" VerticalAlignment="Stretch">

            <Border.Background>
                  <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                        <GradientStop Color="#FF080808"/>
                        <GradientStop Color="#FFFFFFFF" Offset="1"/>
                  </LinearGradientBrush>
```

```
            </Border.Background>
      </Border>
      <Image Source="{Binding Photo}">
      <StackPanel HorizontalAlignment="Right" VerticalAlignment="Top"
            Width="200" Height="120" Margin="0,55,10,0">
            <TextBlock FontSize="12" Text="{Binding BirthDate}" />
            <TextBlock FontSize="12" Text="{Binding Address}" />
            <TextBlock FontSize="12" Text="{Binding HomePhone}" />
            <TextBlock FontSize="12" Text="{Binding City}" />
            <TextBlock FontSize="12" Text="{Binding Country}" />
      </StackPanel>
</DataTemplate>
```

In the above sample, notice that data binding is defined directly in the data template. By using Silverlight's declarative *{Binding}* syntax such as used in the Image and TextBlock objects above, the property's value will be automatically extracted from the specified datasource, which is assigned to the *ItemsSource* property.

Item templating is commonly used together with databinding. As such, you should assign the data template to the Item's template programmatically at runtime. The best event for assigning the item template is at **ItemBind** event.

See following C# example:

```
private void WebCoverFlow_ItemDataBind(object sender,
                  ISNet.Silverlight.WebAqua.WebCoverFlowItemEventArgs e)
{
      e.Item.ItemTemplate = this.Resources["ItemTemplate"] as DataTemplate;
      e.Item.ReflectionTemplate = this.Resources["ReflectionTemplate"] as
                                    DataTemplate;
}
```

In addition to *ItemTemplate*, WebCoverFlow also provides *ReflectionTemplate* property which embeds the specified data template to the cover's reflection presentation.

By utilizing the new powerful architecture of binding and templating concept, you can use WebCoverFlow for any kind of user interface functions, in addition to its main function as media player purpose. The following screenshot shows the result of the above sample.

## New Layout Properties

WebCoverFlow 2.0 comes with more layout and appearance related properties so that designers can have more granular control over the design and styles.

Several noteworthy new layout properties in WebCoverFlow 2.0 are:

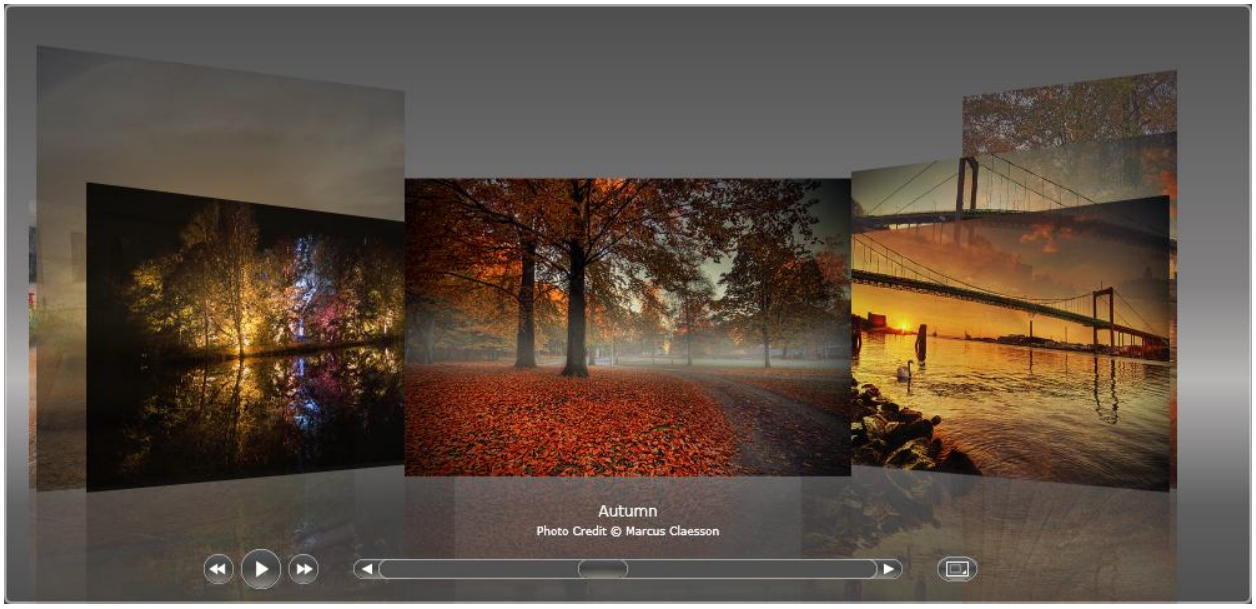
- *HoverSpreadColorTheme* brush. Customizes the brush for the button's color theme in hover state.
- *ItemReflection* brush. Customizes the brush for the cover's reflection effect.
- *SpreadColorTheme* brush. Customizes the brush for the button's color theme in normal state.
- *CoverFlowPadding.* The space between coverflow inner container and main frame container in pixel.
- *ReflectionGap.* The space between cover and its reflection in pixel.
- *SubTitleShadow.* Determines whether shadow effect should be applied to sub title's text.
- *SubTitleShadowOffset.* The location of the subtitle's shadow relative to the subtitle's text.
- *SubTitleShadowOpacity*. The opacity level of the subtitle's shadow, a number between 0 and 1.
- *TitleShadow*. Determines whether shadow effect should be applied to title's text.
- *TitleShadowOffset*. The location of the title's shadow relative to the title's text.
- *TitleShadowOpacity*. The opacity level of the title's shadow, a number between 0 and 1.

## Version 2.0 Concepts

This section discusses the new WebCoverFlow's concept in version 2.0. Since there are new enhanced way in layouting, styling, templating  in Silverlight 2.0 we decide to revamp our internal structure of WebCoverFlow's.NET which offers more flexibility to the users to come out with their own design.

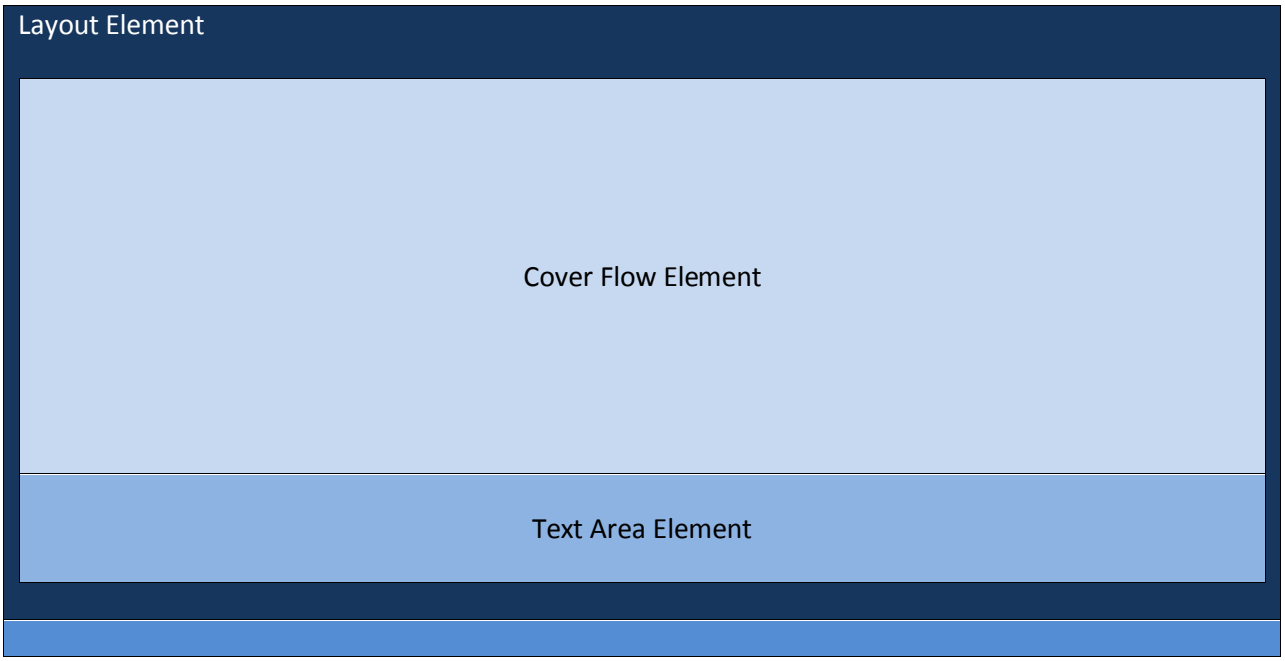
## Concept: Layout Structure

### *Layout Structure: Overview*




The container can be divided into two sections:

1. Layout Element
    a. CoverFlow Element
    b. TextArea Element
2. Control Panel Element



Control Panel Element

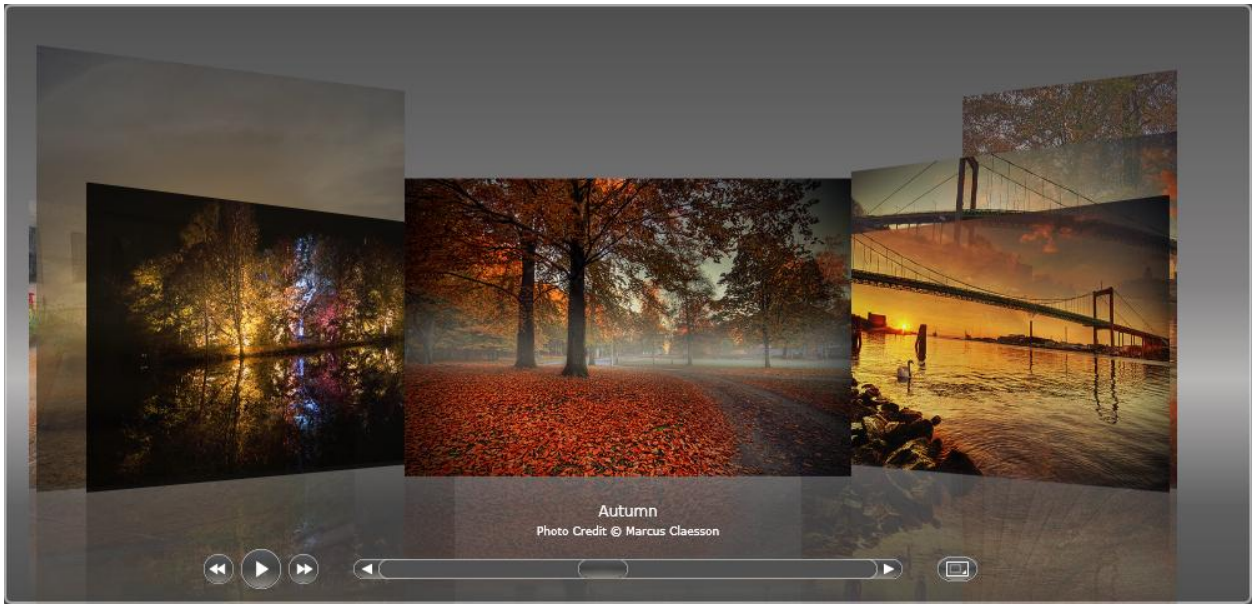
### Control Panel Area

This area contains a control panel (slide show controller) that can be used to navigate between CoverFlow items.

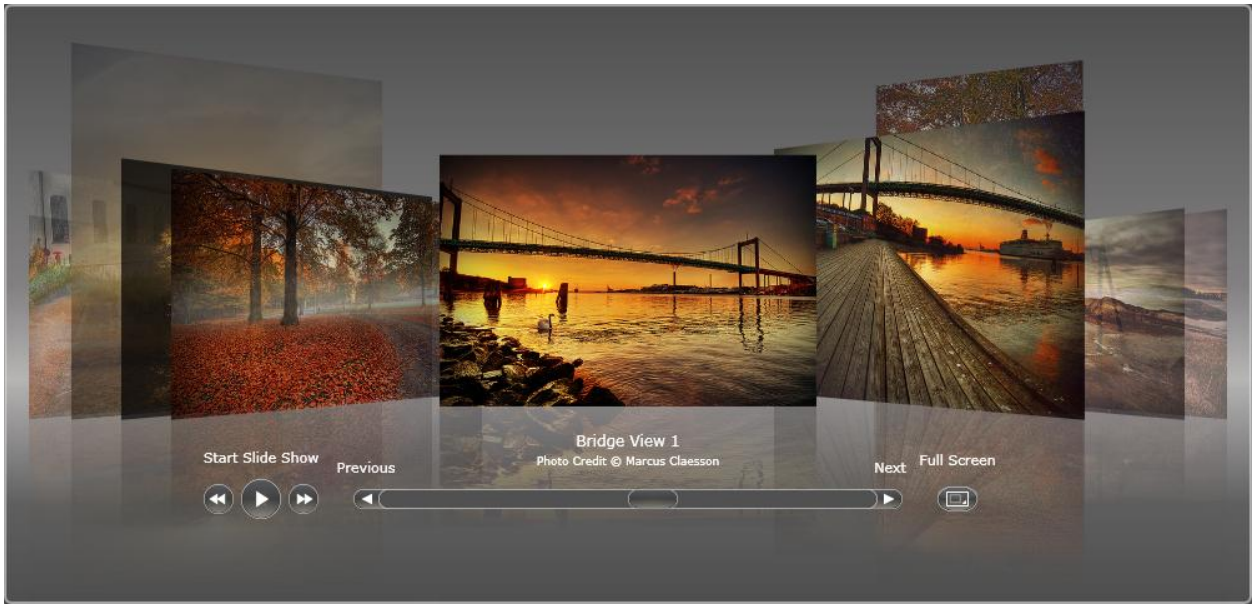
Control Panel

*Panel Position*

The panel position value determines the distance between the control panel and the Silverlight boundary.



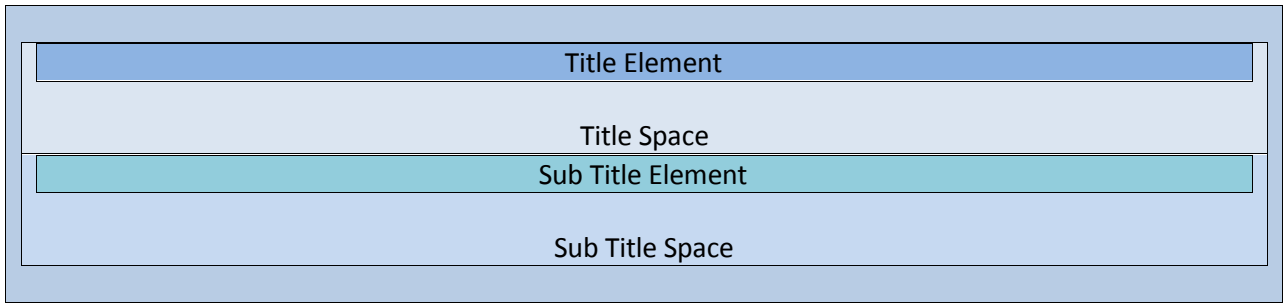

*Control Panel Position: 8*



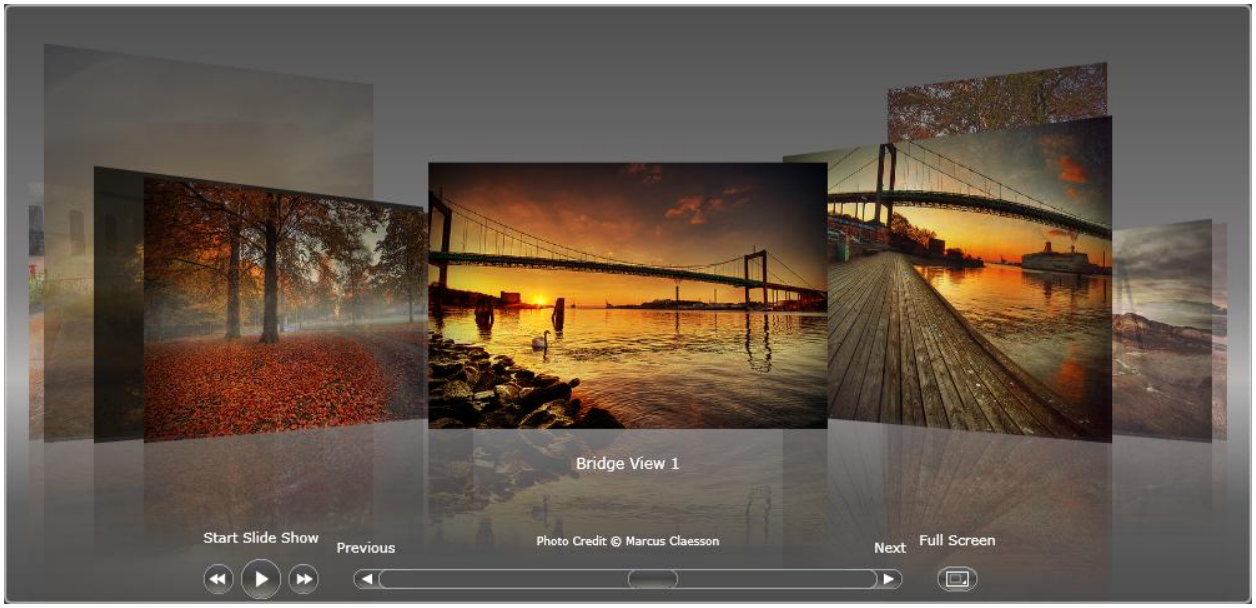

*Control Panel Position: 64*

### Text Area

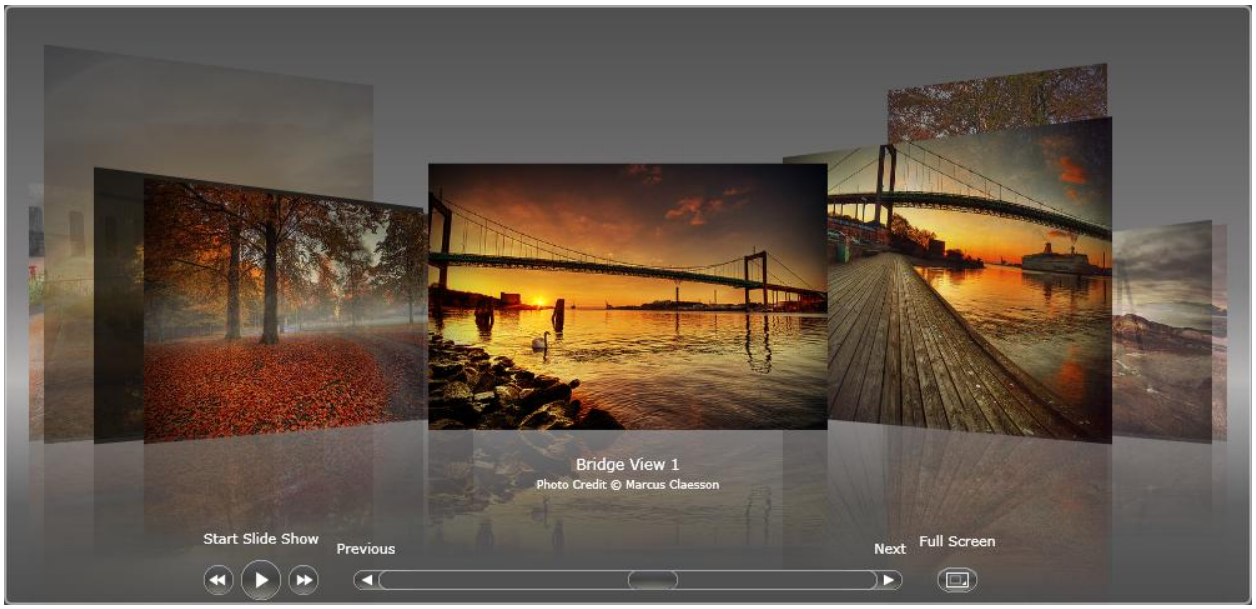
This area contains of two text elements, Title and SubTitle. The Title element is placed on top of the SubTitle element.



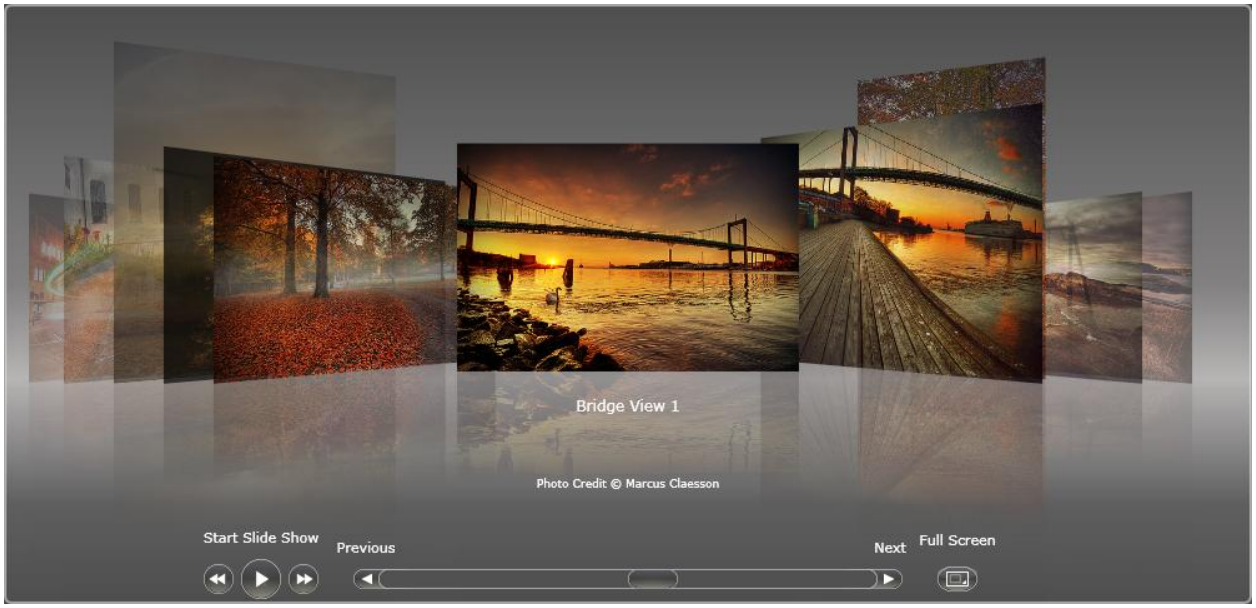

The title space is used to give a distance between the title element and sub title element. While the sub title space is used to give distance between sub title element with the next element.




*Title Space: 64*




*Sub Title Space: 64*

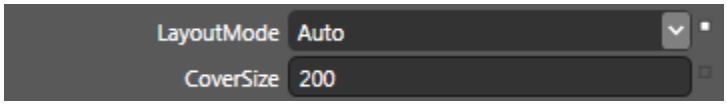



*Title Space: 64 & Sub Title Space: 64*

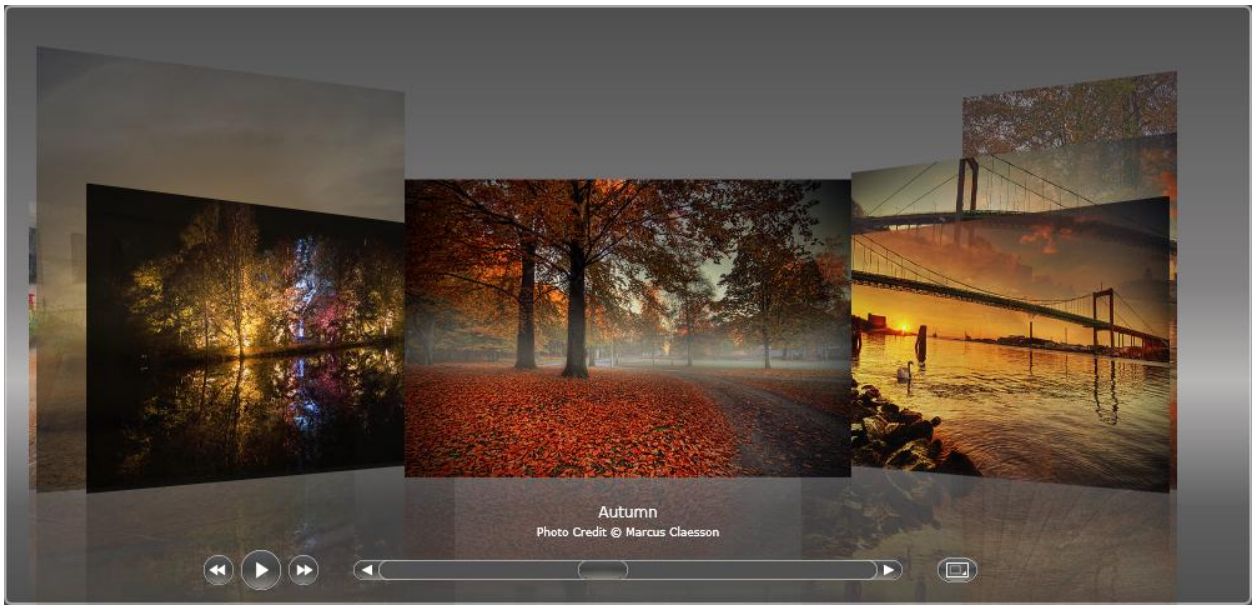
Note that you should specify enough amount of height for the text to be displayed, if your text spaces is lower than the actual text size the text will be truncated.

### Cover Area

This area is where all the cover flow items reside. If the LayoutMode is set to auto, the cover size will be automatically scaled based on the dimension of the control. On the other hand if the LayoutMode is set to manual, the size of CoverFlow item will use CoverSize property.

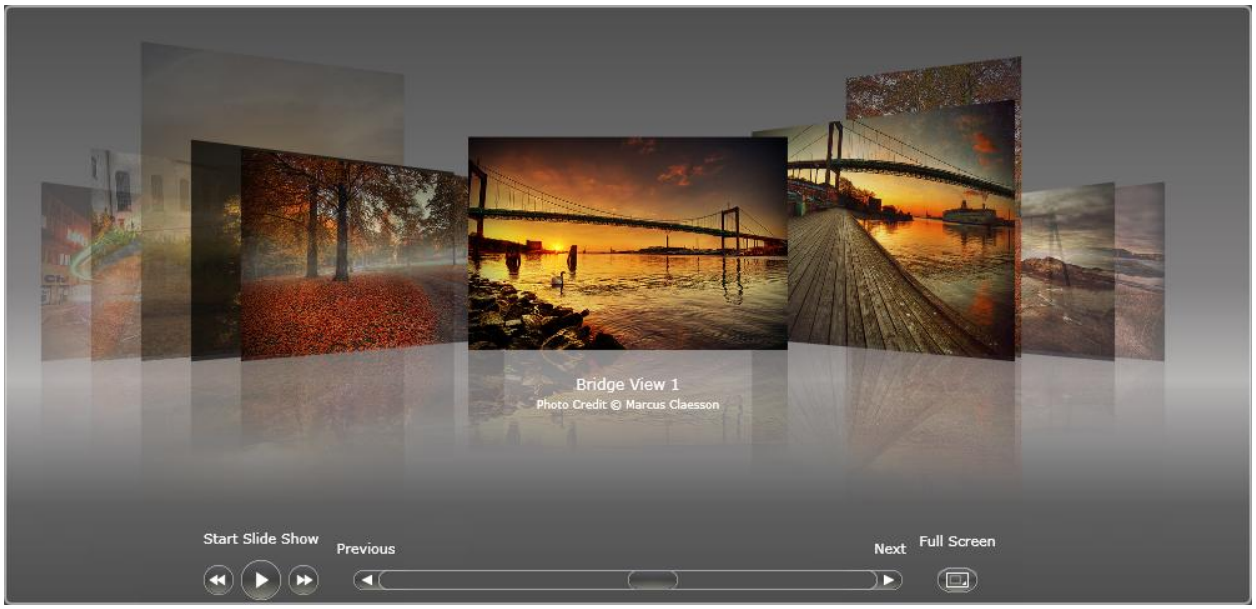


*LayoutMode: Auto*



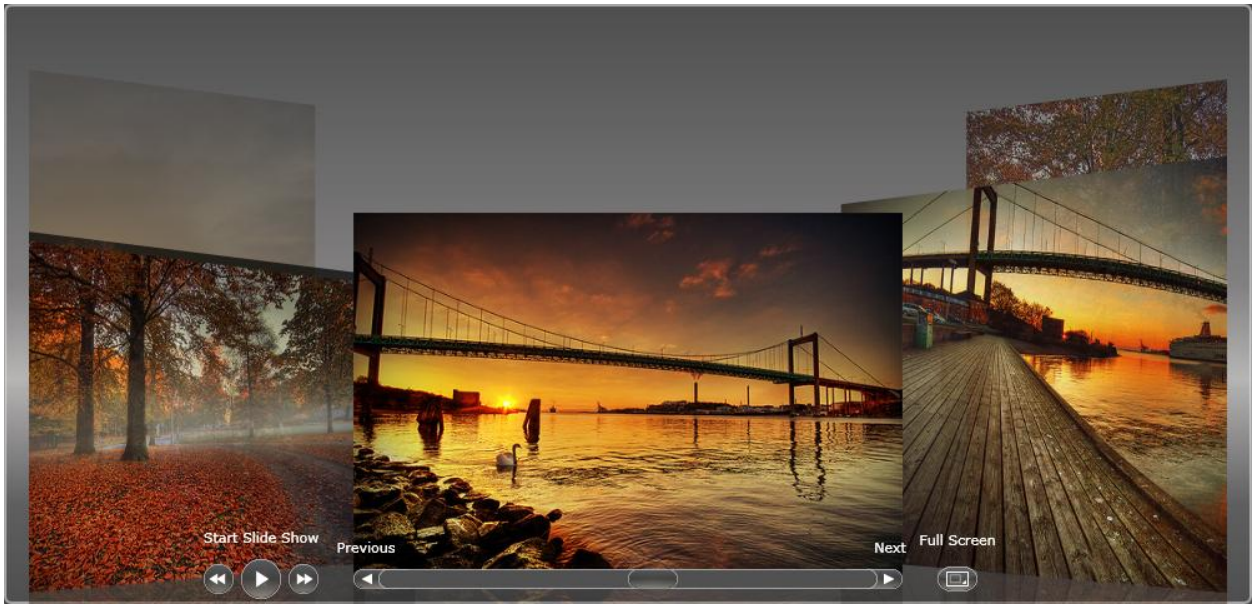

In this mode the size of CoverFlow items are calculated based on control dimension and the layout settings, for example CoverFlow padding, text area space, control panel position etc. So basically the other elements have higher priority than the CoverFlow items.

*LayoutMode: Manual*




*CoverSize: 256*

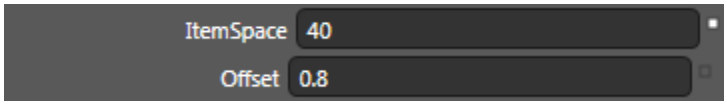
In this mode the size of CoverFlow items are determined by CoverSize property. Unlike in LayoutMode: Auto. The size of CoverFlow items is independent from any other element. As shown in the picture below the CoverFlow items is overlapping with control panel element and.



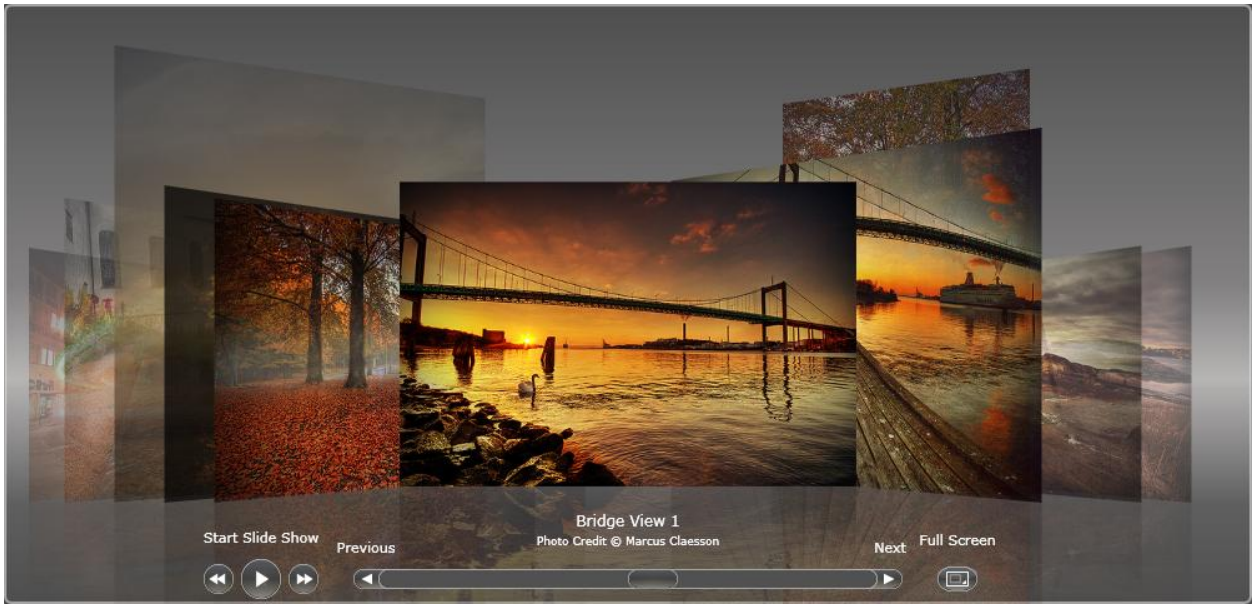

*CoverSize: 440*

### ItemSpaces & Offset

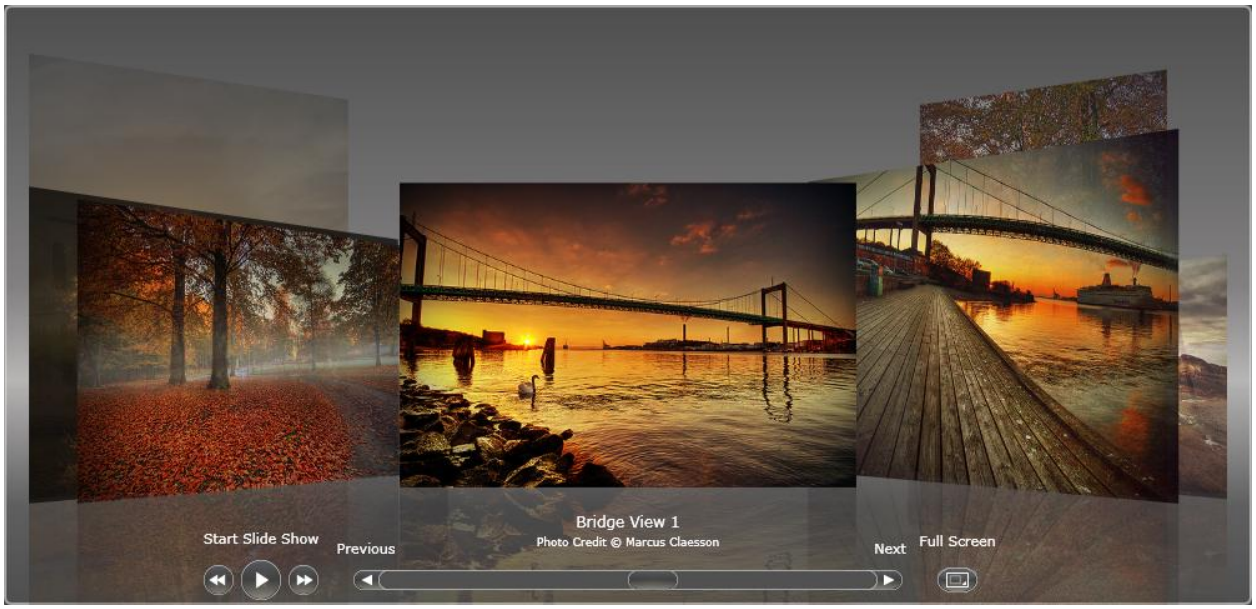
You can also customize the position of preview items, with the following properties



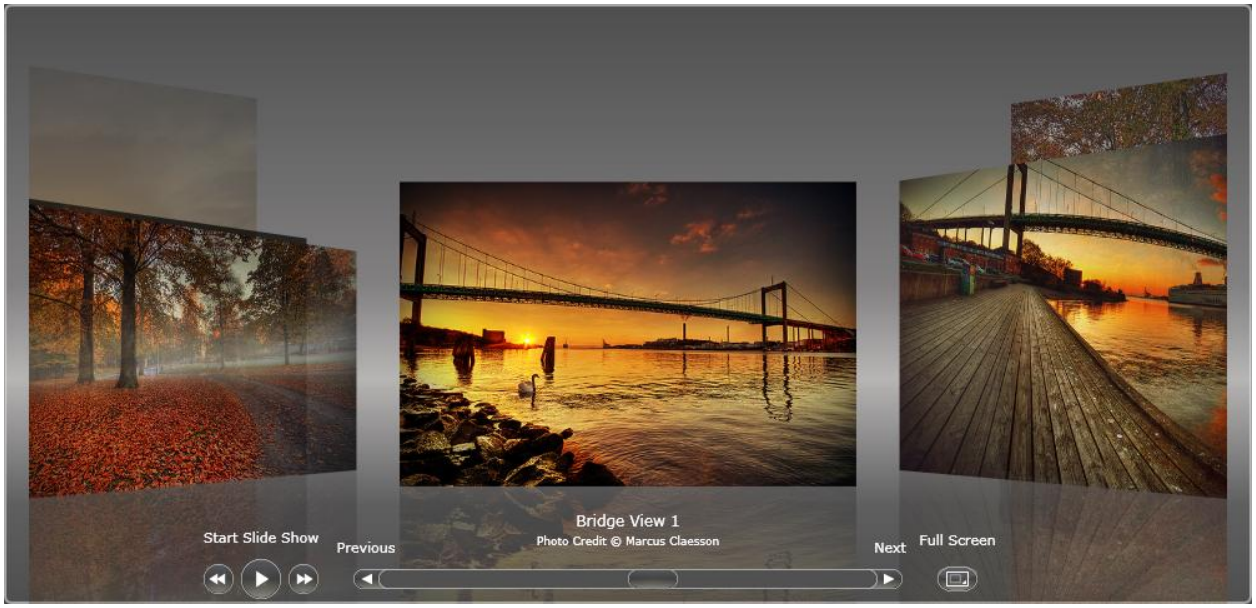

Offset determine the space between the focus item with the first item on the left and right using the following formula [distance = offset * actualCoverSize]. Meanwhile the distance between the rests of items is determined by ItemSpace.



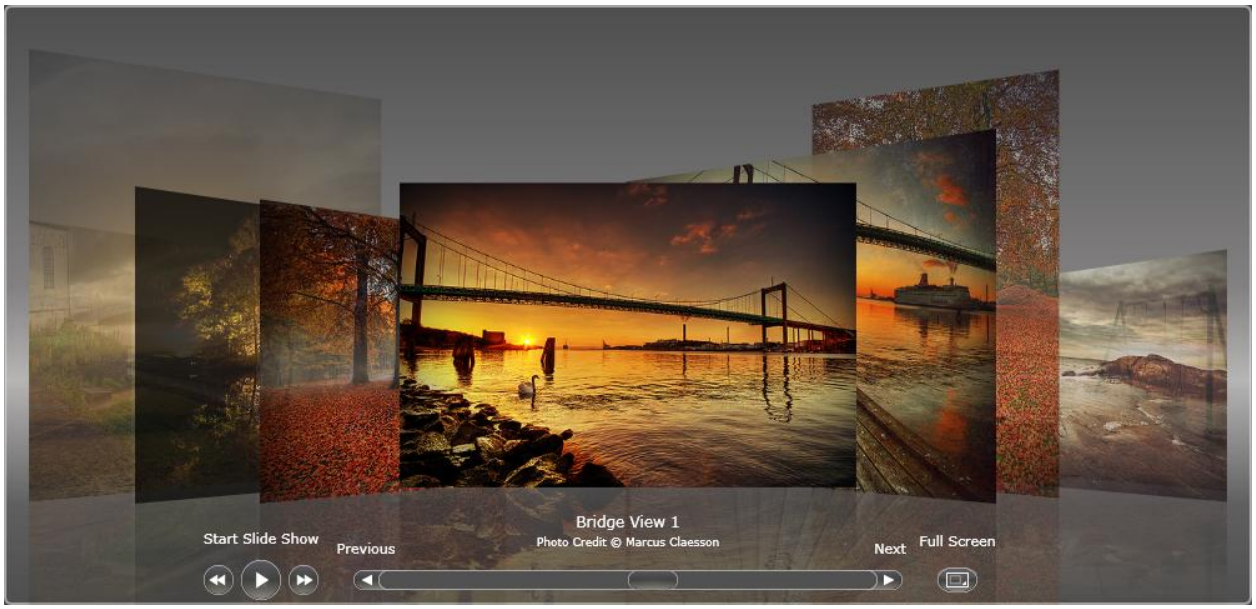

*Offset: 0.5*




*Offset: 0.8*



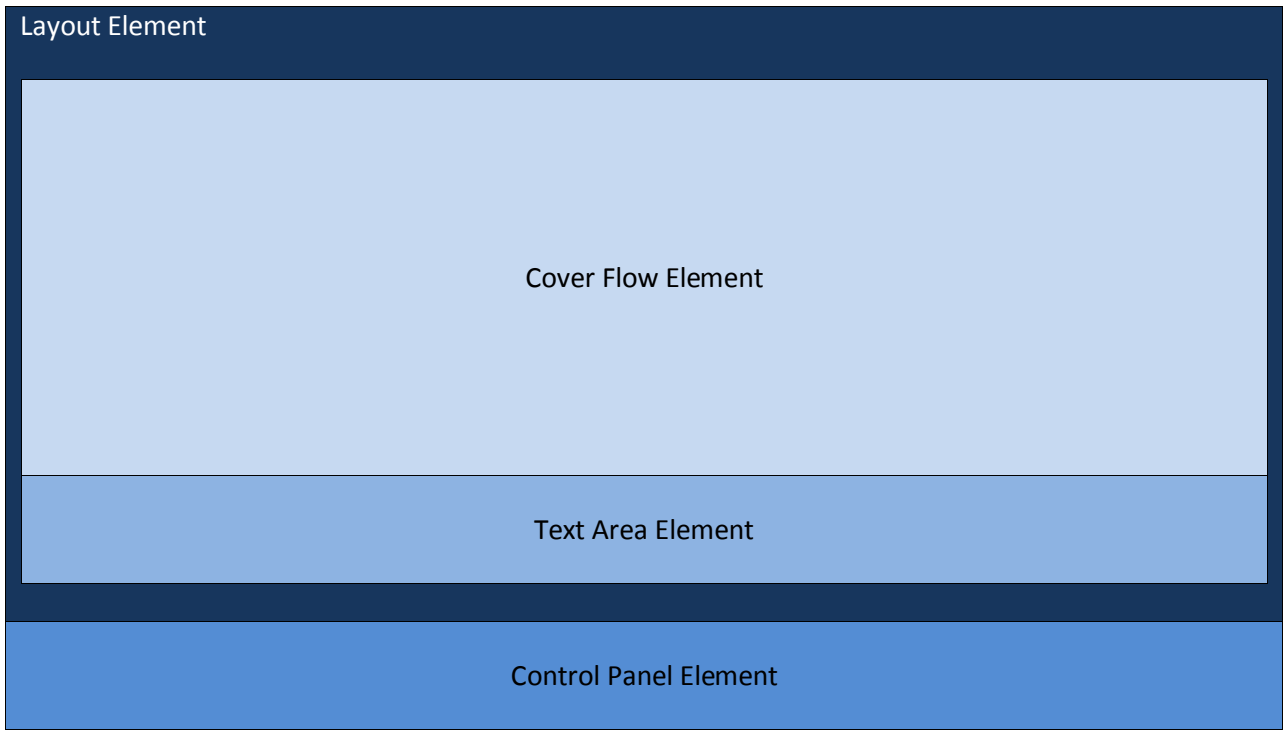

*Offset: 1.0*



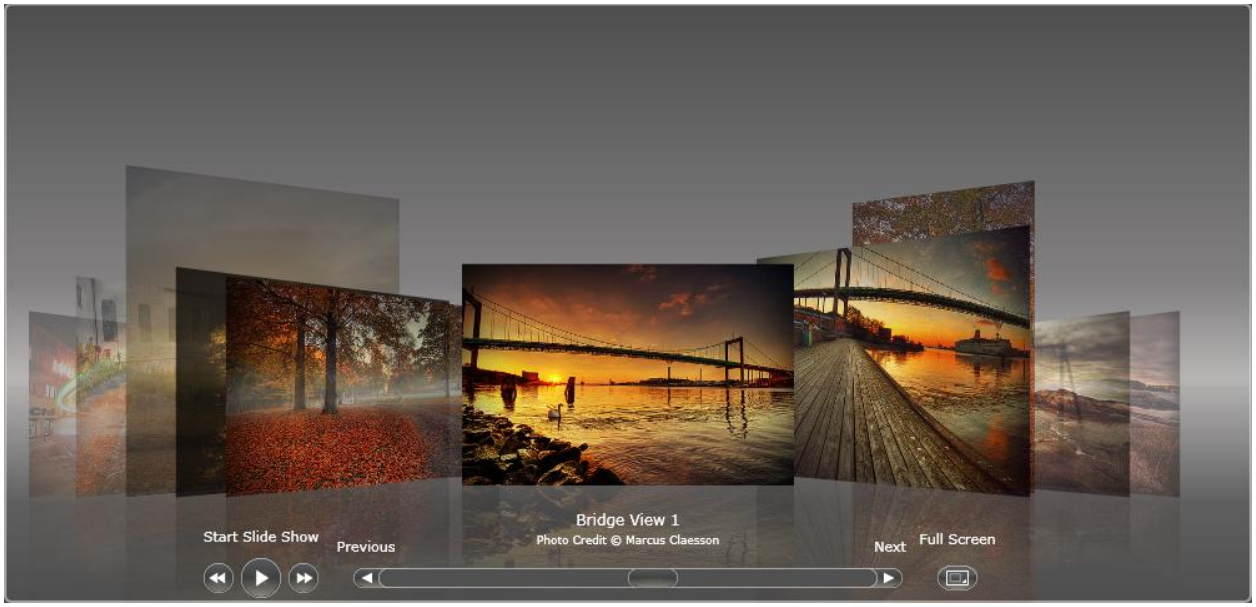

*Offset: 0.4 & ItemSpace: 100*

## CoverFlow Padding

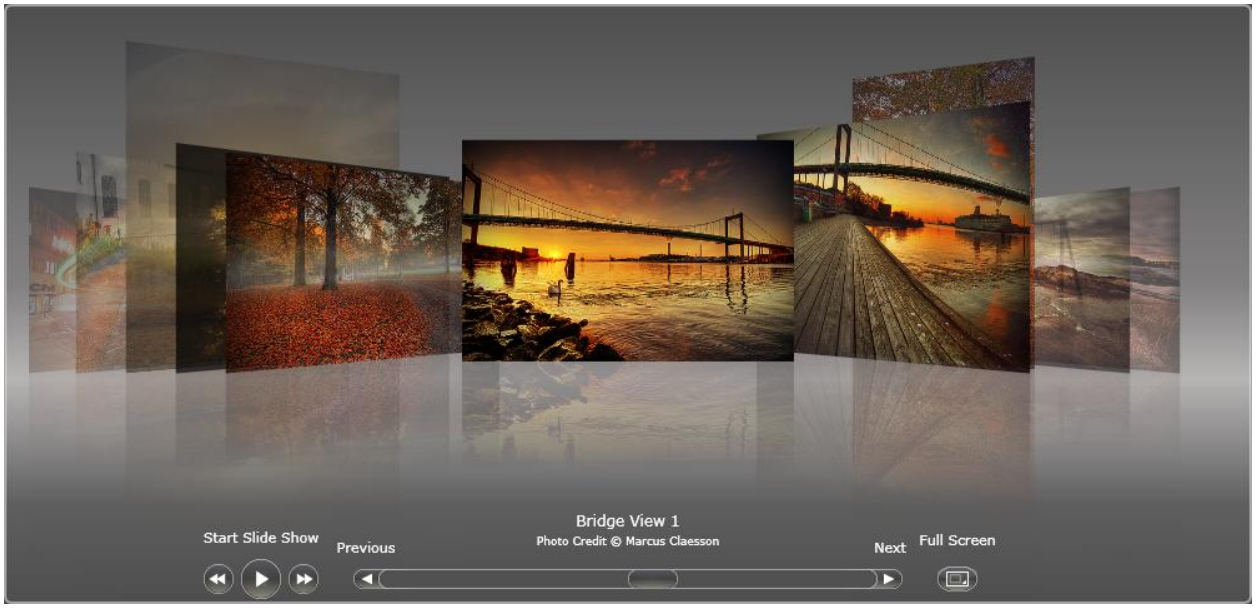
The CoverFlow padding effect the LayoutElement



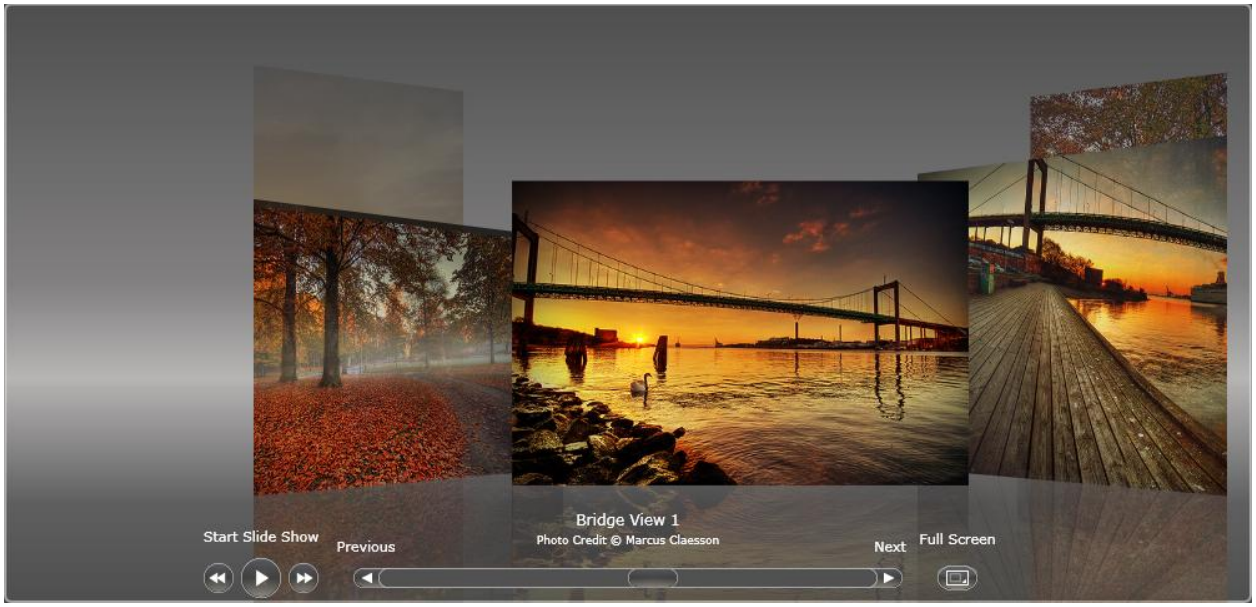

Here's several result when you play with the CoverFlowPadding property.



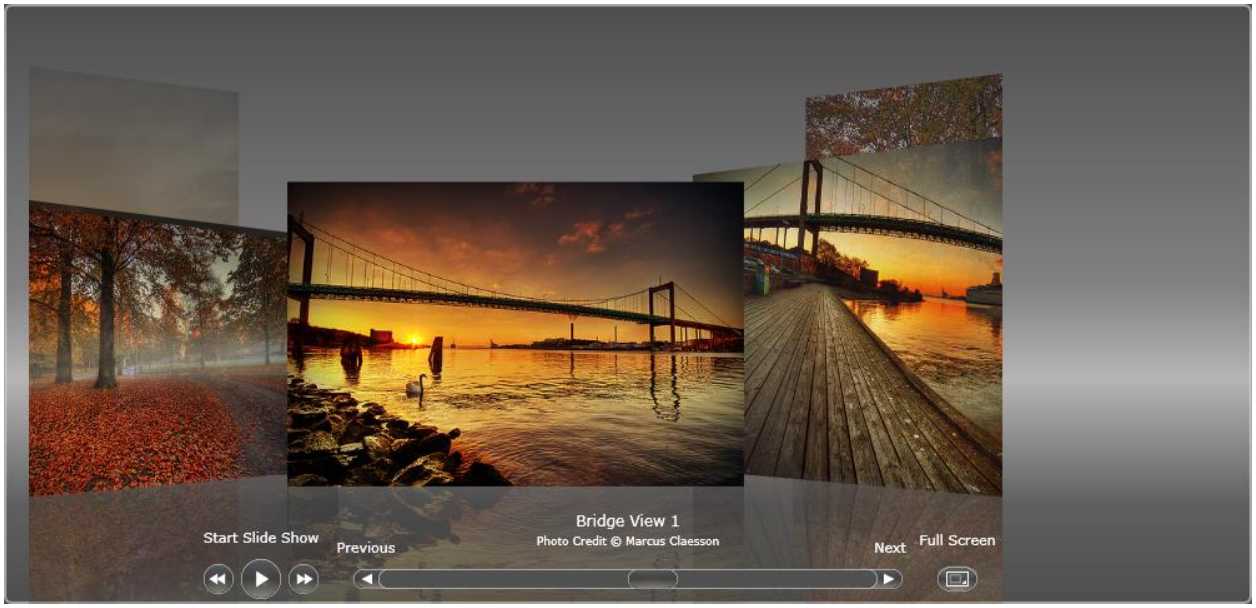

*CoverFlowPadding: 20, 120, 20, 20 (Top: 120)*




*CoverFlowPadding: 20, 20, 20, 120 (Bottom: 120)*




*CoverFlowPadding: 200, 20, 20, 20 (Left: 200)*




*CoverFlowPadding: 20, 20,200, 20 (Right: 200)*

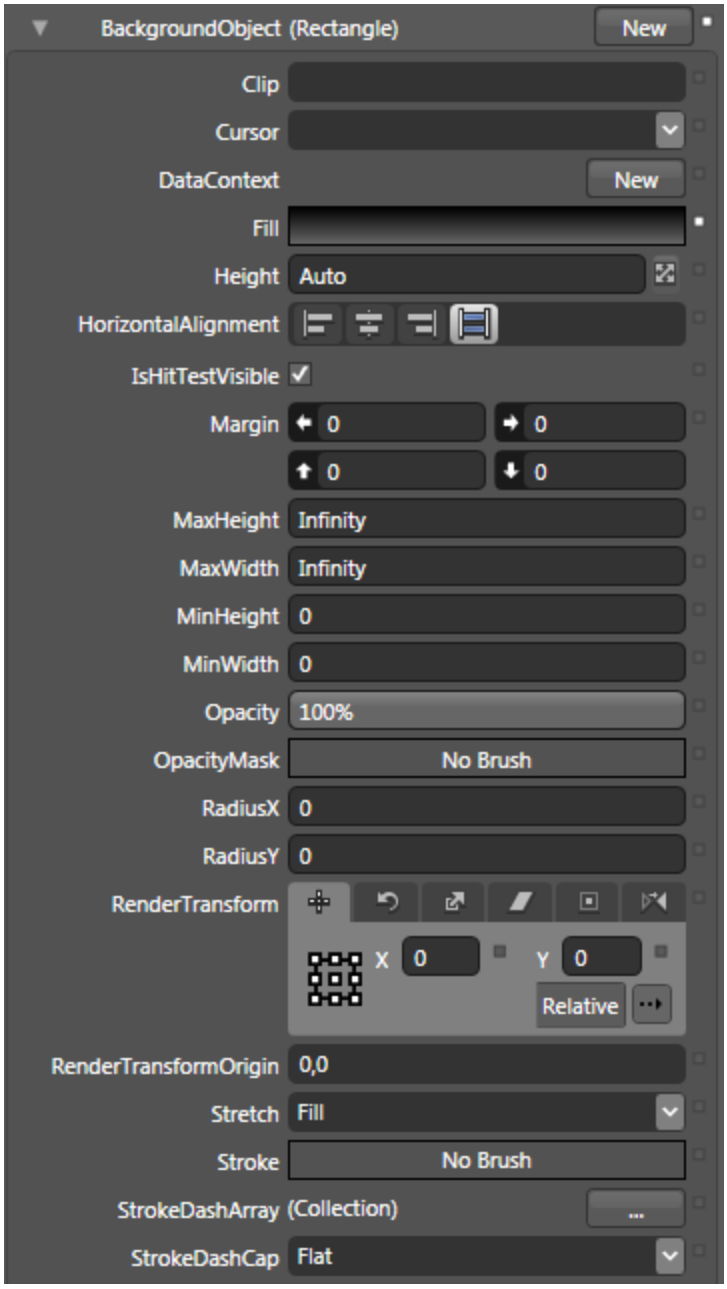
### Concept: Background

There are three types of background settings that you can apply to WebCoverflow.NET. Those are:

1. Simple Mode
2. Image Mode
3. Custom Mode

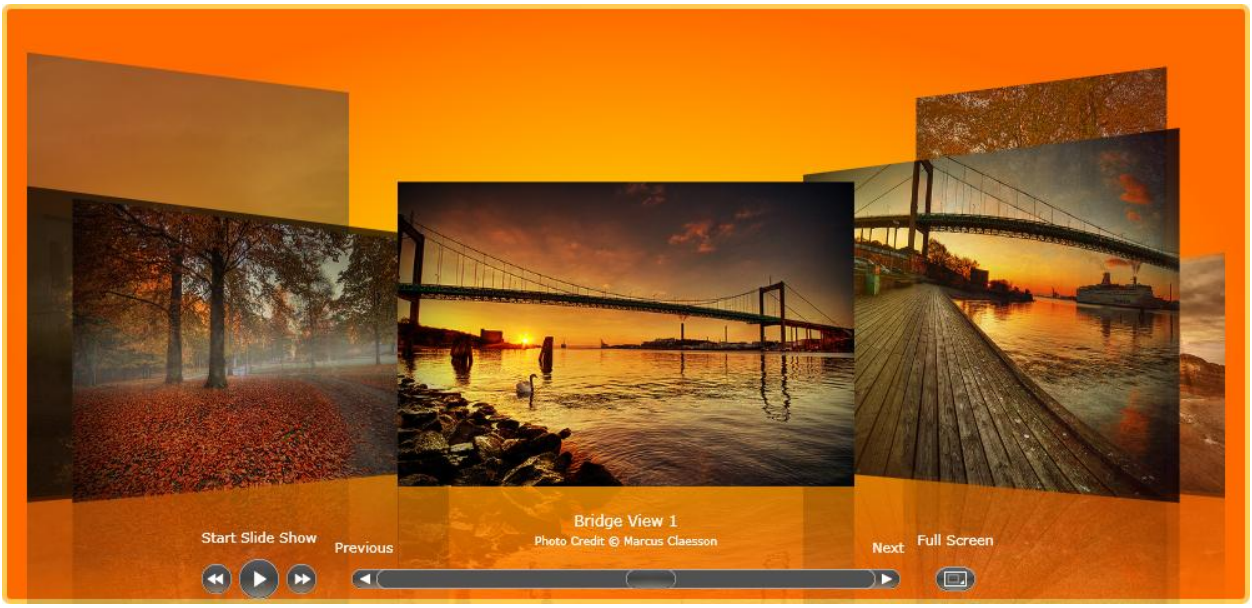
#### *Simple Mode*

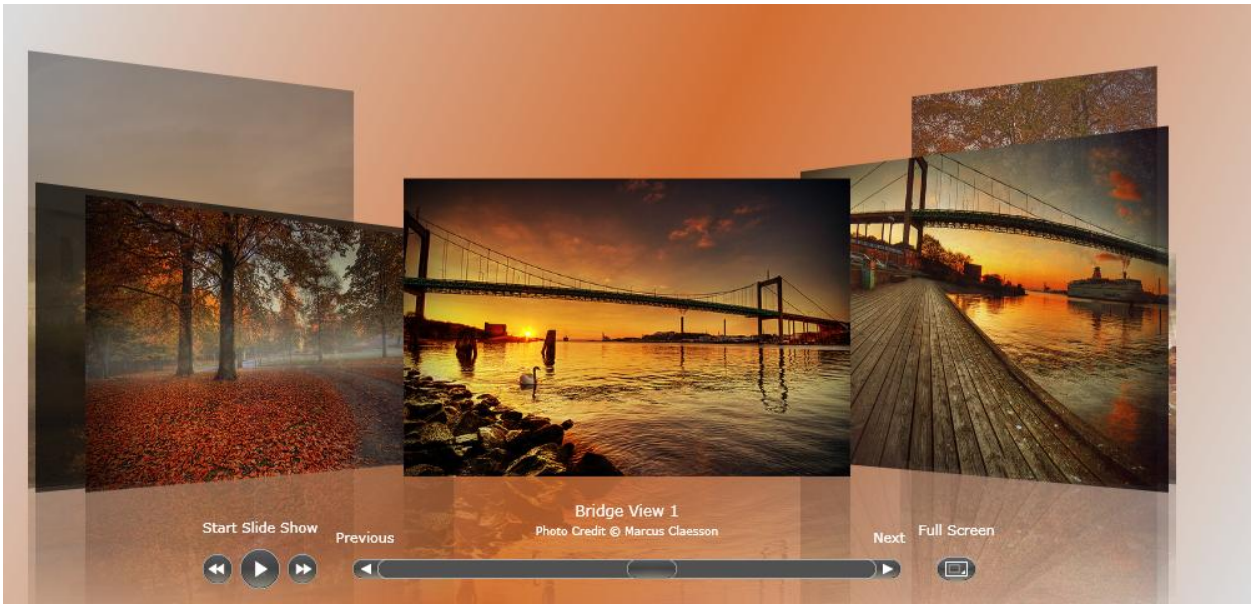
In simple mode, the background object is a Rectangle object which you can control completely using the BackgroundObject property.




You can specify the width and/or height of the rectangle / background through height's property and width's property. If you prefer the background's dimension to follow the control's dimension you can set them to ***Auto***.This will allow the background object to fill its container.

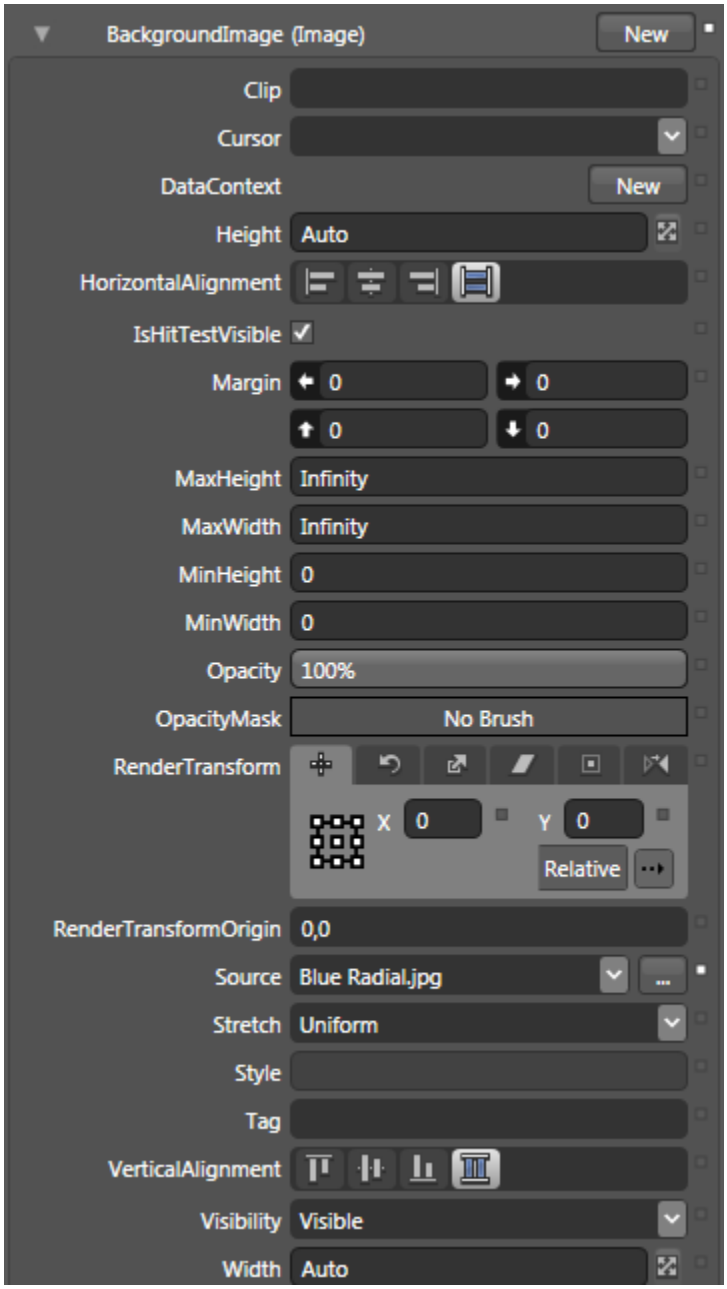
Here's some of the sample image what you can achieve by modifying the background object.


Start Slide Show
Previous
Bridge View 1
Photo Credit © Marcus Claesson
Next
Full Screen

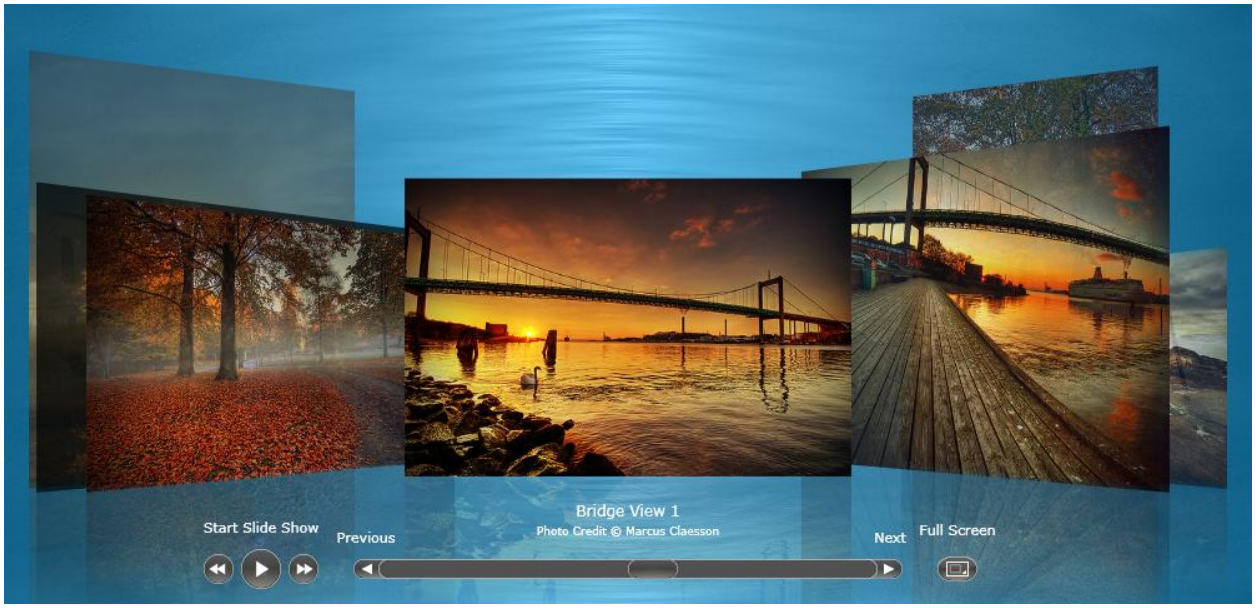

Start Slide Show
Previous
Bridge View 1
Photo Credit © Marcus Claesson
Next
Full Screen

### Background: Image Mode

In image mode you can specify an image to represent the background. You can control the image from the BackgroundImage property.




Here's the result

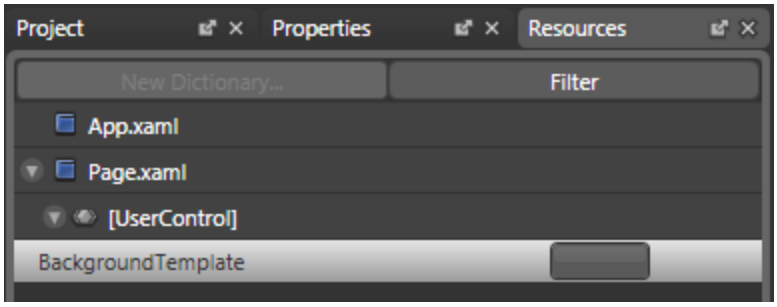



### *Custom Image Mode*

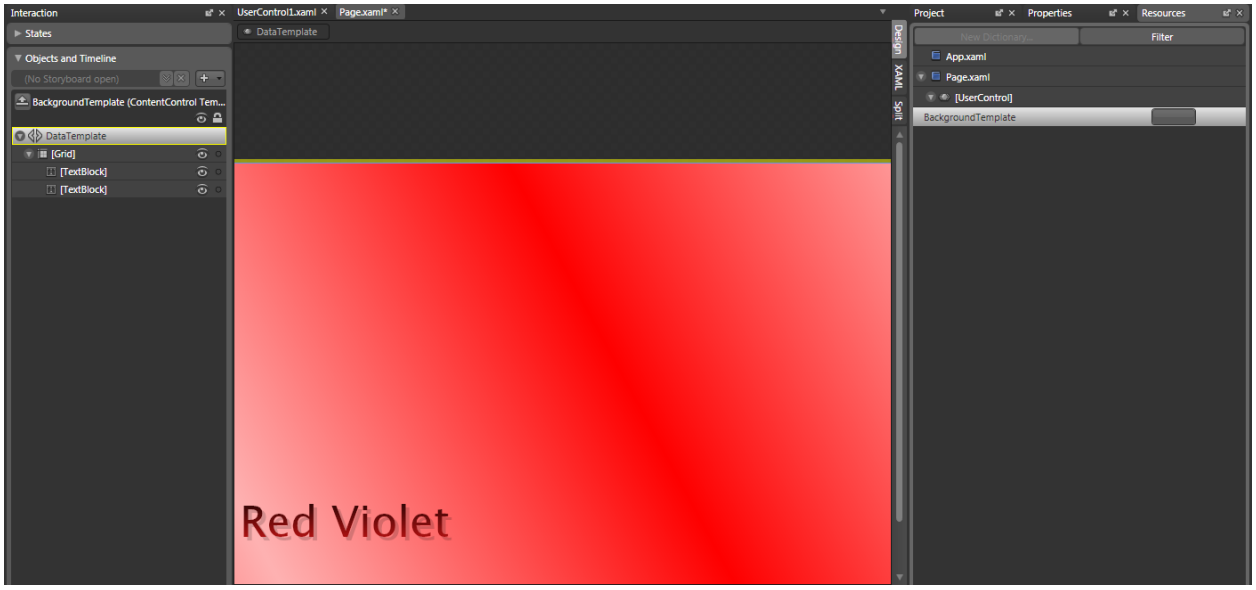
In custom image mode you can create a template object and attach it to WebCoverFlow background container.

This is one way to create the template object.

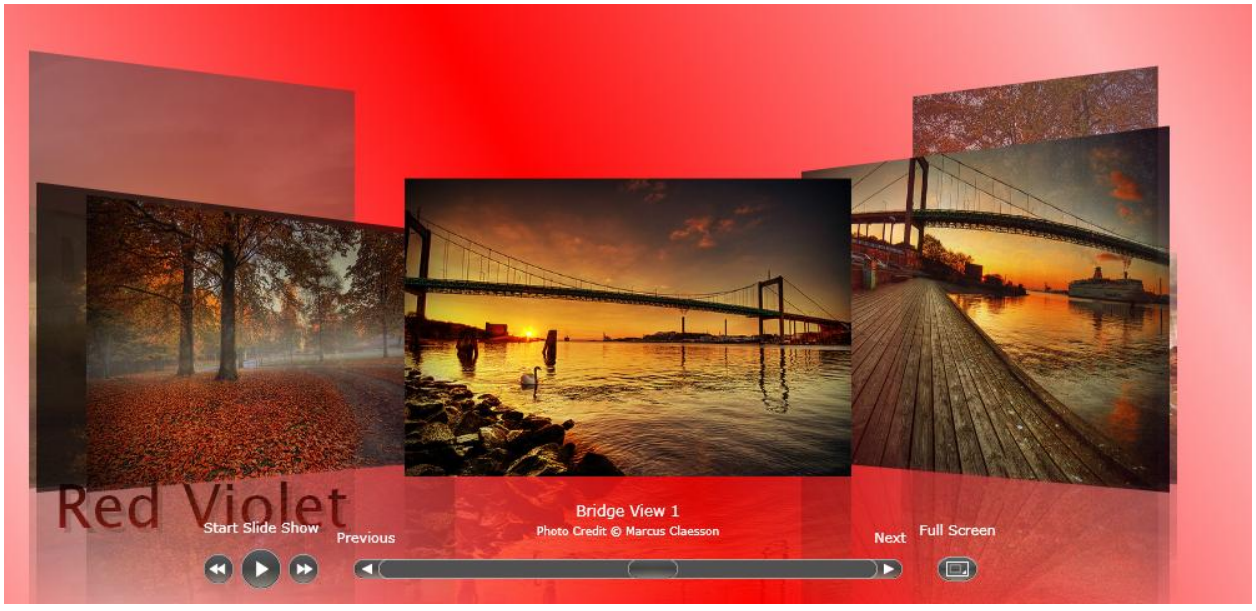

You can then go to resources tab and edit your template.




Create your custom template.




When you go back you'll get something as follow.



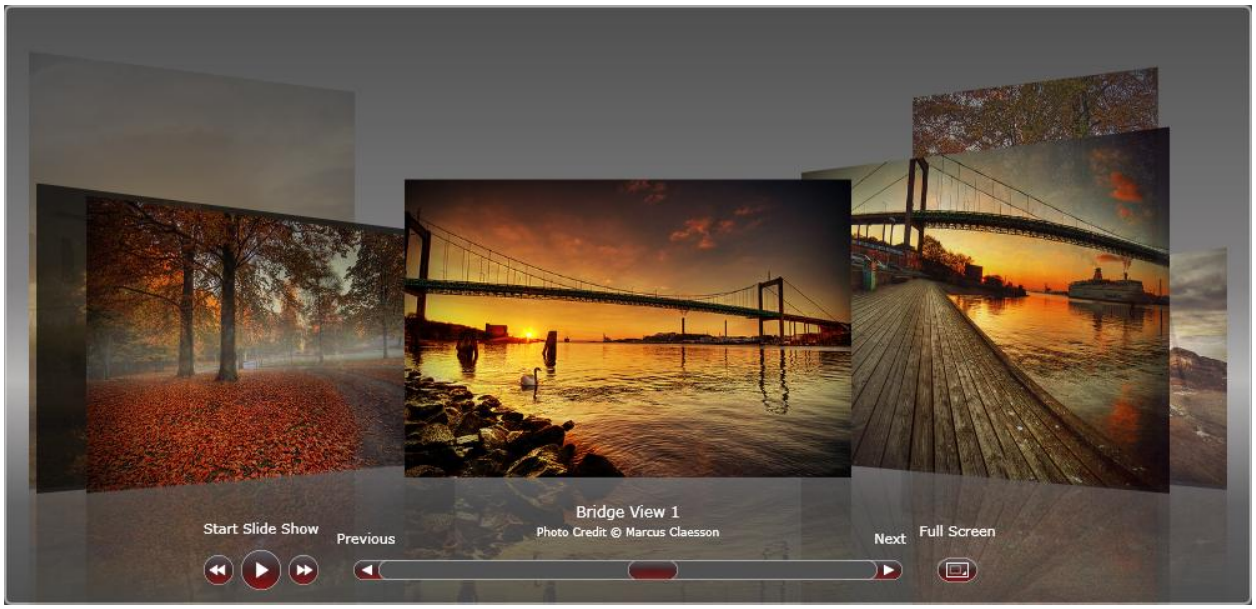
Here's template (Xaml):

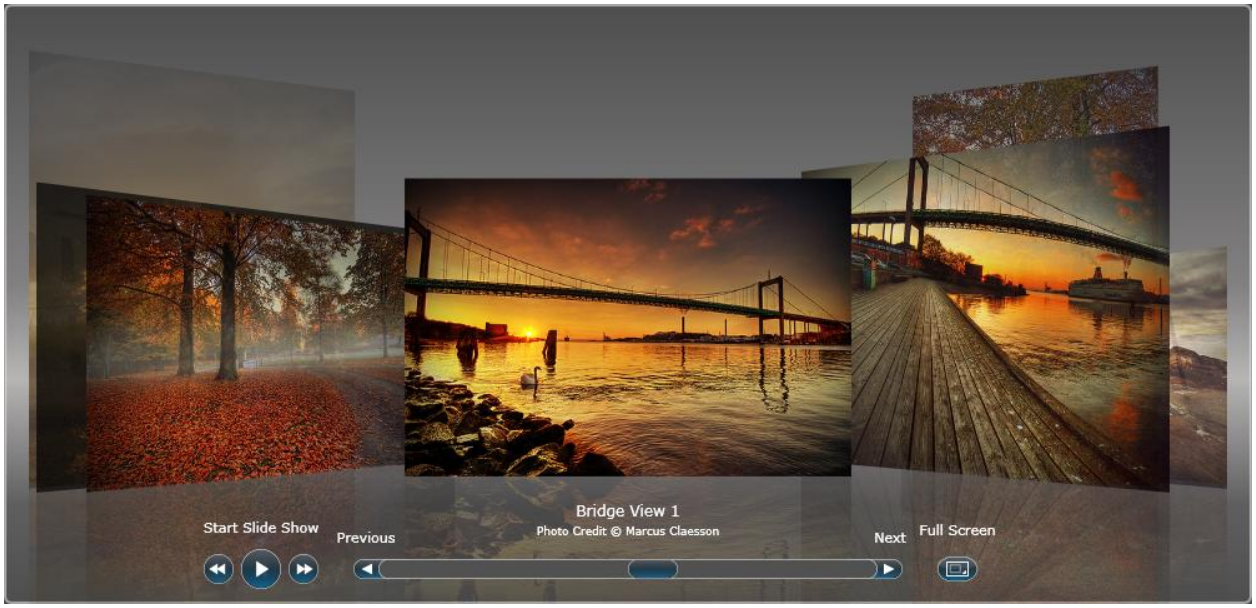
```
<UserControl.Resources>
    <DataTemplate x:Key="BackgroundTemplate">
        <Grid>
            <Grid.Background>
                <LinearGradientBrush EndPoint="0.4,0.3"
                    StartPoint="0.0,0.75" MappingMode="RelativeToBoundingBox"
                    SpreadMethod="Reflect">
                    <GradientStop Color="#FFFFB3B3"/>
                    <GradientStop Color="#FFFF0000" Offset="1"/>
                </LinearGradientBrush>
            </Grid.Background>
        <TextBlock HorizontalAlignment="Left" Margin="44,0,0,36"
            VerticalAlignment="Bottom" FontFamily="Lucida Sans Unicode"
            FontSize="48" Foreground="#4C000000" Text="Red Violet"
            TextWrapping="Wrap"/>
        <TextBlock HorizontalAlignment="Left" Margin="40,0,0,40"
            VerticalAlignment="Bottom" FontFamily="Lucida Sans Unicode"
            FontSize="48" Text="Red Violet" TextWrapping="Wrap">
            <TextBlock.Foreground>
                <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                    <GradientStop Color="#FF000000"/>
                    <GradientStop Color="#FFF21818" Offset="1"/>
                </LinearGradientBrush>
            </TextBlock.Foreground>
        </TextBlock>
        </Grid>
    </DataTemplate>
</UserControl.Resources>
```

### Concept: Color Theme

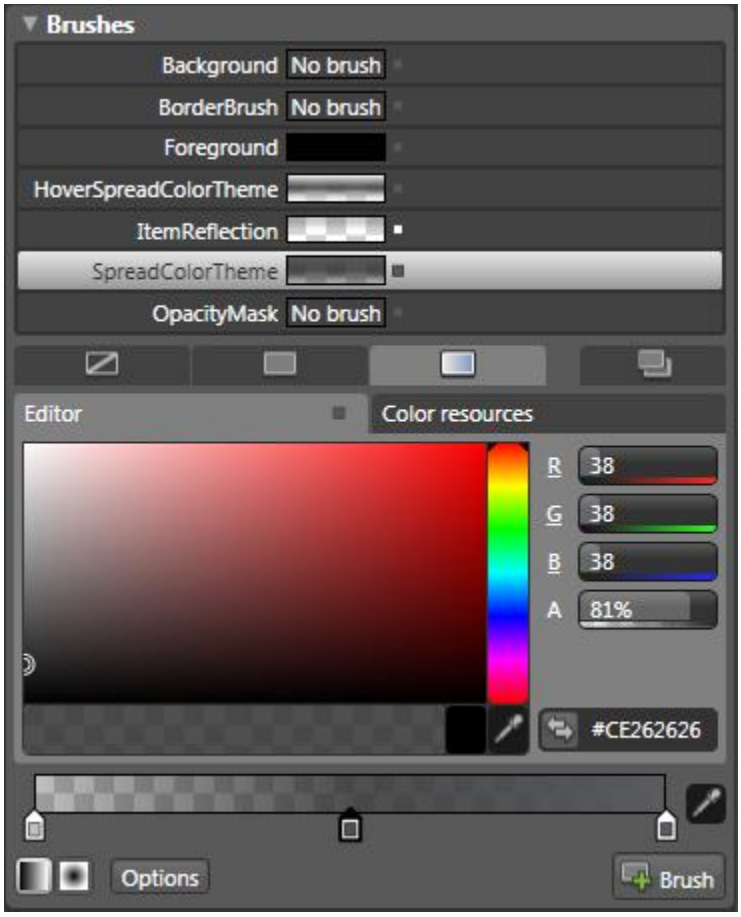
The control panel objects (slideshow player, video player, scrollbar, full screen button, etc) have a predefined shape and effect. However you can control the color scheme through:

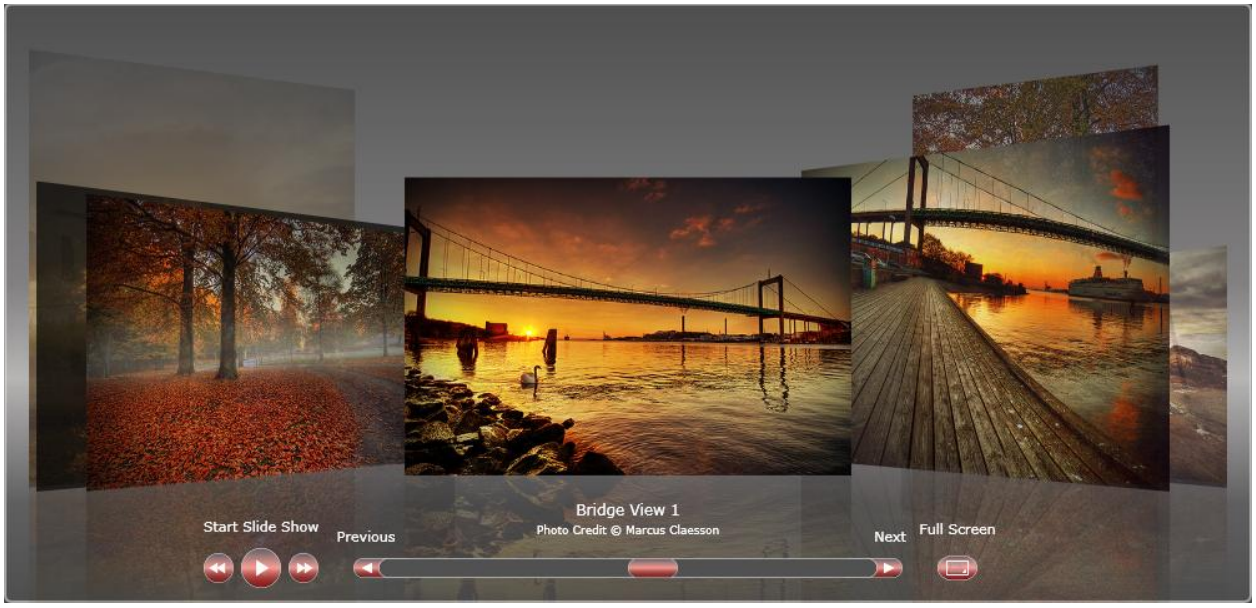
1. Color Theme
   Color theme controls the lower part color of the buttons.

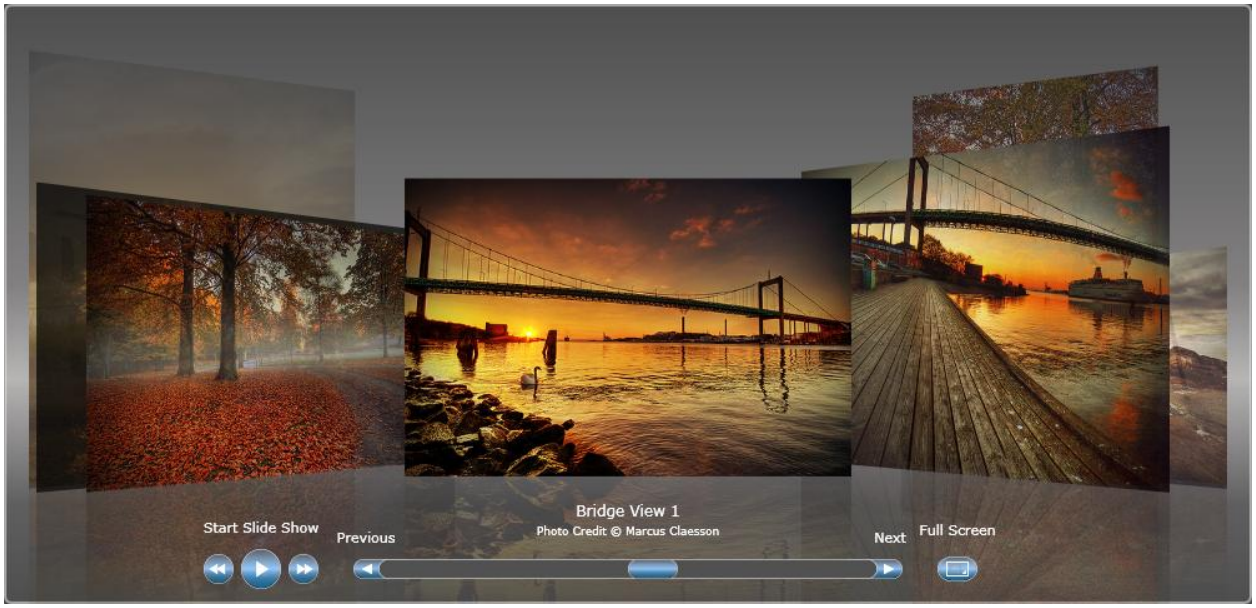



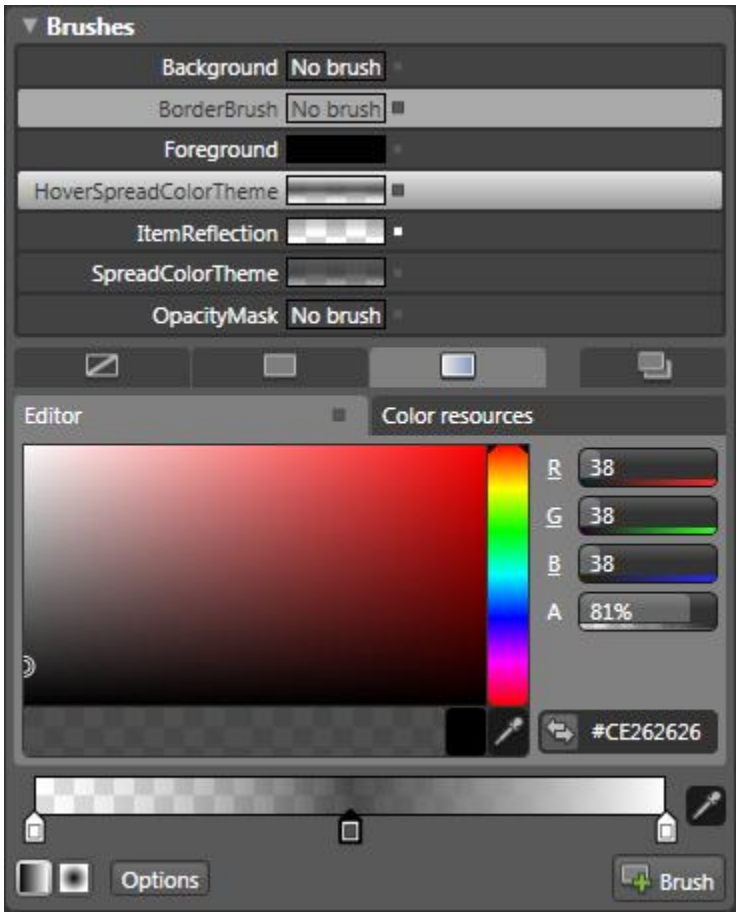

2. Spread Color
   Spread Color controls the medium up part color of the buttons.


Brushes
Background
No brush
BorderBrush
No brush
Foreground
HoverSpreadColorTheme
ItemReflection
SpreadColorTheme
OpacityMask
No brush
Editor
Color resources
R
38
G
38
B
38
A
81%
#CE262626
Options
Brush


Start Slide Show
Previous
Bridge View 1
Photo Credit © Marcus Claesson
Next
Full Screen



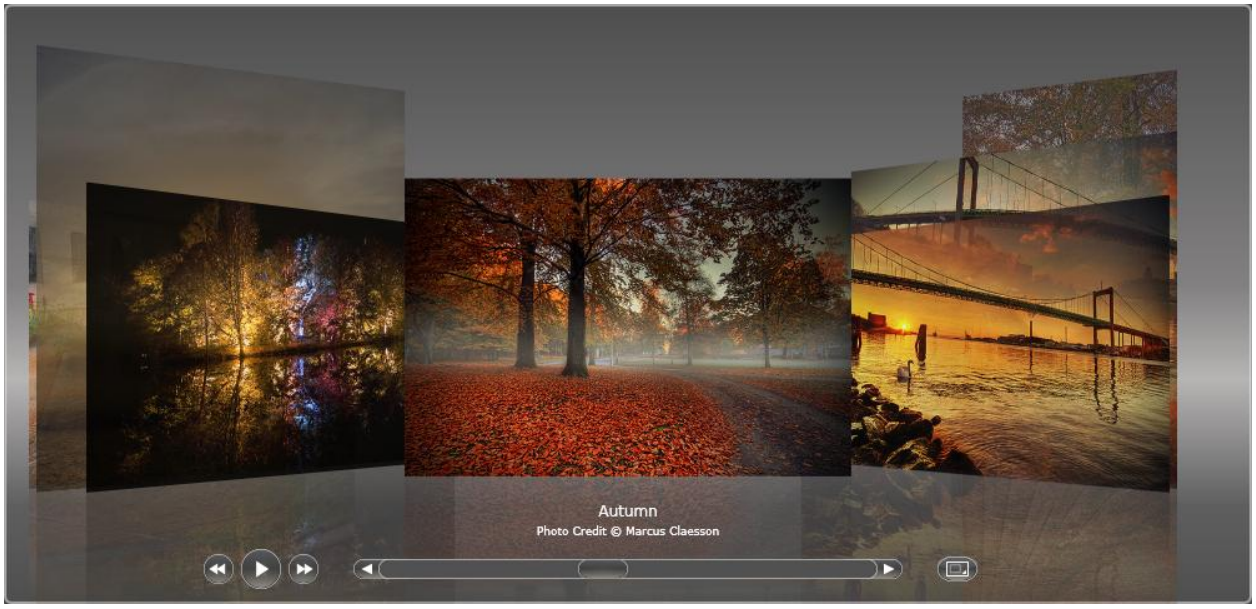

3. Hover Spread Color
   Hover Spread Color controls the medium up part color of the buttons when the button is hovered at run time.



## Concept: Visual Effect

WebCoverFlow.NET delivers two type of visual effect (Flow Effect). One is adapted from Apple iTunes's CoverFlow, which using a perspective point of view, and the other one is an innovative idea from Intersoft which using a scale point of view.

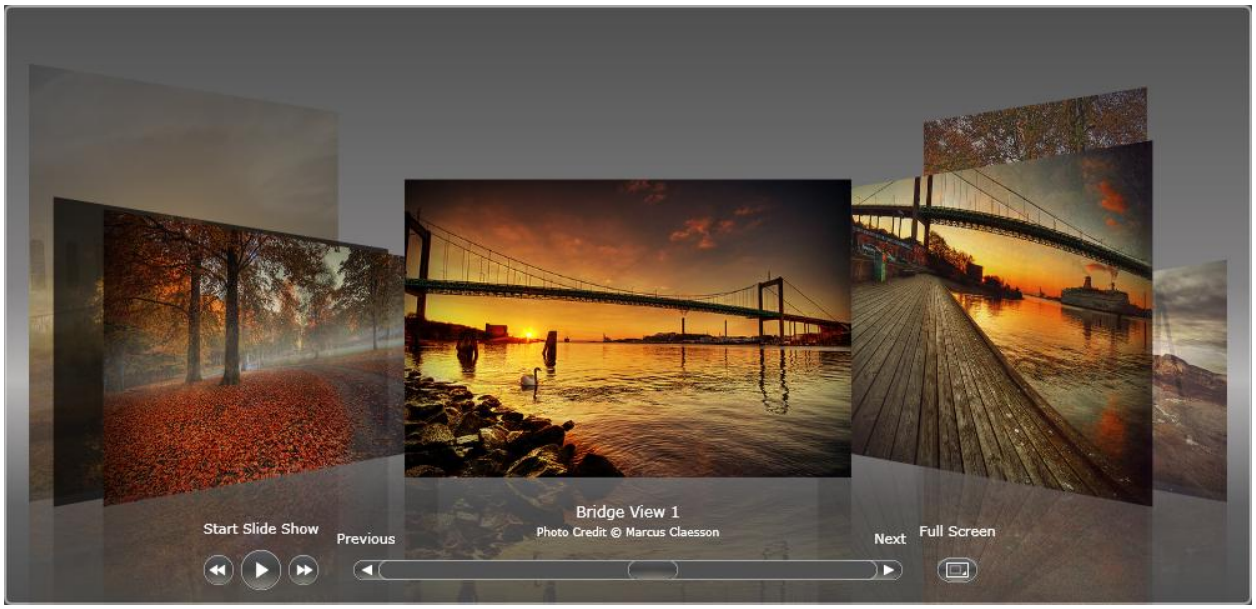
### *Flow Effect: Perspective Mode*



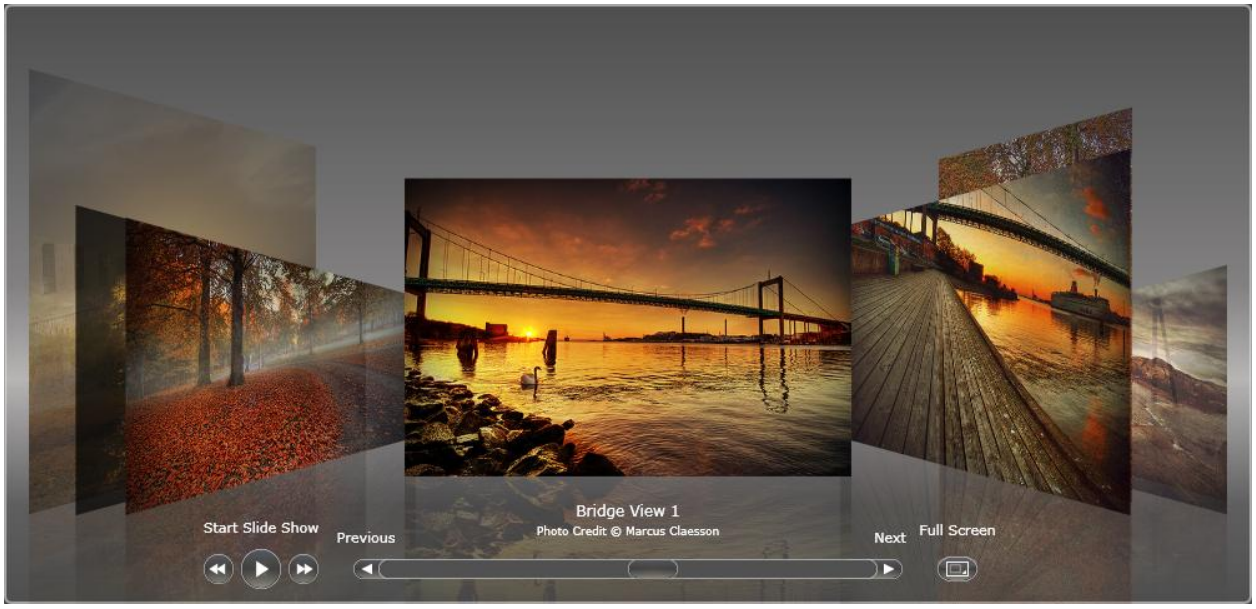

This effect of this mode is controlled by the following properties

- Perspective Top
- Perspective Bottom

The effect is not pure perspective due to limitation in Silverlight, therefore we use this two properties to simulate the perspective effect.

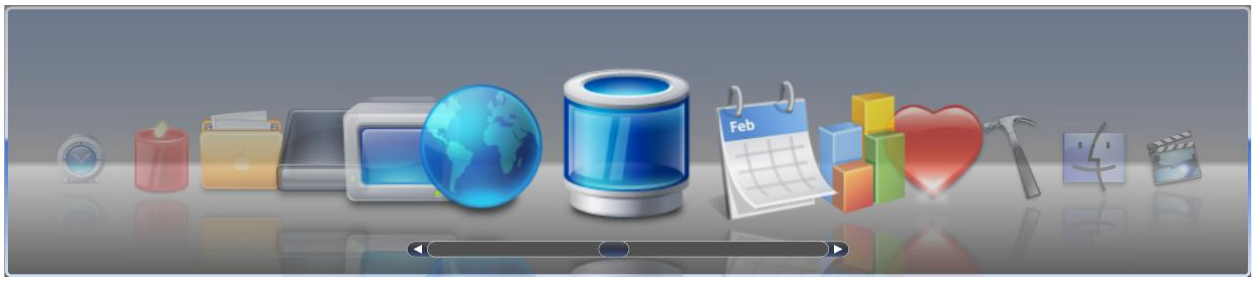


*PerspectiveTop: 0.25 & PerspectiveBottom: 10*



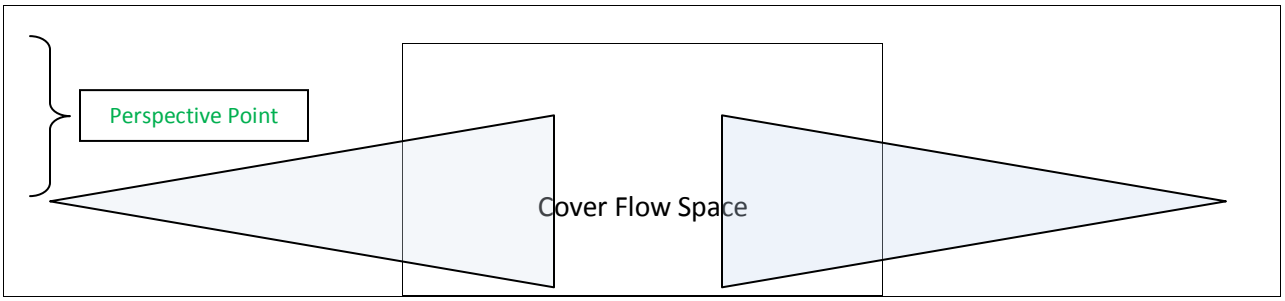

*PerspectiveTop: 0.35 & PerspectiveBottom: 15*

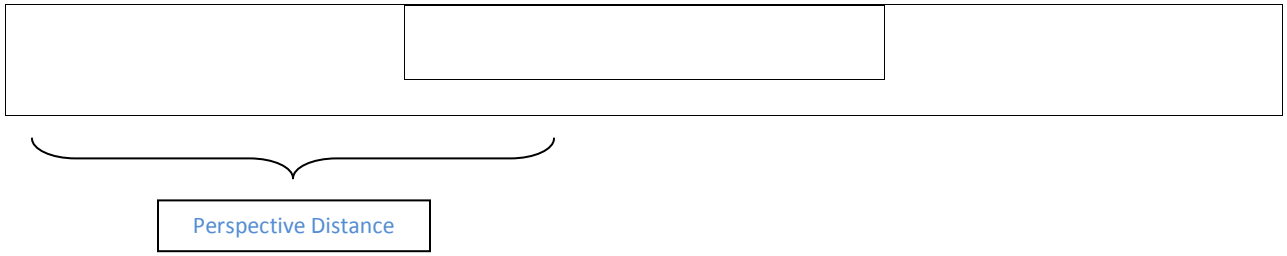
### *Flow Effect: Scale Mode*




This effect of this mode is controlled by the following properties

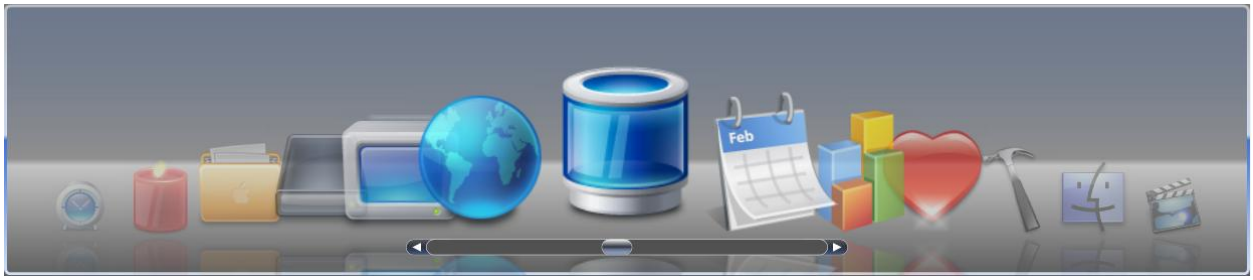
- Perspective Point
- Perspective Distance







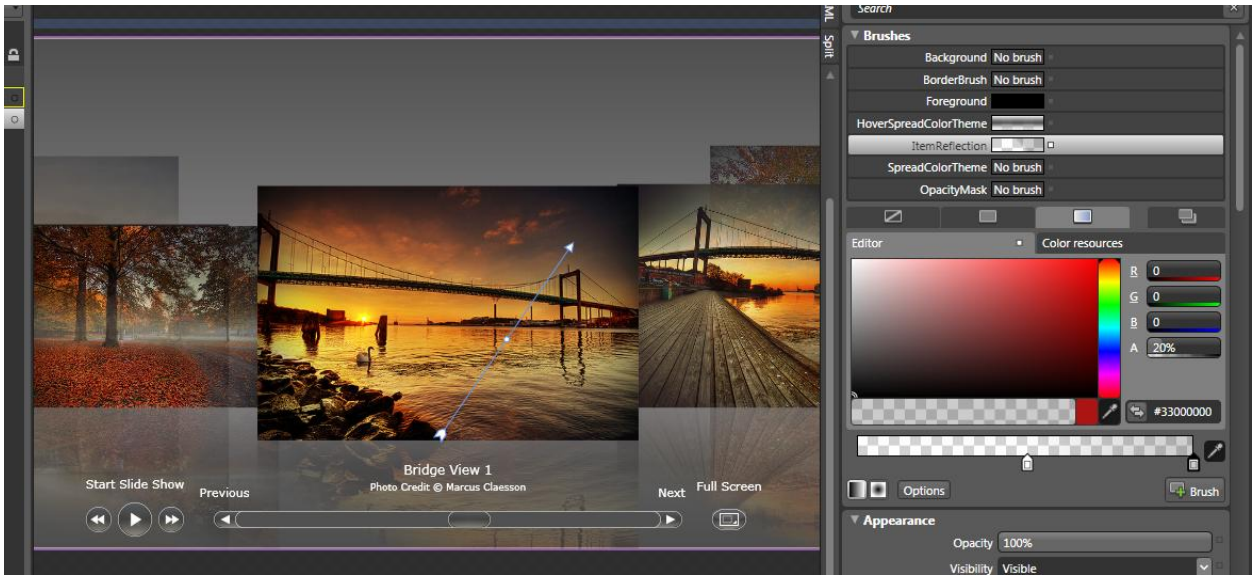
*Perspective Point: 1.0*




*Perspective Point: 0.5*

### Concept: Reflection Effect

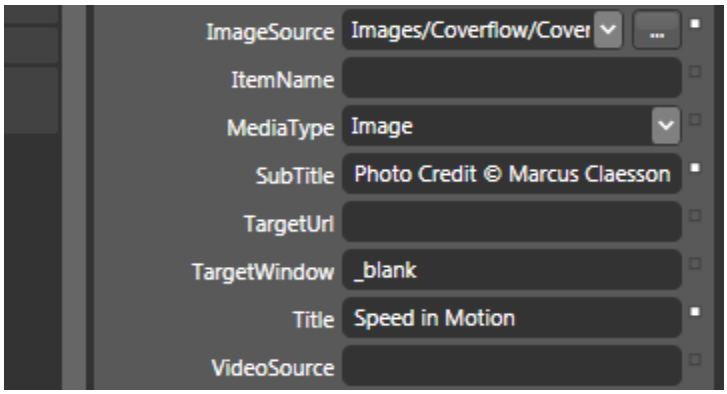
Reflection effect is one of the build-in effects embedded in WebCoverFlow.NET, which can be control from the following property.




The Button Reflection property is a Linear Gradient Brush type so you can control the direction of the gradient, and the gradient stops using the designer as shown in the picture above.

### Concept: Cover Flow Item

Each of Cover Flow Item has the following definition.




Several important things to be noted:

1. Each item must have a unique Item Name
2. Image Source indicates which image to be loaded
3. Title and Sub Title determined the text of the Cover Flow Item
4. Video source must be specified with path that point to a video file, when Media Type is set to Video.
5. Target Url and Target Window is used to open a new page.

When Cover Flow Item's media type is set to video, you need to specify a source image which will act as a preview image and assign the video source with a video file. If you do not specify a source image, the WebCoverFlow will use a default picture as the preview image.

All the image will be scaled proportionally based on the cover size / auto cover size property. Therefore it's a good idea to use a larger resolution file as the image source.

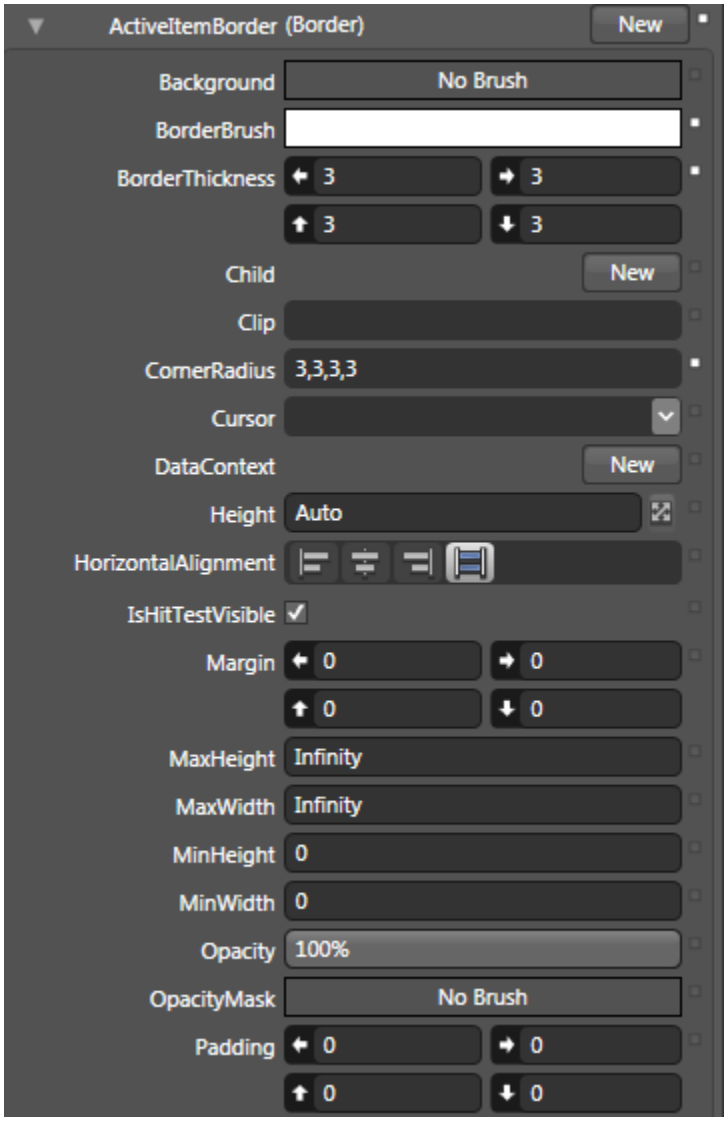
You can also switch to full screen mode by clicking the full screen button. If the LayoutMode is set to automatic, all the cover flow item will be resized to full screen area.

### *Multiple Focus Item*

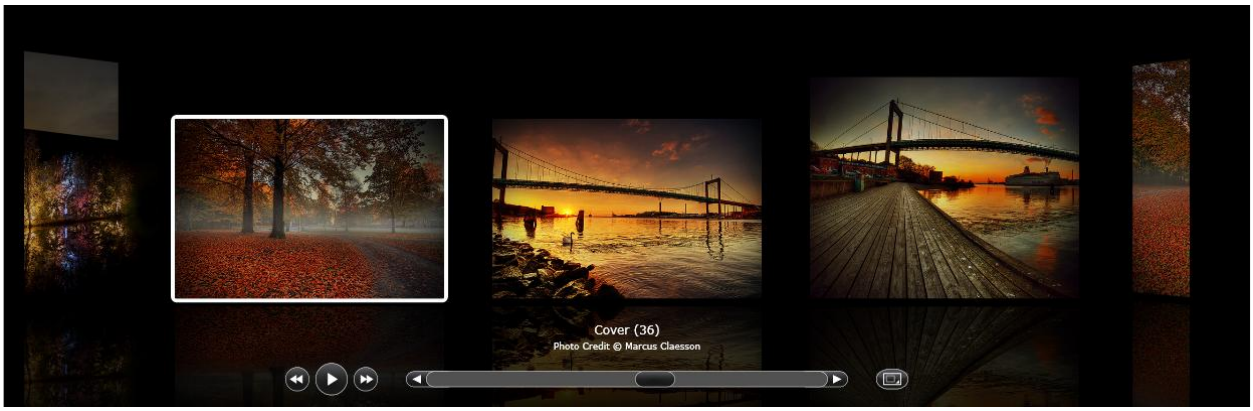
Multiple focus item allow user to have several items in display mode, which determined by TotalFocusItem property.


This value should be an odd value (for example: 1, 3, 5, 7, etc).

When the Multiple Focus Item feature is enabled (by specifying the TotalFocusItem with a value larger than one), you can show a border indicating that particular item is the one is active, by defining the ActiveItemBorder through its property.




After than in run time you'll see something as follows:




With the white bordered item being the active item.

### *Startup Position*

Start up position allow user to pick which item going to be selected or flowed into when the page is loaded. There are four types of startup position:

1. Center
2. First
3. Last
4. Custom

Center will flow the item to the center item when the page is loaded, similar thing happened when you set it to First (it will go to the first item) and when you set to Last (it will go to the last item). Furthermore if you want it to be flowed into specific position, you can use custom type and then specify the index from CustomStartupPosition property.

### *Maximum Preview Item*

Maximum Preview Item is used to determine the number of item in the left and right hand side of the focused item.


*Maximum Preview Item: 0*


*Maximum Preview Item: 1*


*Maximum Preview Item: 2*


*Maximum Preview Item: 4*

### Concept: Item Template & Reflection Template

Template item is introduced in version 2.0. With this feature you can use your own template object as content for WebCoverFlow's item and its reflection.

What you need to do firstly is to create the template object.

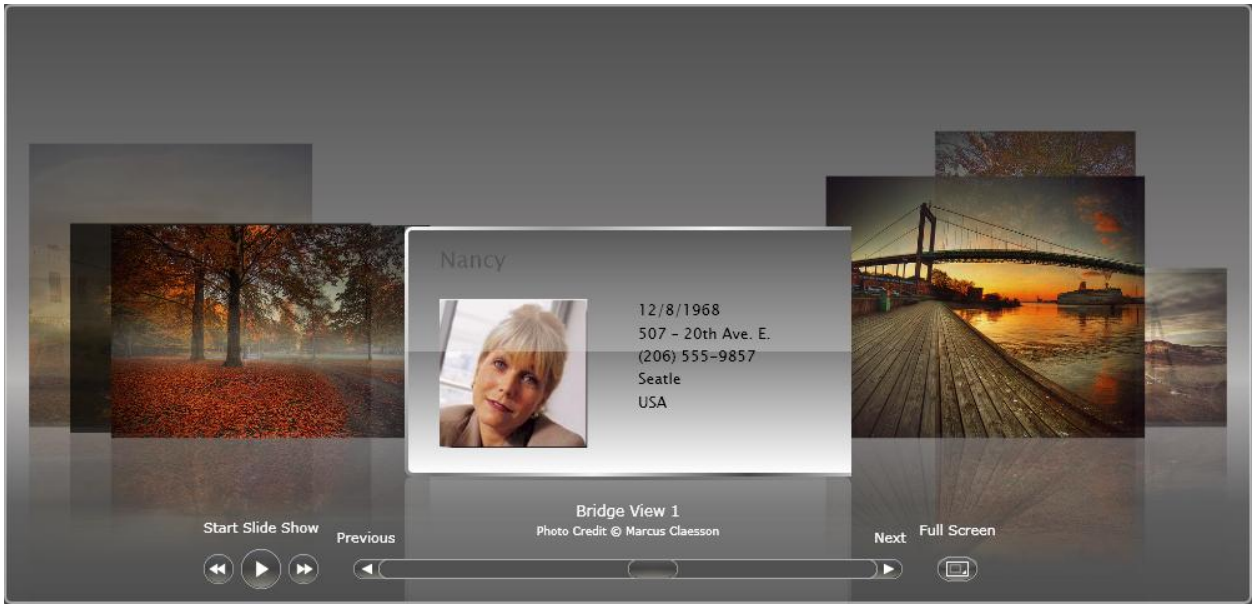
```
<UserControl.Resources>
    <DataTemplate x:Key="ReflectionTemplate">
        <Border BorderThickness="3,3,3,3" CornerRadius="5,5,5,5"
            HorizontalAlignment="Stretch" VerticalAlignment="Stretch">
            <Border.BorderBrush>
                <LinearGradientBrush EndPoint="1,1" StartPoint="0,0">
                    <GradientStop Color="#FFFFFFFF"/>
                    <GradientStop Color="#FFFFFFFF" Offset="1"/>
                    <GradientStop Color="#FFF2F2F2" Offset="0.5"/>
                    <GradientStop Color="#FF707070" Offset="0.25"/>
                    <GradientStop Color="#FF474747" Offset="0.75"/>
                </LinearGradientBrush>
            </Border.BorderBrush>
            <Border.Background>
                <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                    <GradientStop Color="#FF080808"/>
                    <GradientStop Color="#FFFFFFFF" Offset="1"/>
                </LinearGradientBrush>
            </Border.Background>
        </Border>
    </DataTemplate>

    <DataTemplate x:Key="ItemTemplate">
        <Border BorderThickness="3,3,3,3" CornerRadius="5,5,5,5" Width="400"
            Height="200" VerticalAlignment="Stretch">
            <Border.BorderBrush>
                <LinearGradientBrush EndPoint="1,1" StartPoint="0.0,0.0">
                    <GradientStop Color="#FFFFFFFF"/>
                    <GradientStop Color="#FFFFFFFF" Offset="1"/>
                    <GradientStop Color="#FFF2F2F2" Offset="0.5"/>
                    <GradientStop Color="#FF707070" Offset="0.25"/>
                    <GradientStop Color="#FF474747" Offset="0.75"/>
                </LinearGradientBrush>
            </Border.BorderBrush>
            <Border.Background>
                <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                    <GradientStop Color="#FF080808"/>
                    <GradientStop Color="#FFFFFFFF" Offset="1"/>
                </LinearGradientBrush>
            </Border.Background>
            <Grid>
                <Image HorizontalAlignment="Left" Margin="25,55,0,20"
                    VerticalAlignment="Stretch"
                    RenderTransformOrigin="0.5,0.5" Source="Nancy.jpg"/>
                <TextBlock HorizontalAlignment="Stretch"
                    VerticalAlignment="Top" Text="{Binding FirstName}"
                    Margin="25,10,0,0" FontSize="18"/>
                <Rectangle HorizontalAlignment="Stretch"
                    VerticalAlignment="Stretch" Margin="-3,-3,-3,-3">
                    <Rectangle.Fill>
```

```
                    <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                        <GradientStop Color="#33FFFFFF"/>
                        <GradientStop Color="#00FFFFFF" Offset="1"/>
                        <GradientStop Color="#66DEDEDE" Offset="0.5"/>
                        <GradientStop Color="#00D8D8D8" Offset="0.5"/>
                        <GradientStop Color="#57F8F8F8" Offset="0.0"/>
                    </LinearGradientBrush>
                    </Rectangle.Fill>
                </Rectangle>
                <StackPanel HorizontalAlignment="Right"
                    VerticalAlignment="Top" Width="200" Height="120"
                    Margin="0,55,10,0">
                    <TextBlock FontSize="12" Text="12/8/1968" />
                    <TextBlock FontSize="12" Text="507 - 20th Ave. E." />
                    <TextBlock FontSize="12" Text="(206) 555-9857" />
                    <TextBlock FontSize="12" Text="Seatle" />
                    <TextBlock FontSize="12" Text="USA" />
                </StackPanel>
            </Grid>
        </Border>
    </DataTemplate>
</UserControl.Resources>
```

After that you only need to specify the ItemTemplate and / or Reflection Template property in the Item object.

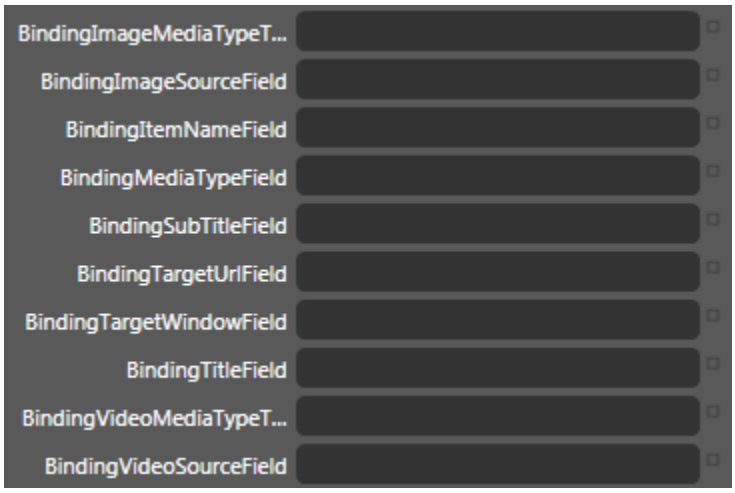
```
<ISNet_Silverlight_WebAqua:WebCoverFlow>
    <ISNet_Silverlight_WebAqua:WebCoverFlow.Items>
        <ISNet_Silverlight_WebAqua:WebCoverFlowItem
            Title="Bridge View 1" SubTitle="Photo Credit © Marcus Claesson"
            ItemTemplate="{StaticResource ItemTemplate}"
            ReflectionTemplate="{StaticResource ReflectionTemplate}"/>
            ……
            ……
            ……

    </ISNet_Silverlight_WebAqua:WebCoverFlow.Items>
</ISNet_Silverlight_WebAqua:WebCoverFlow>
```

Then the WebCoverFlow will automatically use your template object for its content.




### Concept: Data Binding & Template Item

Instead of filling the WebCoverFlow Item collection by creating a WebCoverFlowItem object and filling the info to it, you can provide or reuse a generic collection of your business object to WebCoverFlow and set the data binding member as follows.




This will map your business object member to WebCoverFlowItem member. For example you specify BindingTitleField with Text property in your business object, this will map WebCoverFlowItem's Title property with your business object's Text property.

Another thing that you should pay attention is that when a data binding occurred. The item data context is passed along to the ItemTemplate and / or ReflectionTemplate, therefore you can also retrieved the data from your business object (data context) to your button template as illustrated bellow.

Template Object

```
<UserControl.Resources>
    <DataTemplate x:Key="ReflectionTemplate">
        <Border BorderThickness="3,3,3,3" CornerRadius="5,5,5,5"
            HorizontalAlignment="Stretch" VerticalAlignment="Stretch">
            <Border.BorderBrush>
                <LinearGradientBrush EndPoint="1,1" StartPoint="0,0">
                    <GradientStop Color="#FFFFFFFF"/>
                    <GradientStop Color="#FFFFFFFF" Offset="1"/>
                    <GradientStop Color="#FFF2F2F2" Offset="0.5"/>
                    <GradientStop Color="#FF707070" Offset="0.25"/>
                    <GradientStop Color="#FF474747" Offset="0.75"/>
                </LinearGradientBrush>
            </Border.BorderBrush>
            <Border.Background>
                <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                    <GradientStop Color="#FF080808"/>
                    <GradientStop Color="#FFFFFFFF" Offset="1"/>
                </LinearGradientBrush>
            </Border.Background>
        </Border>
    </DataTemplate>

    <DataTemplate x:Key="ItemTemplate">
        <Border BorderThickness="3,3,3,3" CornerRadius="5,5,5,5" Width="400"
            Height="200" VerticalAlignment="Stretch">
            <Border.BorderBrush>
                <LinearGradientBrush EndPoint="1,1" StartPoint="0.0,0.0">
                    <GradientStop Color="#FFFFFFFF"/>
                    <GradientStop Color="#FFFFFFFF" Offset="1"/>
                    <GradientStop Color="#FFF2F2F2" Offset="0.5"/>
                    <GradientStop Color="#FF707070" Offset="0.25"/>
                    <GradientStop Color="#FF474747" Offset="0.75"/>
                </LinearGradientBrush>
            </Border.BorderBrush>
            <Border.Background>
                <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                    <GradientStop Color="#FF080808"/>
                    <GradientStop Color="#FFFFFFFF" Offset="1"/>
                </LinearGradientBrush>
            </Border.Background>
            <Grid>
                <Image HorizontalAlignment="Left" Margin="25,55,0,20"
                    VerticalAlignment="Stretch"
                    RenderTransformOrigin="0.5,0.5" Source="{Binding Photo}">
                <TextBlock HorizontalAlignment="Stretch"
                    VerticalAlignment="Top" Text="{Binding FirstName}"
                    Margin="25,10,0,0" FontSize="18"/>
                <Rectangle HorizontalAlignment="Stretch"
                    VerticalAlignment="Stretch" Margin="-3,-3,-3,-3">
                    <Rectangle.Fill>
                    <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                        <GradientStop Color="#33FFFFFF"/>
                        <GradientStop Color="#00FFFFFF" Offset="1"/>
                        <GradientStop Color="#66DEDEDE" Offset="0.5"/>
                        <GradientStop Color="#00D8D8D8" Offset="0.5"/>
                        <GradientStop Color="#57F8F8F8" Offset="0.0"/>
                    </LinearGradientBrush>
```

```
                    </Rectangle.Fill>
                </Rectangle>
                <StackPanel HorizontalAlignment="Right"
                    VerticalAlignment="Top" Width="200" Height="120"
                    Margin="0,55,10,0">
                    <TextBlock FontSize="12" Text="{Binding BirthDate}" />
                    <TextBlock FontSize="12" Text="{Binding Address}" />
                    <TextBlock FontSize="12" Text="{Binding HomePhone}" />
                    <TextBlock FontSize="12" Text="{Binding City}" />
                    <TextBlock FontSize="12" Text="{Binding Country}" />
                </StackPanel>
            </Grid>
        </Border>
    </DataTemplate>
</UserControl.Resources>
```

As you can see now we're going to bind Photo, FirstName, BirthDate, Address, HomePhone, City and Contry properties from our business object to this template object.

Data Binding (cs Code)

```
_coverFlow.ItemsSource = BindingManager.GetEmployees();
```

To perform the data bind what you need to do is to fill the ItemsSource property with a generic collection. BindingManager.GetEmployees() returns a generic collection of type employee.

```
    public class Employee
    {
        public int EmployeeID { get; set; }
        public string LastName { get; set; }
        public string FirstName { get; set; }
        public string Title { get; set; }
        public string TitleOfCourtesy { get; set; }
        public DateTime BirthDate { get; set; }
        public DateTime HireDate { get; set; }
        public string Address { get; set; }
        public string City { get; set; }
        public string Region { get; set; }
        public string PostalCode { get; set; }
        public string Country { get; set; }
        public string HomePhone { get; set; }
        public string Extension { get; set; }
        public string Notes { get; set; }
        public string Photo { get; set; }
    }
```

To attach a template object you can use the ItemDataBind event and attached the template object there.

```
private void WebCoverFlow_ItemDataBind(object sender,
ISNet.Silverlight.WebAqua.WebCoverFlowItemEventArgs e)
{
    e.Item.ItemTemplate = this.Resources["ItemTemplate"] as DataTemplate;
    e.Item.ReflectionTemplate = this.Resources["ReflectionTemplate"] as
DataTemplate;
```

}

Then when it's completed you'll have something like this.

### Concept: Virtual Flow™

WebCoverFlow.NET comes with an innovative technology called Virtual Flow™. In this mode, the WebCoverFlow won't create elements for all the items. Instead it will create certain number element, which determined by PageSize property, and re-use it for the flow.

The total number element creates is PageSize x 3 and it is advised-able to use an odd number.

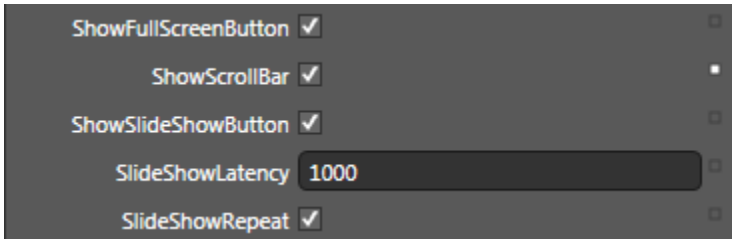
When using this mode, initially all the picture will shows as default picture, then when the download is finished, the image will be switch (fading off – fading in) with the original one. The PageSize play important part how many image will be downloaded at one and how many will be keep. The larger the PageSize, the more image will be downloaded at the same time, so it's up to the developer to determine which size is the most optimum one.

To enable Virtual Flow™, you only need to enable load on demand from property window


VirtualFlow™ is primarily useful in advanced scenarios or enterprise Web applications that require high scalability. For instance, an enterprise Web application that displays video-based knowledge base could get the total number of items growing significantly as the business growth.

### Concept: Slide Show

WebCoverFlow.NET can also act as a SlideShow control, there's a slide show player on the bottom left position which can be turn hide / show.

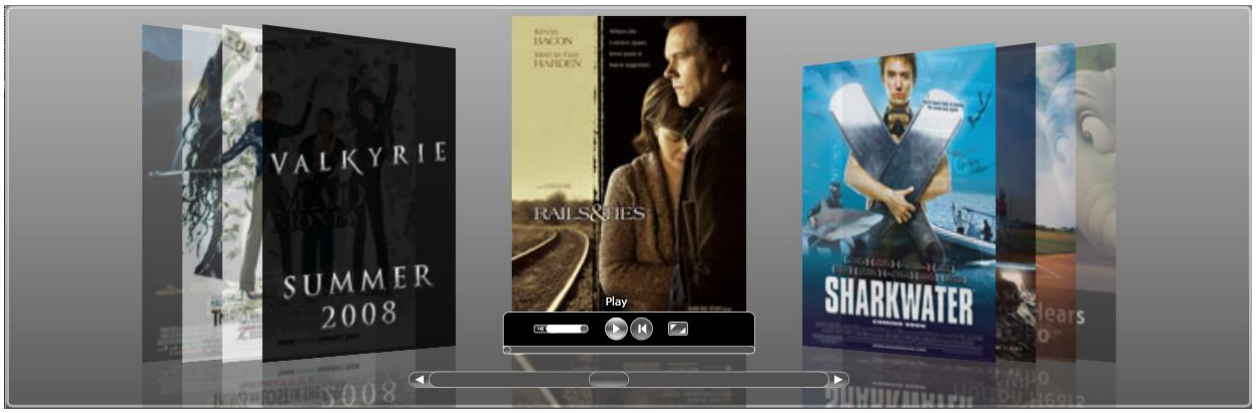



You can set the SlideShowButton to false if you want to hide it, and vice versa.

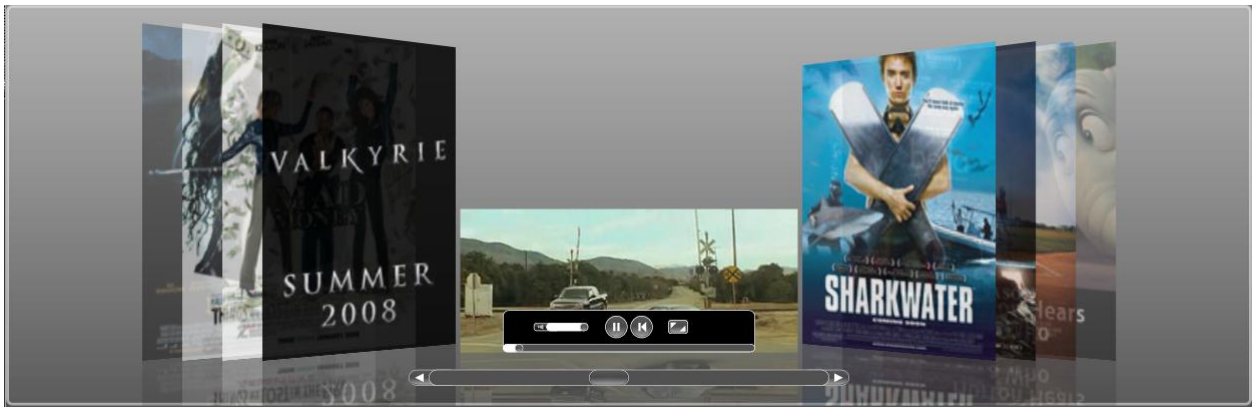
SlidShowLatency indicates the timeout between slide, and SlideShowRepeat indicates whether the slideshow back to start point when its already reach the end.

### Concept: Video Player

WebCoverFlow.NET can also be used as Video Player. When you create a CoverFlowItem with MediaType of Video and a VideoSource is set to a valid url, the WebCoverFlow will give you a video player panel which can be showed when you hover the focused CoverFlowItem.




As you can see the CoverFlow will use the ImageSource specified at first before the video is played. When the video is played it will use the MediaElement as follow.




You can full screen the video by clicking the full screen button, and furthermore you can also full screen the CoverFlow for a better viewing by clicking the full screen button at CoverFlow level.