Control Templates 2

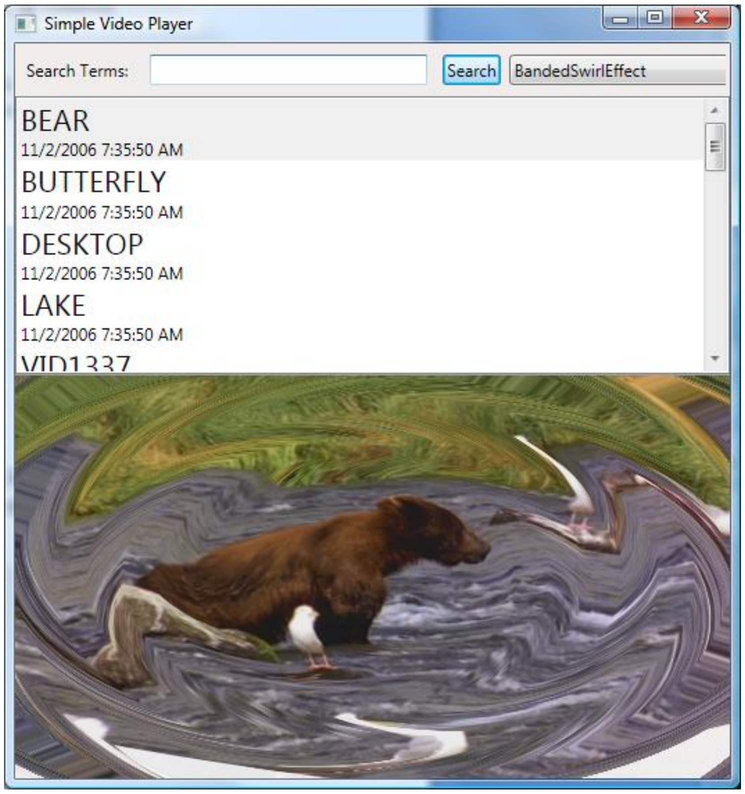
Estimated time for completion: **60 minutes**

Goals:

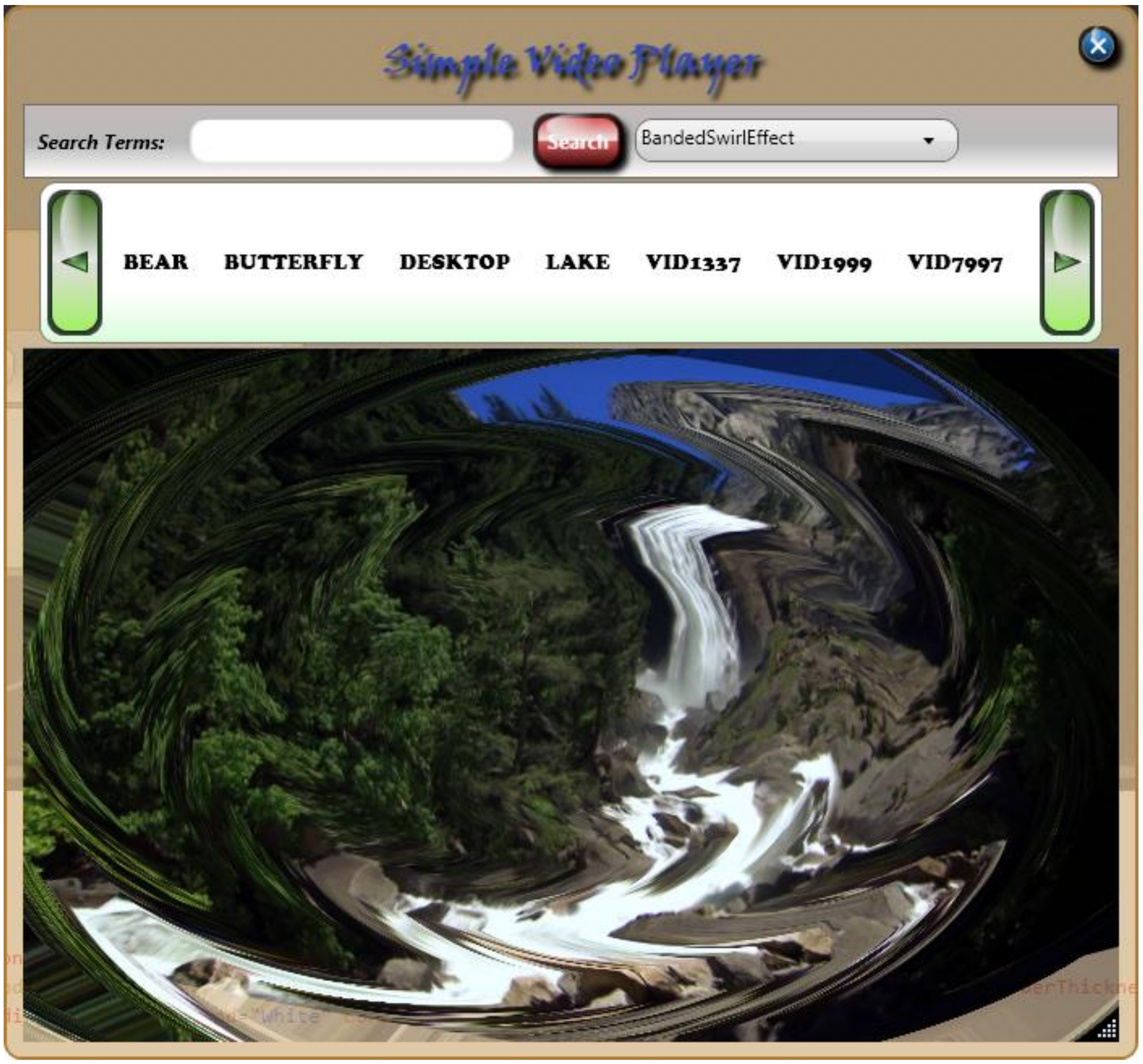
- Replace out a variety of control templates to radically alter a UI without changing any of the procedural code.
- Understand how to integrate with existing template structures so the control behavior remains consistent.
- Replace other visual styles such as panels; focus style, etc. in controls.

Overview:

This lab will continue exploring the control template structures in WPF. You will start with a completely functional application that locates and displays videos on the local machine. The main UI looks like this:



Through the replacement of the control templates in the controls used in this UI you will end up with an application that looks something like this:



No changes will be made in the C# code behind – everything is already in place there. Instead, you will spend your time in Blend, modifying and creating control templates.

This lab is really intended to give you some hands-on experience manipulating a variety of templates using Blend, and as a result is very lengthy. It's very likely you will not complete it in the time allotted so plan to complete it post class sometime.

Also, you do not have to follow the instructions step-by-step. Feel free to deviate, or even to experiment. The point is to get the experience, not to necessarily end up with exactly what is shown (after all, that's what the **after** project is for).

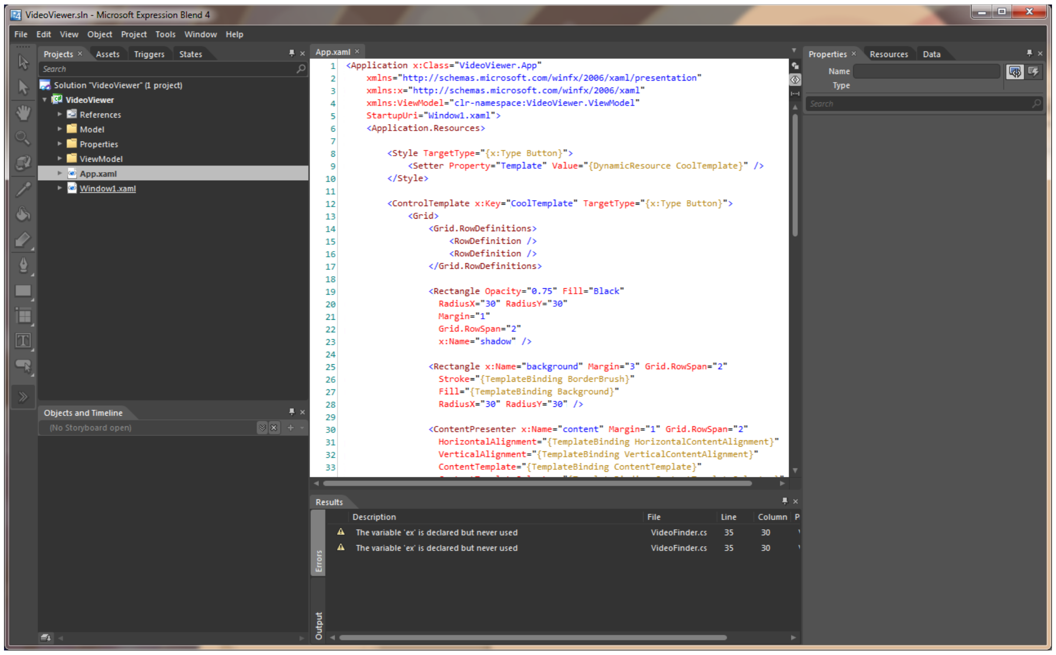
The lab instructions and existing templates assume the Aero theme – if you are using some other theme the template may not be quite as described. You can either force the aero theme (remember how to do that?) or try to work with the other template – it should be close.

Part 1 – Styling the buttons with focus and accelerator support

In this part, you will bring over the button Control Template you created in the first templates lab and add it to this project. As part of that process we will fix a couple of issues with the template.

Steps:

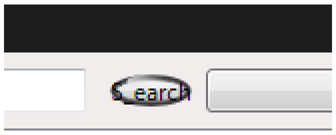
1. Open the starter project located in the before folder associated with the lab using Expression Blend. Note there are different solution folders for Blend 3 vs. Blend 4 – they do not use the same exact project format so open the appropriate starter project based on the version of Blend you are using.
2. Run the application (F5) if you'd like to see the starting picture. At a minimum, build the project so the code has been compiled.
    a. Click the search button with no text to see all videos, or type a portion of video name to filter to just those videos.
    b. Select a pixel-shader effect using the combo box and select a video to see it applied.
3. Using a text editor, open the app.xaml where you placed the CoolButton template and style generated in the previous lab. Copy it to the clipboard.
4. Open the app.xaml file in your new project – it is on the projects tab. Select the XAML view (small icon just above the vertical scrollbar). Go ahead and paste the button template. Since it is a default style it should apply to the search button automatically.


VideoViewer.sln - Microsoft Expression Blend 4
File Edit View Object Project Tools Window Help
Projects Assets Triggers States
Search
Solution 'VideoViewer' (1 project)
VideoViewer
References
Model
Properties
ViewModel
App.xaml
Window1.xaml
Objects and Timeline
(No Storyboard open)
App.xaml
Properties Resources Data
Name
Type
Search
Results
Errors
Output

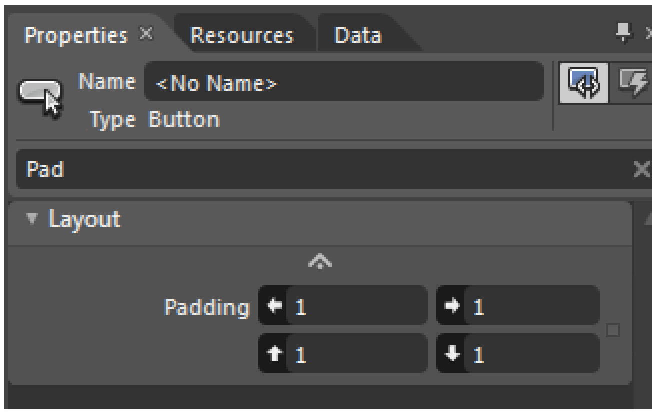
```
<Application x:Class="VideoViewer.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:ViewModel="clr-namespace:VideoViewer.ViewModel"
    StartupUri="Window1.xaml">
    <Application.Resources>

        <Style TargetType="{x:Type Button}">
            <Setter Property="Template" Value="{DynamicResource CoolTemplate}" />
        </Style>

        <ControlTemplate x:Key="CoolTemplate" TargetType="{x:Type Button}">
            <Grid>
                <Grid.RowDefinitions>
                    <RowDefinition />
                    <RowDefinition />
                </Grid.RowDefinitions>

                <Rectangle Opacity="0.75" Fill="Black"
                  RadiusX="30" RadiusY="30"
                  Margin="1"
                  Grid.RowSpan="2"
                  x:Name="shadow" />

                <Rectangle x:Name="background" Margin="3" Grid.RowSpan="2"
                  Stroke="{TemplateBinding BorderBrush}"
                  Fill="{TemplateBinding Background}"
                  RadiusX="30" RadiusY="30" />

                <ContentPresenter x:Name="content" Margin="1" Grid.RowSpan="2"
                  HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"
                  VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
                  ContentTemplate="{TemplateBinding ContentTemplate}"
```

```
Description                                   File             Line   Column
The variable 'ex' is declared but never used  VideoFinder.cs   35     30
The variable 'ex' is declared but never used  VideoFinder.cs   35     30
```

5. Switch back to the window – look at the search button:



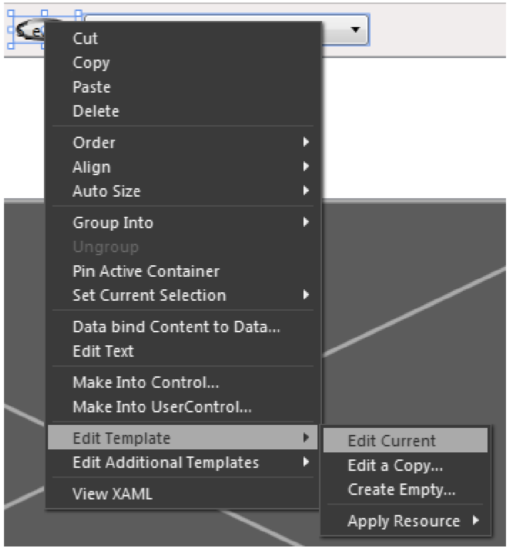
6. Notice how the text is running off the edge? Try applying additional **Padding** to the button to give it some more space. (**Padding** is in the additional section under Layout in the property explorer – you can just type Pad into the search box to find it).



   a. Does that fix the problem? Can you explain why?

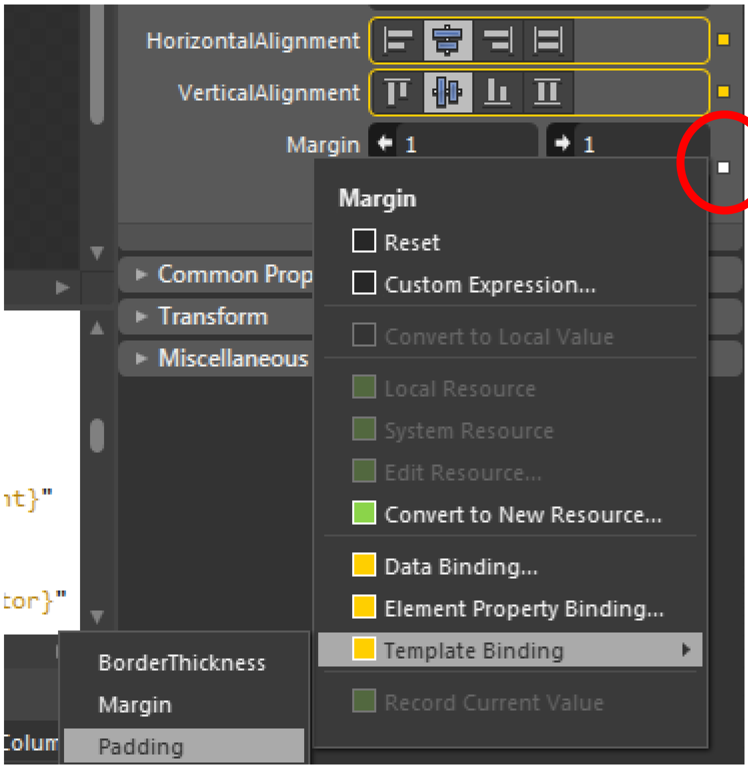
7. The issue is the control template is not using the padding property – so let's change the template to fix that.

   a. Right click on the button and select "Edit Template | Edit Current".

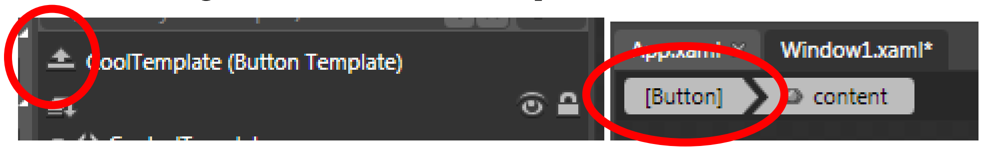


   b. It should open the template in the designer where only the button is visible. Click on the content element in the objects and timeline.

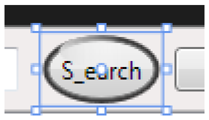
   c. Notice that the margin is hard-coded to be "1" on all edges. Template bind that to "Padding" by clicking the small circle next to the Margin property and selecting "Template Binding" | "Padding".



8. Back out of the template by either clicking the “Up” button in the objects and timeline, or double-clicking the breadcrumb at the top of the screen.

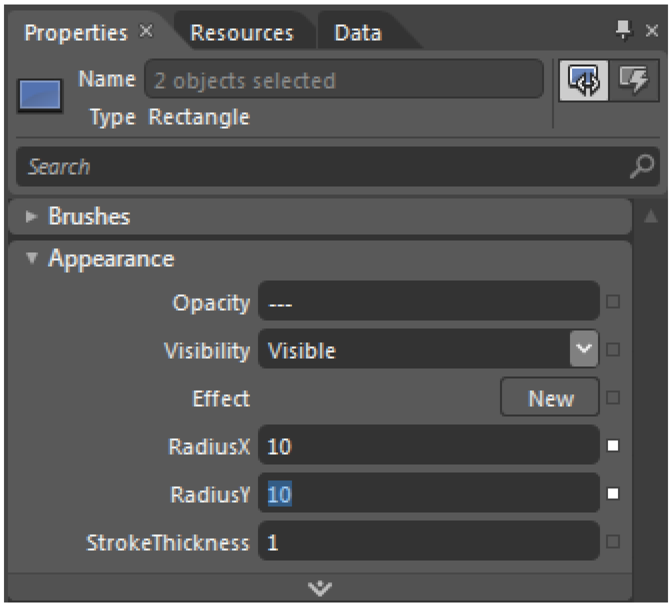


9. Notice now the button looks much better, although it’s a bit too round and there’s an issue with the accelerator:



10. Go back to the template –

    a. Change the **RadiusX** and **RadiusY** properties for the shadow and background to be 10 instead of 30 to give it a more cylindrical shape.



Properties × Resources Data

Name 2 objects selected

Type Rectangle

Search

Opacity ---

Visibility Visible

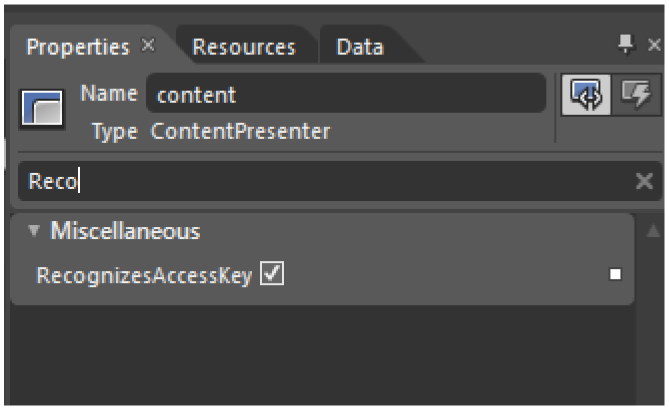
Effect New

RadiusX 10

RadiusY 10

StrokeThickness 1

b. Next, find the **RecognizesAccessKey** property on the **ContentPresenter** – this property is what tells the presenter to support the "_" accelerator key display. Go ahead and check it so that we get our accelerator key support.



Properties × Resources Data

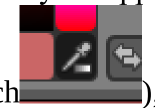
Name content

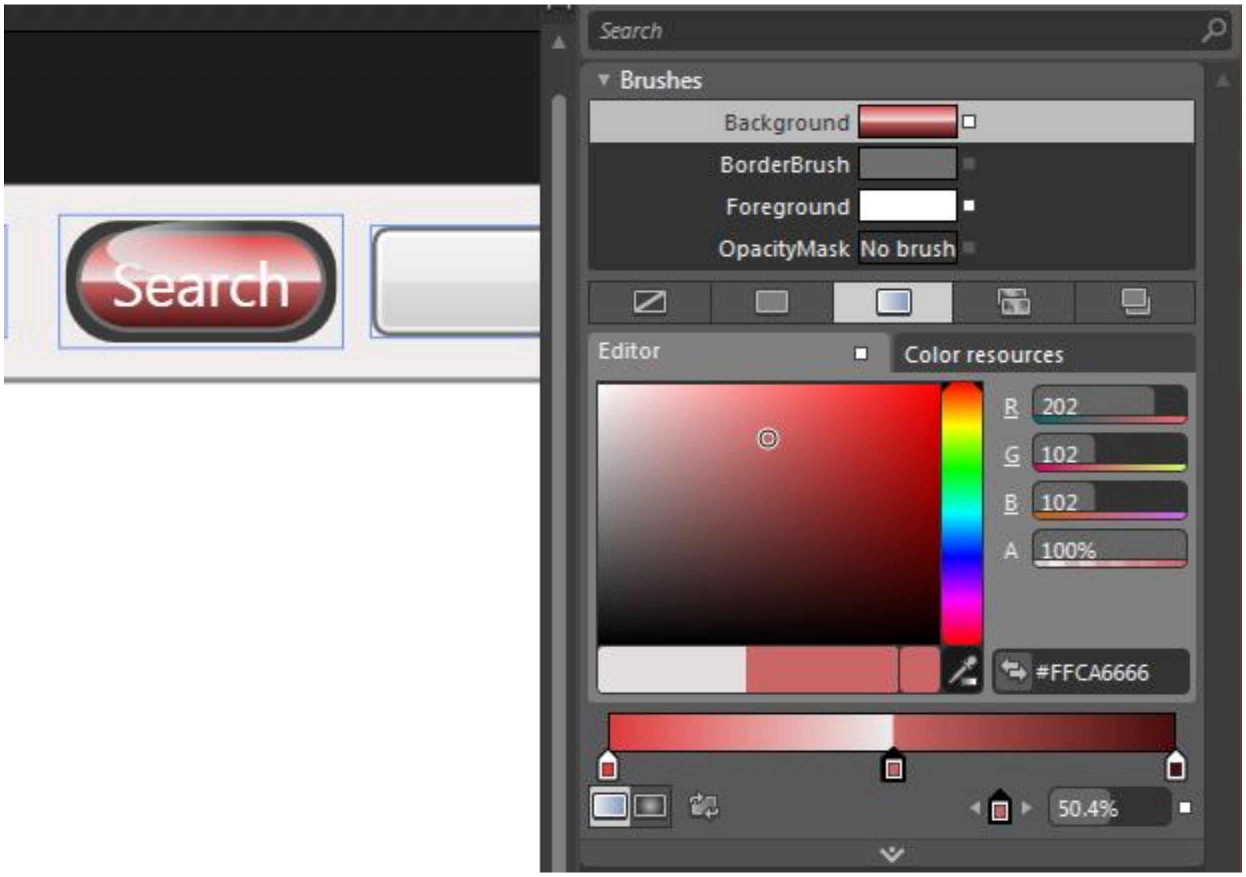
Type ContentPresenter

Reco

RecognizesAccessKey ✓

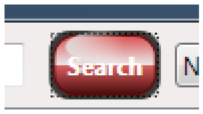
11. Switch back to the main display – the button should look much better now. Change the background color to be some gradient you like. Adjust the foreground to make the text legible given your background. Here's a red with white combination – pick anything you like.

You can use the eyedropper tool to "steal" colors from the below screen shot if you like. It has 4 gradient stops – the middle two are overlapping to create the hard line in the center. To use the eye dropper, click on it (it's the icon at the corner of the brushes color swatch ), and then move the cursor to the color you want in whatever window you like and click on it.



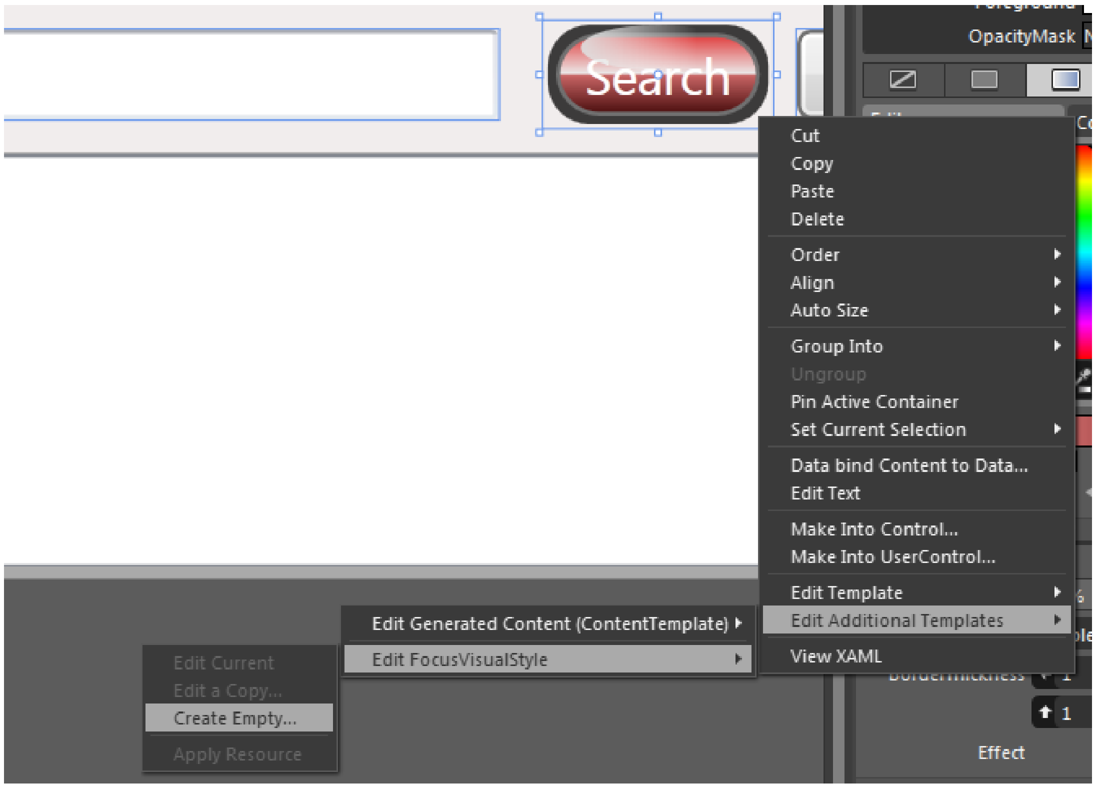
Background: gradient brush
BorderBrush
Foreground
OpacityMask: No brush
R 202
G 102
B 102
A 100%
#FFCA6666

12. Run the application and tab to the button. Notice how the current focus style is a square around the edge?

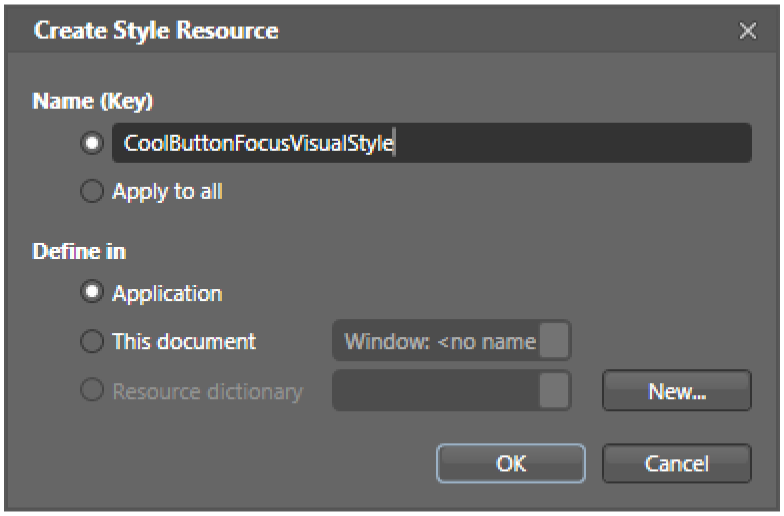


13. This may be fine in some cases, but we'd like to have it be a little more evident and to follow the actual shape of the control more definitively. Close the application and switch back to Blend.

14. Right-click on the button and select "Edit Additional Templates" | "Edit FocusVisualStyle" | "Create Empty".



15. Define the style in the application resources and name it CoolButtonFocusVisualStyle:

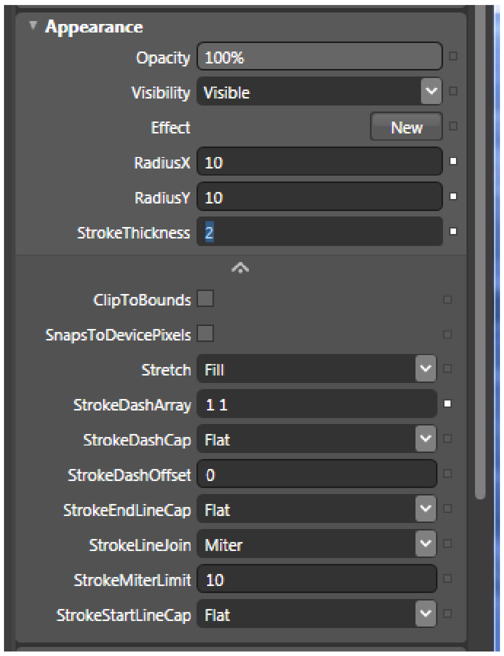

Create Style Resource

Name (Key)
(●) CoolButtonFocusVisualStyle
( ) Apply to all

Define in
(●) Application
( ) This document   Window: <no name
( ) Resource dictionary   New...

16. Add a **Rectangle** to the **Grid**. On the rectangle

   a. Remove the **Fill** brush
   b. Change the **Stroke** to be template bound to the Foreground brush
   c. Change the **Stroke** thickness to be “2” and the **RadiusX** and **RadiusY** to be “10”
   d. Set the **StrokeDashArray** (in Advanced Appearance properties) to be “1 1”.

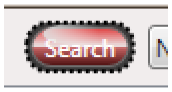


Appearance

Opacity	100%
Visibility	Visible
Effect	New
RadiusX	10
RadiusY	10
StrokeThickness	2
ClipToBounds	☐
SnapsToDevicePixels	☐
Stretch	Fill
StrokeDashArray	1 1
StrokeDashCap	Flat
StrokeDashOffset	0
StrokeEndLineCap	Flat
StrokeLineJoin	Miter
StrokeMiterLimit	10
StrokeStartLineCap	Flat

Note: in some cases, Blend might not put you into the proper editing mode when you create the focus style – this appears to be a bug in Blend. If this happens, you can hand-edit the App.xaml file and do the above changes to the CoolButtonFocusVisualStyle it created.

17. Exit out of the template and run the application to see the new focus effect.

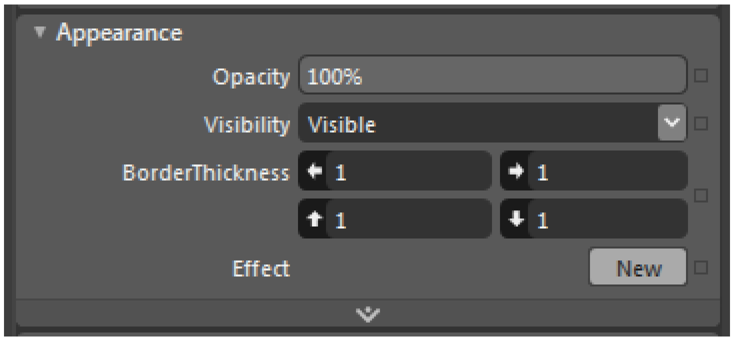


18. If you look in the window XAML you will find that it applied the focus template directly to the button. Since we have a default style, this may not be what you desire. You can cut it out of the button tag and put it into the default style if you like.

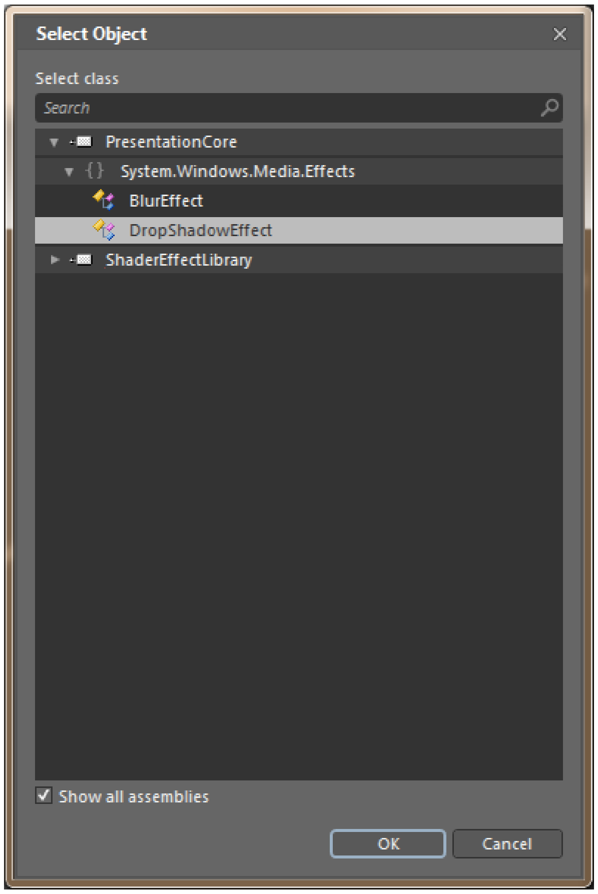
```
<Application x:Class="VideoViewer.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:ViewModel="clr-namespace:VideoViewer.ViewModel"
    StartupUri="Window1.xaml">
    <Application.Resources>

        <Style TargetType="{x:Type Button}">
            <Setter Property="Template" Value="{DynamicResource CoolTemplate}" />
            <Setter Property="FocusVisualStyle" Value="{DynamicResource CoolButtonFocusVisualStyle}" />
        </Style>
```

19. As a last step, apply a drop shadow effect to the button.

    a. Find the **Effect** property in the Appearance section and click the "New" button.



b. In the resulting dialog, select **DropShadowEffect** to add it to the button.

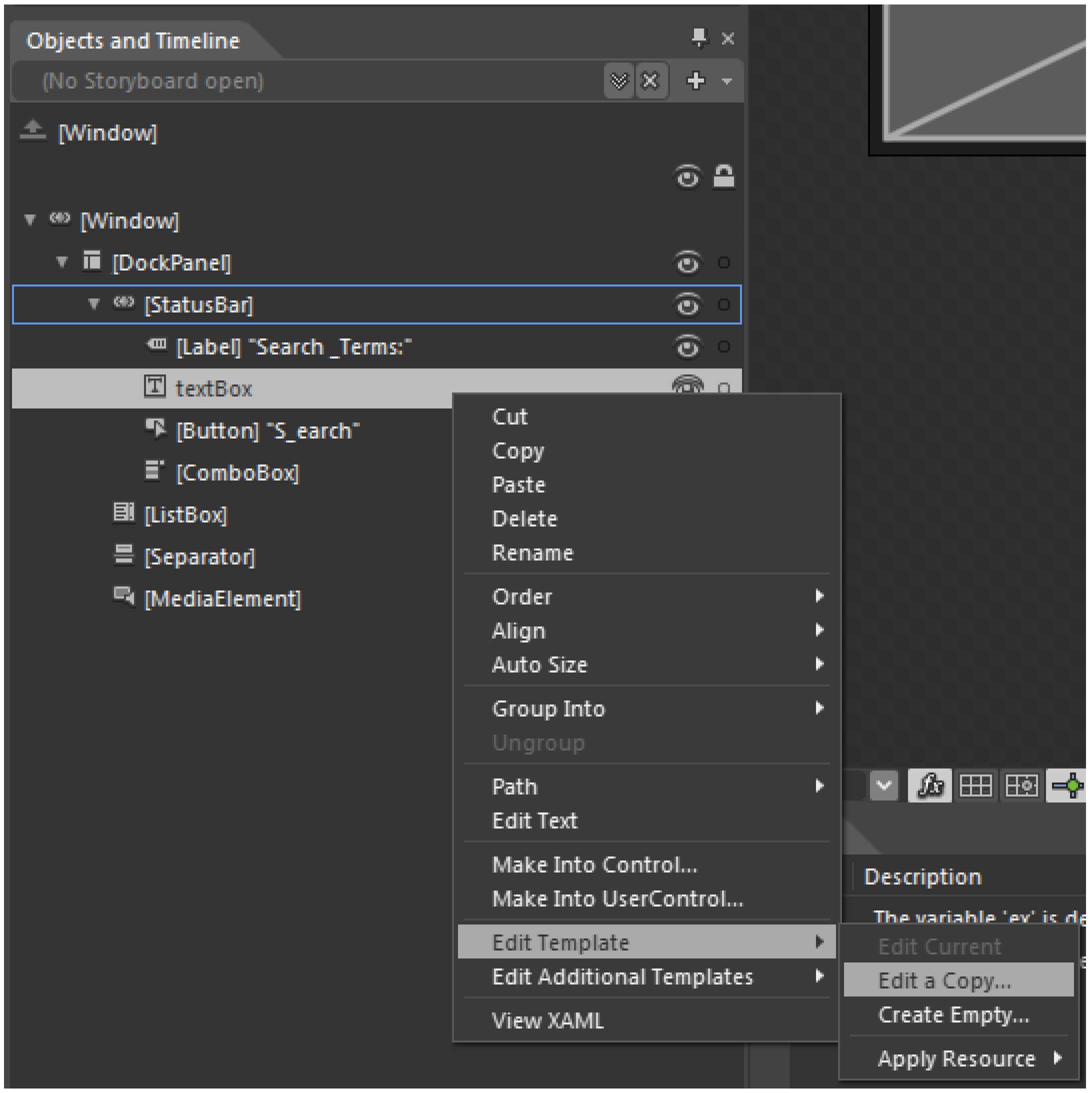


Part 2 – Styling the TextBox

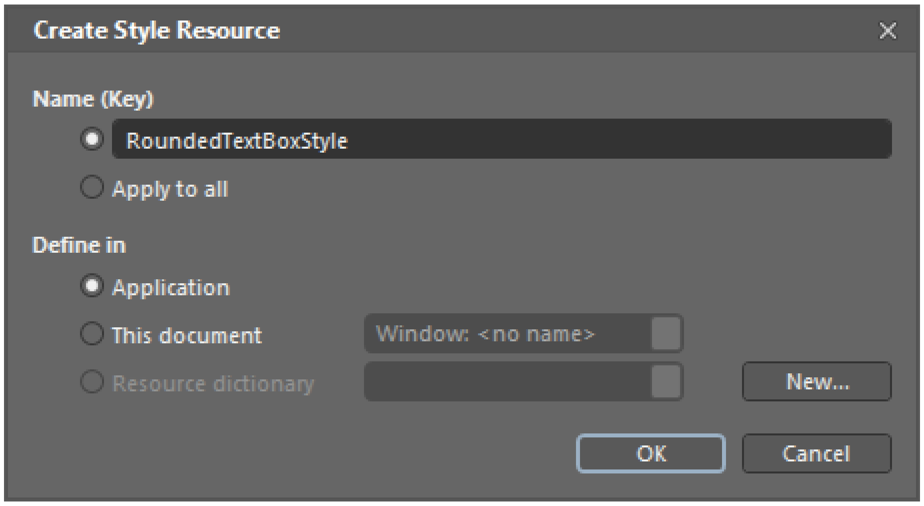
In this part, you will create a new control template for the TextBox which supports a more "rounded" look.

Steps:

1. Select the TextBox in the designer or Objects and Timeline pane and edit a copy of the control template (right-click, "Edit Template" | "Edit a Copy").



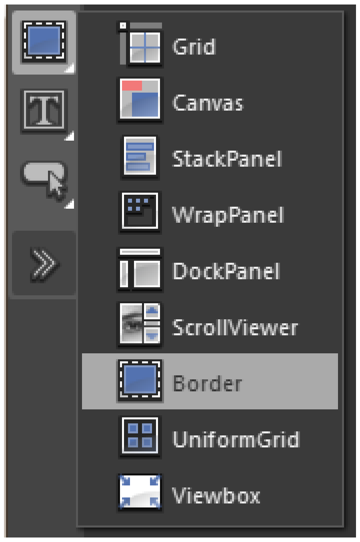
2. Place it into the application resources and name it "RoundedTextBoxStyle".


Create Style Resource

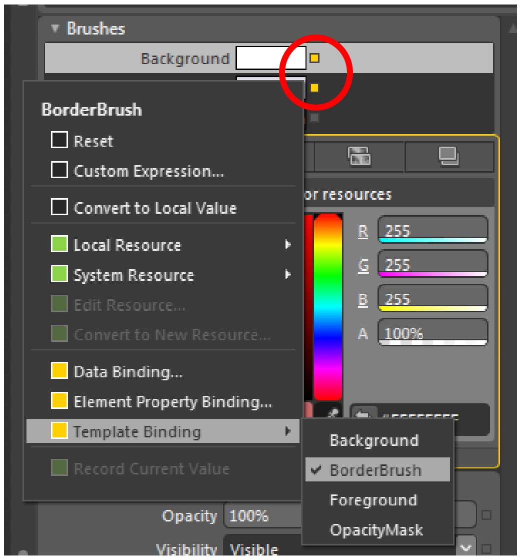
Name (Key)
- [✓] RoundedTextBoxStyle
- Apply to all

Define in
- [✓] Application
- This document Window: <no name>
- Resource dictionary

3. Examine the control template – note how it is composed of a border (a **ListBoxChrome** control) and a **ScrollViewer** named "PART_ContentHost". What we are going to do is replace the **ListboxChrome** with a regular border – leaving the **ScrollViewer** in place.
4. Click on the PART_ContentHost to select it and cut it to the clipboard.
5. Next, delete the **ListBoxChrome** object and replace it with a regular **Border** control (this can be found asset toolbox, or click and hold on the panel section of the toolbar).



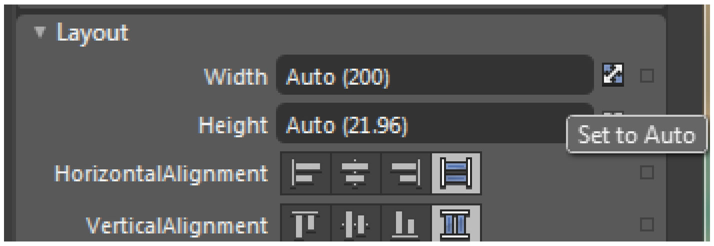
6. Set the border properties so we get the control values:
   a. Bind the **Background**, **Border** and **BorderThickness** to the same values on the TextBox using a Template Binding.



Click the Advanced Properties button next to the property and go down to the Template Binding menu item to see available choices.

   b. Change the Corner Radius for the border to be 10 to give it a rounded look.
   c. Change the **Width** and **Height** to be Auto by clicking the "Auto" button next to the properties. (Hover over the elements next to the Width / Height properties if you aren't sure which one is the auto button – the tooltip will show you.) Note:

this may not be necessary on one or both properties depending on the template – if it's already set to Auto you can skip this step.



7. Paste the **PART_ContentHost** into the border (select the border and Edit | Paste).
8. Run the application – notice how close the blinking cursor is to the edge of the border? Go and fix that in the template by applying a Margin to the content host. 4 pixels should do the trick.
9. Run the application again and see if it still works.

Part 3 – Styling the ComboBox

In this part, you will create a new control template for the ComboBox which supports a more "rounded" look similar to the TextBox we just did.

Steps:

1. Select the ComboBox in the designer and edit a copy of the control template (right-click, "Edit Template" | "Edit a Copy").
   a. Place it into the application resources and name it "RoundedComboBoxStyle".
2. Examine the control template – note how it is composed of a **Grid** holder (MainGrid), a **Popup** (PART_Popup), a **ToggleButton** and **ContentPresenter**.
   a. We need to leave the popup in place – this is the drop down portion of the **ComboBox**. What we want to replace is the **ToggleButton** – this is what is displaying the UI when the popup is in the closed state. The **ContentPresenter** is then used to display the currently selected item – you can see it is template bound to drive this.
3. Select the **ToggleButton** and edit the control template for that element.
   a. We want to replace the "Chrome" element here – it's currently a **ButtonChrome** visual which doesn't support rounded corners, so we'll replace it with a Border like we did before.
4. Click on the **Grid** to select it and cut it to the clipboard.
5. Next, delete the **ButtonChrome** object and replace it with a regular Border control (this is generally in the asset toolbox).
6. Set the **Border** properties appropriately
   a. Template bind the **Background**, **Border** and **BorderThickness**
   b. Set the Corner Radius property to be "30" to give it a rounded edge.

c. Change the **Width** and **Height** to be Auto by clicking the "Auto" button next to the properties like you did with the TextBox.

d. This time, to adjust for the content, change the **Padding** of the border (we could have done that before as well) – set it to "10" to give enough space and make it about the same size as the **TextBox** is.

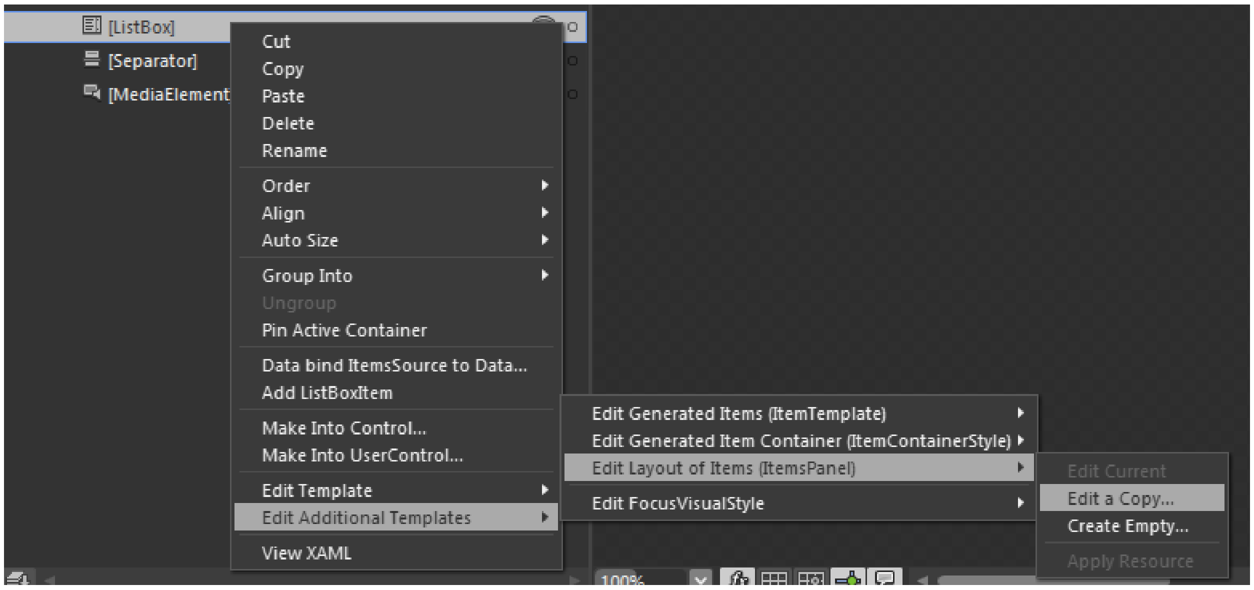
7. Paste the Grid into the border (select the border and Edit | Paste).
8. Run the application again and see if it still works.

Part 4 – Styling the ListBox

In this part, you will do a complete re-styling of the ListBox – formatting it so it displays elements left to right instead of top-bottom and changing the selection style as well as how it scrolls.

Steps:

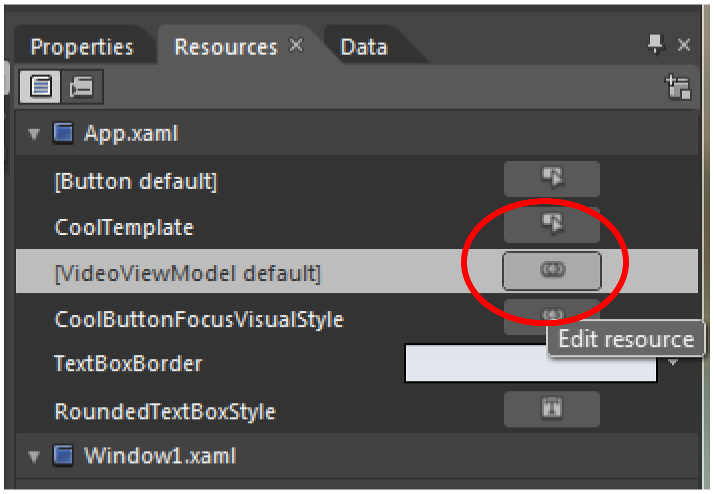
1. First try setting a nice background color on the ListBox – use a subtle gradient shade moving from light to dark going down.
2. Next, let's work with the panel – remember that all ItemsControls have an ItemsPanel which decides how elements are laid out within the control. To edit this, select the ListBox in the Objects and Timelines view and right-click on it. Select "Edit Additional Templates" | "Edit Layout of Items (ItemsPanel)" | "Edit a Copy".



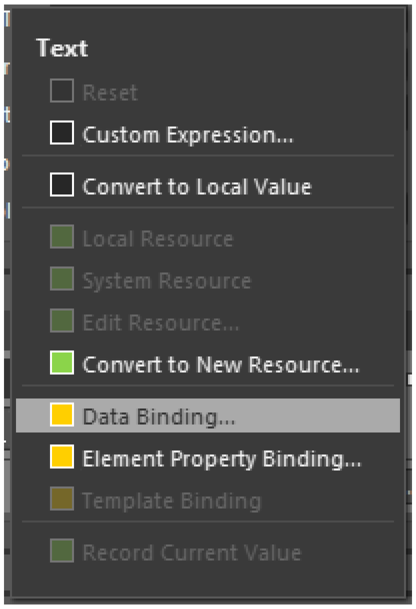
3. Place it into the application resources and name it "HorizontalWrappingList".
4. The design surface should show the default which is a VirtualizingStackPanel. Change the properties on the panel to be Horizontal orientation instead of Vertical.
5. Run the application – notice that now the items run left to right – still using the default data template.
6. The data template being applied has no resource key so it gets applied wherever a VideoViewModel is presented. Let's edit that so it has a different visual appearance.

   a. Go to the “Resources” tab – it’s located next to Properties and Projects tabs.
   b. Expand the “App.xaml” section and look for the DataTemplate – it should be at the top. It has no name because a developer created it in Visual Studio. Feel free to name it something if you like by double-clicking on the name. If you do this, it applies a resource key and breaks the default linkage to the ListBox. To re-establish that, right click on the ListBox and select “Edit Other Templates” | “Edit Generated Items (ItemTemplate)” | “Apply Resource” and select your name.

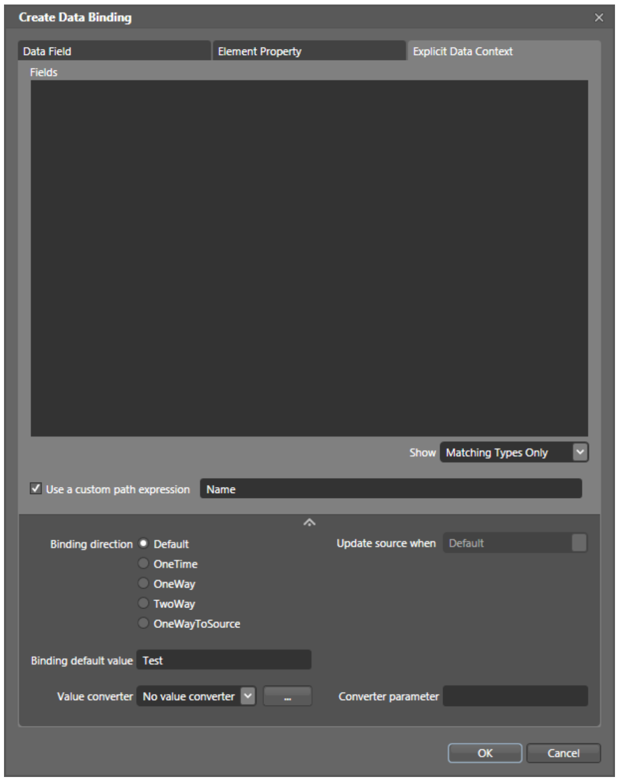
7. To edit the DataTemplate resource, click the button next to the name. This switches you to the specific resource.



8. It’s currently just two TextBlocks in a StackPanel. We want to replace it with a single TextBlock (really the first one) so delete the second TextBlock and then drag the first TextBlock on top of the DataTemplate icon to replace the stack panel with it.
9. Change the TextBlock properties to give it a nice font (Cooper Black looks nice), set a font size of 10pt, a MaxWidth of “100” and turn on Text wrapping. Go ahead and put a 10 pixel margin around it to separate it from other elements.
10. You may need to re-establish the binding on the TextBlock – look at the Text property in the property pane and see if it is surrounded with a yellow rectangle. If not, the binding was removed (this can happen when you add/remove the element out of the tree).
11. To re-establish it, click the advanced property button next to the Text property and select “Data Binding”.



12. In Blend 3 you will need to select the “Explicit Data Context” tab and check the “Use Custom Path Expression” checkbox.
    a. Enter “Name” into the path expression
    b. Expand the Advanced Properties (at the bottom of the window) and set a default binding value of “Test” onto the binding – that way we can see our designer choices.



Show: Matching Types Only

[✓] Use a custom path expression: Name

Binding direction: [● Default] OneTime OneWay TwoWay OneWayToSource

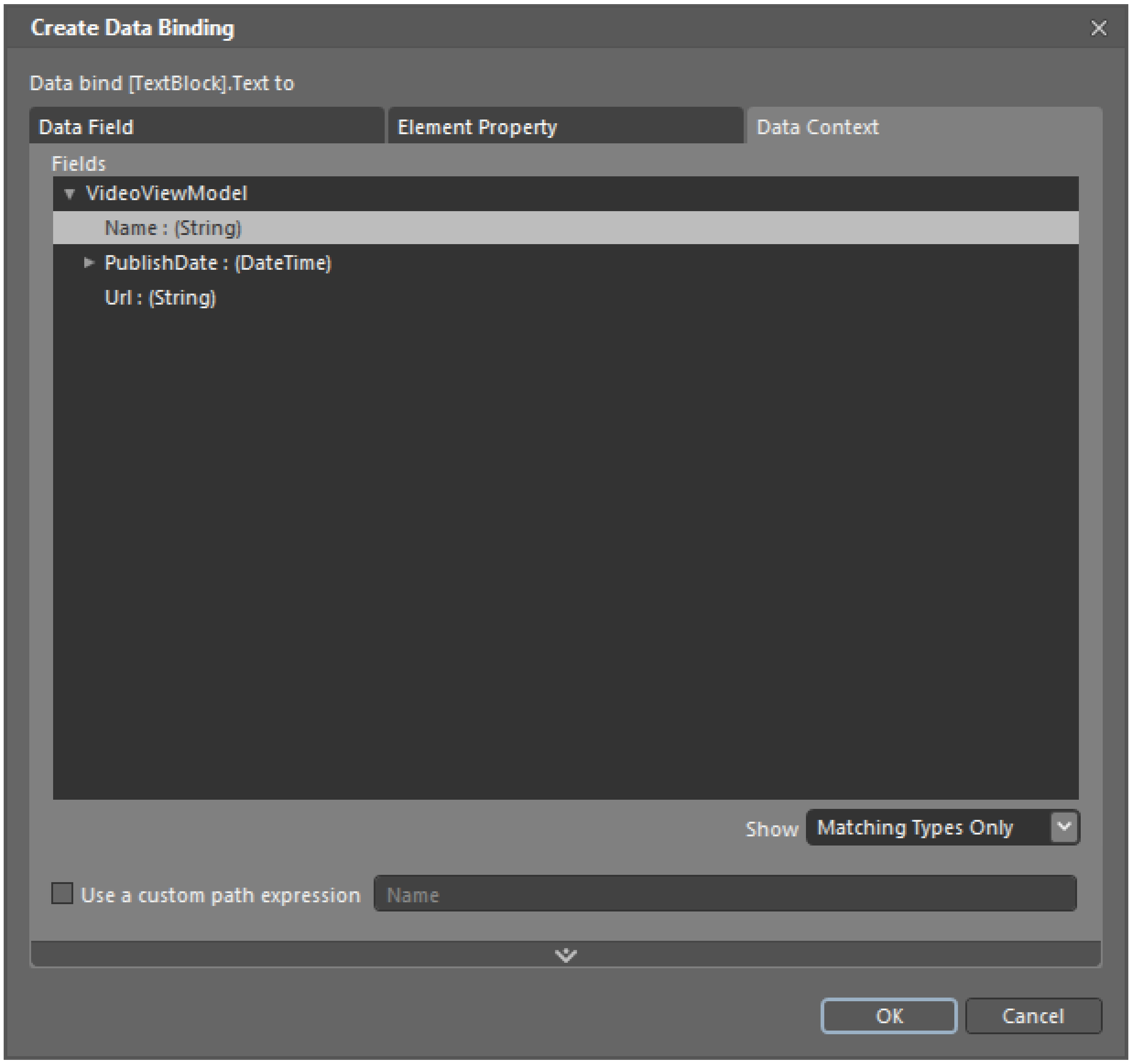
Update source when: Default

Binding default value: Test

Value converter: No value converter

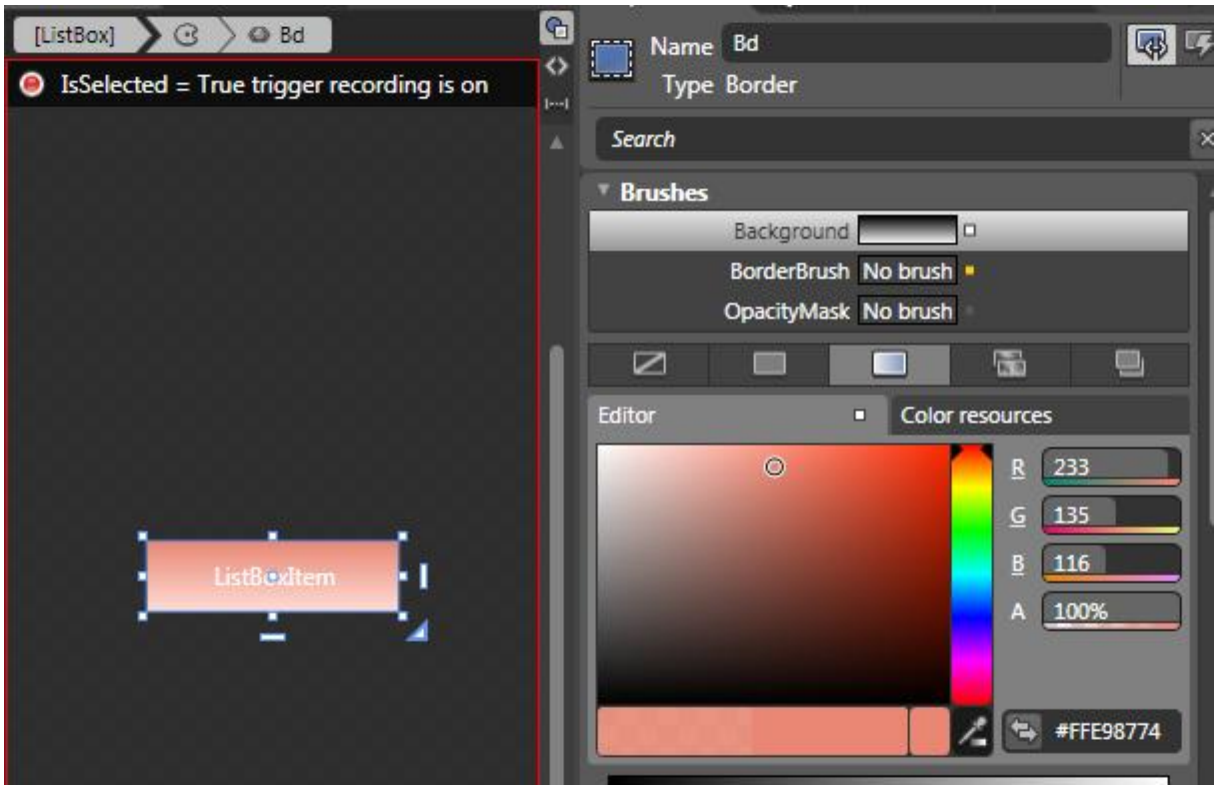
Converter parameter:

13. In Blend 4, it should be smart enough to figure out the actual DataContext type – you should see the data elements on the type and be able to select "Name":



14. If not, follow the same instructions used above for Blend 3.
15. Run the application to see the results of your work.
16. The next step is to change the selection style – this is controlled through the ItemContainerStyle. To edit this style, right-click on the ListBox and select "Edit Additional Templates" | "Edit Generated Item Container (ItemContainerStyle)" | "Edit a copy".
    a. Place it into your application resources and name it "VideoNameContainerStyle".
17. The default is to place a border around a ContentPresenter. Select the border and note the default properties.
    a. It does not provide a background or a border.

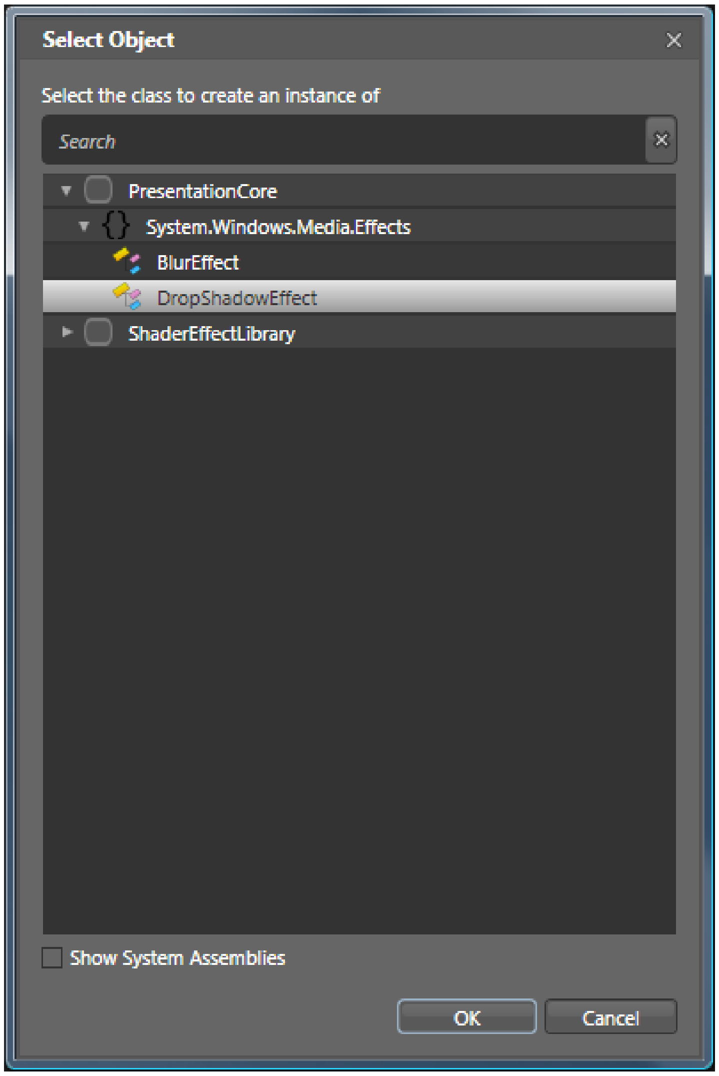
18. Look at the "Triggers" for the border (Select the Triggers tab).
    a. It has a trigger on selection, inactive selection and disabled states.
19. Change the default background color for selection – click on the " IsSelected='true' " trigger and edit the background brush to be a gradient color. You will need to convert the brush style to a local value in order to change it because it is currently picking up the system style – just click the advanced properties next to the brush and select "Convert to Local Value".
20. Use whatever gradient colors you like:



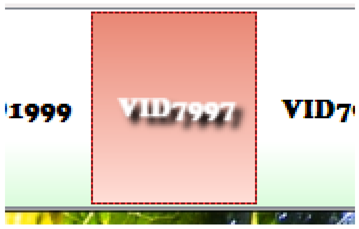
21. Try to do the same thing for the border brush – notice how the tool will not let you select a new brush?
    a. This is because the default brush is already data bound – you will need to remove the binding before you can edit it in the trigger.
22. Turn off the trigger recording (click the red circle at the top of the designer view).
23. Reset the BorderBrush and BorderThickness properties – they should retain their current values (No Brush and 0 respectively) but can now be altered.
24. Turn on trigger recording for the IsSelected state again and now set the border color and border thickness to what you desire.
25. Next, do the same for the inactive/selected state.
26. Run the application to see the results. Click the Search button and note the inactive selection – then shift focus to the entry by clicking on it to see your selected state. In this

instance, the default rectangular focus rectangle looks respectable but you can change it if you like.

27. Let's make the selected state even more obvious – go back to the template and select the ContentPresenter in the template. Activate the "IsSelected" trigger and add a DropShadow effect to the content presenter by clicking the "Effect" button in the property pane. It will display the effect dialog – select DropShadowEffect:

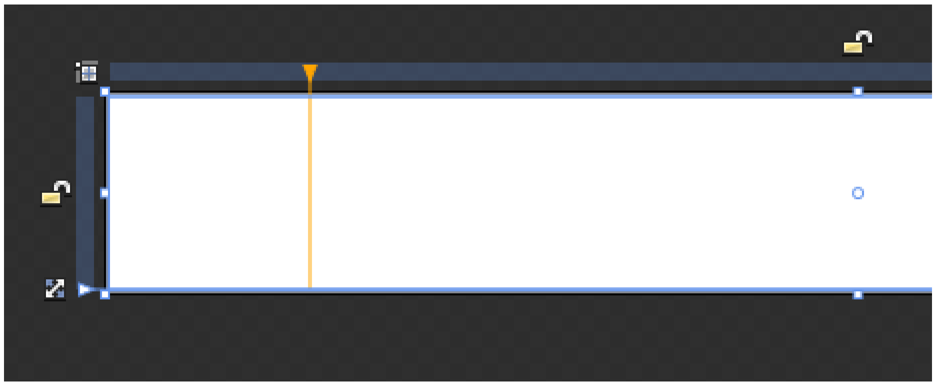


28. Run the application again to see your effect:

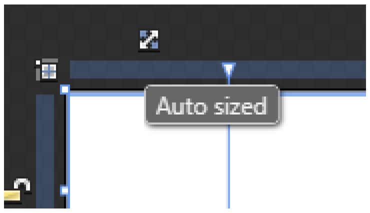


29. The next step is to change the ListBox control template – we want to remove the horizontal scroll bar and replace it with buttons on either end of the listbox.

30. Edit the Control Template for the ListBox. Place the new style into application resources and name it “HorizontalSlidingListBox”.
31. Notice how the default template is composed of a ListBoxChrome border and a ScrollViewer. To change the scrolling visuals we will need to change the Control Template for that scroll viewer.
32. Edit the Control Template for the ScrollViewer (just right click on it and select “Edit Template | Edit a Copy” just like you normally do). Place it alongside the ListBox template in application resources and name it “ButtonScrollViewer”.
33. Examine this template – it is a little complex with a Grid that lays out scroll bars and a scroll presenter. Since the ScrollBars are named, it might be dangerous to remove them, so instead let’s just collapse them to make them still be in the template but not placed into the rendering tree.
    a. Reset the Visibility property on each one (click the advanced menu button and select Reset) – it is currently data bound to turn it on and off based on space. You will need to do this individually for each scroll bar – multi selection will not allow you to reset the property.
    b. Change the property value to be “Collapsed”. Since this is directly set onto the control it will override everything else except a direct setter or animation. Neither of which will happen here.
34. Next, let’s add two new columns to the Grid to place our buttons into.
    a. Select the Grid in the template and move the mouse along the blue border around the visual in the designer. This is the grid placement area – it controls the row/column definitions. Above this should be icons indicating the Height/Width properties (shown as a lock and double-sided arrow below) – if you don’t see these icons, click the little icon on the top left corner of the Grid display to show them.



    b. Add a new column on the left which is auto-sized. It will likely be created as fixed size – just click the icon to switch between star-sizing, auto sizing and fixed size.



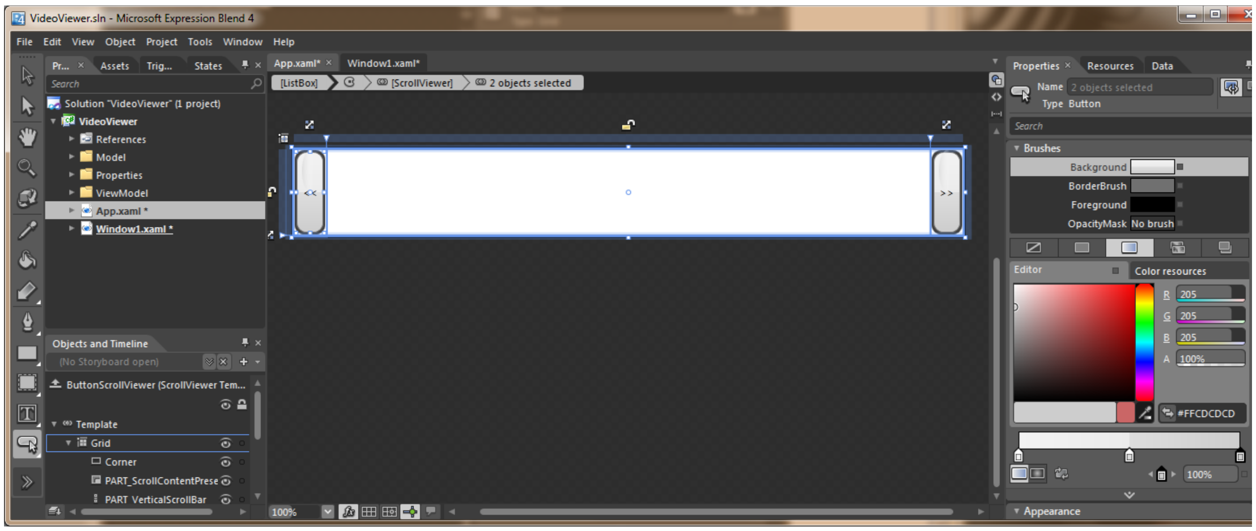
c. Make sure the ContentPresenter is now in the center column which is star-sized. There is already a column on the right (it holds the scrollbar) so we'll just use that.

d. You may need to adjust the MinWidth of the column – set it to zero so it doesn't reserve any specific space (click on the column separator, or edit the ColumnDefinitions property in the XAML view or property explorer).

e. Finally, add two buttons into the display – position them into the left and right column. Set some text onto the button like "<<" and ">>" for now.

   i. Change them to both be auto-sized

   ii. Set the Padding property to have 10 unit padding on the left and right.



35. Run the application and try out the ListBox. Does it work?

36. The buttons are not impacting the scrolling because the ScrollViewer control doesn't know about them and it's hooked into them. We can fix that by telling the button to raise a command when it is clicked – specifically we want the ScrollBar.PageLeftCommand and ScrollBar.PageRightCommand.

37. The Command property is located in the Misc. section of the property explorer – click on the advanced button and select "Custom Expression…". Type in "{x:Static ScrollBar.PageLeftCommand}" and "{x:Static Scrollbar.PageRightCommand}" for the two buttons.

38. Run the application again and try the buttons. They should now scroll the content by a single page.

39. The final step is to clean up the button display – select the LEFT button – make sure the blue selection rectangle is around the button and add a Canvas into the view.
    a. Set the Width/Height to be 16 for the canvas
    b. Draw some path into the canvas – the example will use a triangle. Fill it with some color.
    c. Do the same steps for the other button – you can just copy the visual and apply a flip rotation to it if you want.
    d. Set the background and border brush on the buttons to complementary colors. Try setting the colors on one button and then converting the brushes to resources (click the advanced button and select "Convert to New Resource") and then applying the same resources to the other button.
    e. Set some padding on the buttons to provide enough space to work with.
40. Run the application now to see the results.

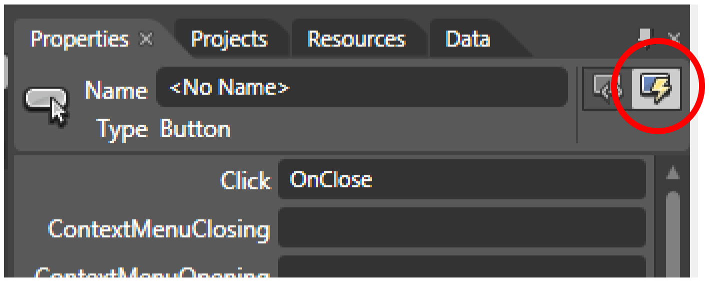
Part 5 – Styling the Window

In this final part we will style the overall Window to give it a transparent appearance, rounded edges and a nicer close + title.

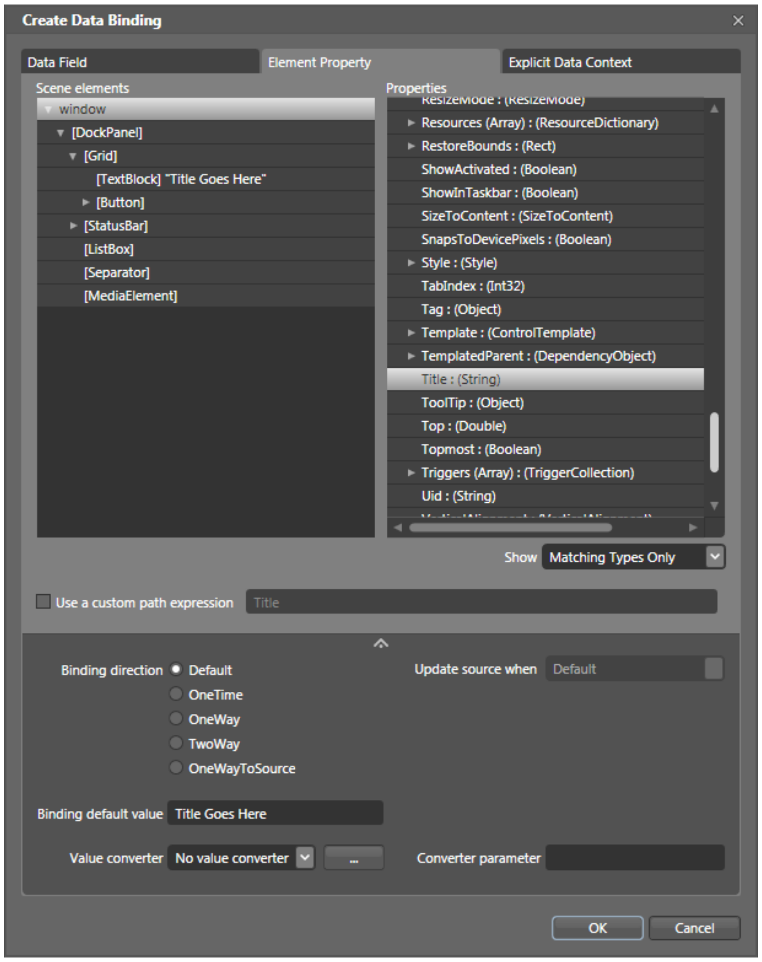
Steps:

1. Edit the control template for the Window. It is a control like any other and has a control template.
2. Select the overall Border – this is what we will style. We could replace the whole template if we want, but this is enough for now.
3. Change the background color to be a partially transparent color. Change the border to be a complementary color with a border thickness of '1". Add some padding and set the corner radius to "10" to make it rounded.
4. Run the application. Notice how we still get the Windows supplied caption? This is what is causing our transparency to not work properly. Change the Window properties to have a **WindowStyle** of "None" and **AllowsTransparency** to be checked.
5. Run the application. Now it should be partially transparent but it cannot be moved, sized or closed! Use the ALT-F4 keystroke to close the window.
6. To support normal windows behavior we will need to add proper adornments into our visuals to support each feature.
7. Let's start with title and close button. Add a new Grid into the visual tree at the very top, docked to the top. Into the **Grid** place a **TextBlock** and **Button**.
8. Place a 10x10 **Canvas** into the button and draw an "X" using the path tool. Fill it with White and set the **Button** to auto-size around it. You might need to set some padding on the button to ensure it looks proper. Horizontally align the button to the right to push it over on the grid. Add a **DropShadow** to make it pop out of the background.

9. We could use a **Command** here to close the window bound to the view model, but a code-behind handler was already setup by the programmer – use the Events tab to bind the **Click** event to the OnClose handler:



10. When you Tab out, it will switch to the code behind in Blend – if it creates the method again, just delete the new implementation.
11. DataBind the **TextBlock** to the **Window.Title** property using the data binding dialog – select the Element tab and then the **Window.Title** property from the dialog. Go ahead and supply a default value as well.



```
window
  [DockPanel]
    [Grid]
      [TextBlock] "Title Goes Here"
      [Button]
    [StatusBar]
    [ListBox]
    [Separator]
    [MediaElement]
```

```
ResizeMode : (ResizeMode)
Resources (Array) : (ResourceDictionary)
RestoreBounds : (Rect)
ShowActivated : (Boolean)
ShowInTaskbar : (Boolean)
SizeToContent : (SizeToContent)
SnapsToDevicePixels : (Boolean)
Style : (Style)
TabIndex : (Int32)
Tag : (Object)
Template : (ControlTemplate)
TemplatedParent : (DependencyObject)
Title : (String)
ToolTip : (Object)
Top : (Double)
Topmost : (Boolean)
Triggers (Array) : (TriggerCollection)
Uid : (String)
```

Show: Matching Types Only

Use a custom path expression: Title

Binding direction: [✓ Default] OneTime OneWay TwoWay OneWayToSource

Update source when: Default

Binding default value: Title Goes Here

Value converter: No value converter

Converter parameter:

12. Finally, set a font and foreground color on the **TextBlock** and up the font size to be something readable. Add a **DropShadow** onto this element as well.
13. The last thing we need to do is provide the drag behavior. Here the programmer has also provided a method for us called OnDrag() in the code behind. Wire up a **MouseDown** handler on the window element to this OnDrag method:
14. Run the application – it should now be draggable and closable with the button.
15. The last step is to provide resize behavior. The Window control template already has a resize grip in it so edit the Window control template again, expand the Adorner and change the visibility of the **ResizeGrip** to “Visible”. While you are there, remove the trigger related to the **CanResizeWithGrip** property that’s in the template – this is not necessary for our window.
16. Change the ResizeMode of the Window to “CanResizeWithGrip” and then run the application to see it resize!

The final solution provides a gradient fill onto the **StatusBar** at the top and changes the **CornerRadius** of the **Listbox** while applying a margin set to push it in a bit. The final screen shot was shown at the top of the lab – see if you can make your solution look like that.

Solution

There is a full solution to this lab in the **after** folder for both Blend 3 and Blend 4.