ComponentOne

Studio for Silverlight Extended Library

By GrapeCity, Inc.

Corporate Headquarters
ComponentOne, a division of GrapeCity
201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 • USA

Internet: **info@ComponentOne.com**

Web site: **http://www.componentone.com**

Sales

E-mail: sales@componentone.com

Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

This manual was produced using ComponentOne Doc-To-Help™.

Table of Contents

ComponentOne Studio for Silverlight Extended Library Overview

The **ComponentOne Studio for SilverlightExtended Library** is part of **ComponentOne Studio for Silverlight**. It contains every control in the **C1.Silverlight.Extended.dll** assembly, which includes specialized controls that are used less frequently than the ones in the **C1.Silverlight.dll** assembly. Most of the controls in the **C1.Silverlight.Extended.dll** are visually rich controls not available in **WinForms** or in **WPF**.

Main Classes

The following main classes are included in the **C1.Silverlight.Extended.dll** assembly:

- **C1Accordion:** An **ItemsControl** where each item is shown inside a **C1Expander** control and only one is allowed to be expanded at a time. This is similar to the Microsoft Outlook navigation bar.
- **C1Book:** Presents **UIElement** objects as if they were pages in a book. You can see two elements at a time, and turn pages with the mouse, just as you would turn pages in a regular paper book. The control provides page-turning effects and shadows that make the experience visually interesting and yet familiar.
- **C1ColorPicker:** Enables users to browse colors using a palette or to create colors using the RGB and HSB color models.
- **C1Expander:** A **C1HeaderedContentControl** whose content can be collapsed and expanded by clicking on the header.
- **C1HtmlHost:** Provides a frame that can host arbitrary HTML content. The control can display content from arbitrary URIs (**SourceUri** property), or it can display HTML documents (**SourceHtml** property).

Note: the **C1HtmlHost** control requires that the host Silverlight plug-in have its **Windowless** property set to **True**.

- **C1PropertyGrid:** Provides an editable form for any object. Includes more than 10 built-in editors and support for custom editors. Also supports methods.
- **C1CoverFlow:** Presents an animated, three dimensional display of selectable items, similar to the Apple iTunes® application (supports any UIElement).
- **C1Reflector:** A **ContentControl** that adds a real 3D reflection to its content (supports any UIElement)

Help with ComponentOne Studio for Silverlight

Getting Started

For information on installing **ComponentOne Studio for Silverlight**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with Studio for Silverlight.

What's New

For a list of the latest features added to **ComponentOne Studio for Silverlight**, visit What's New in Studio for Silverlight.

Accordion

Display a list of expandable items with **ComponentOne Accordion™ for Silverlight**. Select an item to expand it and collapse all others, automatically organizing your UI and optimizing the use of screen real estate.

Getting Started

- Working with the C1Accordion Control (page 18)- Quick Start (page 13)- Task-Based Help (page 31)

Accordion for Silverlight Key Features

ComponentOne Accordion for Silverlight allows you to create customized, rich applications. Make the most of **Accordion for Silverlight** by taking advantage of the following key features:

- **Expand Direction**

  The **C1Accordion** control has the ability to expand in four different directions. The ExpandDirection property indicates which direction the control expands and can be set to **Top**, **Right**, **Bottom**, or **Left**. For more information, see the Expand Direction (page 21)topic.

- **Custom Header**

  An accordion pane's header can be customized with both text and controls. For more information on the customizable header element, see Accordion Pane Header (page 19).

- **Configure Items in an Organized Pattern**

  Accordion is designed to maximize space. Configure the size and position of C1Accordion to hide items until needed.

- **Add Objects of any Data Type**

  Because the C1Accordion control inherits from **ItemsControl**, you can add objects of any data type to its Items collection and use a **DataTemplate** to create a visual representation of the items.

Accordion for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **Accordion for Silverlight**. In this quick start, you'll start in Visual Studio to create a new project with a C1Accordion control. You will also customize the accordion, add accordion panes filled with content to it, and then observe some of the run-time features of the control.

Step 1 of 4: Creating an Application with a C1Accordion Control

In this step, you'll begin in Visual Studio to create a Silverlight application using **Accordion for Silverlight**.

Complete the following steps:

1. In Visual Studio, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Silverlight Application**. Enter a **Name** for your project and click **OK**. The **New Silverlight Application** dialog box will appear.

3. Click **OK** to close the **New Silverlight Application** dialog box and create your project.
4. In the XAML window of the project, resize the **UserControl** by changing `DesignWidth="400" DesignHeight="300"` to `DesignWidth="Auto" DesignHeight="Auto"` in the `<UserControl>` tag so that it appears similar to the following:

```
<UserControl x:Class="SilverlightApplication24.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d=http://schemas.microsoft.com/expression/blend/2008
  xmlns:mc="http://schemas.openxmlformats.org/markup-
  compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="Auto" d:DesignHeight="Auto">
```

   The **UserControl** will now resize to accommodate any content placed within it.
5. In the XAML window of the project, place the cursor between the `<Grid>` and `</Grid>` tags and click once. Note that you cannot currently add Silverlight controls directly to the design area in Visual Studio, so you must add them to the XAML window as directed in the next step.
6. Navigate to the Toolbox and double-click the C1Accordion icon to add the control to the grid. The XAML markup resembles the following:

```
<Grid x:Name="LayoutRoot">
<c1:C1Accordion></c1:C1Accordion>
</Grid>
```

You've successfully created a Silverlight application containing a C1Accordion control. In the next step, you will customize the appearance and behavior of the C1Accordion control.

Step 2 of 4: Customizing the C1Accordion Control

In the last step, you created a Silverlight project and added a C1Accordion control to it. In this step, you will customize the behavior and appearance of the C1Accordion control.

Complete the following steps:

1. Add `Height="250"` to the `<c1:C1Accordion>` tag to set the height of the control. The XAML markup appears as follows:

```
<c1:C1Accordion Height="250">
```

2. Add `Width="400"` to the `<c1:C1Accordion>` tag to set the width of the control. The XAML markup appears as follows:

```
<c1:C1Accordion Height="250" Width="400">
```

3. Add `ExpandDirection="Left"` to the `<c1:C1Accordion>` tag so that the C1Accordion control will expand from the bottom rather than expanding from the top, which is its default. The XAML markup appears follows:

```
<c1:C1Accordion Height="250" Width="400" ExpandDirection="Left">
```

4. Add `Fill="True"` to the `<c1:C1Accordion>` tag so that each pane will expand to fill the specified width of the C1Accordion control. The XAML markup appears follows:

```
<c1:C1Accordion Height="250" Width="400" ExpandDirection="Left"
Fill="True">
```

5. Add `AllowCollapseAll="False"` to the `<c1:C1Accordion>` tag to prevent users from collapsing all panes at the same time. The XAML markup appears follows:

```
<c1:C1Accordion Height="250" Width="400" ExpandDirection="Left"
Fill="True" AllowCollapseAll="False">
```

In this step, you customized the appearance and behavior of the C1Accordion control. In the next step, you will add customized accordion panes to the control.

Step 3 of 4: Adding Accordion Panes

In the last step, you customized the appearance and behavior of the C1Accordion control. In the next step, you will add accordion panes, which you will customize and add content to.

Complete the following steps.

1. Place your cursor between the `<c1:C1Accordion>` and `</c1:C1Accordion>` tags and press ENTER.
2. Navigate to the Toolbar and click the C1AccordionItem icon to add an accordion pane to the control. Repeat this twice so that a total of three C1AccordionItems are added to the C1Accordion control. The XAML markup will resemble the following:

```
<c1:C1AccordionItem></c1:C1AccordionItem>

<c1:C1AccordionItem></c1:C1AccordionItem>

<c1:C1AccordionItem></c1:C1AccordionItem>
```

3. Add `Header="Pane 1"` to the first `<c1:C1AccordionItem>` tag, `Header="Pane 2"` to the second `<c1:C1AccordionItem>` tag, and `Header="Pane 3"` to the third `<c1:C1AccordionItem>` tag so that the XAML markup resembles the following:

```
<c1:C1AccordionItem Header="Pane 1"></c1:C1AccordionItem>

<c1:C1AccordionItem Header="Pane 2"></c1:C1AccordionItem>

<c1:C1AccordionItem Header="Pane 3"></c1:C1AccordionItem>
```

4. Add `Background="Aqua"` to the first `<c1:C1AccordionItem>` tag, `Background="AliceBlue"` to the second `<c1:C1AccordionItem>` tag, and `Background="LawnGreen"` to the third `<c1:C1AccordionItem>` tag so that the XAML markup resembles the following:

```
<c1:C1AccordionItem Header="Pane 1"
Background="Aqua"></c1:C1AccordionItem>

<c1:C1AccordionItem Header="Pane 2"
Background="AliceBlue"></c1:C1AccordionItem>

<c1:C1AccordionItem Header="Pane 3"
Background="LawnGreen"></c1:C1AccordionItem>
```

5. Add content to the first two accordion panes by completing the following steps:
   a. Add `Content="This is text content"` to the first `<c1:C1AccordionItem>` tag so that the XAML resembles the following:

```
<c1:C1AccordionItem Header="Pane 1" Background="Aqua" Content="This
is text content">
```

   b. Place your cursor between the second set of `<c1:C1AccordionItem>` and `</c1:C1AccordionItem>` tags and press ENTER.
   c. Navigate to the Toolbox and double-click the **Calendar** icon to add a **Calendar** control as the second pane's content. The XAML resembles the following:

```
<c1:C1AccordionItem Header="Pane 2" Background="AliceBlue">
    <sdk:Calendar></sdk:Calendar>
```

```
</c1:C1AccordionItem>
```

6. Add `IsExpanded="True"` to the second `<c1:C1AccordionItem>` tag so that the XAML resembles the following:

```
<c1:C1AccordionItem Header="Pane 2" Background="AliceBlue"
IsExpanded="True">
```

This means that the second accordion pane, which holds the **Calendar** control, will be expanded at run time.

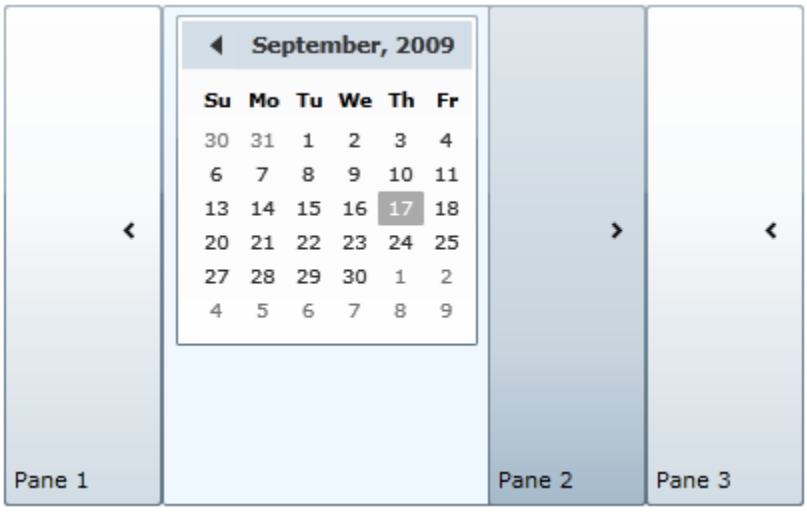
In this step, you added three accordion panes to the C1Accordion control and then added content to two of the accordion panes. In the next step, you will run the project and observe the run time features of the control.

Step 4 of 4: Running the Project

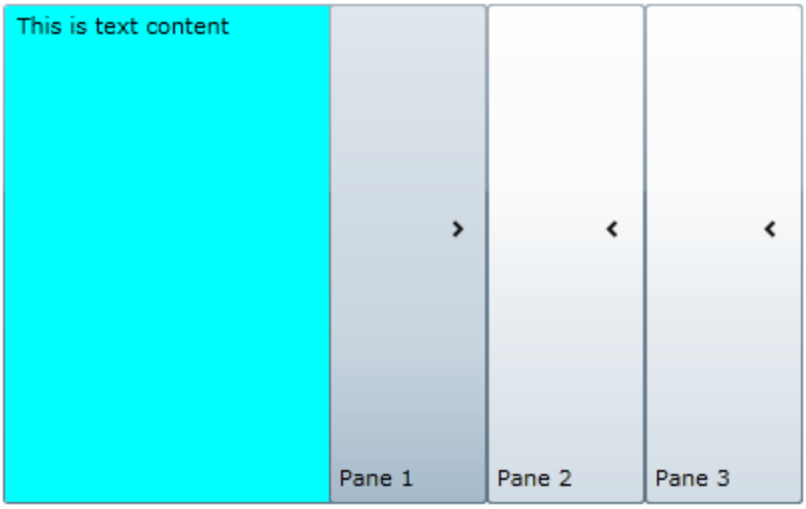
In the last step, you added accordion panes and content to the C1Accordion control. In this step, you will run the project and observe some of the run-time features of the C1Accordion control.

Complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time. Observe that the second pane, which holds the **Calendar** control, is expanded.



2. Click **Pane 1** and observe that the first pane expands to reveal its content.



3. Click **Pane 3** and observe that the last pane expands. Note that the third pane expands to the same width as the other panes despite the fact that it has no content.

4. Click **Pane 3** and observe that you can't close the pane. This is because you set the AllowCollapseAll property to **False**, which means that one accordion pane must be expanded at all times.

Congratulations! You have completed the **Accordion for Silverlight** quick start tutorial. In this tutorial, you created a Silverlight project containing a C1Accordion control, modified the appearance and behavior of the control, added accordion panes and accordion pane content to the control, and then observed some of the run-time features of the control.

Accordion XAML Quick Reference

This topic is dedicated to providing a quick overview of the XAML used to complete various C1Accordion tasks. For more information, see the Accordion for Silverlight Task-Based Help (page 31) section.

C1Accordion with C1AccordionItems

The following XAML markup creates a C1Accordion control with three C1AccordionItems.

```
<c1:C1Accordion Height="100" HorizontalAlignment="Left"
Name="c1Accordion1" VerticalAlignment="Top" Width="200">
     <c1:C1AccordionItem IsTabStop="False" />
     <c1:C1AccordionItem IsTabStop="False" />
     <c1:C1AccordionItem IsTabStop="False" />
</c1:C1Accordion>
```

C1AccordionItems with Content

The following XAML markup creates a C1Accordion control containing three C1AccordionItems. It is similar to the markup under C1Accordion with C1AccordionItems, but this time each C1AccordionItem contains content. The first item contains a **TextBlock**, the second item contains a **Button** control, and the third item contains a **Calendar** control.

```
<c1:C1Accordion Height="276" HorizontalAlignment="Left"
Name="c1Accordion1" VerticalAlignment="Top" Width="355">
    <c1:C1AccordionItem IsTabStop="False">
       <TextBlock Height="23" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="textBlock1" Text="TextBlock"
VerticalAlignment="Top">Hello world, my name is Studio for
Silverlight.</TextBlock>
    </c1:C1AccordionItem>
    <c1:C1AccordionItem IsTabStop="False">
        <Button Content="Button" Height="23" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="button1" VerticalAlignment="Top" Width="75" />
     </c1:C1AccordionItem>
     <c1:C1AccordionItem IsTabStop="False">
         <sdk:Calendar Height="169" HorizontalAlignment="Left"
Margin="15,15,0,0" Name="calendar1" VerticalAlignment="Top" Width="230"
/>
     </c1:C1AccordionItem>
</c1:C1Accordion>
```

Working with the C1Accordion Control

The C1Accordion control is a container that can hold a series of expandable and collapsible panes for storing text, images, and controls. The C1Accordion control is an **ItemsControl**, which means that the control is designed to host a series of objects. The C1AccordionItem class represents the items, or accordion panes, that can be hosted by the C1Accordion control.

When you add the C1Accordion control to a project, it exists as nothing more than a container. But once the control is added to your project, you can easily add multiple accordion panes to it in Blend, in XAML, or in code. The following image depicts a C1Accordion control with three accordion pane items, the first of which is expanded.

The image above shows the accordion panes sans header text or content, but customizing the header and adding content to the pane is as simple as setting a few properties. You can also modify behaviors, such as the expandability and the direction, of the control.

The following topics provide an overview of the C1Accordion control's elements and features.

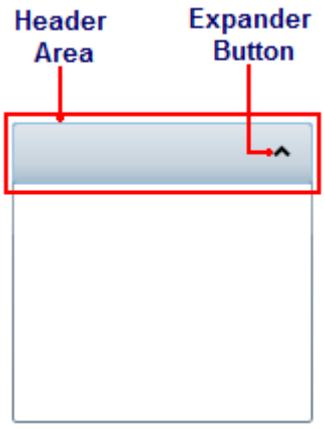
C1Accordion Elements

This section provides a visual and descriptive overview of the elements that comprise the C1Accordion control. The control is comprised of two elements – the header and the content area – that combine to make the complete C1Accordion control.

Accordion Pane Header

By default, the header element of an accordion pane appears at the top of the control and the expander button appears on the right side of the header. When the C1AccordionItem item (accordion pane) is first placed on the page, the header element contains no text.

The following image labels the header area of an accordion pane.



To add text to the header element, simply set the **Header** property to a string. Once the text is added, you can style it using several font properties (see Text Properties (page 28)). You can also add Silverlight controls to the header. For task-based help about adding content to the header, see Adding Content to Header Elements (page 32).

The placement of the header element and expander button will change depending on the expand direction of the control. For more information on expand directions, see the Expand Direction (page 21) topic.

Attribute Syntax versus Property Element Syntax

When you want to add something simple to the header, such as an unformatted string, you can simply use the common XML attributes in your XAML markup, such as in the following:

```
<c1:C1AccordionItem Header="Hello World"/>
```

However, there may be times where you want to add more complex elements, such as grids or panels, to the content area. In this case you would use property element syntax, such as in the following:

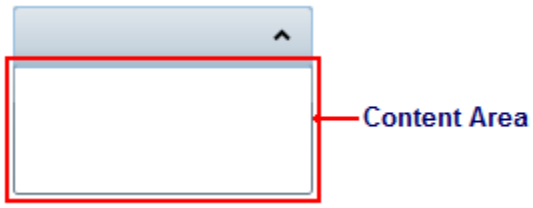
```
<c1:C1AccordionItem Width="150" Height="55" Name="C1AccordionItem1">
            <c1:C1AccordionItem.Header>
```

```
        <Grid HorizontalAlignment="Stretch">
           <Grid.ColumnDefinitions>
              <ColumnDefinition Width="Auto" />
              <ColumnDefinition Width="*" />
           </Grid.ColumnDefinitions>
              <TextBlock Text="C1AccordionItem Header" />
         </Grid>
            </c1:C1AccordionItem.Header>
</c1:C1AccordionItem>
```

Accordion Pane Content Area

An accordion pane's content area initially consists of an empty space. In the content area, you can add grids, text, images, and arbitrary controls. When working in Blend, elements in the content area of the control can be added and moved on the control through a simple drag-and-drop operation.

The following image labels the content area of an accordion pane.



You can add text to the content area by setting the item's **Content** property or by adding a **TextBox** element to the content area. Adding Silverlight elements to the content area at run time is simple: You can either use simple drag-and-drop operations or XAML in Visual Studio or Blend. If you'd prefer to add a control at run time, you can use C# or Visual Basic code. For task-based help about adding content to the content area, see Adding Content to Content Areas (page 34).

A C1AccordionItem item can only accept one child element at a time. However, you can get around this limitation by adding a panel-based control as its child element. Panel-based controls, such as a **StackPanel** control, are able to hold multiple elements. The panel-based control meets the one control limitation of the C1AccordionItem item, but its ability to hold multiple elements will allow you to show several controls in the content area of the accordion pane.

Attribute Syntax versus Property Element Syntax

When you want to add something simple to the content area, such as an unformatted string or a single control, you can simply use the common XML attributes in your XAML markup, such as in the following:

```
<c1:C1AccordionItem Content="Hello World"/>
```

However, there may be times where you want to add more complex elements, such as grids or panels, to the content area. In this case you can use property element syntax, such as in the following:

```
<c1:C1AccordionItem Width="150" Height="55" Name="C1AccordionItem1">
            <c1:C1AccordionItem.Content>
                <StackPanel>
                    <TextBlock Text="Hello"/>
                    <TextBlock Text="World"/>
                </StackPanel>
```

```
            </c1:C1AccordionItem.Content>
</c1:C1AccordionItem>
```

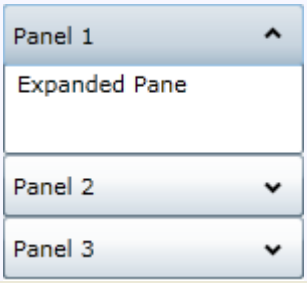
Expanding and Collapsing Accordion Panes

This section details the options for customizing the way that the accordion panes expand and collapse.

Accordion Pane Initial Expand State

By default, the **IsExpanded** property of each accordion pane is set to **False**, which means that the pane appears in its collapsed state when the page is loaded. If you want the pane to be expanded upon page load, you can set the **IsExpanded** property to **True**.

The following table illustrates the difference between the two expand states.

IsExpanded	Result
IsExpanded=False	
IsExpanded=True	

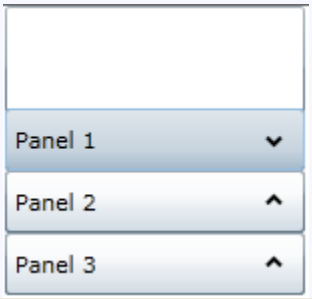
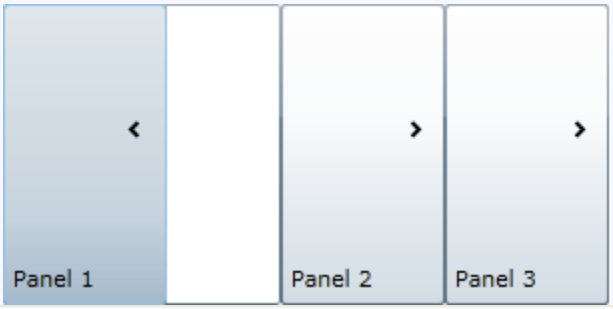
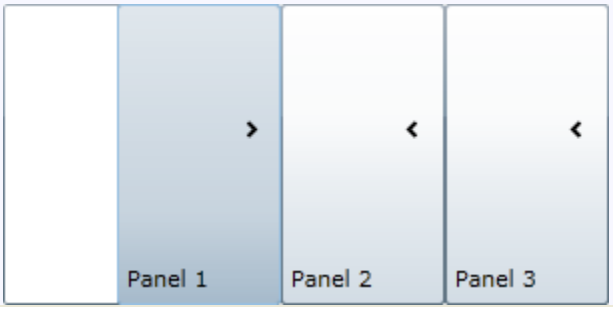
You can set the expand states in Blend, XAML, or code.

Expand Direction

The C1Accordion control includes the option to specify the expand direction using the ExpandDirection property. In addition to setting the direction of expansion, changing the ExpandDirection also changes the header's orientation to the content area of the control. By default the ExpandDirection property is set to **Down** and the control expands from top to bottom.

The following table illustrates each ExpandDirection setting.

ExpandDirection	Result

Down	
Up	
Right	
Left	

You can set the collapsing and expanding direction in Blend, XAML, or code.

Collapsing Panes

By default, the C1Accordion control's AllowCollapseAll property is set to **True**, meaning that all panes of an accordion can be collapsed by the user. This is useful in cases where you want to conserve screen real estate so that the user is free of unnecessary distractions. Think of it like the musical instrument accordion, which can be tightly compressed for storage and then decompressed when a player wants to use it.

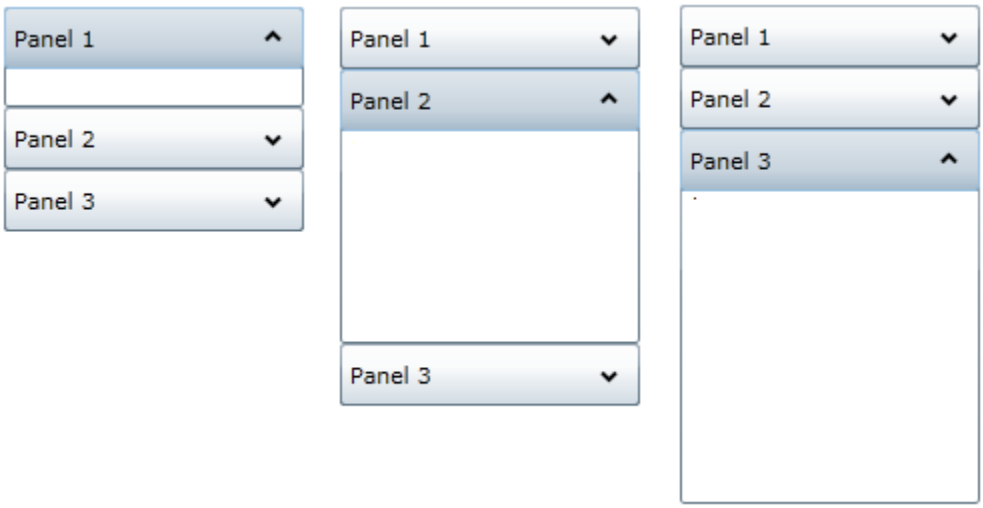
But there are times when this won't fit into the design of your user interface. For example, you may want to use the accordion as a menu element. In that case, you'd want the C1Accordion control to fill to a specific height or width at all times, and the control won't fulfill that need if all of its panes can be collapsed. Therefore, you would set the AllowCollapseAll property to **False** so that one pane has to remain open at all times. You would then set the

height or width property as desired and then set the Fill property to **True** so that the C1Accordion control and its panes always fill to a specified height or width.

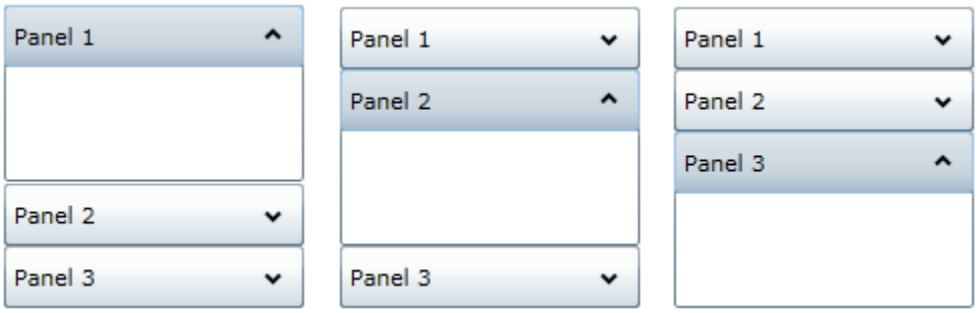
> **Note**: The C1Accordion control is only capable of expanding one pane at a time.

Expansion Fill

The default behavior of an accordion pane (C1AccordionItem) is to fill only to the height (for accordions that expand up or down) or width (for accordions that expand right or left) of an accordion pane's contents. This will cause the accordion to grow to different heights or widths depending on the height or width of the content for a given pane. This is illustrated in the example below.



To avoid this, set the height or width of the C1Accordion control and then set its Fill property to **True**. This will cause each panel to expand to fill the width or height that you specified, thus providing a uniformed width or height for accordion pane. You can observe the difference in the example below.



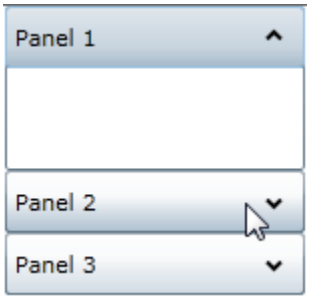
Accordion for Silverlight Layout and Appearance

The following topics detail how to customize the C1Accordion control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the grid and take advantage of Silverlight's XAML-based styling. You can also use templates to format and layout the control and to customize the control's actions.

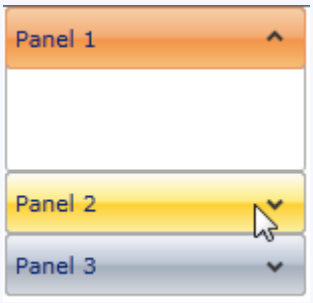
Accordion Theming

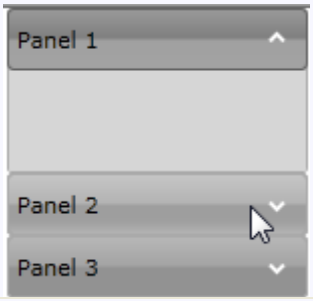
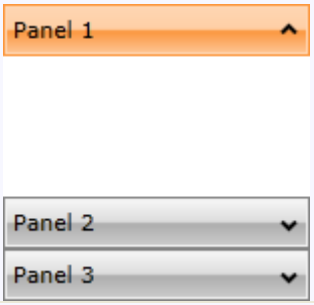
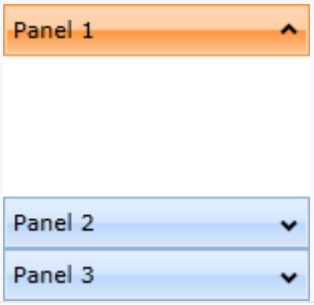
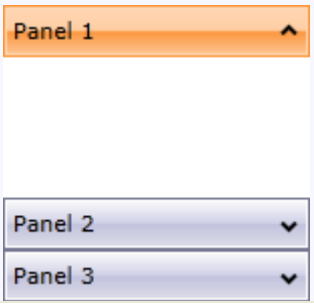
Silverlight themes are a collection of image settings that define the look of a control or controls. The benefit of using themes is that you can apply the theme across several controls in the application, thus providing consistency without having to repeat styling tasks.

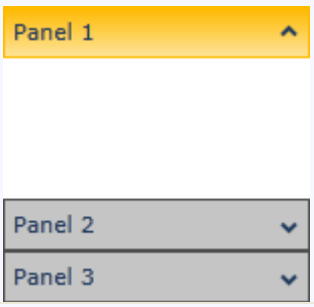
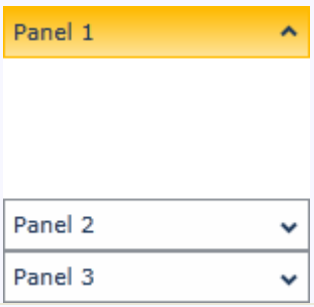
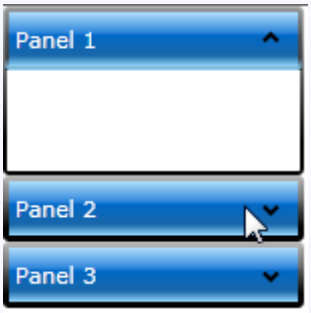
When the C1Accordion control is added to your project, it appears with the default blue theme:

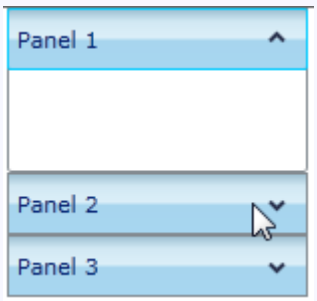


You can also theme the C1Accordion control with one of our seven included Silverlight themes: BureauBlack, Cosmopolitan, ExpressionDark, ExpressionLight, RainierOrange, ShinyBlue, and WhistlerBlue. The table below provides a sample of each theme.

Full Theme Name	Appearance
C1ThemeBureauBlack	
C1ThemeCosmopolitan	

C1ThemeExpressionDark	
C1ThemeExpressionLight	
C1ThemeOffice2007Black	
C1ThemeOffice2007Blue	
C1ThemeOffice2007Silver	

C1ThemeOffice2010Black	
C1ThemeOffice2010Blue	
C1ThemeOffice2010Silver	
C1ThemeRainierOrange	
C1ThemeShinyBlue	

C1ThemeWhistlerBlue	

You can add any of these themes to a C1Accordion control by declaring the theme around the control in markup and then setting the **Theme.Apply** mode to **Auto**. For task-based help about adding themes to the C1Accordion control, see the Using C1Accordion Themes (page 38).

C1Accordion and C1AccordionItem ClearStyle Properties

Accordion for Silverlight supports ComponentOne's new ClearStyle technology that allows you to easily change control colors without having to change control templates. By just setting a few color properties you can quickly style the entire grid.

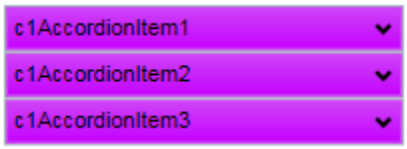
The following table outlines the brush properties of the C1Accordion control:

Brush	Description
Background	Gets or sets the brush of the control's background.
ExpandedBackground	Gets or sets the brush of the header background that appears when items of the C1Accordion control are expanded.
MouseOverBrush	Gets or sets the System.Windows.Media.Brush used to highlight the control's C1AccordionItems when it is moused over.
HeaderBackground	Gets or sets the background brush of all headers of all C1AccordionItem headers contained within the C1Accordion control.

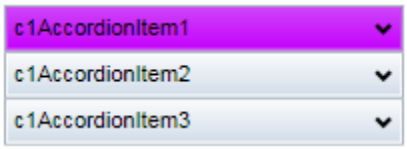
The following table outlines the brush properties of the **C1AccordionItem** control:

Brush	Description
Background	Gets or sets the brush of the control's background.
ExpandedBackground	Gets or sets the brush of the header background when the C1AccordionItem is expanded.
MouseOverBrush	Gets or sets the System.Windows.Media.Brush used to highlight the control when it is moused over.
HeaderBackground	Gets or sets the background brush of the C1AccordionItem header.

You can completely change the appearance of the **C1Accordion** and **C1AccordionItem** controls by setting a few properties, such as the **C1Accordion** control's **HeaderBackground** property, which sets the background color for every accordion item in a **C1Accordion** control. For example, if you set the **C1Accordion** control's **HeaderBackground** property to "#FFC500FF", each header in the **C1Accordion** control would appear similar to the following:



You can also choose to modify the accordion items independently by setting the **C1AccordionItem**'s **HeaderBackground** property. For example, if you set the **HeaderBackground** property of the first accordion item to "#FFC500FF", the control would appear similar to the following:



It's that simple with ComponentOne's ClearStyle technology. For more information on ClearStyle, see the ComponentOne ClearStyle Technology topic.

Accordion for Silverlight Appearance Properties

ComponentOne Accordion for Silverlight includes several properties that allow you to customize the appearance of the control. You can change the appearance of the text displayed in the control and customize graphic elements of the control. The following topics describe some of these appearance properties.

Text Properties

The following properties let you customize the appearance of text in the accordion pane.

Property	Description
FontFamily	Gets or sets the font family of the control. This is a dependency property.
FontSize	Gets or sets the font size. This is a dependency property.
FontStretch	Gets or sets the degree to which a font is condensed or expanded on the screen. This is a dependency property.
FontStyle	Gets or sets the font style. This is a dependency property.
FontWeight	Gets or sets the weight or thickness of the specified font. This is a dependency property.
TextAlignment	Gets or sets how the text should be aligned in the accordion pane content area. This is a dependency property.
Header	Gets or sets the header of an accordion item.
HeaderFontFamily	Gets or sets the font family of the header.
HeaderFontStretch	Gets or sets the font stretch of the header.
HeaderFontStyle	Gets or sets the font style of the header.

HeaderFontWeight	Gets or sets the font weight of the header.

Content Positioning Properties

The following properties let you customize the position of header and content area content in the accordion and in the accordion's panes.

Property	Description
HeaderPadding	Gets or sets the padding of the header.
HeaderHorizontalContentAlignment	HorizontalContentAlignment of the header.
HeaderVerticalContentAlignment	Gets or sets the vertical content alignment of the header.
HorizontalContentAlignment	Gets or sets the horizontal alignment of the control's content. This is a dependency property.
VerticalContentAlignment	Gets or sets the vertical alignment of the control's content. This is a dependency property.

Color Properties

The following properties let you customize the colors used in the accordion and its panes.

Property	Description
Background	Gets or sets a brush that describes the background of a control. This is a dependency property.
Foreground	Gets or sets a brush that describes the foreground color. This is a dependency property.
HeaderBackground	Gets or sets the background brush of the header.
HeaderForeground	Gets or sets the foreground brush of the header.

Border Properties

The following properties let you customize the border of the control and its panes.

Property	Description
BorderBrush	Gets or sets a brush that describes the border background of a control. This is a dependency property.
BorderThickness	Gets or sets the border thickness of a control. This is a dependency property.

Size Properties

The following properties let you customize the size of the accordion and its panes.

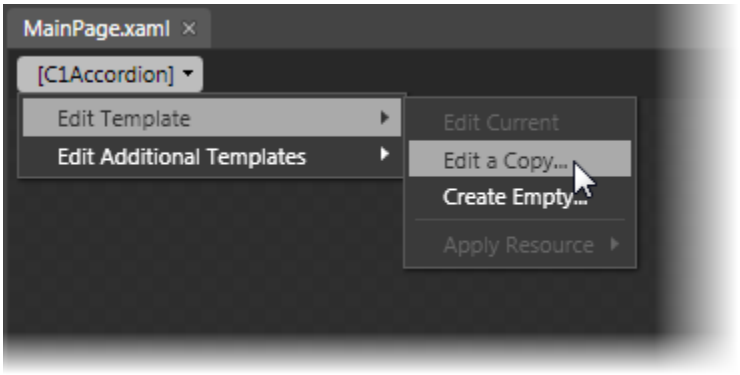
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
MinHeight	Gets or sets the minimum height constraint of the element. This is a dependency property.
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

Templates

One of the main advantages to using a Silverlight control is that controls are "lookless" with a fully customizable user interface. Just as you design your own user interface (UI), or look and feel, for Silverlight applications, you can provide your own UI for data managed by **ComponentOne Accordion for Silverlight**. Extensible Application Markup Language (XAML; pronounced "Zammel"), an XML-based declarative language, offers a simple approach to designing your UI without having to write code.

Accessing Templates

You can access templates in Microsoft Expression Blend by selecting the C1Accordion control and, in the menu, selecting **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or select **Create Empty** to create a new blank template.



If you want to edit the C1AccordionItem template, simply select the C1AccordionItem control and, in the menu, select **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or **Create Empty**, to create a new blank template.

Note: If you create a new template through the menu, the template will automatically be linked to that template's property. If you manually create a template in XAML you will have to link the appropriate template property to the template you've created.

Note that you can use the Template property to customize the template.

Item Templates

ComponentOne Accordion for Silverlight's accordion control is an **ItemsControl**s that serves as a container for other elements. As such, the control includes templates to customize items places within the accordion. These templates include an **ItemTemplate** an **ItemContainerStyle** template. You use the **ItemTemplate** to specify the visualization of the data objects and the **ItemsPanel** to define the panel that controls the layout of items.

Accessing Templates

You can access these templates in Microsoft Expression Blend by selecting the C1Accordion control and, in the menu, selecting **Edit Additional Templates**. Choose **Edit Generated Items (ItemTemplate)** or **Edit Layout of Items (ItemsPanel)** and select **Create Empty** to create a new blank template or **Edit a Copy**.

A dialog box will appear allowing you to name the template and determine where to define the template.

Accordion for Silverlight Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1Accordion control in general. If you are unfamiliar with the **ComponentOne Accordion for Silverlight** product, please see the **Accordion for Silverlight** Quick Start first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne Accordion for Silverlight** product.

Each task-based help topic also assumes that you have created a new Silverlight project.

Adding Accordion Panes to the C1Accordion Control

In this topic, you will add an accordion pane to a C1Accordion control in Blend, in XAML, and in code.

At Design Time in Blend

To add a pane to the C1Accordion control, complete the following steps:

1. Click the C1Accordion control once to select it.
2. Under the **Assets** tab, double-click the C1AccordionItem icon.

   An accordion pane is added to the C1Accordion control.
3. Click the C1AccordionItem item once to select it.
4. Under the **Properties** panel, set the **Width** property to "150".

In XAML

To add a pane to the C1Accordion control, place the following markup between the `<c1:C1Accordion>` and `</c1:C1Accordion>` tags:

```
<c1:C1AccordionItem Content="C1AccordionItem" Width="150">
</c1:C1AccordionItem>
```

In Code

To add accordion panes in code, complete the following steps:

1. Add `x:Name="C1Accordion1"` to the `<c1:C1Accordion>` tag so that the object will have a unique identifier for you to call in code.

2. Enter Code view and import the following namespace:

   - Visual Basic
     ```
     Imports C1.Silverlight.Extended
     ```
   - C#
     ```
     using C1.Silverlight.Extended;
     ```

3. Add the following code beneath the **InitializeComponent()** method:

   - Visual Basic
     ```
     'Create the accordion pane and add content
     Dim C1AccordionItem1 As New C1AccordionItem()
     C1AccordionItem1.Content = "C1AccordionItem1"
     C1AccordionItem1.Width = 150
     'Add the accordion pane to the C1Accordion control
     C1Accordion1.Items.Add(C1AccordionItem1)
     ```
   - C#
     ```
     //Create the accordion pane and add content
     C1AccordionItem C1AccordionItem1 = new C1AccordionItem();
     C1AccordionItem1.Content = "C1AccordionItem1";
     C1AccordionItem1.Width = 150;
     //Add the accordion pane to the C1Accordion control
     C1Accordion1.Items.Add(C1AccordionItem1);
     ```

4. Run the program.

Adding Content to Header Elements

You can easily add both simple text and Silverlight controls to an accordion pane's header. The topics in this section will provide step-by-step instructions about adding text content and controls to the header. For more information on the header element, you can also visit the Accordion Pane Header (page 19) topic.

Adding Text to the Header

By default, an accordion pane's header is empty. You can add text to the control's header by setting the **Header** property to a string in Blend, in XAML, or in code. This topic assumes that you have added a C1Accordion control with at least one C1AccordionItem item to your project.

At Design Time in Blend

To set the **Header** property in Blend, complete the following steps:

1. Click the C1AccordionItem item once to select it.
2. Under the **Properties** panel, set the **Header** property to a string (for example, "Hello World").

In XAML

To set the **Header** property in XAML, add `Header="Hello World"` to the `<c1:C1AccordionItem>` tag so that it appears similar to the following:

```
<c1:C1AccordionItem Header="Hello World" Width="150" Height="55">
```

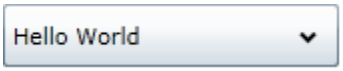
In Code

To set the **Header** property in code, complete the following steps:

1. Add x:Name="C1AccordionItem1" to the <c1:C1AccordionItem> tag so that the object will have a unique identifier for you to call in code.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic
     ```
     C1AccordionItem1.Header = "Hello World"
     ```
   - C#
     ```
     C1AccordionItem1.Header = "Hello World";
     ```
3. Run the program.

This Topic Illustrates the Following:

The header of the accordion pane now reads "Hello World". The end result of this topic should resemble the following:



Adding a Control to the Header

Any accordion pane header element is able to accept a Silverlight control. In this topic, you will add a **Button** control to the header in XAML and in code. This topic assumes that you have added a C1Accordion control with at least one C1AccordionItem item to your project.

In XAML

To add a **Button** control to the header in XAML, place the following XAML markup between the <c1:C1AccordionItem> and </c1:C1AccordionItem> tags:

```
<c1:C1AccordionItem.Header>
    <Button Content="Button" Height="Auto" Width="50"/>
</c1:C1AccordionItem.Header>
```

In Code

To add a **Button** control to the header in code, complete the following steps:

1. Add x:Name="C1AccordionItem1" to the <c1:C1Accordion> tag so that the object will have a unique identifier for you to call in code.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

```
'Create the Button control
Dim NewButton As New Button()
NewButton.Content = "Button"
'Set the Button Control's Width and Height properties
NewButton.Width = 50
NewButton.Height = Double.NaN
'Add the Button to the header
```

```
C1AccordionItem1.Header = (NewButton)
```

- C#

```
InitializeComponent();
//Create the Button control
Button NewButton = new Button();
NewButton.Content = "Button";
//Set the Button Control's Width and Height properties
NewButton.Width = 50;
NewButton.Height = Double.NaN;
//Add the Button to the header
C1AccordionItem1.Header = (NewButton);
```

3. Run the program.

This Topic Illustrates the Following:

As a result of this topic, the control will appear in the header. The final result will resemble the following image:

Adding Content to Content Areas

You can easily add both simple text and Silverlight controls to an accordion pane's header. The topics in this section will provide step-by-step instructions about how to add text content and controls to the header.

For more information on the header element, you can also visit the Accordion Pane Content Area (page 20) topic.

Adding Text to the Content Area

You can easily add a simple line of text to the content area of an accordion pane by setting the **Content** property to a string in Blend, in XAML, or in code. This topic assumes that you have added a C1Accordion control with at least one C1AccordionItem item to your project.

> **Note**: You can also add text to the content area by adding a **TextBox** control to the content area and then setting the **TextBox** control's **Text** property. To learn how to add a control to the content area, see Adding a Control to the Content Area (page 35).

At Design Time in Blend

To set the **Content** property in Blend, complete the following steps:

1. Click the C1AccordionItem item once to select it.
2. Under the **Properties** panel, set the **Content** property to a string (for example, "Hello World").
3. Run the program and expand the accordion pane.

In XAML

To set the **Content** property in XAML, complete the following:

1. Add `Content="Hello World"` to the `<c1:C1AccordionItem>` tag so that it appears similar to the following:

```
      <c1:C1AccordionItem Content="Hello World" Width="150"
Height="55">
```

2. Run the program and expand the accordion pane.

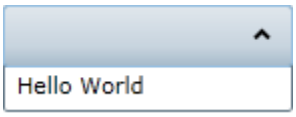
In Code

To set the **Content** property in code, complete the following steps:

1. Add `x:Name="C1AccordionItem1"` to the `<c1:C1AccordionItem>` tag so that the object will have a unique identifier for you to call in code.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic
     ```
     C1AccordionItem1.Content = "Hello World"
     ```
   - C#
     ```
     C1AccordionItem1.Content = "Hello World";
     ```
3. Run the program and expand the accordion pane.

This Topic Illustrates the Following:

The content of your accordion pane now reads "Hello World". The end result of this topic should resemble the following:



Adding a Control to the Content Area

Each accordion pane (C1AccordionItem) will accept one child control in its content area. In this topic, you will learn how to add a Silverlight button control in Blend, in XAML, and in code.

This topic assumes that you have added a C1Accordion control with at least one C1AccordionItem item to your project.

At Design Time in Blend

To add a **Button** control to the content area, complete the following steps:

1. Under the **Objects and Timeline** tab, select **[C1AccordionItem]**.
2. Navigate to the **Assets** tab and click the **Controls** drop-down arrow.
3. Select **All** to open a list of all available Silverlight controls.
4. Double click the **Button** icon to add it to the accordion pane's content area.
5. Under the **Objects and Timeline** tab, select **[Button]** so that the **Button** control's properties take focus in the **Properties** panel.
6. Next to the **Width** property, click the **Set to Auto** button. This will ensure that the height of the button control is the same height as the accordion pane's content area.
7. Next to the **Height** property, click the **Set to Auto** button. This will ensure that the height of the button control is the same height as the accordion pane's content area.

8. Run the program and expand the accordion pane.

In XAML

To add a **Button** control to the content area in XAML, complete the following:

1. Place the following markup between the `<c1:C1AccordionItem>` and `</c1:C1AccordionItem>` tags:

```
<Button Content="Button" Height="Auto" Width="Auto"/>
```

2. Run the program and expand the accordion pane

In Code

To add a **Button** control to the content area in code, complete the following:

1. Add `x:Name="C1Accordion1"` to the `<c1:C1AccordionItem>` tag so that the object will have a unique identifier for you to call in code.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

```
'Create the Button control
Dim NewButton As New Button()
NewButton.Content = "Button"
'Set the Button Control's Width and Height properties
NewButton.Width = Double.NaN
NewButton.Height = Double.NaN
'Add the Button to the content area
C1AccordionItem1.Content = (NewButton)
```

   - C#

```
//Create the Button control
Button NewButton = new Button();
NewButton.Content = "Button";
//Set the Button Control's Width and Height properties
NewButton.Width = double.NaN;
NewButton.Height = double.NaN;
//Add the Button to the content area
C1AccordionItem1.Content = (NewButton);
```

3. Run the program and expand the accordion pane.

This Topic Illustrates the Following:

When accordion pane is expanded, the button control will appear in its content area, resembling the following image:

Adding Multiple Controls to the Content Area

You cannot set an accordion pane's (C1AccordionItem) **Content** property to more than one control at a time. However, you can circumvent this issue by adding a panel-based control that can accept more than one control, such as a **StackPanel** control, to the content area of the accordion pane. When you add multiple controls to the panel-based control, each one will appear within the accordion pane's content area.

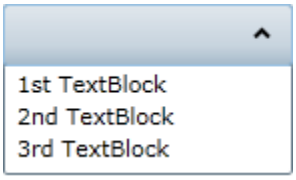
This topic assumes that you have added a C1Accordion control with at least one C1AccordionItem item to your project.

To add multiple controls to the content area, complete these steps:

1. Place the following XAML markup between the `<c1:C1AccordionItem>` and `</c1:C1AccordionItem>` tags:

```
<c1:C1AccordionItem.Content>
    <StackPanel>
        <TextBlock Text="1st TextBlock"/>
        <TextBlock Text="2nd TextBlock"/>
        <TextBlock Text="3rd TextBlock"/>
    </StackPanel>
</c1:C1AccordionItem.Content>
```

2. Run the program.
3. Expand the accordion pane and observe that each of the three **TextBlock** controls appear in the content area. The result will resemble the following:



Changing the Expand Direction

By default, the C1Accordion control's accordion panes expand from top-to-bottom because the ExpandDirection property is set to **Down**. You can easily change the expand direction by setting the ExpandDirection property to **Up**, **Right**, or **Left** in Blend, in XAML, or in code.

This topic assumes that you have added a C1Accordion control with at least one C1AccordionItem to your project.

At Design Time in Blend

To set the ExpandDirection property in Blend, complete the following steps:

1. Click the C1Accordion control once to select it.
2. Under the **Properties** panel, click the ExpandDirection drop-down arrow and select one of the options from the list. For this example, select **Right**.

In XAML

To set the ExpandDirection property to **Right** in XAML, add `ExpandDirection="Right"` to the `<c1:C1Accordion>` tag so that it appears similar to the following:

```
<c1:C1Accordion ExpandDirection=Right Width="150" Height="55">
```

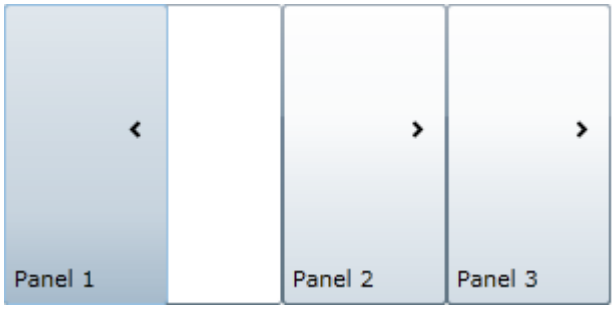
In Code

To set the ExpandDirection property in code, complete the following steps:

1. Add `x:Name="C1Accordion1"` to the `<c1:C1AccordionItem>` tag so that the object will have a unique identifier for you to reference in code.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic
     ```
     C1Accordion1.ExpandDirection =
     C1.Silverlight.Extended.ExpandDirection.Right
     ```
   - C#
     ```
     C1Accordion1.ExpandDirection =
     C1.Silverlight.Extended.ExpandDirection.Right;
     ```
3. Run the program.

This Topic Illustrates the Following:

By following the instructions in this topic, you have learned how to set the ExpandDirection property. In this topic, you set the ExpandDirection property to **Right**, which will make the C1Accordion control resemble the following:



Using C1Accordion Themes

The C1Accordion control comes equipped with a light blue default theme, but you can also apply seven themes (see Accordion Theming (page 24)) to the control. In this topic, you will change the C1Accordion control's theme to **C1ThemeRainierOrange** in Blend and in Visual Studio.

In Blend

Complete the following steps:

1. Click the **Assets** tab.
2. In the search bar, enter "C1ThemeRainierOrange".

   The **C1ThemeRainierOrange** icon appears.
3. Double-click the **C1ThemeRainierOrange** icon to add it to your project.
4. In the search bar, enter "C1Accordion" to search for the C1Accordion control.
5. Double-click the C1Accordion icon to add the C1Accordion control to your project.
6. Under the **Objects and Timeline** tab, select **[C1Accordion]** and, using a drag-and-drop operation, place it underneath **[C1ThemeRainierOrange]**.

7. Return to the **Assets** tab.
8. In the search bar, enter "C1AccordionItem" to search for the C1AccordionItem item.
9. Double-click the C1AccordionItem icon to add it to the C1Accordion control. Repeat this twice so that you have a total of three accordion panes.
10. Under the **Objects and Timeline** tab, select **[C1Accordion]** to reveal its list of properties under the **Properties** panel. Set the following properties:
    - Set the **Width** property to "150".
    - Set the **Height** property to "150".
11. Under the **Objects and Timeline** tab, select the first **[C1AccordionItem]** to reveal its list of properties under the **Properties** panel. Complete the following:
    - Set the **Width** property to "150".
    - Check the **Margin** property and make sure each margin is set to "0".
12. Under the **Objects and Timeline** tab, select the second**[C1AccordionItem]** to reveal its list of properties under the **Properties** panel. Complete the following:
    - Set the Width property to "150".
    - Check the **Margin** property and make sure each margin is set to "0".
13. Under the **Objects and Timeline** tab, select the third **[C1AccordionItem]** to reveal its list of properties under the **Properties** panel. Complete the following:
    - Set the **Width** property to "150".
    - Check the **Margin** property and make sure each margin is set to "0".
14. Run the project.

In Visual Studio

Complete the following steps:

1. Open the **.xaml** page in Visual Studio.
2. Place your cursor between the <Grid></Grid> tags.
3. In the Toolbox, double-click the **C1ThemeRainierOrange** icon to declare the theme. Its tags will appear as follows:

```
<c1:C1ThemeRainierOrange></c1:C1ThemeRainierOrange>
```

4. Place your cursor between the <c1:C1ThemeRainierOrange> and </c1:C1ThemeRainierOrange> tags.
5. In the Toolbox, double-click the C1Accordion icon to add the control to the project. Its tags will appear as children of the <c1:C1ThemeRainierOrange> tags. Any control that appears as a child item of that theme will adopt that theme.
6. Add Height="150" and Width="150" to the <c1:C1Accordion> tags so that the markup resembles the following:

```
      <c1:C1Accordion Height="150" Width="150">
```

7. To create the accordion items, enter the following markup between the <c1:C1Accordion> and </c1:C1Accordion> tags:

```
   <c1:C1AccordionItem Header="Panel 1"></c1:C1AccordionItem>
```

```
<c1:C1AccordionItem Header="Panel 2"></c1:C1AccordionItem>
<c1:C1AccordionItem Header="Panel 3"></c1:C1AccordionItem>
```

8. Run your project. By placing your control within the `<c1:C1ThemeRanierOrange>` tags, you have applied the theme to the control.

This Topic Illustrates the Following:

The result of this topic resembles the following image:

Changing the Accordion's Appearance

In addition to using themes to change the appearance of the Accordion control, you can completely customize the Accordion's appearance at design time and in XAML. We will set the properties for the HeaderBackground, ExpandedBackground, and MouseOverBrush.

Follow these steps to change the Accordion's appearance:

At Design Time in Visual Studio

Follow these steps:

1. Add three C1AccordionItems to your C1Accordion control (see Adding Accordion Panes to the C1Accordion Control (page 31)).
2. Select the first C1AccordionItem to view the Properties window.
3. Locate and select the HeaderBackground property.
4. Use the dropdown arrow to reveal the color-picker and choose a color for the HeaderBackground property.
5. Choose colors for the following C1AccordionItem properties:
   - ExpandedBackground
   - MouseOverBrush
6. Repeat the steps for the two remaining C1AccordionItems.
7. Run your application. Your C1Accordion control should reflect the changes you have made to its appearance.

In XAML

Follow these steps:

1. Locate the `<Grid> </Grid>` tags in your markup. Insert the following XAML markup between the tags to add a C1Accordion control and three C1AccordionItems:

```
<c1:C1Accordion>
   <c1:C1AccordionItem> </c1:C1AccordionItem>
   <c1:C1AccordionItem> </c1:C1AccordionItem>
   <c1:C1AccordionItem> </c1:C1AccordionItem>
```

```
</c1:C1Accordion>
```

2. Click in the first `<c1:C1AccordionItem>` tag and add the following XAML to set the HeaderBackground, ExpandedBackground, and MouseOverBrush properties:

```
ExpandedBackground="#FFF5E006" HeaderBackground="#FF32640C"
MouseOverBrush="#FF007C00"
```

The whole tag should appear as follows:

```
<c1:C1AccordionItem BorderBrush="Black" ExpandedBackground="#FFF5E006"
HeaderBackground="#FF32640C" MouseOverBrush="#FF007C00">
```

3. Click the second `<c1:C1AccordionItem>` tag and add the following XAML to set the HeaderBackground, ExpandedBackground, and the MouseOverBrush properties:

```
 ExpandedBackground="#FFF5E006" FocusBrush="#FFC900F2"
HeaderBackground="#FF0071E4" MouseOverBrush="#FF007C00"
```

The whole tag should appear as follows:

```
<c1:C1AccordionItem ExpandedBackground="#FFF5E006"
HeaderBackground="#FF0071E4" MouseOverBrush="#FF007C00">
```

4. Click in the third `<c1:C1AccordionItem>` tag and add the following XAML to set the HeaderBackground, ExpandedBackground, and MouseOverBrush properties:

```
MouseOverBrush="#FF8E45FF" HeaderBackground="#FFF5C318"
ExpandedBackground="#FFF88325"
```

The whole tag should appear as follows:

```
<c1:C1AccordionItem MouseOverBrush="#FF8E45FF"
HeaderBackground="#FFF5C318" ExpandedBackground="#FFF88325">
```

5. Press F5 to run your application. The C1Accordion control should appear as in the following image:

Filling Out the Accordion's Height

The default behavior of an accordion pane (C1AccordionItem) is to fill only to the height (for accordions that expand up or down) or width (for accordions that expand right or left) of its contents. If you want the C1Accordion control to fill to a specific height or width, you will have to set the height or width of the control and then set the control's Fill property to **True**.

At Design Time in Blend

To set the Fill property in Blend, complete the following steps:

1. Add three accordion panes to your C1Accordion control (see Adding Accordion Panes to the C1Accordion Control (page 31)).

2. Click the C1Accordion control once to select it.
3. Under the **Properties panel**, complete the following:
   - Select the **Fill** check box.
   - Set the **Height** property to "200".
4. Run the program.

In XAML

To set the Fill property to **True** in XAML, complete the following:

1. Add `Fill="True"` to the `<c1:C1Accordion>` tag so that it appears similar to the following:

```
<c1:C1Accordion Fill="True">
```

2. Add `Height="200"` to the `<c1:C1Accordion>` tag so that it appears similar to the following:

```
<c1:C1Accordion Fill="True" Height="200">
```

In Code

To set the Fill property in code, complete the following steps:

1. Add `x:Name="C1Accordion1"` to the `<c1:C1Accordion>` tag so that the object will have a unique identifier for you to reference in code.
2. Add and `Height="200"` to the `<c1:C1Accordion>` tag. This ensures that each accordion pane expands to fill a height of 200 pixels.
3. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic
     ```
     C1Accordion1.Fill = True
     ```
   - C#
     ```
     C1Accordion1.Fill = true;
     ```
4. Run the program.

Book

Present information using a familiar book metaphor with **C1Book**, a page-turning book control for innovative Silverlight navigation. With **C1Book** you can present **UIElement** objects as if they were pages in a regular paper book. You can see two elements at a time, add shadows, turn pages with the mouse, and more with **ComponentOne Book™ for Silverlight**.

Getting Started

Get started with the following topics:

- Key Features (page 43)

- Quick Start (page 43)

- Task-Based Help (page 106)

Book for Silverlight Key Features

ComponentOne Book for Silverlight allows you to create customized, rich applications. Make the most of **Book for Silverlight** by taking advantage of the following key features:

- **Familiar Book Metaphor**

  Book enables you to present information innovatively using a familiar mental model – that of a book. But **Book for Silverlight** is not a typical static book, it's dynamic and interactive, and it takes the familiar metaphor further using Silverlight.

- **Real Book-like Visuals**

  Book enables you to customize the look and feel of the book pages; for example, show page folds and display inner and outer shadows. It not only looks like a book, but can be interacted with like a book.

- **Flexible Data Binding**

  C1Book is an **ItemsControl**, so you can bind it to any data source. Each item in the data source can be a **UIElement** or a generic object that gets converted into a **UIElement** using templates.

- **Custom Styles for Book Pages and Cover**

  Book supports different templates for odd and even pages, and it is possible to define custom pages like the cover.

Book for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **Book for Silverlight**. In this quick start you'll create a new project, add a C1Book control to your application, and customize the appearance and behavior of the control.

This example uses Microsoft Expression Blend 3 to create and customize a Silverlight application, but you can also complete the following steps in Visual Studio 2008. You will create a simple project using a C1Book control. The C1Book control will be used to create a book that contains a variety of content.

Step 1 of 3: Creating the Book Application

In this step you'll create a Silverlight application in Microsoft Expression Blend using **Book for Silverlight**. When you add a C1Book control to your application, you'll have a complete, functional book-like interface that you can add images, controls, and other elements to. To set up your project and add C1Book controls to your application, complete the following steps:

1. In Expression Blend, select **File | New Project**.

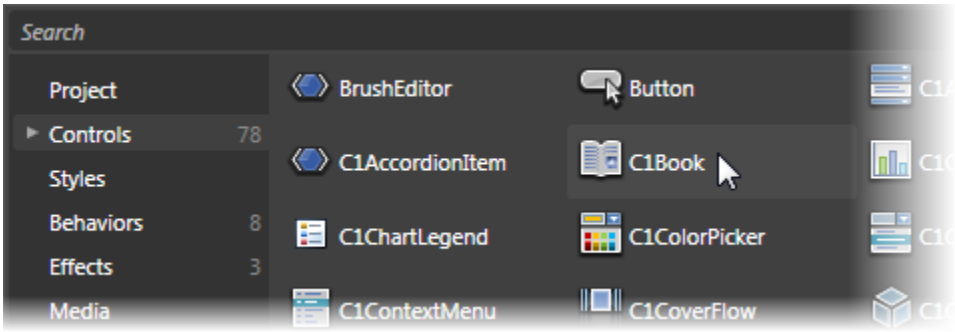
2. In the **New Project** dialog box, select the **Silverlight** project type in the left pane and in the right-pane select **Silverlight Application + Website**. Enter a **Name** and **Location** for your project, select a **Language** in the drop-down box, and click **OK**.

   A new application will be created and should open with the **MainPage.xaml** file displayed in Design view.

3. Navigate to the **Projects** window and right-click the **References** folder in the project files list. In the context menu choose **Add Reference**, locate and select the **C1.Silverlight.dll** and **C1.Silverlight.Extended.dll** assemblies, and click **Open**. The dialog box will close and the references will be added to your project.

4. In the Toolbox click on the **Assets** button (the double chevron icon) to open the **Assets** dialog box.

5. In the **Asset Library** dialog box, choose the **Controls** item in the left pane, and then click on the **C1Book** icon in the right pane:



   The **C1Book** icon will appear in the Toolbox under the **Assets** button.

6. Click once on the design area of the **UserControl** to select it. Unlike in Visual Studio, in Blend you can add Silverlight controls directly to the design surface as in the next step.

7. Double-click the **C1Book** icon in the Toolbox to add the control to the panel.

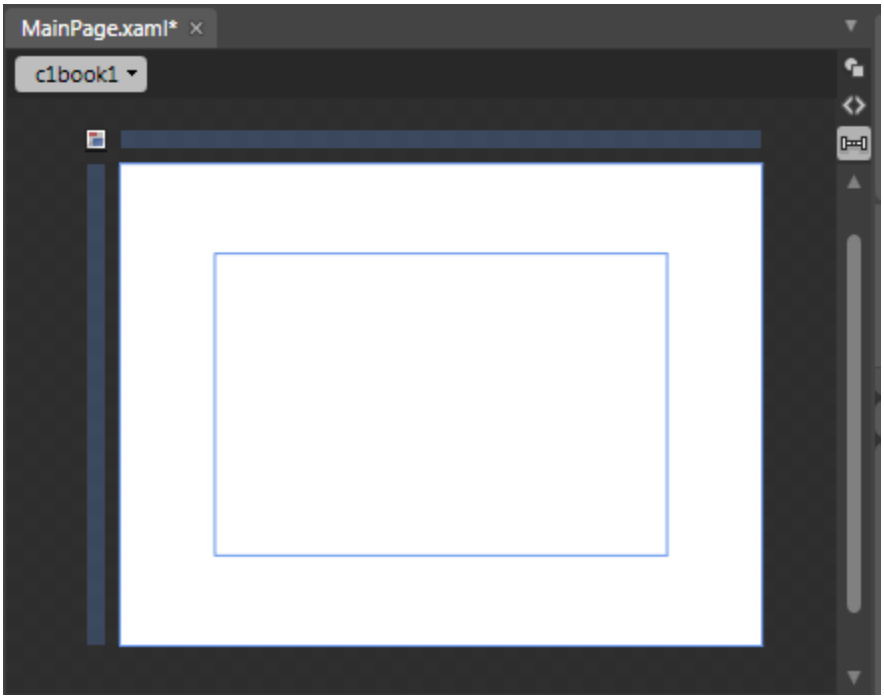
8. Click once on the **C1Book** control in design view, navigate to the Properties window and set the following properties:

   - Set **Name** to "c1book1" to give the control a name so it is accessible in code.
   - Set **Width** to "450" and **Height** to "300".
   - Set **HorizontalAlignment** and **VerticalAlignment** to **Center** to center the control in the panel.
   - Check the **IsFirstPageOnTheRight** check box to set the first page to appear on the right side.
   - Set the **TurnInterval** property to 600 to increase the time taken for the page turn animation.

   The XAML will appear similar to the following:

```
<c1:C1Book x:Name="c1book1" Height="300" HorizontalAlignment="Center"
VerticalAlignment="Center" Width="450" IsFirstPageOnTheRight="True"
TurnInterval="600"/>
```

   The page's Design view should now look similar to the following image (with the C1Book control selected on the form):



You've successfully set up your application's user interface, but C1Book control currently contains no content. In the next step you'll add content to the C1Book control, and then you'll add code to your application to add functionality to the control.

Step 2 of 3: Adding Content to the Book Control

In this step you'll add content to the C1Book control in design-time, XAML markup, and code. You'll add standard Microsoft controls and content to create a virtual book with several pages that can be turned. To customize your project and add content to the C1Book control in your application, complete the following steps:

1. Click once on the C1Book control to select it.
2. Navigate to the Toolbox and double-click the **TextBlock** control to add it to the project.

   Notice in XAML view that the **TextBlock** control was added within the C1Book control's markup.
3. Select the **TextBlock** in the **Objects and Timeline** window, navigate to the Properties window, and set the following properties:
   - **Text** to "Hello World!"
   - **HorizontalAlignment** to **Center**
   - **VerticalAlignment** to **Center**
4. Select the C1Book in the **Objects and Timeline** window, navigate to the Toolbox, and double-click the **Button** item twice to add two button controls to the page.
5. In XAML view, update the Button controls' markup so that it appears similar to the following:

```
<Button x:Name="btnLast" Content="Last" Height="100" Width="100"
Click="btnLast_Click"/>

<Button x:Name="btnNext" Content="Next" Width="150" Height="150"
Click="btnNext_Click"/>
```

This will give the controls names so they are accessible in code, resize the controls, and add event handlers that you will add code for in the next steps.

6. In the Projects window, expand the **MainPage.xaml** item and then double-click the code file (**MainPage.xaml.vb** or **MainPage.xaml.cs**).
7. In Code view, add the following import statements to the top of the page:
   - Visual Basic
     ```
     Imports C1.Silverlight
     Imports C1.Silverlight.Extended
     ```
   - C#
     ```
     using C1.Silverlight;
     using C1.Silverlight.Extended;
     ```
8. Add the following code just after the page's constructor to add handlers to the **Click** events:
   - Visual Basic
     ```
     Private Sub btnLast_Click(ByVal sender as Object, ByVal e as
     System.Windows.RoutedEventArgs)
         Me.c1book1.CurrentPage = Me.c1book1.CurrentPage - 1
     End Sub
     Private Sub btnNext_Click(ByVal sender as Object, ByVal e as
     System.Windows.RoutedEventArgs)
         Me.c1book1.CurrentPage = Me.c1book1.CurrentPage + 2
     End Sub
     ```
   - C#
     ```
     private void btnLast_Click(object sender,
     System.Windows.RoutedEventArgs e)
     {
         this.c1book1.CurrentPage = this.c1book1.CurrentPage - 1;
     }
     private void btnNext_Click(object sender,
     System.Windows.RoutedEventArgs e)
     {
         this.c1book1.CurrentPage = this.c1book1.CurrentPage + 1;
     }
     ```

   The buttons will now allow the user to navigate to the last or next page at run time.
9. Add code to the page's constructor so that it appears similar to the following:
   - Visual Basic
     ```
     Public Sub New()
         InitializeComponent()
         Dim txt1 as New TextBlock
         txt1.VerticalAlignment = VerticalAlignment.Center
         txt1.HorizontalAlignment = HorizontalAlignment.Center
         txt1.Text = "The End."
         c1book1.Items.Add(txt1)
     End Sub
     ```
   - C#
     ```
     public MainPage()
     {
         InitializeComponent();
         TextBlock txt1 = new TextBlock();
         txt1.VerticalAlignment = VerticalAlignment.Center;
         txt1.HorizontalAlignment = HorizontalAlignment.Center;
     ```

```
    txt1.Text = "The End.";
    c1book1.Items.Add(txt1);
}
```

This will add a **TextBlock** to the C1Book control in code.

10. Save your project and return to XAML view.
11. In XAML view, add the following markup just after the `<Button x:Name="btnNext"/>` tag:

- XAML to add

```
<Grid x:Name="checkers" Background="White" ShowGridLines="True">
    <Grid.RowDefinitions>
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
    </Grid.ColumnDefinitions>
    <Rectangle Fill="Red" Grid.Row="0"  Grid.Column="0" Margin="5" />
    <Rectangle Fill="Black" Grid.Row="0" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="0" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="0" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="3" Margin="5"  />
</Grid>
<Grid x:Name="checkers2" Background="White" ShowGridLines="True">
    <Grid.RowDefinitions>
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
        <RowDefinition Height=".25*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
        <ColumnDefinition Width=".25*" />
    </Grid.ColumnDefinitions>
    <Rectangle Fill="Red" Grid.Row="0" Grid.Column="0" Margin="5" />
    <Rectangle Fill="Black" Grid.Row="0" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="0" Grid.Column="2" Margin="5"  />
   <Rectangle Fill="Black" Grid.Row="0" Grid.Column="3" Margin="5"  />
```

```
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="1" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="1" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="2" Grid.Column="3" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="0" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="1" Margin="5"  />
    <Rectangle Fill="Black" Grid.Row="3" Grid.Column="2" Margin="5"  />
    <Rectangle Fill="Red" Grid.Row="3" Grid.Column="3" Margin="5"  />
</Grid>
```

This markup will add two Grids with multiple Rectangle elements. This markup demonstrates how you can add multiple controls to a page of the C1Book control as long as the controls are all in one panel, such as a Grid or StackPanel.

In this step you completed adding content to the C1Book control. In the next step you'll run the application and observe run-time interactions.

Step 3 of 3: Running the Book Application

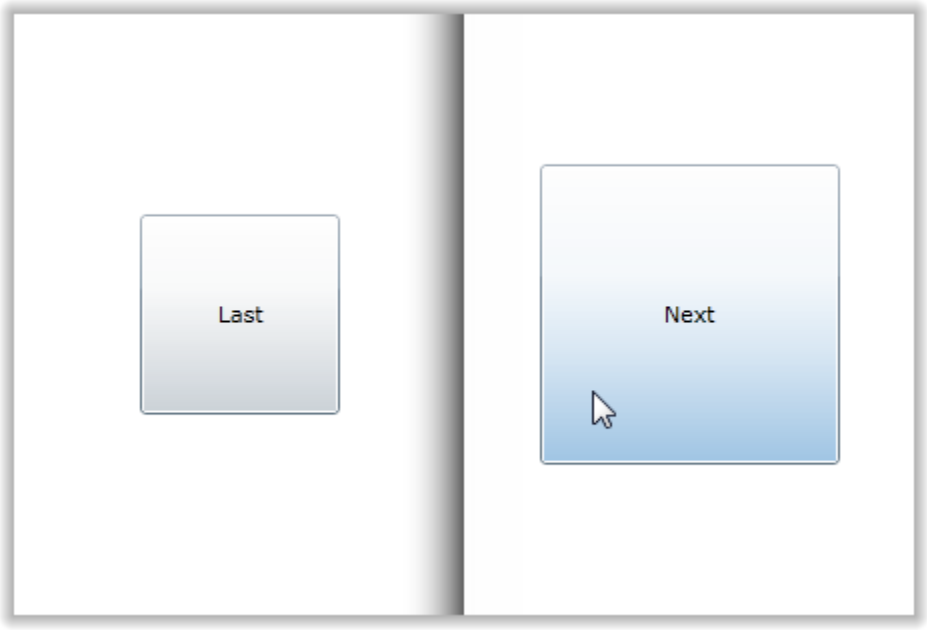
Now that you've created a Silverlight application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **Book for Silverlight**'s run-time behavior, complete the following steps:

1. From the **Project** menu, select **Run Project** to view how your application will appear at run time.

   The application will appear similar to the following:

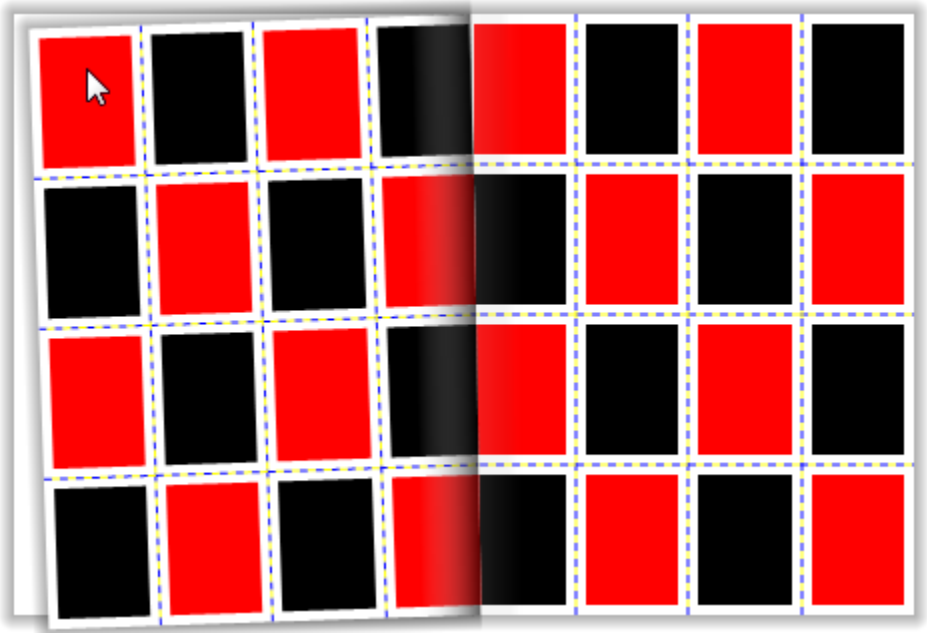
You set the IsFirstPageOnTheRight property so that only one page is initially visible. Notice that when you hover over the lower or upper-right corner of the C1Book control the page folds back slightly to prompt you to turn the page; see Book Zones (page 52) for more information.

2. Click the upper-right corner of the page and notice that the page turns and the second and third pages are visible:



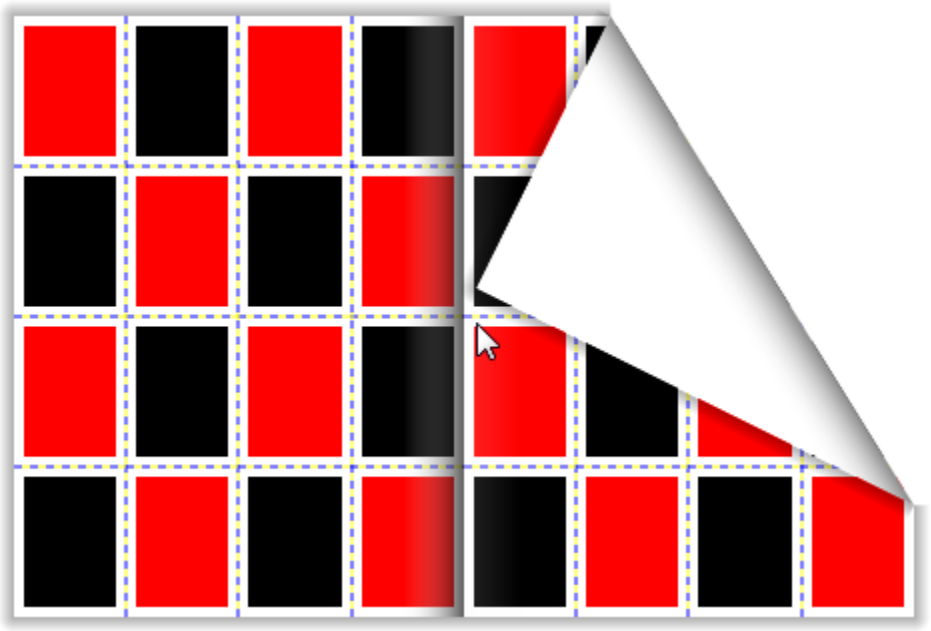
You customized the time it takes for the page to turn by setting the TurnInterval property.

3. Click the **Next** button. The next pages are displayed:

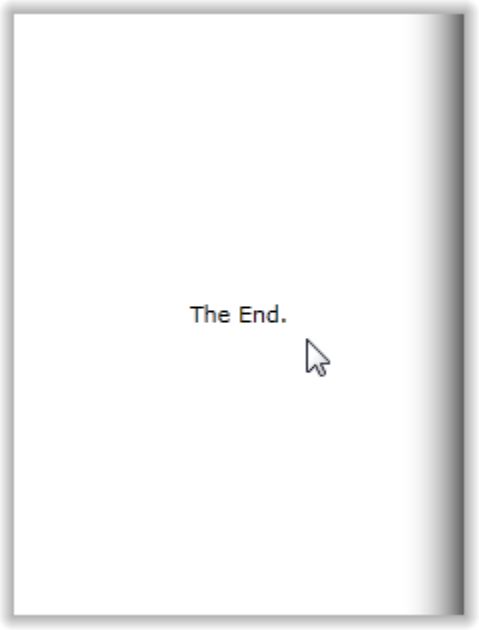


Note that you can return to the previous page by clicking the left top or bottom corner.

4. Click and drag the top-right page corner to the left to turn to the next page:



Notice that the last page contains the **TextBlock** content that was added in code:



Congratulations! You've completed the **Book for Silverlight** quick start and created a simple Silverlight application, added and customized a **Book for Silverlight** control, and viewed some of the run-time capabilities of the control.

Book XAML Quick Reference

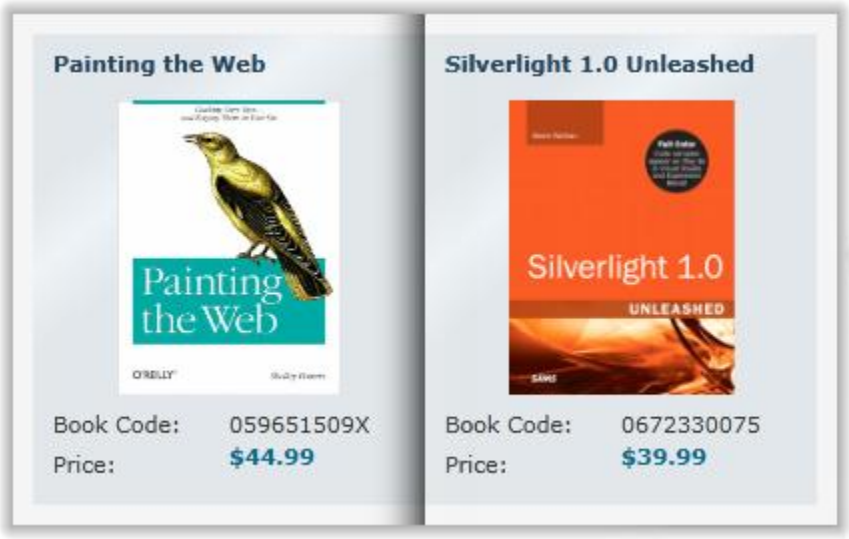
This topic is dedicated to providing a quick overview of the XAML used to create a C1Book control with five items and its current page set to 3 (page 4, since this is on a zero-based index). For more information, see the CoverFlow for Silverlight Task-Based Help (page 169) section.

```
<c1:C1Book x:Name="c1book1" Height="300" Width="450" CurrentPage="3">>
                    <TextBlock Text="Hello World! 1"/>
                    <TextBlock Text="Hello World! 2"/>
                    <TextBlock Text="Hello World! 3"/>
                    <TextBlock Text="Hello World! 4"/>
                    <TextBlock Text="Hello World! 5"/>
</c1:C1Book>
```

Working with Book for Silverlight

ComponentOne Book for Silverlight includes the C1Book control, a simple book control that acts as a container, allowing you to add controls, images, and more in a familiar book format. When you add the C1Book control to a XAML window, it exists as a container, similar to a panel, that can be customized and include added content.

The control's interface looks similar to the following image:



Basic Properties

ComponentOne Book for Silverlight includes several properties that allow you to set the functionality of the control. Some of the more important properties are listed below. Note that you can see Book for Silverlight Appearance Properties (page 58) for more information about properties that control appearance.

The following properties let you customize the C1Book control:

Property	Description
CurrentPage	Gets or sets the current page that is displayed.
CurrentZone	Gets the zone of the book under the mouse. See Book Zones (page 52) for more information.
FoldSize	Gets or sets the size of the fold (in pixels). See Page Fold Size (page 54) for more information.
IsFirstPageOnTheRight	Gets or sets whether the first page of the book is displayed in the right side of the book. See First Page Display (page 56) for more information.
IsMouseOver	Returns **true** if the mouse is over the control.
LeftPageTemplate	Gets or sets the template of the page shown in the left side of the book.
PageFoldAction	Gets or sets the action that will raise the turn animation. See Page Turning Options (page 55) for more information.
RightPageTemplate	Gets or sets the template of the page shown in the right side of the book.
ShowInnerShadows	Gets or sets whether the inner shadows are shown. See Page Shadows (page 57) for more information.
ShowOuterShadows	Gets or sets whether the outer shadows are shown. See Page Shadows (page 57) for more information.
ShowPageFold	Gets or sets whether the fold is displayed. See Page Fold Visibility (page 55) for more information.
TurnInterval	Gets or sets the amount (in milliseconds) of time of the turn animation.

Basic Events

ComponentOne Book for Silverlight includes several events that allow you to set interaction and customize the control. Some of the more important events are listed below.

The following events let you customize the C1Book control:

Event	Description
CurrentPageChanged	Fires when the CurrentPage property changes.
CurrentZoneChanged	Occurs when current zone changed. For more information, see Book Zones (page 52).
DragPageFinished	Occurs when page ends of being dragged.
DragPageStarted	Occurs when the page starts to be dragged.
IsMouseOverChanged	Event raised when the IsMouseOver property has changed.

Book Zones

The C1Book control includes several zones. These zones let you customize what happens when users interact with various sections of the control. You can use the CurrentZone property to get the user's current zone and you can use the CurrentZoneChanged event to customize what happens when users move to a different zone.

There are six separate zones in the C1Book control. For an illustration of each zone, note the mouse's position in each of the images in the following table:

Zone	Description	Example

Out	Specifies the zone outside the borders of the book.	
BottomLeft	Specifies bottom left fold zone.	
TopLeft	Specifies top left fold zone.	

Center	Specifies the center of the book (no fold zone).	
TopRight	Specifies top right fold zone.	
BottomRight	Specifies bottom right fold zone.	

Page Fold Size

One way to customize the appearance of the book as users flip pages is by setting the size of the page fold using the FoldSize property. Page folds, which appear when users mouse over certain book zones (page 52), serve as a cue to users that a page can be turned.

When you set the FoldSize property you will be setting the size of all of the page folds – this includes the right top and bottom folds and the left top and bottom folds. So for example, when FoldSize is **40**, the bottom left and bottom right folds appear similar to the following image:

If set to a higher number, the folds will appear more prominent. When FoldSize is **80**, the bottom left and bottom right folds appear similar to the following image:

Page Fold Visibility

By default, users will see a page fold when their mouse is over certain book zones (page 52). If you choose to, however, you can change the page fold visibility. You can set the ShowPageFold property to any of the values in the following table to determine how users interact with the C1Book control:

Value	Description
OnMouseOver	The fold will be visible when the user drags the mouse over the edge of the page. This is the default setting.
Never	The fold will not be visible.
Always	The fold will always be visible.

Page Turning Options

By default, when users click once on a page fold the book will progress to the previous or next page. You can customize how pages turn on page click using the PageFoldAction property. For example you can set

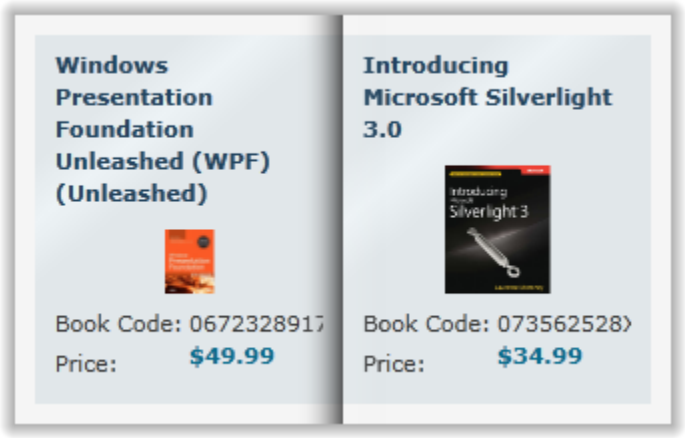
PageFoldAction so that users must double-click on the page fold to turn the page, or you can prevent page turning on mouse click altogether, requiring that users perform a drag-and-drop operation on the page fold to turn a page.

You can set the PageFoldAction property to any of the values in the following table to determine how users interact with the C1Book control:

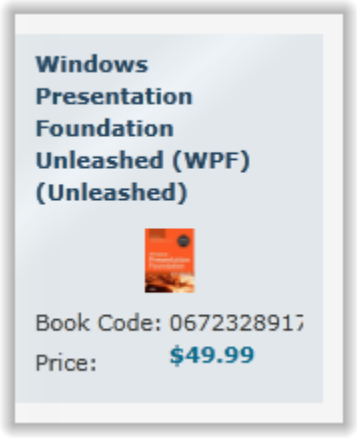
Value	Description
TurnPageOnClick	Turn the page when the user clicks the page fold.
TurnPageOnDoubleClick	Turn the page when the user double clicks the page fold.
None	Turn page when user drags the page fold across the book.

First Page Display

By default, the first page in the C1Book control is displayed on the left hand side. This makes it appear as if the book is open:



If you choose, however, you can change the display of the first page to appear on the right by setting the IsFirstPageOnTheRight property to **True**. When the first page is set to display on the right side, it will appear similar to a cover, as if the book is closed:



Page Shadows

The C1Book control includes shadows that give the control a three-dimensional appearance. The control includes inner and outer shadows. Inner shadows appear in the center of the book, and behind the right page when turning. Outer shadows appear around the outside of the control and behind the left page when turning. For example, the following image illustrates inner and outer shadows:

If you do not want shadows to be displayed, you can change their visibility by setting the ShowInnerShadows and ShowOuterShadows properties to **False**.

Book Navigation

At run time users can navigate through the C1Book control using the mouse. Users can click in one of the book zones (page 52) or can perform a drag-and-drop operation to turn the page. The C1Book control includes navigation-related methods, properties, and events to make it easier for you to determine what page a user is currently viewing, and to set the application's actions as users navigate through a book.

The CurrentPage property gets or sets the page that is currently displayed at run time. Note that when you turn a page, the page displayed on the left of the two-page spread will be the CurrentPage. Page numbering begins with **0** and page 0 is always displayed on the left. So if IsFirstPageOnTheRight property is set to **True**, the first initial page of the book displayed on the right side will be page 1 with a hidden page 0 on the left side.

You can set the displayed page using the CurrentPage property, but you can also use the TurnPage method to change the current page at run time. The TurnPage method turns to book pages forward or back one page.

You can use the CurrentPageChanged event to specify actions that happen when the current page is changed. You can also use the DragPageStarted and DragPageFinished events to specify actions to take when the user turns the page using a drag-and-drop operation.

Book for Silverlight Layout and Appearance

The following topics detail how to customize the C1Book control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the grid and take advantage of Silverlight's XAML-based styling. You can also use templates to format and lay out the control and to customize the control's actions.

Book for Silverlight Appearance Properties

ComponentOne Book for Silverlight includes several properties that allow you to customize the appearance of the control. You can change the color, border, and height of the control. The following topics describe some of these appearance properties.

Color Properties

The following properties let you customize the colors used in the control itself:

Property	Description
Background	Gets or sets a brush that describes the background of a control. This is a dependency property.
Foreground	Gets or sets a brush that describes the foreground color. This is a dependency property.

Border Properties

The following properties let you customize the control's border:

Property	Description
BorderBrush	Gets or sets a brush that describes the border background of a control. This is a dependency property.
BorderThickness	Gets or sets the border thickness of a control. This is a dependency property.

Size Properties

The following properties let you customize the size of the control:

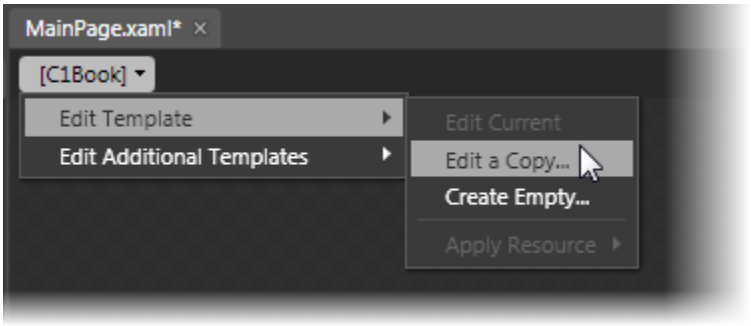
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
MinHeight	Gets or sets the minimum height constraint of the element. This is a dependency property.
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

Book Templates

One of the main advantages to using a Silverlight control is that controls are "lookless" with a fully customizable user interface. Just as you design your own user interface (UI), or look and feel, for Silverlight applications, you can provide your own UI for data managed by **ComponentOne Book for Silverlight**. Extensible Application Markup Language (XAML; pronounced "Zammel"), an XML-based declarative language, offers a simple approach to designing your UI without having to write code.

Accessing Templates

You can access templates in Microsoft Expression Blend by selecting the C1Book control and, in the menu, selecting **Edit Control Parts (Templates)**. Select **Edit a Copy** to create an editable copy of the current template or **Create Empty**, to create a new blank template.



Note: If you create a new template through the menu, the template will automatically be linked to that template's property. If you manually create a template in XAML you will have to link the appropriate template property to the template you've created.

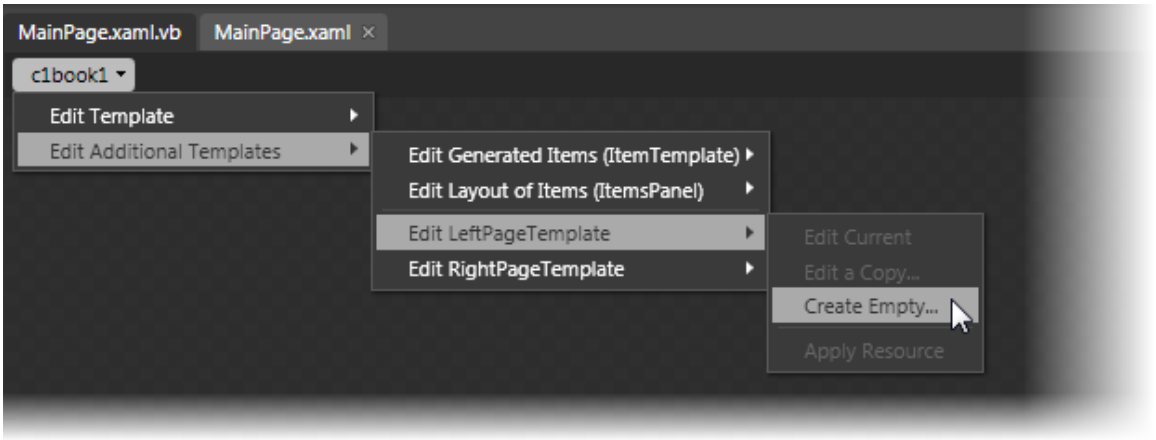
Note that you can use the Template property to customize the template.

Page Templates

ComponentOne Book for Silverlight's book control includes two page templates: the LeftPageTemplate and the RightPageTemplate. These templates control the appearance and layout of the left and right pages of the book. These templates function as master pages -- items that you add to these page templates will appear on every page that template effects. This is useful, for example, if you want to add a watermark or company logo to every page without adding it to every single page in the book individually.

Accessing Page Templates

You can access page templates in Microsoft Expression Blend by selecting the C1Book control and, in the menu, selecting **Edit Additional Templates**. Choose **Edit LeftPageTemplate** or **Edit RightPageTemplate** and select **Create Empty** to create a new blank template.



The **Create ControlTemplate Resource** dialog box will appear allowing you to name the template and determine where to define the template. By default, the template will appear blank with an empty Grid control:

```
<ControlTemplate x:Key="NewLeftPageTemplate">
    <Grid/>
</ControlTemplate>
```

You can customize the template as you would any other **ControlTemplate**.

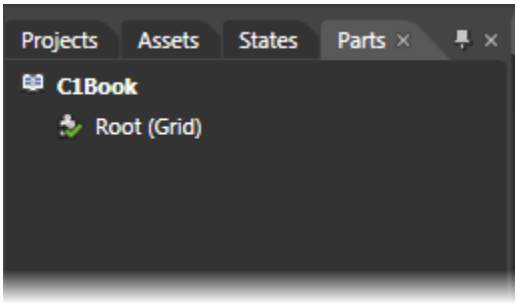
Book Styles

ComponentOne Book for Silverlight's C1Book control provides several style properties that you can use to change the appearance of the control. Some of the included styles are described in the table below:

Style	Description
FontStyle	Gets or sets the font style. This is a dependency property.
Style	Gets or sets the style used by this element when it is rendered. This is a dependency property.

Book Template Parts

In Microsoft Expression Blend, you can view and edit template parts by creating a new template (for example, click the C1Book control to select it and choose **Object | Edit Template | Edit a Copy**). Once you've created a new template, the parts of the template will appear in the **Parts** window:



Note that you may have to select the **ControlTemplate** for its parts to be visible in the **Parts** window.

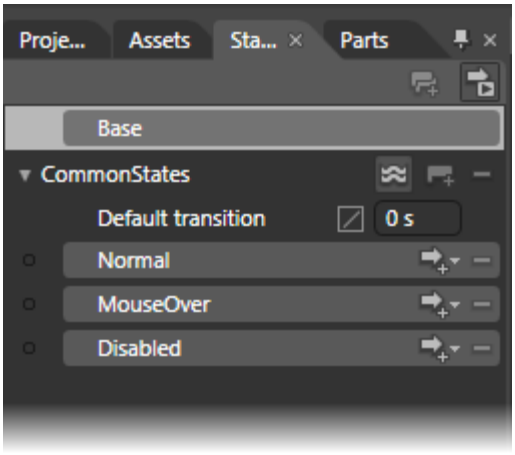
In the Parts window, you can double-click any element to create that part in the template. Once you have done so, the part will appear in the template and the element's icon in the **Parts** pane will change to indicate selection.

Template parts available in the C1Book control include:

Name	Type	Description
Root	FrameworkElement	Provides a framework of common APIs for objects that participate in Silverlight layout. Also defines APIs related to data binding, object tree, and object lifetime feature areas in Silverlight.

Book Visual States

In Microsoft Expression Blend, you can add custom states and state groups to define a different appearance for each state of your user control – for example, the visual state of the control could change on mouse over. You can view and edit visual states by creating a new template and adding a new template part (page 60). Once you've done so the available visual states for that part will be visible in the **Visual States** window:



Common states include **Normal** for the normal appearance of the item, **MouseOver** for the item on mouse over, and **Disabled** for when the item is not enabled. Focus states include **Unfocused** for when the item is not in focus and **Focused** when the item is in focus.

Book for Silverlight Tutorials

The following tutorials assume that you are familiar with programming in Visual Basic .NET or C#. The tutorials provide step-by-step instructions; no prior knowledge of **Book for Silverlight** is needed. By following the steps outlined in this section, you will be able to create projects demonstrating **Book for Silverlight** features.

You are encouraged to run the tutorial projects, and experiment with your own modifications.

Using Book for Silverlight as a Flip Calendar

The C1Book.Orientation property sets the control's horizontal or vertical orientation. By setting the Orientation property to **Vertical**, you can use the C1Book control as a flip calendar. You can find the sample for this tutorial at [Your Documents]\ComponentOne Samples\Studio for Silverlight\C1.Silverlight.Extended\CS\C1Extended_Demo.

Using Book for Silverlight as a Flip Calendar Part 1: Setting Up the Application

Follow these steps to create your application:

1. Create a new Silverlight application and name it **CalendarBook**.
2. Right-click on **References** and choose **Add Reference** from the list. Add references to **C1.Silverlight.dll** and **C1.Silverlight.Extended.dll** to your project.
3. In addition to the initial **MainPage.xaml** page, you will need to create five pages to contain the calendar components.
    a. Add a page to your project by right-clicking on **CalendarBook** and choosing **Add | New Item** from the list.
    b. Choose **Add Silverlight Page** from the **Add New Items** dialog box. Name the page **CalendarBookCover.xaml**.
    c. Follow the above steps four more times and name the pages as follows:
        - **CalendarBookBottom.xaml**
        - **CalendarBookTop.xaml**

- **CalendarBookResources.xaml**
- **CalendarTopSpiral.xaml**

4. Click the arrow next to **CalendarBookResources.xaml** and delete the **CalendarBookResources.xaml.cs** page.

Using Book for Silverlight as a Flip Calendar Part 2: Creating the Calendar

In the last part you set up the application. In this part, you will use code and XAML markup to create a calendar using the C1Book control. You will change each page that you added to your application from a `<navigation:Page>` page type to a `<UserControl>` page type both in the XAML markup and in the code.

1. Double-click on **CalendarBookResources.xaml** to open the page and switch to the XAML view. Replace the markup on the page with the following xaml markup to create your Resource Dictionary:

   - XAML to add

```
<?xml version="1.0" encoding="utf-8"?>
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">


    <!-- Spiral Styles -->
    <Style x:Key="SpiralTop" TargetType="Border">
        <Setter Property="Background">
            <Setter.Value>
                <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                    <GradientStop Color="White" Offset="1" />
                    <GradientStop Color="#FF9D9D9D" />
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
        <Setter Property="CornerRadius" Value="2,2,0,0" />
        <Setter Property="BorderBrush" Value="#FF434343" />
        <Setter Property="BorderThickness" Value="1,1,1,0" />
    </Style>
    <Style x:Key="SpiralTopDark" TargetType="Border">
        <Setter Property="Background">
            <Setter.Value>
                <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                    <GradientStop Color="#FF484848" Offset="1" />
                    <GradientStop Color="#FFC5C5C5" />
                </LinearGradientBrush>
            </Setter.Value>
```

```
            </Setter>
            <Setter Property="CornerRadius" Value="2,2,0,0" />
            <Setter Property="BorderBrush">
                <Setter.Value>
                    <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                        <GradientStop Color="#FF676767" Offset="0" />
                        <GradientStop Color="#FF1A1A1A" Offset="1" />
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
            <Setter Property="BorderThickness" Value="1,1,1,0" />
        </Style>
        <Style x:Key="SpiralBottom" TargetType="Border">
            <Setter Property="Background">
                <Setter.Value>
                    <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                        <GradientStop Color="#FF9D9D9D" Offset="1" />
                        <GradientStop Color="White" />
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
            <Setter Property="CornerRadius" Value="0,0,2,2" />
            <Setter Property="BorderBrush" Value="#FF434343" />
            <Setter Property="BorderThickness" Value="1,0,1,1" />
        </Style>
        <Style x:Key="SpiralHole" TargetType="Rectangle">
            <Setter Property="Fill" Value="#FF1D2A33" />
            <Setter Property="RadiusX" Value="2" />
            <Setter Property="RadiusY" Value="2" />
        </Style>


    </ResourceDictionary>
```

2. Open your **MainPage.xaml** page by double-clicking on it. Replace the markup on the page with the following xaml markup to create the **C1Book** control and add some content to the control:

   - XAML to add

```
<UserControl x:Class="CalendarBook.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    xmlns:local="clr-namespace:CalendarBook"
    mc:Ignorable="d"
    d:DesignHeight="350" d:DesignWidth="475" >

    <UserControl.Resources>
        <Style x:Key="SpiralTopDark" TargetType="Border">
            <Setter Property="Background">
                <Setter.Value>
                    <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                        <GradientStop Color="#FF484848" Offset="1" />
                        <GradientStop Color="#FFC5C5C5" />
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
            <Setter Property="CornerRadius" Value="2,2,0,0" />
            <Setter Property="BorderBrush" Value="#FF1A1A1A" />
            <Setter Property="BorderThickness" Value="1,1,1,0" />
        </Style>
        <Style x:Key="C1BookItemStyle" TargetType="c1:C1BookItem">
            <Setter Property="Foreground" Value="#FF000000" />
            <Setter Property="HorizontalContentAlignment"
Value="Stretch" />
            <Setter Property="VerticalContentAlignment" Value="Stretch"
/>
            <Setter Property="Background" Value="White" />
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="c1:C1BookItem">
                        <Grid Background="{x:Null}">
                            <ContentControl x:Name="ContentControl">
                                <ContentControl.Template>
                                    <ControlTemplate>
                                        <ContentPresenter />
```

```
                                    </ControlTemplate>
                                </ContentControl.Template>
                                <ContentPresenter
Cursor="{TemplateBinding Cursor}" HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}" Margin="{TemplateBinding Padding}"
VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
Content="{TemplateBinding Content}" ContentTemplate="{TemplateBinding
ContentTemplate}" />
                            </ContentControl>
                        </Grid>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
    </UserControl.Resources>


    <Grid x:Name="LayoutRoot" Background="White">
        <local:CalendarTopSpiral VerticalAlignment="Center" />
        <c1:C1Book x:Name="calendar2010" Orientation="Vertical"
PageFoldAction="TurnPageOnClick" IsFirstPageOnTheRight="True"
MinWidth="310" MaxWidth="500" MaxHeight="500" Grid.RowSpan="2"
ItemContainerStyle="{StaticResource C1BookItemStyle}">
            <local:CalendarBookCover d:IsHidden="True" />
        </c1:C1Book>
    </Grid>
</UserControl>
```

3. View **MainPage.xaml.cs** by right-clicking on the page and choosing **View Code** from the list. Add the following statements to the top of the page:

   - Visual Basic

     ```
     Imports System.Globalization
     Imports System.Windows.Threading
     ```

   - C#

     ```
     using System.Globalization;
     using System.Windows.Threading;
     ```

4. Add the following code directly after the **InitializeComponent()** method:

   - Visual Basic

     ```
     For Each month As var In MonthData.Months
          Dim top As var = New CalendarBookTop
          top.DataContext = month
          AddHandler top.MonthRequested, AddressOf Me.top_MonthRequested
     ```

```
        calendar2010.Items.Add(top)
        Dim bottom As var = New CalendarBookBottom
        bottom.DataContext = month
        calendar2010.Items.Add(bottom)
    Next
    UnknownUnknownPublic Class MonthData

        Public Sub New(ByVal index As Integer)
            MyBase.New
            Day = New DateTime(2010, index, 1)
            Index = index
            Dim text As var = Day.ToString("MMMM")
            Name = (text(0).ToString.ToUpper + text.Substring(1).ToLower)
        End Sub

        Public Property Day As DateTime
        End Property

        Public Property Name As String
        End Property

        Public Property Index As Integer
        End Property

        Public Shared _months As List(Of MonthData) = New List(Of
    MonthData)

        Public Shared ReadOnly Property Months As List(Of MonthData)
            Get
                If (_months.Count = 0) Then
                    Dim i As Integer = 1
                    Do While (i <= 12)
                        _months.Add(New MonthData(i))
                        i = (i + 1)
                    Loop
                End If
                Return _months
            End Get
```

```
    End Property
End Class
Unknown

    Private Sub top_MonthRequested(ByVal sender As Object, ByVal e As
MonthEventArgs)
        Dim timer As var = New DispatcherTimer()
{Interval=TimeSpan.FromSeconds(0.05Unknown}
        timer.Tick = (timer.Tick + s)
        e2
        Dim index As var = (e.Month.Index * 2)
        If ((calendar2010.CurrentPage = index)  _
                    OrElse (calendar2010.CurrentPage  _
                    = (index - 1))) Then
            timer.Stop
        Else
            calendar2010.TurnPage((index > calendar2010.CurrentPage))
        End If

        timer.Start
    End Sub
```

- C#

```
foreach (var month in MonthData.Months)
            {
                var top = new CalendarBookTop();
                top.DataContext = month;
                top.MonthRequested += new
EventHandler<MonthEventArgs>(top_MonthRequested);
                calendar2010.Items.Add(top);

                var bottom = new CalendarBookBottom();
                bottom.DataContext = month;
                calendar2010.Items.Add(bottom);
            }
        }

        void top_MonthRequested(object sender, MonthEventArgs e)
        {
```

```
            var timer = new DispatcherTimer { Interval =
TimeSpan.FromSeconds(0.05) };
            timer.Tick += (s, e2) =>
            {
                var index = e.Month.Index * 2;
                if (calendar2010.CurrentPage == index ||
calendar2010.CurrentPage == index - 1)
                {
                    timer.Stop();

                }
                else
                {
                    calendar2010.TurnPage(index >
calendar2010.CurrentPage);
                }
            };
            timer.Start();
        }
    }

    public class MonthData
    {
        public MonthData(int index)
        {
            Day = new DateTime(2010, index, 1);
            Index = index;

            var text = Day.ToString("MMMM");
            Name = text[0].ToString().ToUpper() +
text.Substring(1).ToLower();
        }

        public DateTime Day { get; set; }
        public string Name { get; set; }
        public int Index { get; set; }

        public static List<MonthData> _months = new List<MonthData>();
        public static List<MonthData> Months
```

```
        {
            get
            {
                if (_months.Count == 0)
                {
                    for (int i = 1; i <= 12; i++)
                    {
                        _months.Add(new MonthData(i));
                    }
                }
                return _months;
            }
        }
    }
}
```

5. Double-click on **CalendarBookTop.xaml** to view the page. Add the following xaml markup to the page, replacing the existing markup:

   - XAML to add

```
<?xml version="1.0" encoding="utf-8"?>
<UserControl x:Class="CalendarBook.CalendarBookTop"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml">
    <UserControl.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="CalendarBookResources.xaml"
/>
            </ResourceDictionary.MergedDictionaries>
            <Style x:Key="MonthButton" TargetType="Button">
                <Setter Property="Background" Value="#FF0C2934" />
                <Setter Property="FontFamily" Value="Times New Roman"
/>
                <Setter Property="Foreground" Value="#ff10657a" />
                <Setter Property="Padding" Value="1 2 1 1" />
                <Setter Property="BorderThickness" Value="1" />
                <Setter Property="BorderBrush">
                    <Setter.Value>
                        <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
```

```
                            <GradientStop Color="#FFA3AEB9" Offset="0"
/>
                            <GradientStop Color="#FF8399A9"
Offset="0.375" />
                            <GradientStop Color="#FF718597"
Offset="0.375" />
                            <GradientStop Color="#FF617584" Offset="1"
/>
                        </LinearGradientBrush>
                    </Setter.Value>
                </Setter>
                <Setter Property="Template">
                    <Setter.Value>
                        <ControlTemplate TargetType="Button">
                            <Grid>
                                <VisualStateManager.VisualStateGroups>
                                    <VisualStateGroup
x:Name="CommonStates">
                                        <VisualState x:Name="Normal" />
                                        <VisualState
x:Name="MouseOver">
                                            <Storyboard>

<ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
Duration="00:00:00.0010000" Storyboard.TargetName="rectangle"
Storyboard.TargetProperty="(UIElement.Visibility)">

<DiscreteObjectKeyFrame KeyTime="00:00:00">

<DiscreteObjectKeyFrame.Value>

<Visibility>Visible</Visibility>

</DiscreteObjectKeyFrame.Value>

</DiscreteObjectKeyFrame>

</ObjectAnimationUsingKeyFrames>
                                            </Storyboard>
                                        </VisualState>
                                        <VisualState x:Name="Pressed"
/>
                                        <VisualState x:Name="Disabled">
```

```
                                        <Storyboard>

<DoubleAnimationUsingKeyFrames
Storyboard.TargetName="DisabledVisualElement"
Storyboard.TargetProperty="Opacity">

<SplineDoubleKeyFrame KeyTime="0" Value=".55" />

</DoubleAnimationUsingKeyFrames>
                                        </Storyboard>
                                    </VisualState>
                                </VisualStateGroup>
                                <VisualStateGroup
x:Name="FocusStates">
                                    <VisualState x:Name="Focused">
                                        <Storyboard>

<DoubleAnimationUsingKeyFrames
Storyboard.TargetName="FocusVisualElement"
Storyboard.TargetProperty="Opacity">

<SplineDoubleKeyFrame KeyTime="0" Value="1" />

</DoubleAnimationUsingKeyFrames>
                                        </Storyboard>
                                    </VisualState>
                                    <VisualState x:Name="Unfocused"
/>
                                </VisualStateGroup>
                            </VisualStateManager.VisualStateGroups>
                            <Grid.RowDefinitions>
                                <RowDefinition Height="Auto" />
                                <RowDefinition />
                            </Grid.RowDefinitions>
                            <Rectangle x:Name="rectangle"
Fill="White" Stroke="#FF8899A9" Grid.RowSpan="2" RadiusX="2"
RadiusY="2" StrokeThickness="2" Visibility="Collapsed">
                                <Rectangle.Effect>
                                    <DropShadowEffect
ShadowDepth="0" Color="#FF00D0FF" BlurRadius="8" />
                                </Rectangle.Effect>
                            </Rectangle>
```

```
                                <Rectangle Fill="White" Stretch="Fill"
Stroke="#FF8899A9" StrokeThickness="2" Grid.RowSpan="2" RadiusX="2"
RadiusY="2" />

                                <Path Fill="#FFD5E3EA" Stretch="Fill"
UseLayoutRounding="False" Data="M357,155.15535 L427.00003,155.15535
L427.00003,190.99835 L357,190.99835 z M286,155.15535 L355,155.15535
L355,190.99835 L286,190.99835 z M214.00002,155.15535 L284,155.15535
L284,190.99835 L214.00002,190.99835 z M143,155.15535 L212,155.15535
L212,190.99835 L143,190.99835 z M71,155.15535 L141,155.15535
L141,190.99835 L71,190.99835 z M0,155.15535 L69,155.15535 L69,164.99834
L69,182.99834 L69,184.99835 L69,185.99834 L69,190.99835 L64,190.99835
L62,190.99835 L5,190.99835 C2.2385864,190.99835 0,188.75977 0,185.99834
L0,184.99835 L0,182.99834 z M429,155.15535 L498,155.15535 L498,172.9982
L498,181.77263 C498,184.53406 488.1828,190.99835 488.1828,190.99835
L480.00003,190.99835 L429,190.99835 L429,183.99858 z M429,123.3285
L498,123.3285 L498,153.16634 L429,153.16634 z M357,123.3285
L427.00003,123.3285 L427.00003,153.16634 L357,153.16634 z M286,123.3285
L355,123.3285 L355,153.16634 L286,153.16634 z M214.00002,123.3285
L284,123.3285 L284,153.16634 L214.00002,153.16634 z M143,123.3285
L212,123.3285 L212,153.16634 L143,153.16634 z M71,123.3285
L141,123.3285 L141,153.16634 L71,153.16634 z M0,123.3285 L69,123.3285
L69,153.16634 L0,153.16634 z M429,92.496246 L498,92.496246
L498,121.33949 L429,121.33949 z M357,92.496246 L427.00003,92.496246
L427.00003,121.33949 L357,121.33949 z M286,92.496246 L355,92.496246
L355,121.33949 L286,121.33949 z M214.00002,92.496246 L284,92.496246
L284,121.33949 L214.00002,121.33949 z M143,92.496246 L212,92.496246
L212,121.33949 L143,121.33949 z M71,92.496246 L141,92.496246
L141,121.33949 L71,121.33949 z M0,92.496246 L69,92.496246 L69,121.33949
L0,121.33949 z M429,61.664253 L498,61.664253 L498,90.507492
L429,90.507492 z M357,61.664253 L427.00003,61.664253
L427.00003,90.507492 L357,90.507492 z M286,61.664253 L355,61.664253
L355,90.507492 L286,90.507492 z M214.00002,61.664253 L284,61.664253
L284,90.507492 L214.00002,90.507492 z M143,61.664253 L212,61.664253
L212,90.507492 L143,90.507492 z M71,61.664253 L141,61.664253
L141,90.507492 L71,90.507492 z M0,61.664253 L69,61.664253 L69,90.507492
L0,90.507492 z M429,30.832001 L498,30.832001 L498,59.675255
L429,59.675255 z M357,30.832001 L427.00003,30.832001
L427.00003,59.675255 L357,59.675255 z M286,30.832001 L355,30.832001
L355,59.675255 L286,59.675255 z M214.00002,30.832001 L284,30.832001
L284,59.675255 L214.00002,59.675255 z M143,30.832001 L212,30.832001
L212,59.675255 L143,59.675255 z M71,30.832001 L141,30.832001
L141,59.675255 L71,59.675255 z M0,30.832001 L69,30.832001 L69,59.675255
L0,59.675255 z M429,0 L498,0 L498,28.843246 L429,28.843246 z M357,0
L427.00003,0 L427.00003,28.843231 L357,28.843231 z M286,0 L355,0
L355,28.843231 L286,28.843231 z M214.00002,0 L284,0 L284,28.843231
L214.00002,28.843231 z M143,0 L212,0 L212,28.843231 L143,28.843231 z
M71,0 L141,0 L141,28.843246 L71,28.843246 z M0,0 L69,0 L69,28.843246
L0,28.843246 z" Grid.Row="1" Margin="4,2,4,4" />

                                <Border BorderBrush="#FF8899A9"
BorderThickness="2,2,2,1" CornerRadius="2" Background="#FFAAC3CB" />

                                <ContentPresenter
x:Name="contentPresenter" HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}" Margin="{TemplateBinding Padding}"
VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
```

```
Content="{TemplateBinding Content}" ContentTemplate="{TemplateBinding
ContentTemplate}" />
                                <Path Fill="#FFFFFFFF" Stretch="Fill"
Height="14" HorizontalAlignment="Right" VerticalAlignment="Bottom"
Width="14" UseLayoutRounding="False" Grid.Row="1" Data="M0,3
C0,1.3431457 1.3431457,0 3,0 L17,0 C18.656855,0 20,1.3431457 20,3 C20,3
3,20 3,20 C1.3431457,20 0,18.656855 0,17 z" Stroke="#FF8899A9"
StrokeThickness="2" />
                                <Rectangle
x:Name="DisabledVisualElement" Fill="#FFFFFFFF" RadiusX="3" RadiusY="3"
IsHitTestVisible="false" Opacity="0" Grid.RowSpan="2" />
                                <Rectangle x:Name="FocusVisualElement"
Stroke="#FF6DBDD1" StrokeThickness="1" RadiusX="2" RadiusY="2"
Margin="1" IsHitTestVisible="false" Opacity="0" />
                            </Grid>
                        </ControlTemplate>
                    </Setter.Value>
                </Setter>
            </Style>
        </ResourceDictionary>
    </UserControl.Resources>
    <Border Background="White" CornerRadius="10,10,5,5">
        <Grid>
            <Grid>
                <Grid.RowDefinitions>
                    <RowDefinition Height="75" />
                    <RowDefinition />
                </Grid.RowDefinitions>
                <Border BorderThickness="2,2,2,3"
CornerRadius="5,5,0,0">
                    <Border.BorderBrush>
                        <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                            <GradientStop Color="#FF425E76" Offset="0"
/>
                            <GradientStop Color="#FF788C9F"
Offset="0.651" />
                            <GradientStop Color="#FF425E76" Offset="1"
/>
                        </LinearGradientBrush>
                    </Border.BorderBrush>
                    <Border.Background>
```

```
                        <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                            <GradientStop Color="#FF245A76" Offset="0"
/>
                            <GradientStop Color="#FF6B91AA"
Offset="0.583" />
                            <GradientStop Color="#FF759AB5"
Offset="0.665" />
                        </LinearGradientBrush>
                    </Border.Background>
                </Border>
                <StackPanel Orientation="Horizontal"
HorizontalAlignment="Right" VerticalAlignment="Bottom">
                    <TextBlock Text="20" TextWrapping="Wrap"
FontSize="150" FontFamily="Times New Roman">
                        <TextBlock.Foreground>
                            <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                                <GradientStop Color="#FF134659"
Offset="0.2" />
                                <GradientStop Color="#FF0C2934"
Offset="0.3" />
                                <GradientStop Color="#FF13455A"
Offset="0.404" />
                            </LinearGradientBrush>
                        </TextBlock.Foreground>
                    </TextBlock>
                    <TextBlock Text="10" TextWrapping="Wrap"
FontSize="150" FontFamily="Times New Roman" Margin="-4,0,0,0">
                        <TextBlock.Foreground>
                            <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                                <GradientStop Color="#FFC4C9CD"
Offset="0.2" />
                                <GradientStop Color="DarkGray"
Offset="0.3" />
                                <GradientStop Color="#FFFCFCFC"
Offset="0.4" />
                            </LinearGradientBrush>
                        </TextBlock.Foreground>
                    </TextBlock>
                </StackPanel>
```

```
                <TextBlock x:Name="month" Text="{Binding Name}"
TextWrapping="Wrap" FontSize="32" FontFamily="Times New Roman"
Foreground="White" Margin="20,0,0,0" VerticalAlignment="Center"
HorizontalAlignment="Left">
                    <TextBlock.Effect>
                        <DropShadowEffect BlurRadius="2"
ShadowDepth="2" Color="#FF2C4A64" />
                    </TextBlock.Effect>
                </TextBlock>
                <Path Stretch="Fill" UseLayoutRounding="False"
Data="M0,0 L493.79904,0 C497.28104,1.4098798 499.49084,8.2246914
500,22.265001 L500,61.752743 L493.36795,60.268562 C456.3688,52.313362
416.7272,48.000004 375.5,48.000008 C191.56329,48.000004
39.188137,133.8588 12.234431,245.94026 L11.340348,250 L0,250 z">
                    <Path.Fill>
                        <LinearGradientBrush EndPoint="0.428,0.832"
StartPoint="0.246,-0.03">
                            <GradientStop Color="#0CFFFFFF" Offset="0"
/>
                            <GradientStop Color="#4CFFFFFF" Offset="1"
/>
                        </LinearGradientBrush>
                    </Path.Fill>
                </Path>
                <ItemsControl x:Name="months" Grid.Row="1"
Margin="8,8,8,31">
                    <ItemsControl.ItemsPanel>
                        <ItemsPanelTemplate>
                            <c1:C1UniformGrid Rows="2" Columns="6" />
                        </ItemsPanelTemplate>
                    </ItemsControl.ItemsPanel>
                    <ItemsControl.ItemTemplate>
                        <DataTemplate>
                            <Button Content="{Binding Month.Name}"
Style="{StaticResource MonthButton}" Margin="8" Click="Button_Click"
IsEnabled="{Binding Current}" />
                        </DataTemplate>
                    </ItemsControl.ItemTemplate>
                </ItemsControl>
            </Grid>
            <Grid x:Name="Spiral" Height="25"
VerticalAlignment="Bottom">
                <Grid.RowDefinitions>
```

```
        <RowDefinition Height="1*" />
        <RowDefinition Height="2*" />
        <RowDefinition Height="4*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="10*" />
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="2*" />
        <ColumnDefinition Width="0.5*" />
        <ColumnDefinition Width="2*" />
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="10*" />
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="2*" />
        <ColumnDefinition Width="0.5*" />
        <ColumnDefinition Width="2*" />
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="10*" />
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="2*" />
        <ColumnDefinition Width="0.5*" />
        <ColumnDefinition Width="2*" />
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="10*" />
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="2*" />
        <ColumnDefinition Width="0.5*" />
        <ColumnDefinition Width="2*" />
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="100*" />
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="2*" />
        <ColumnDefinition Width="0.5*" />
        <ColumnDefinition Width="2*" />
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="10*" />
        <ColumnDefinition Width="1*" />
        <ColumnDefinition Width="2*" />
```

```
                    <ColumnDefinition Width="0.5*" />
                    <ColumnDefinition Width="2*" />
                    <ColumnDefinition Width="1*" />
                    <ColumnDefinition Width="10*" />
                    <ColumnDefinition Width="1*" />
                    <ColumnDefinition Width="2*" />
                    <ColumnDefinition Width="0.5*" />
                    <ColumnDefinition Width="2*" />
                    <ColumnDefinition Width="1*" />
                    <ColumnDefinition Width="10*" />
                    <ColumnDefinition Width="1*" />
                    <ColumnDefinition Width="2*" />
                    <ColumnDefinition Width="0.5*" />
                    <ColumnDefinition Width="2*" />
                    <ColumnDefinition Width="1*" />
                    <ColumnDefinition Width="10*" />
                </Grid.ColumnDefinitions>
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.ColumnSpan="5" Grid.RowSpan="2" Grid.Column="1" />
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.ColumnSpan="5" Grid.RowSpan="2" Grid.Column="7" />
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.ColumnSpan="5" Grid.RowSpan="2" Grid.Column="13" />
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.ColumnSpan="5" Grid.RowSpan="2" Grid.Column="19" />
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.RowSpan="2" Grid.Column="25" Grid.ColumnSpan="5" />
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.RowSpan="2" Grid.Column="31" Grid.ColumnSpan="5" />
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.RowSpan="2" Grid.Column="37" Grid.ColumnSpan="5" />
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.RowSpan="2" Grid.Column="43" Grid.ColumnSpan="5" />
                <Border Style="{StaticResource SpiralTop}"
Grid.RowSpan="2" Grid.Column="2" Grid.Row="1" />
                <Border Style="{StaticResource SpiralTop}"
Grid.Column="4" Grid.RowSpan="2" Grid.Row="1" />
                <Border Style="{StaticResource SpiralTop}"
Grid.Column="8" Grid.RowSpan="2" Grid.Row="1" />
                <Border Style="{StaticResource SpiralTop}"
Grid.Column="10" Grid.RowSpan="2" Grid.Row="1" />
```

```
                <Border Style="{StaticResource SpiralTop}"
Grid.Column="14" Grid.RowSpan="2" Grid.Row="1" />
                <Border Style="{StaticResource SpiralTop}"
Grid.Column="16" Grid.RowSpan="2" Grid.Row="1" />
                <Border Style="{StaticResource SpiralTop}"
Grid.Column="20" Grid.RowSpan="2" Grid.Row="1" />
                <Border Style="{StaticResource SpiralTop}"
Grid.Column="22" Grid.RowSpan="2" Grid.Row="1" />
                <Border Style="{StaticResource SpiralTop}"
Grid.Column="26" Grid.RowSpan="2" Grid.Row="1" />
                <Border Style="{StaticResource SpiralTop}"
Grid.Column="28" Grid.RowSpan="2" Grid.Row="1" />
                <Border Style="{StaticResource SpiralTop}"
Grid.Column="32" Grid.RowSpan="2" Grid.Row="1" />
                <Border Style="{StaticResource SpiralTop}"
Grid.Column="34" Grid.RowSpan="2" Grid.Row="1" />
                <Border Style="{StaticResource SpiralTop}"
Grid.Column="38" Grid.RowSpan="2" Grid.Row="1" />
                <Border Style="{StaticResource SpiralTop}"
Grid.Column="40" Grid.RowSpan="2" Grid.Row="1" />
                <Border Style="{StaticResource SpiralTop}"
Grid.Column="44" Grid.RowSpan="2" Grid.Row="1" />
                <Border Style="{StaticResource SpiralTop}"
Grid.Column="46" Grid.RowSpan="2" Grid.Row="1" />
            </Grid>
        </Grid>
    </Border>
</UserControl>
```

6. Right-click on the page and select **View Code** from the list to view the code. Use the following code to replace the code on the page, starting with the class. This will change the page to being a UserControl.

   - Visual Basic

```
Public Class CalendarBookTop
     Inherits UserControl

     Public Sub New()
         MyBase.New
         InitializeComponent
         Loaded = (Loaded + AddressOf Me.CalendarBookTop_Loaded)
     End Sub

     Private Sub CalendarBookTop_Loaded(ByVal sender As Object, ByVal e
As RoutedEventArgs)
```

```
            For Each month As var In MonthData.Months
                months.Items.Add(New CalendarItem, {, Month=month,
Current=!(DataContext==monthUnknown)
            Next
        End Sub
    End Class
    UnknownUnknownPublic Class MonthEventArgs
        Inherits EventArgs

        Public Property Month As MonthData
        End Property
    End Class
    Public Class CalendarItem

        Public Property Current As Boolean
        End Property

        Public Property Month As MonthData
        End Property
    End Class
    Unknown

        Private Sub Button_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
            Dim btn As var = CType(sender,Button)
            Dim item As var = CType(btn.DataContext,CalendarItem)
            If (Not (MonthRequested) Is Nothing) Then
                MonthRequested(Me, New MonthEventArgs, {,
Month=item.Month)
            End If
        End Sub

        Public Event MonthRequested As EventHandler(Of MonthEventArgs)
```

- C#

```
    public partial class CalendarBookTop : UserControl
       {
           public CalendarBookTop()
           {
               InitializeComponent();
```

```
            Loaded += new RoutedEventHandler(CalendarBookTop_Loaded);
            }

            void CalendarBookTop_Loaded(object sender, RoutedEventArgs e)
            {
                foreach (var month in MonthData.Months)
                {
                    months.Items.Add(new CalendarItem()
                    {
                        Month = month,
                        Current = !(DataContext == month)
                    });
                }
            }

            private void Button_Click(object sender, RoutedEventArgs e)
            {
                var btn = (Button) sender;
                var item = btn.DataContext as CalendarItem;
                if (MonthRequested != null)
                {
                    MonthRequested(this, new MonthEventArgs() { Month =
item.Month });
                }
            }

            public event EventHandler<MonthEventArgs> MonthRequested;
        }

        public class MonthEventArgs
            : EventArgs
        {
            public MonthData Month { get; set; }
        }

        public class CalendarItem
        {
            public bool Current { get; set; }
```

```
        public MonthData Month { get; set; }
    }
}
```

7. Double-click on the **CalendarBookBottom.xaml** file to open it. Replace the markup on the page with the following XAML markup:

- XAML to add

```
<?xml version="1.0" encoding="utf-8"?>

<UserControl x:Class="CalendarBook.CalendarBookBottom"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:System_Windows_Controls_Primitives="clr-
namespace:System.Windows.Controls.Primitives;assembly=System.Windows.Co
ntrols" xmlns:controls="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls">


    <UserControl.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="CalendarBookResources.xaml"
/>
            </ResourceDictionary.MergedDictionaries>


            <Style x:Key="TitleDayBkgStyle" TargetType="Border">
                <Setter Property="Background">
                    <Setter.Value>
                        <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                            <GradientStop Color="#FF245A76" Offset="0"
/>
                            <GradientStop Color="#FF6B91AA"
Offset="0.583" />
                            <GradientStop Color="#FF759AB5"
Offset="0.665" />
                        </LinearGradientBrush>
                    </Setter.Value>
                </Setter>
                <Setter Property="BorderThickness" Value="0,0,0,2" />
                <Setter Property="BorderBrush" Value="#FF6587A7" />
            </Style>


            <!-- Month Styles -->
            <Style x:Key="TitleDayStyle" TargetType="TextBlock">
```

```
                <Setter Property="Foreground" Value="White" />
                <!--#FF0C2934-->
                <Setter Property="FontSize" Value="14" />
                <Setter Property="FontWeight" Value="Bold" />
                <Setter Property="HorizontalAlignment" Value="Center"
/>
                <Setter Property="VerticalAlignment" Value="Bottom" />
                <Setter Property="Margin" Value="0 30 0 5" />
                <Setter Property="FontFamily" Value="Times New Roman"
/>
            </Style>
            <Style x:Key="CalendarItemStyle"
TargetType="System_Windows_Controls_Primitives:CalendarItem">
                <Setter Property="Template">
                    <Setter.Value>
                        <ControlTemplate
TargetType="System_Windows_Controls_Primitives:CalendarItem">
                            <Grid>
                                <Grid.Resources>
                                    <SolidColorBrush
x:Key="DisabledBrush" Color="#8CFFFFFF" />
                                </Grid.Resources>
                                <VisualStateManager.VisualStateGroups>
                                    <VisualStateGroup
x:Name="CommonStates">
                                        <VisualState x:Name="Normal" />
                                        <VisualState x:Name="Disabled">
                                            <Storyboard>
                                                <DoubleAnimation
Duration="0" Storyboard.TargetName="DisabledVisual"
Storyboard.TargetProperty="Opacity" To="1" />
                                            </Storyboard>
                                        </VisualState>
                                    </VisualStateGroup>
                                </VisualStateManager.VisualStateGroups>
                                <Border>
                                    <Grid>
                                        <Grid.Resources>
                                            <ControlTemplate
x:Key="HeaderButtonTemplate" TargetType="Button">
                                                <Grid Cursor="Hand">
```

```
<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="CommonStates">

<VisualState x:Name="Normal" />

<VisualState x:Name="MouseOver">

<Storyboard>

<ColorAnimation Duration="0" Storyboard.TargetName="ContentBrush"
Storyboard.TargetProperty="Color" To="#FF73A9D8" />

</Storyboard>

</VisualState>

<VisualState x:Name="Disabled">

<Storyboard>

<DoubleAnimation Duration="0" Storyboard.TargetName="Content"
Storyboard.TargetProperty="Opacity" To=".5" />

</Storyboard>

</VisualState>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>
                                                        <ContentControl
x:Name="Content" HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}" Margin="1,5,1,9"
VerticalAlignment="{TemplateBinding VerticalContentAlignment}"
IsTabStop="False" Content="{TemplateBinding Content}"
ContentTemplate="{TemplateBinding ContentTemplate}">

<ContentControl.Foreground>

<SolidColorBrush x:Name="ContentBrush" Color="#FF333333" />

</ContentControl.Foreground>
                                                            </ContentControl>
                                                    </Grid>
```

```
                                        </ControlTemplate>
                                        <DataTemplate
x:Name="DayTitleTemplate">
                                            <Border
Style="{StaticResource TitleDayBkgStyle}" CornerRadius="5,5,0,0">
                                                <TextBlock
Text="{Binding}" Style="{StaticResource TitleDayStyle}" />
                                            </Border>
                                        </DataTemplate>
                                        <ControlTemplate
x:Key="PreviousButtonTemplate" TargetType="Button">
                                            <Grid Cursor="Hand">

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="CommonStates">

<VisualState x:Name="Normal" />

<VisualState x:Name="MouseOver">

<Storyboard>

<ColorAnimation Duration="0" Storyboard.TargetName="IconBrush"
Storyboard.TargetProperty="Color" To="#FF73A9D8" />

</Storyboard>

</VisualState>

<VisualState x:Name="Disabled">

<Storyboard>

<DoubleAnimation Duration="0" Storyboard.TargetName="IconBrush"
Storyboard.TargetProperty="Opacity" To=".5" />

</Storyboard>

</VisualState>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>
```

```
                                                    <Rectangle
Fill="#11E5EBF1" Stretch="Fill" Opacity="1" />
                                                    <Grid>
                                                        <Path
Stretch="Fill" Height="10" HorizontalAlignment="Left" Margin="14,-
6,0,0" VerticalAlignment="Center" Width="6" Data="M288.75,232.25
L288.75,240.625 L283,236.625 z">
                                                            <Path.Fill>

<SolidColorBrush x:Name="IconBrush" Color="#FF333333" />

</Path.Fill>
                                                        </Path>
                                                    </Grid>
                                                </Grid>
                                            </ControlTemplate>
                                            <ControlTemplate
x:Key="NextButtonTemplate" TargetType="Button">
                                                <Grid Cursor="Hand">

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="CommonStates">

<VisualState x:Name="Normal" />

<VisualState x:Name="MouseOver">

<Storyboard>

<ColorAnimation Duration="0" Storyboard.TargetName="IconBrush"
Storyboard.TargetProperty="Color" To="#FF73A9D8" />

</Storyboard>

</VisualState>

<VisualState x:Name="Disabled">

<Storyboard>

<DoubleAnimation Duration="0" Storyboard.TargetName="IconBrush"
Storyboard.TargetProperty="Opacity" To=".5" />

</Storyboard>
```

```
</VisualState>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>
                                                    </Grid>
                                                </ControlTemplate>
                                            </Grid.Resources>
                                            <!--<Button
x:Name="PreviousButton" Height="20" HorizontalAlignment="Left"
Width="28" Visibility="Collapsed" Template="{StaticResource
PreviousButtonTemplate}"/>
                                                    <Button
x:Name="HeaderButton" HorizontalAlignment="Center"
VerticalAlignment="Center" FontSize="10.5" FontWeight="Bold"
Template="{StaticResource HeaderButtonTemplate}" Grid.Column="1"/>
                                                    <Button
x:Name="NextButton" Height="20" HorizontalAlignment="Right" Width="28"
Visibility="Collapsed" Template="{StaticResource NextButtonTemplate}"
Grid.Column="2"/>-->
                                            <Grid>
                                                <Grid.ColumnDefinitions>
                                                    <ColumnDefinition
Width="*" />
                                                    <ColumnDefinition
Width="*" />
                                                    <ColumnDefinition
Width="*" />
                                                    <ColumnDefinition
Width="*" />
                                                    <ColumnDefinition
Width="*" />
                                                    <ColumnDefinition
Width="*" />
                                                    <ColumnDefinition
Width="*" />
                                                </Grid.ColumnDefinitions>
                                                <Grid.RowDefinitions>
                                                    <RowDefinition
Height="2*" />
                                                    <RowDefinition
Height="*" />
                                                    <RowDefinition
Height="*" />
```

```
                                                <RowDefinition
Height="*" />
                                                <RowDefinition
Height="*" />
                                                <RowDefinition
Height="*" />
                                                <RowDefinition
Height="*" />
                                            </Grid.RowDefinitions>
                                            <Border
Style="{StaticResource TitleDayBkgStyle}" CornerRadius="5,5,0,0"
Grid.ColumnSpan="7" />
                                            <Path Fill="#FFD5E3EA"
Stretch="Fill" UseLayoutRounding="False" Grid.ColumnSpan="7"
Grid.RowSpan="6" Data="M357,155.15535 L427.00003,155.15535
L427.00003,190.99835 L357,190.99835 z M286,155.15535 L355,155.15535
L355,190.99835 L286,190.99835 z M214.00002,155.15535 L284,155.15535
L284,190.99835 L214.00002,190.99835 z M143,155.15535 L212,155.15535
L212,190.99835 L143,190.99835 z M71,155.15535 L141,155.15535
L141,190.99835 L71,190.99835 z M0,155.15535 L69,155.15535 L69,164.99834
L69,182.99834 L69,184.99835 L69,185.99834 L69,190.99835 L64,190.99835
L62,190.99835 L5,190.99835 C2.2385864,190.99835 0,188.75977 0,185.99834
L0,184.99835 L0,182.99834 z M429,155.15535 L498,155.15535 L498,172.9982
L498,185.99834 C498,188.75977 495.76144,190.99835 493,190.99835
L491,190.99835 L480.00003,190.99835 L429,190.99835 L429,183.99858 z
M429,123.3285 L498,123.3285 L498,153.16634 L429,153.16634 z
M357,123.3285 L427.00003,123.3285 L427.00003,153.16634 L357,153.16634 z
M286,123.3285 L355,123.3285 L355,153.16634 L286,153.16634 z
M214.00002,123.3285 L284,123.3285 L284,153.16634 L214.00002,153.16634 z
M143,123.3285 L212,123.3285 L212,153.16634 L143,153.16634 z
M71,123.3285 L141,123.3285 L141,153.16634 L71,153.16634 z M0,123.3285
L69,123.3285 L69,153.16634 L0,153.16634 z M429,92.496246 L498,92.496246
L498,121.33949 L429,121.33949 z M357,92.496246 L427.00003,92.496246
L427.00003,121.33949 L357,121.33949 z M286,92.496246 L355,92.496246
L355,121.33949 L286,121.33949 z M214.00002,92.496246 L284,92.496246
L284,121.33949 L214.00002,121.33949 z M143,92.496246 L212,92.496246
L212,121.33949 L143,121.33949 z M71,92.496246 L141,92.496246
L141,121.33949 L71,121.33949 z M0,92.496246 L69,92.496246 L69,121.33949
L0,121.33949 z M429,61.664253 L498,61.664253 L498,90.507492
L429,90.507492 z M357,61.664253 L427.00003,61.664253
L427.00003,90.507492 L357,90.507492 z M286,61.664253 L355,61.664253
L355,90.507492 L286,90.507492 z M214.00002,61.664253 L284,61.664253
L284,90.507492 L214.00002,90.507492 z M143,61.664253 L212,61.664253
L212,90.507492 L143,90.507492 z M71,61.664253 L141,61.664253
L141,90.507492 L71,90.507492 z M0,61.664253 L69,61.664253 L69,90.507492
L0,90.507492 z M429,30.832001 L498,30.832001 L498,59.675255
L429,59.675255 z M357,30.832001 L427.00003,30.832001
L427.00003,59.675255 L357,59.675255 z M286,30.832001 L355,30.832001
L355,59.675255 L286,59.675255 z M214.00002,30.832001 L284,30.832001
L284,59.675255 L214.00002,59.675255 z M143,30.832001 L212,30.832001
L212,59.675255 L143,59.675255 z M71,30.832001 L141,30.832001
L141,59.675255 L71,59.675255 z M0,30.832001 L69,30.832001 L69,59.675255
L0,59.675255 z M429,0 L498,0 L498,28.843246 L429,28.843246 z M357,0
L427.00003,0 L427.00003,28.843231 L357,28.843231 z M286,0 L355,0
```

```
L355,28.843231 L286,28.843231 z M214.00002,0 L284,0 L284,28.843231
L214.00002,28.843231 z M143,0 L212,0 L212,28.843231 L143,28.843231 z
M71,0 L141,0 L141,28.843246 L71,28.843246 z M0,0 L69,0 L69,28.843246
L0,28.843246 z" Grid.Row="1" />
                                        </Grid>
                                        <Grid x:Name="MonthView">
                                            <Grid.ColumnDefinitions>
                                                <ColumnDefinition
Width="*" />
                                                <ColumnDefinition
Width="*" />
                                                <ColumnDefinition
Width="*" />
                                                <ColumnDefinition
Width="*" />
                                                <ColumnDefinition
Width="*" />
                                                <ColumnDefinition
Width="*" />
                                                <ColumnDefinition
Width="*" />
                                            </Grid.ColumnDefinitions>
                                            <Grid.RowDefinitions>
                                                <RowDefinition
Height="2*" />
                                                <RowDefinition
Height="*" />
                                                <RowDefinition
Height="*" />
                                                <RowDefinition
Height="*" />
                                                <RowDefinition
Height="*" />
                                                <RowDefinition
Height="*" />
                                                <RowDefinition
Height="*" />
                                            </Grid.RowDefinitions>
                                        </Grid>
                                        <Grid x:Name="YearView"
Margin="6,-3,7,6" Visibility="Collapsed" Grid.ColumnSpan="3"
Grid.Row="1">
                                            <Grid.ColumnDefinitions>
                                                <ColumnDefinition
Width="Auto" />
```

```
                                                <ColumnDefinition
Width="Auto" />
                                                <ColumnDefinition
Width="Auto" />
                                                <ColumnDefinition
Width="Auto" />
                                            </Grid.ColumnDefinitions>
                                            <Grid.RowDefinitions>
                                                <RowDefinition
Height="Auto" />
                                                <RowDefinition
Height="Auto" />
                                                <RowDefinition
Height="Auto" />
                                            </Grid.RowDefinitions>
                                        </Grid>
                                    </Grid>
                                </Border>
                                <Rectangle x:Name="DisabledVisual"
Fill="{StaticResource DisabledBrush}" Stretch="Fill"
Stroke="{StaticResource DisabledBrush}" StrokeThickness="1" RadiusX="2"
RadiusY="2" Margin="0,2,0,2" Opacity="0" Visibility="Collapsed" />
                            </Grid>
                        </ControlTemplate>
                    </Setter.Value>
                </Setter>
            </Style>
            <Style x:Key="MonthCalendarStyle"
TargetType="controls:Calendar">
                <Setter Property="IsTabStop" Value="False" />
                <Setter Property="Template">
                    <Setter.Value>
                        <ControlTemplate
TargetType="controls:Calendar">
                            <Grid x:Name="Root">

<System_Windows_Controls_Primitives:CalendarItem x:Name="CalendarItem"
Style="{StaticResource CalendarItemStyle}" />
                            </Grid>
                        </ControlTemplate>
                    </Setter.Value>
                </Setter>
```

```
            </Style>
        </ResourceDictionary>
    </UserControl.Resources>
    <Border Background="White" CornerRadius="5">
        <Grid x:Name="LayoutRoot">
            <Grid>
                <controls:Calendar x:Name="calendar"
DisplayMode="Month" IsTodayHighlighted="True" Style="{StaticResource
MonthCalendarStyle}" DisplayDate="{Binding Day}" />
                <Path Stretch="Fill" UseLayoutRounding="False"
Data="M4.7479415,0 L495.21542,0 C498.09106,0.33296099
499.69119,1.9419931 500,4.842844 L500,61.752743 L493.36795,60.268562
C456.3688,52.313362 416.7272,48.000004 375.5,48.000008
C191.56329,48.000004 39.188137,133.8588 12.234431,245.94026
L11.340348,250 L4.5437288,250 C1.9792502,249.50926 0.470622,247.99745
0,245.4818 L0,4.791791 C0.41878578,1.9642336 1.9324206,0.29401842
4.7479415,0 z" RenderTransformOrigin="0.5,0.5"
IsHitTestVisible="False">
                    <Path.RenderTransform>
                        <TransformGroup>
                            <ScaleTransform ScaleY="-1" />
                            <SkewTransform AngleX="0" AngleY="0" />
                            <RotateTransform Angle="0" />
                            <TranslateTransform />
                        </TransformGroup>
                    </Path.RenderTransform>
                    <Path.Fill>
                        <LinearGradientBrush EndPoint="0.428,0.832"
StartPoint="0.246,-0.03">
                            <GradientStop Color="#0CFFFFFF"
Offset="0.337" />
                            <GradientStop Color="#4CFFFFFF"
Offset="0.435" />
                        </LinearGradientBrush>
                    </Path.Fill>
                </Path>
            </Grid>

            <Grid x:Name="SpiralBottom" Height="25"
VerticalAlignment="Top">
                <Grid.RowDefinitions>
                    <RowDefinition Height="0.56*" />
```

```
        <RowDefinition Height="0.28*" />
        <RowDefinition Height="0.16*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="0.044*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.002*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.042*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.002*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.044*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.002*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.042*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.002*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.432*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.044*" />
        <ColumnDefinition Width="0.006*" />
        <ColumnDefinition Width="0.01*" />
        <ColumnDefinition Width="0.002*" />
```

```
                    <ColumnDefinition Width="0.008*" />
                    <ColumnDefinition Width="0.004*" />
                    <ColumnDefinition Width="0.042*" />
                    <ColumnDefinition Width="0.006*" />
                    <ColumnDefinition Width="0.01*" />
                    <ColumnDefinition Width="0.004*" />
                    <ColumnDefinition Width="0.008*" />
                    <ColumnDefinition Width="0.004*" />
                    <ColumnDefinition Width="0.044*" />
                    <ColumnDefinition Width="0.006*" />
                    <ColumnDefinition Width="0.01*" />
                    <ColumnDefinition Width="0.002*" />
                    <ColumnDefinition Width="0.008*" />
                    <ColumnDefinition Width="0.004*" />
                    <ColumnDefinition Width="0.042*" />
                </Grid.ColumnDefinitions>
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.ColumnSpan="5" Grid.RowSpan="2" Grid.Column="1" Grid.Row="1" />
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.ColumnSpan="5" Grid.RowSpan="2" Grid.Column="7" Grid.Row="1" />
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.ColumnSpan="5" Grid.RowSpan="2" Grid.Column="13" Grid.Row="1" />
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.ColumnSpan="5" Grid.RowSpan="2" Grid.Column="19" Grid.Row="1" />
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.RowSpan="2" Grid.Column="25" Grid.Row="1" Grid.ColumnSpan="5" />
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.RowSpan="2" Grid.Column="31" Grid.Row="1" Grid.ColumnSpan="5" />
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.RowSpan="2" Grid.Column="37" Grid.Row="1" Grid.ColumnSpan="5" />
                <Rectangle Style="{StaticResource SpiralHole}"
Grid.RowSpan="2" Grid.Column="43" Grid.Row="1" Grid.ColumnSpan="5" />
                <Border Style="{StaticResource SpiralBottom}"
Grid.RowSpan="2" Grid.Column="2" />
                <Border Style="{StaticResource SpiralBottom}"
Grid.Column="4" Grid.RowSpan="2" />
                <Border Style="{StaticResource SpiralBottom}"
Grid.Column="8" Grid.RowSpan="2" />
                <Border Style="{StaticResource SpiralBottom}"
Grid.Column="10" Grid.RowSpan="2" />
                <Border Style="{StaticResource SpiralBottom}"
Grid.Column="14" Grid.RowSpan="2" />
```

```
                <Border Style="{StaticResource SpiralBottom}"
Grid.Column="16" Grid.RowSpan="2" />
                <Border Style="{StaticResource SpiralBottom}"
Grid.Column="20" Grid.RowSpan="2" />
                <Border Style="{StaticResource SpiralBottom}"
Grid.Column="22" Grid.RowSpan="2" />
                <Border Style="{StaticResource SpiralBottom}"
Grid.Column="26" Grid.RowSpan="2" />
                <Border Style="{StaticResource SpiralBottom}"
Grid.Column="28" Grid.RowSpan="2" />
                <Border Style="{StaticResource SpiralBottom}"
Grid.Column="32" Grid.RowSpan="2" />
                <Border Style="{StaticResource SpiralBottom}"
Grid.Column="34" Grid.RowSpan="2" />
                <Border Style="{StaticResource SpiralBottom}"
Grid.Column="38" Grid.RowSpan="2" />
                <Border Style="{StaticResource SpiralBottom}"
Grid.Column="40" Grid.RowSpan="2" />
                <Border Style="{StaticResource SpiralBottom}"
Grid.Column="44" Grid.RowSpan="2" />
                <Border Style="{StaticResource SpiralBottom}"
Grid.Column="46" Grid.RowSpan="2" />
            </Grid>
        </Grid>
    </Border>
</UserControl>
```

8. Right-click on the page and select **View Code** from the list. Find the class statement and change `Page` to `UserControl`.
9. Open **CalendarBookCover.xaml** and replace the existing XAML markup with the following:
   - XAML to add

```
<?xml version="1.0" encoding="utf-8"?>
<UserControl x:Class="CalendarBook.CalendarBookCover" mc:Ignorable="d"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006">
    <UserControl.Resources>
        <Style x:Key="SpiralHole" TargetType="Rectangle">
            <Setter Property="Fill" Value="#FF1D2A33" />
            <Setter Property="RadiusX" Value="2" />
            <Setter Property="RadiusY" Value="2" />
        </Style>
```

```
        <Style x:Key="SpiralTop" TargetType="Border">
            <Setter Property="Background">
                <Setter.Value>
                    <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                        <GradientStop Color="White" Offset="1" />
                        <GradientStop Color="#FF9D9D9D" />
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
            <Setter Property="CornerRadius" Value="2,2,0,0" />
            <Setter Property="BorderBrush" Value="#FF434343" />
            <Setter Property="BorderThickness" Value="1,1,1,0" />
        </Style>
        <Style x:Key="SpiralBottom" TargetType="Border">
            <Setter Property="Background">
                <Setter.Value>
                    <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                        <GradientStop Color="#FF9D9D9D" Offset="1" />
                        <GradientStop Color="White" />
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
            <Setter Property="CornerRadius" Value="0,0,2,2" />
            <Setter Property="BorderBrush" Value="#FF434343" />
            <Setter Property="BorderThickness" Value="1,0,1,1" />
        </Style>
        <Style x:Key="SpiralTopDark" TargetType="Border">
            <Setter Property="Background">
                <Setter.Value>
                    <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                        <GradientStop Color="#FF484848" Offset="1" />
                        <GradientStop Color="#FFC5C5C5" />
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
            <Setter Property="CornerRadius" Value="2,2,0,0" />
```

```
            <Setter Property="BorderBrush">
                <Setter.Value>
                    <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                        <GradientStop Color="#FF676767" Offset="0" />
                        <GradientStop Color="#FF1A1A1A" Offset="1" />
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
            <Setter Property="BorderThickness" Value="1,1,1,0" />
        </Style>
    </UserControl.Resources>
    <Border x:Name="cover" CornerRadius="6">
        <Border.Background>
            <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                <GradientStop Color="#FF245A76" Offset="0.009" />
                <GradientStop Color="#FF6B91AA" Offset="0.583" />
                <GradientStop Color="#FF759AB5" Offset="0.665" />
                <GradientStop Color="#FFCFD9E0" Offset="1" />
            </LinearGradientBrush>
        </Border.Background>
        <Border BorderThickness="5" CornerRadius="5">
            <Border.BorderBrush>
                <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                    <GradientStop Color="#998FB4CC" Offset="0.63" />
                    <GradientStop Color="Transparent" />
                    <GradientStop Color="#664D5153" Offset="0.996" />
                </LinearGradientBrush>
            </Border.BorderBrush>
            <Grid>
                <Grid.RowDefinitions>
                    <RowDefinition Height="0.708*" />
                    <RowDefinition Height="0.292*" />
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="*" />
                    <ColumnDefinition Width="296" />
```

```
                    <ColumnDefinition Width="*" />
                </Grid.ColumnDefinitions>
                <StackPanel Orientation="Horizontal"
VerticalAlignment="Bottom" Margin="0,0,0,-35" Width="296"
Grid.Column="1">
                    <TextBlock Text="20" TextWrapping="Wrap"
FontSize="150" FontFamily="Times New Roman"
HorizontalAlignment="Center" VerticalAlignment="Bottom">
                                        <TextBlock.Foreground>
                                            <LinearGradientBrush
EndPoint="0.5,1" StartPoint="0.5,0">
                                                <GradientStop
Color="#FF134659" Offset="0.2" />
                                                <GradientStop
Color="#FF0C2934" Offset="0.3" />
                                                <GradientStop
Color="#FF13455A" Offset="0.404" />
                                            </LinearGradientBrush>
                                        </TextBlock.Foreground>
                    </TextBlock>
                    <TextBlock Text="10" TextWrapping="Wrap"
FontSize="150" FontFamily="Times New Roman"
HorizontalAlignment="Center" VerticalAlignment="Bottom" Margin="-
4,0,0,0">
                                        <TextBlock.Foreground>
                                            <LinearGradientBrush
EndPoint="0.5,1" StartPoint="0.5,0">
                                                <GradientStop
Color="#FFC4C9CD" Offset="0.2" />
                                                <GradientStop
Color="DarkGray" Offset="0.3" />
                                                <GradientStop
Color="#FFFCFCFC" Offset="0.4" />
                                            </LinearGradientBrush>
                                        </TextBlock.Foreground>
                    </TextBlock>
                </StackPanel>
                <StackPanel Orientation="Horizontal"
VerticalAlignment="Top" RenderTransformOrigin="0.5,0.5" Grid.Row="1"
Width="296" Grid.Column="1">
                    <StackPanel.OpacityMask>
                        <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
```

```
                            <GradientStop Color="#66000000" Offset="1"
/>
                            <GradientStop />
                        </LinearGradientBrush>
                    </StackPanel.OpacityMask>
                    <StackPanel.RenderTransform>
                        <TransformGroup>
                            <ScaleTransform ScaleY="-1" />
                            <SkewTransform AngleX="0" AngleY="0" />
                            <RotateTransform Angle="0" />
                            <TranslateTransform />
                        </TransformGroup>
                    </StackPanel.RenderTransform>
                    <Path Stretch="Fill" Height="70.125" Width="141.43"
RenderTransformOrigin="0.500000006743084,0.264705882352941"
UseLayoutRounding="False" Data="M79.179497,0 L92.868904,0
L92.623459,1.9136314 C91.909348,8.3864136 91.552299,14.790527
91.552299,21.125973 C91.552299,34.797852 93.236862,45.955078
96.605988,54.597656 C99.389175,61.873047 103.53954,65.510742
109.0571,65.510742 C111.6938,65.510742 114.42816,64.32666
117.26019,61.958496 C120.0922,59.590332 122.24063,55.623047
123.70547,50.056641 C125.95155,41.658203 127.07459,29.817383
127.07459,14.534184 C127.07459,10.286133 126.9098,6.3024445
126.58021,2.5831146 L126.30667,0 L139.67915,0 L139.83356,0.68682861
C140.89784,5.8961792 141.42999,11.586182 141.42999,17.756832
C141.42999,29.280273 139.76984,39.070313 136.44954,47.126953
C133.12924,55.183594 128.89343,61.030762 123.74209,64.668457
C118.59074,68.306152 113.62251,70.125 108.83737,70.125
C99.364761,70.125 91.479057,64.53418 85.18026,53.352539
C79.858017,43.928711 77.196899,32.576172 77.196899,19.294922
C77.196899,13.630859 77.624146,8.3604736 78.478638,3.4837608 z
M46.943062,0 L58.719505,0 L58.593468,0.61816788 C58.007538,3.1328125
57.128632,5.6474609 55.956764,8.1621094 C52.343506,16.072266
46.484154,24.446289 38.378723,33.28418 C26.220579,46.56543
18.627842,54.573242 15.600511,57.307617 L41.52813,57.307617
C46.801537,57.307617 50.500252,57.112305 52.62426,56.72168
C54.748276,56.331055 56.664772,55.537598 58.373741,54.341309
C60.082726,53.14502 61.571976,51.448242 62.841492,49.250977
L65.551445,49.250977 L58.593468,68.367188 L0,68.367188 L0,65.657227
C17.236248,49.93457 29.36998,37.092773 36.401192,27.131836
C42.992958,17.793457 46.494835,9.2060966 46.90683,1.3697548 z">
                        <Path.RenderTransform>
                            <TransformGroup>
                                <ScaleTransform />
                                <SkewTransform />
                                <RotateTransform />
                                <TranslateTransform />
```

```
                            </TransformGroup>
                        </Path.RenderTransform>
                        <Path.Fill>
                            <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                                <GradientStop Color="#FF134659"
Offset="0.2" />
                                <GradientStop Color="#FF0C2934"
Offset="0.3" />
                                <GradientStop Color="#FF13455A"
Offset="0.404" />
                            </LinearGradientBrush>
                        </Path.Fill>
                    </Path>
                    <Path Stretch="Fill" Height="70.125"
Margin="19,0,0,0" Width="125.317"
RenderTransformOrigin="0.492986569617602,0.264705882352941"
UseLayoutRounding="False" Data="M63.066498,0 L76.755951,0
L76.510506,1.9136314 C75.796394,8.3864136 75.439339,14.790527
75.439339,21.125973 C75.439339,34.797852 77.123909,45.955078
80.493042,54.597656 C83.276245,61.873047 87.426628,65.510742
92.944199,65.510742 C95.58091,65.510742 98.315285,64.32666
101.14731,61.958496 C103.97933,59.590332 106.12777,55.623047
107.59261,50.056641 C109.8387,41.658203 110.96175,29.817383
110.96175,14.534184 C110.96175,10.286133 110.79695,6.3024445
110.46736,2.5831146 L110.19383,0 L123.56635,0 L123.72074,0.68682861
C124.78504,5.8961792 125.31719,11.586182 125.31719,17.756832
C125.31719,29.280273 123.65704,39.070313 120.33673,47.126953
C117.01643,55.183594 112.78059,61.030762 107.62923,64.668457
C102.47787,68.306152 97.509621,70.125 92.724472,70.125
C83.251831,70.125 75.366096,64.53418 69.067284,53.352539
C63.745026,43.928711 61.083893,32.576172 61.083893,19.294922
C61.083893,13.630859 61.511139,8.3604736 62.365631,3.4837608 z
M12.744122,0 L24.829065,0 L24.829065,50.862305 C24.829065,56.428711
25.060999,59.895508 25.524868,61.262695 C25.988731,62.629883
26.953085,63.679688 28.417925,64.412109 C29.882769,65.144531
32.861282,65.55957 37.353462,65.657227 L37.353462,68.367188
L0,68.367188 L0,65.657227 C4.6874952,65.55957 7.7148342,65.156738
9.0820198,64.44873 C10.449205,63.740723 11.401352,62.788574
11.938459,61.592285 C12.475569,60.395996 12.744123,56.819336
12.744122,50.862305 z">
                        <Path.RenderTransform>
                            <TransformGroup>
                                <ScaleTransform />
                                <SkewTransform />
                                <RotateTransform />
                                <TranslateTransform />
                            </TransformGroup>
```

```
                        </Path.RenderTransform>
                        <Path.Fill>
                            <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                                <GradientStop Color="#FFC4C9CD"
Offset="0.2" />
                                <GradientStop Color="DarkGray"
Offset="0.3" />
                                <GradientStop Color="#FFFCFCFC"
Offset="0.4" />
                            </LinearGradientBrush>
                        </Path.Fill>
                    </Path>
                </StackPanel>
                <Path Stretch="Fill" UseLayoutRounding="False"
Data="M0,0 L500,0 L500,61.752743 L493.36795,60.268562
C456.3688,52.313362 416.7272,48.000004 375.5,48.000008
C191.56329,48.000004 39.188137,133.8588 12.234431,245.94026
L11.340348,250 L0,250 z" Margin="-4,-4,-4,0" Grid.RowSpan="2"
Grid.ColumnSpan="3">
                    <Path.Fill>
                        <LinearGradientBrush EndPoint="0.428,0.832"
StartPoint="0.246,-0.03">
                            <GradientStop Color="#0CFFFFFF" Offset="0"
/>
                            <GradientStop Color="#4CFFFFFF" Offset="1"
/>
                        </LinearGradientBrush>
                    </Path.Fill>
                </Path>
                <TextBlock Text="calendar" TextWrapping="Wrap"
Foreground="#FF124256" Margin="0,0,8,8" FontSize="24" Grid.Row="1"
VerticalAlignment="Bottom" d:LayoutOverrides="Width"
Grid.ColumnSpan="3" HorizontalAlignment="Right">
                                <TextBlock.Effect>
                                    <DropShadowEffect
ShadowDepth="0" Color="White" />
                                </TextBlock.Effect>
                </TextBlock>
                <Grid x:Name="SpiralBottom" Height="25"
VerticalAlignment="Top" Margin="-5,-5,-5,0" Grid.ColumnSpan="3">
                    <Grid.RowDefinitions>
                        <RowDefinition Height="0.56*" />
                        <RowDefinition Height="0.28*" />
```

```
        <RowDefinition Height="0.16*" />
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="0.044*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.002*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.042*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.002*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.044*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.002*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.042*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.002*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.432*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.008*" />
        <ColumnDefinition Width="0.004*" />
        <ColumnDefinition Width="0.044*" />
        <ColumnDefinition Width="0.006*" />
        <ColumnDefinition Width="0.01*" />
        <ColumnDefinition Width="0.002*" />
        <ColumnDefinition Width="0.008*" />
```

```
                        <ColumnDefinition Width="0.004*" />
                        <ColumnDefinition Width="0.042*" />
                        <ColumnDefinition Width="0.006*" />
                        <ColumnDefinition Width="0.01*" />
                        <ColumnDefinition Width="0.004*" />
                        <ColumnDefinition Width="0.008*" />
                        <ColumnDefinition Width="0.004*" />
                        <ColumnDefinition Width="0.044*" />
                        <ColumnDefinition Width="0.006*" />
                        <ColumnDefinition Width="0.01*" />
                        <ColumnDefinition Width="0.002*" />
                        <ColumnDefinition Width="0.008*" />
                        <ColumnDefinition Width="0.004*" />
                        <ColumnDefinition Width="0.042*" />
                    </Grid.ColumnDefinitions>
                    <Rectangle Style="{StaticResource SpiralHole}"
Grid.ColumnSpan="5" Grid.RowSpan="2" Grid.Column="1" Grid.Row="1" />
                    <Rectangle Style="{StaticResource SpiralHole}"
Grid.ColumnSpan="5" Grid.RowSpan="2" Grid.Column="7" Grid.Row="1" />
                    <Rectangle Style="{StaticResource SpiralHole}"
Grid.ColumnSpan="5" Grid.RowSpan="2" Grid.Column="13" Grid.Row="1" />
                    <Rectangle Style="{StaticResource SpiralHole}"
Grid.ColumnSpan="5" Grid.RowSpan="2" Grid.Column="19" Grid.Row="1" />
                    <Rectangle Style="{StaticResource SpiralHole}"
Grid.RowSpan="2" Grid.Column="25" Grid.Row="1" Grid.ColumnSpan="5" />
                    <Rectangle Style="{StaticResource SpiralHole}"
Grid.RowSpan="2" Grid.Column="31" Grid.Row="1" Grid.ColumnSpan="5" />
                    <Rectangle Style="{StaticResource SpiralHole}"
Grid.RowSpan="2" Grid.Column="37" Grid.Row="1" Grid.ColumnSpan="5" />
                    <Rectangle Style="{StaticResource SpiralHole}"
Grid.RowSpan="2" Grid.Column="43" Grid.Row="1" Grid.ColumnSpan="5" />
                    <Border Style="{StaticResource SpiralBottom}"
Grid.RowSpan="2" Grid.Column="2" />
                    <Border Style="{StaticResource SpiralBottom}"
Grid.Column="4" Grid.RowSpan="2" />
                    <Border Style="{StaticResource SpiralBottom}"
Grid.Column="8" Grid.RowSpan="2" />
                    <Border Style="{StaticResource SpiralBottom}"
Grid.Column="10" Grid.RowSpan="2" />
                    <Border Style="{StaticResource SpiralBottom}"
Grid.Column="14" Grid.RowSpan="2" />
```

```
                    <Border Style="{StaticResource SpiralBottom}"
Grid.Column="16" Grid.RowSpan="2" />
                    <Border Style="{StaticResource SpiralBottom}"
Grid.Column="20" Grid.RowSpan="2" />
                    <Border Style="{StaticResource SpiralBottom}"
Grid.Column="22" Grid.RowSpan="2" />
                    <Border Style="{StaticResource SpiralBottom}"
Grid.Column="26" Grid.RowSpan="2" />
                    <Border Style="{StaticResource SpiralBottom}"
Grid.Column="28" Grid.RowSpan="2" />
                    <Border Style="{StaticResource SpiralBottom}"
Grid.Column="32" Grid.RowSpan="2" />
                    <Border Style="{StaticResource SpiralBottom}"
Grid.Column="34" Grid.RowSpan="2" />
                    <Border Style="{StaticResource SpiralBottom}"
Grid.Column="38" Grid.RowSpan="2" />
                    <Border Style="{StaticResource SpiralBottom}"
Grid.Column="40" Grid.RowSpan="2" />
                    <Border Style="{StaticResource SpiralBottom}"
Grid.Column="44" Grid.RowSpan="2" />
                    <Border Style="{StaticResource SpiralBottom}"
Grid.Column="46" Grid.RowSpan="2" />
                </Grid>
            </Grid>
        </Border>
    </Border>
</UserControl>
```

10. Right-click on the page and select **View Code** from the list. Find the class statement and change `Page` to `UserControl`.
11. Open **CalendarTopSpiral.xaml** and replace the XAML markup on the page with the following:
    - XAML to add

```
<?xml version="1.0" encoding="utf-8"?>
<UserControl mc:Ignorable="d" x:Class="CalendarBook.CalendarTopSpiral"
d:DesignWidth="500" d:DesignHeight="25"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006">
    <UserControl.Resources>
        <Style x:Key="SpiralTopDark" TargetType="Border">
            <Setter Property="Background">
                <Setter.Value>
                    <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
```

```
                        <GradientStop Color="#FF484848" Offset="1" />
                        <GradientStop Color="#FFC5C5C5" />
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
            <Setter Property="CornerRadius" Value="2,2,0,0" />
            <Setter Property="BorderBrush" Value="#FF1A1A1A" />
            <Setter Property="BorderThickness" Value="1,1,1,0" />
        </Style>
    </UserControl.Resources>

    <Grid x:Name="LayoutRoot">
        <Grid x:Name="Spiral" Height="30" MaxWidth="500"
MinWidth="310">
            <Grid.RowDefinitions>
                <RowDefinition Height="0.16*" />
                <RowDefinition Height="0.28*" />
                <RowDefinition Height="0.56*" />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="0.044*" />
                <ColumnDefinition Width="0.004*" />
                <ColumnDefinition Width="0.008*" />
                <ColumnDefinition Width="0.002*" />
                <ColumnDefinition Width="0.008*" />
                <ColumnDefinition Width="0.004*" />
                <ColumnDefinition Width="0.042*" />
                <ColumnDefinition Width="0.004*" />
                <ColumnDefinition Width="0.008*" />
                <ColumnDefinition Width="0.002*" />
                <ColumnDefinition Width="0.008*" />
                <ColumnDefinition Width="0.004*" />
                <ColumnDefinition Width="0.044*" />
                <ColumnDefinition Width="0.004*" />
                <ColumnDefinition Width="0.008*" />
                <ColumnDefinition Width="0.002*" />
                <ColumnDefinition Width="0.008*" />
                <ColumnDefinition Width="0.004*" />
```

```
                <ColumnDefinition Width="0.042*" />
                <ColumnDefinition Width="0.004*" />
                <ColumnDefinition Width="0.008*" />
                <ColumnDefinition Width="0.002*" />
                <ColumnDefinition Width="0.008*" />
                <ColumnDefinition Width="0.004*" />
                <ColumnDefinition Width="0.432*" />
                <ColumnDefinition Width="0.004*" />
                <ColumnDefinition Width="0.008*" />
                <ColumnDefinition Width="0.004*" />
                <ColumnDefinition Width="0.008*" />
                <ColumnDefinition Width="0.004*" />
                <ColumnDefinition Width="0.044*" />
                <ColumnDefinition Width="0.006*" />
                <ColumnDefinition Width="0.01*" />
                <ColumnDefinition Width="0.002*" />
                <ColumnDefinition Width="0.008*" />
                <ColumnDefinition Width="0.004*" />
                <ColumnDefinition Width="0.042*" />
                <ColumnDefinition Width="0.006*" />
                <ColumnDefinition Width="0.01*" />
                <ColumnDefinition Width="0.004*" />
                <ColumnDefinition Width="0.008*" />
                <ColumnDefinition Width="0.004*" />
                <ColumnDefinition Width="0.044*" />
                <ColumnDefinition Width="0.006*" />
                <ColumnDefinition Width="0.01*" />
                <ColumnDefinition Width="0.002*" />
                <ColumnDefinition Width="0.008*" />
                <ColumnDefinition Width="0.004*" />
                <ColumnDefinition Width="0.042*" />
            </Grid.ColumnDefinitions>
            <Border Style="{StaticResource SpiralTopDark}"
Grid.RowSpan="2" Grid.Column="2" Grid.Row="1" />
            <Border Style="{StaticResource SpiralTopDark}"
Grid.Column="4" Grid.RowSpan="2" Grid.Row="1" />
            <Border Style="{StaticResource SpiralTopDark}"
Grid.Column="8" Grid.RowSpan="2" Grid.Row="1" />
```

```
            <Border Style="{StaticResource SpiralTopDark}"
Grid.Column="10" Grid.RowSpan="2" Grid.Row="1" />
            <Border Style="{StaticResource SpiralTopDark}"
Grid.Column="14" Grid.RowSpan="2" Grid.Row="1" />
            <Border Style="{StaticResource SpiralTopDark}"
Grid.Column="16" Grid.RowSpan="2" Grid.Row="1" />
            <Border Style="{StaticResource SpiralTopDark}"
Grid.Column="20" Grid.RowSpan="2" Grid.Row="1" />
            <Border Style="{StaticResource SpiralTopDark}"
Grid.Column="22" Grid.RowSpan="2" Grid.Row="1" />
            <Border Style="{StaticResource SpiralTopDark}"
Grid.Column="26" Grid.RowSpan="2" Grid.Row="1" />
            <Border Style="{StaticResource SpiralTopDark}"
Grid.Column="28" Grid.RowSpan="2" Grid.Row="1" />
            <Border Style="{StaticResource SpiralTopDark}"
Grid.Column="32" Grid.RowSpan="2" Grid.Row="1" />
            <Border Style="{StaticResource SpiralTopDark}"
Grid.Column="34" Grid.RowSpan="2" Grid.Row="1" />
            <Border Style="{StaticResource SpiralTopDark}"
Grid.Column="38" Grid.RowSpan="2" Grid.Row="1" />
            <Border Style="{StaticResource SpiralTopDark}"
Grid.Column="40" Grid.RowSpan="2" Grid.Row="1" />
            <Border Style="{StaticResource SpiralTopDark}"
Grid.Column="44" Grid.RowSpan="2" Grid.Row="1" />
            <Border Style="{StaticResource SpiralTopDark}"
Grid.Column="46" Grid.RowSpan="2" Grid.Row="1" />
        </Grid>
    </Grid>
</UserControl>
```

12. Right-click on the page and select **View Code** from the list. Find the class statement and change `Page` to `UserControl`.

In this part you created a C1Book control and the calendar that the control will display. In the next part, you will run the application.

Using Book for Silverlight as a Flip Calendar Part 3: Running the Application

In the last part, you created the C1Book control and the calendar that the control will display. In this part, you will run the application and observe the control's bevahior at run time.

1. Press F5 to run your application. The application should appear as in the following image:

2. Click the corner of the calendar and observe the control's behavior.

What You've Accomplished

You created a C1Book control that was designed to work as a calendar.

Book for Silverlight Task-Based Help

The following task-based help topics assume that you are familiar with Visual Studio and Expression Blend and know how to use the C1Book control in general. If you are unfamiliar with the **ComponentOne Book for Silverlight** product, please see the Book for Silverlight Quick Start (page 43) first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne Book for Silverlight** product. Most task-based help topics also assume that you have created a new Silverlight project and added a C1Book control named "c1book1" to the project – for information about creating the control, see Creating a Book (page 106).

Creating a Book

You can easily create a C1Book control at design time in Expression Blend, in XAML, and in Code. Note that if you create a C1Book control as in the following steps, it will appear as an empty container. You will need to add items to the control for it to appear as a book at run time. For an example, see Adding Items to a Book (page 108).

At Design Time in Blend

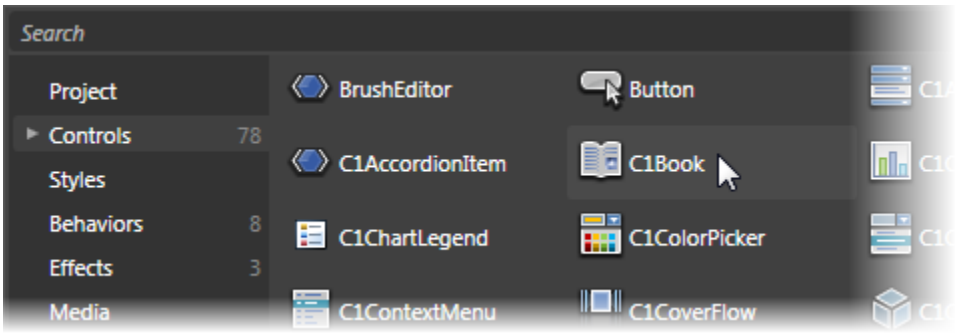
To create a C1Book control in Blend, complete the following steps:

1. Navigate to the **Projects** window and right-click the **References** folder in the project files list. In the context menu choose **Add Reference**, locate and select the **C1.Silverlight.dll** and **C1.Silverlight.Extended.dll** assemblies, and click **Open**.

   The dialog box will close and the references will be added to your project and the controls will be available in the Asset Library.

2. In the Toolbox click on the **Assets** button (the double chevron icon) to open the **Assets** dialog box.

3. In the **Asset Library** dialog box, choose the **Controls** item in the left pane, and then click on the **C1Book** icon in the right pane:



The **C1Book** icon will appear in the Toolbox under the **Assets** button.

4. Click once on the design area of the **UserControl** to select it. Unlike in Visual Studio, in Blend you can add Silverlight controls directly to the design surface as in the next step.
5. Double-click the **C1Book** icon in the Toolbox to add the control to the panel. The C1Book control will now exist in your application.
6. If you choose, you can customize the control by selecting it and setting properties in the Properties window. For example, set the C1Book control's **Name** property to "c1book1".

In XAML

To create a C1Book control using XAML markup, complete the following steps:

1. In the Visual Studio Solution Explorer, right-click the **References** folder in the project files list. In the context menu choose **Add Reference**, select the **C1.Silverlight.dll** and **C1.Silverlight.Extended.dll** assemblies, and click **OK**.
2. Add a XAML namespace to your project by adding `xmlns:c1="clr-namespace:C1.Silverlight.Extended;assembly=C1.Silverlight.Extended"` to the initial `<UserControl>` tag. It will appear similar to the following:

```
<UserControl
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:c1="clr-
namespace:C1.Silverlight.Extended;assembly=C1.Silverlight.Extended"
x:Class="C1Book.MainPage" Width="640" Height="480">
```

3. Add a `<c1:C1Book>` tag to your project within the `<Grid>` tag to create a C1Book control. The markup will appear similar to the following:

```
<Grid x:Name="LayoutRoot" Background="White">
    <c1:C1Book x:Name="c1book1" Height="300" Width="450"/>
    </c1:C1Book>
</Grid>
```

This markup will create an empty C1Book control named "c1book1" and set the control's size.

In Code

To create a C1Book control in code, complete the following steps:

1. In the Visual Studio Solution Explorer, right-click the **References** folder in the project files list. In the context menu choose **Add Reference**, select the **C1.Silverlight.dll** and **C1.Silverlight.Extended.dll** assemblies, and click **OK**.
2. Right-click within the **MainPage.xaml** window and select **View Code** to switch to Code view.
3. Add the following import statements to the top of the page:

   - Visual Basic

   ```
   Imports C1.Silverlight
   Imports C1.Silverlight.Extended
   ```

   - C#

   ```
   using C1.Silverlight;
   using C1.Silverlight.Extended;
   ```

4. Add code to the page's constructor to create the C1Book control. It will look similar to the following:

   - Visual Basic

   ```
   Public Sub New()
       InitializeComponent()
       Dim c1book1 as New C1Book
       c1book1.Height = 300
       c1book1.Width = 450
       LayoutRoot.Children.Add(c1book1)
   End Sub
   ```

   - C#

   ```
   public MainPage()
   {
       InitializeComponent();
       C1Book c1book1 = new C1Book();
       c1book1.Height = 300;
       c1book1.Width = 450;
       LayoutRoot.Children.Add(c1book1);
   }
   ```

   This code will create an empty C1Book control named "c1book1", set the control's size, and add the control to the page.

What You've Accomplished

You've created a C1Book control. Note that when you create a C1Book control as in the above steps, it will appear as an empty container. You will need to add items to the control for it to appear as a book at run time. For an example, see Adding Items to a Book (page 108).

Adding Items to a Book

You can add any sort of arbitrary content to a C1Book control. This includes text, images, layout panels, and other standard and 3rd-party controls. In this example, you'll add a **TextBlock** control to a C1Book control, but you can customize the steps to add other types of content instead.

At Design Time in Blend

To add a **TextBlock** control to the book in Blend, complete the following steps:

1. Click the C1Book control once to select it.
2. Navigate to the Toolbox, and double-click the **TextBlock** item to add the control to the C1Book control.
3. If you choose, you can customize the C1Book and **TextBlock** controls by selecting each control and setting properties in the Properties window. For example, set the **TextBlock**'s **Text** property to "Hello World!".

In XAML

For example, to add a **TextBlock** control to the book add `<TextBlock Text="Hello World!"/>` within the `<c1:C1Book>` tag so that it appears similar to the following:

```
<c1:C1Book x:Name="c1book1" Height="300" Width="450">
    <TextBlock Text="Hello World!"/>
</c1:C1Book>
```

In Code

For example, to add a **TextBlock** control to the book, add code to the page's constructor so it appears like the following:

- Visual Basic

```
Public Sub New()
    InitializeComponent()
    Dim txt1 as New TextBlock
    txt1.Text = "Hello World!"
    c1book1.Items.Add(txt1)
End Sub
```

- C#

```
public MainPage()
{
    InitializeComponent();
    TextBlock txt1 = new TextBlock();
    txt1.Text = "Hello World!";
    c1book1.Items.Add(txt1);
}
```

What You've Accomplished

You've added a control to the C1Book control. Run the application and observe that the **TextBlock** control has been added to the C1Book control. You can similarly add other content and controls.

Clearing Items in a Book

You may choose to allow users to clear all items from the C1Book control at run time, or you may need to clear the items collection when binding and then rebinding the control to another data source.

For example, to clear the book's content, add the following code to your project:

- Visual Basic

```
Me.c1book1.Items.Clear()
```

- C#

```
this.c1book1.Items.Clear();
```

What You've Accomplished

The control's content will be cleared. If you run the application, you will observe that the book is blank.

Displaying the First Page on the Right

The IsFirstPageOnTheRight property gets or sets if the first page of the book is displayed on the right or the left side. See First Page Display (page 56) for more information. By default the C1Book control starts with the first page displayed on the left and two pages displayed, but you can customize this by setting the IsFirstPageOnTheRight property at design time in Microsoft Expression Blend, in XAML, and in code.

At Design Time in Blend

To set the IsFirstPageOnTheRight property at design time in Blend, complete the following steps:

1. Click the C1Book control once to select it.
2. Navigate to the Properties window and check the check box next to the **IsFirstPageOnTheRight** item.

In XAML

For example, to set the IsFirstPageOnTheRight property, add `IsFirstPageOnTheRight="True"` to the `<c1:C1Book>` tag so that it appears similar to the following:

```
<c1:C1Book x:Name="c1book1" Height="300" Width="450"
IsFirstPageOnTheRight="True">
```

In Code

For example, to set the IsFirstPageOnTheRight property, add the following code to your project in the page's constructor:

- Visual Basic

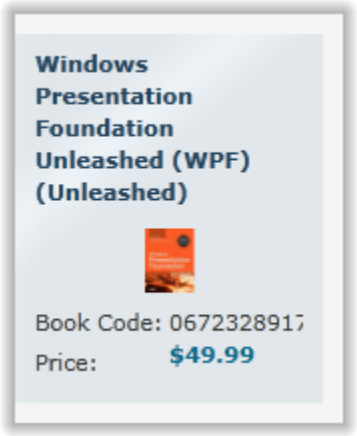
```
Me.c1book1.IsFirstPageOnTheRight = True
```

- C#

```
this.c1book1.IsFirstPageOnTheRight = true;
```

What You've Accomplished

You've set the first page to appear on the right. If you run the application, the first page will appear as a single page, like the book's cover:



Setting the Initial Page

The CurrentPage property gets or sets the value of the C1Book control's current page. By default the C1Book control starts with the first page displayed but you can customize this by setting the CurrentPage property at design time in Microsoft Expression Blend, in XAML, and in code.

At Design Time in Blend

To set the CurrentPage property to **3** at design time in Blend, complete the following steps:

1. Click the C1Book control once to select it.
2. Navigate to the Properties window and click in the text box next to the **CurrentPage** item.
3. Enter a number, for example "3", for the displayed initial page.

In XAML

For example, to set the CurrentPage property to **3**, add `CurrentPage="3"` to the `<c1:C1Book>` tag so that it appears similar to the following:

```
<c1:C1Book x:Name="c1book1" Height="300" Width="450" CurrentPage="3">
```

In Code

For example, to set the CurrentPage property to **3**, add the following code to your project:

- Visual Basic

  ```
  Me.c1book1.CurrentPage = 3
  ```

- C#

  ```
  this.c1book1.CurrentPage = 3;
  ```

What You've Accomplished

You've changed the book's initial starting page. If you run the application, the initial page that appears will be page

Navigating the Book with Code

You can set the displayed page using the CurrentPage property, but you can also use the TurnPage method to change the current page at run time. For more information, see Book Navigation (page 57). In this topic you'll add two buttons to your application, one that will turn to the previous page and one that will turn to the next page of the book.

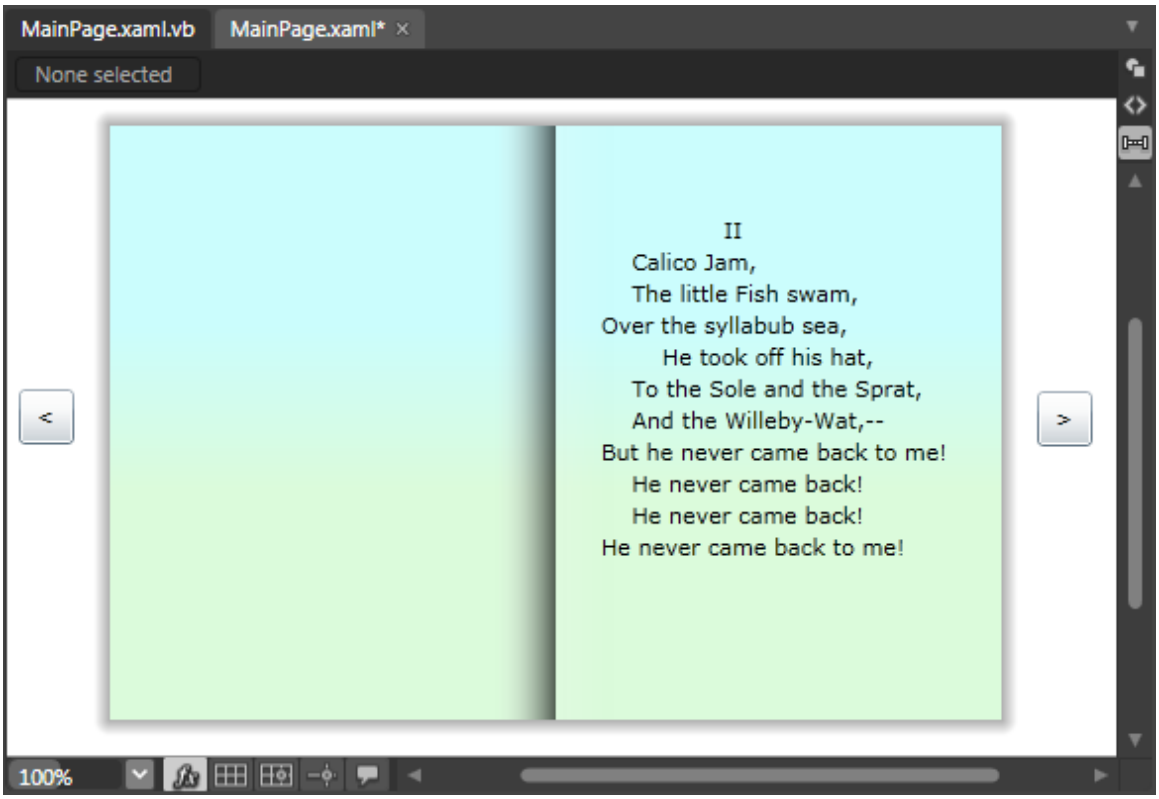
To add additional navigation to your book in Blend, complete the following steps:

1. Navigate to the Toolbox and double-click the **Button** item twice to add two **Button** controls to your application.
2. Select the first button, navigate to the Properties window and set the following properties:
   - Set **Name** to "btn_last".
   - Set **Content** to "<".
   - Set **Height** and **Width** to "28".
3. Select the second button, navigate to the Properties window and set the following properties:
   - Set **Name** to "btn_next".
   - Set **Content** to ">".
   - Set **Height** and **Width** to "28".
4. Relocate the buttons by setting their **Margin** properties. Place the **btn_last** button to the left of the book, and the **btn_next** button to the right of the book.
5. Select the left button, navigate to the Properties window, click the lightning bolt **Events** button, and double-click the text box next to the **Click** item to create the **Button_Click** event handler and switch to Code view.
6. Return to Design view and repeat the previous step with the right button so each button has a **Click** event specified.

The XAML markup will appear similar to the following:

```
<Button x:Name="btn_last" HorizontalAlignment="Left" Margin="49,223,0,229"
Width="28" Height="28" Content="&lt;" Click="btn_last_Click"/>

<Button x:Name="btn_next" HorizontalAlignment="Right"
Margin="0,224,49,228" Width="28" Height="28" Content="&gt;"
Click="btn_next_Click"/>
```

The page should now look similar to the following:



7. Switch to Code view and add the following import statements to the top of the page:

   - Visual Basic

     ```
     Imports C1.Silverlight
     Imports C1.Silverlight.Extended
     ```

   - C#

     ```
     using C1.Silverlight;
     using C1.Silverlight.Extended;
     ```

8. Add code to the **Click** event handlers so they look like the following:

   - Visual Basic

     ```
     Private Sub btn_next_Click(ByVal sender as Object, ByVal e as
     System.Windows.RoutedEventArgs)
         Me.c1book1.TurnPage(True)
     End Sub
     ```

```
Private Sub btn_last_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)
    Me.c1book1.TurnPage(False)
End Sub
```

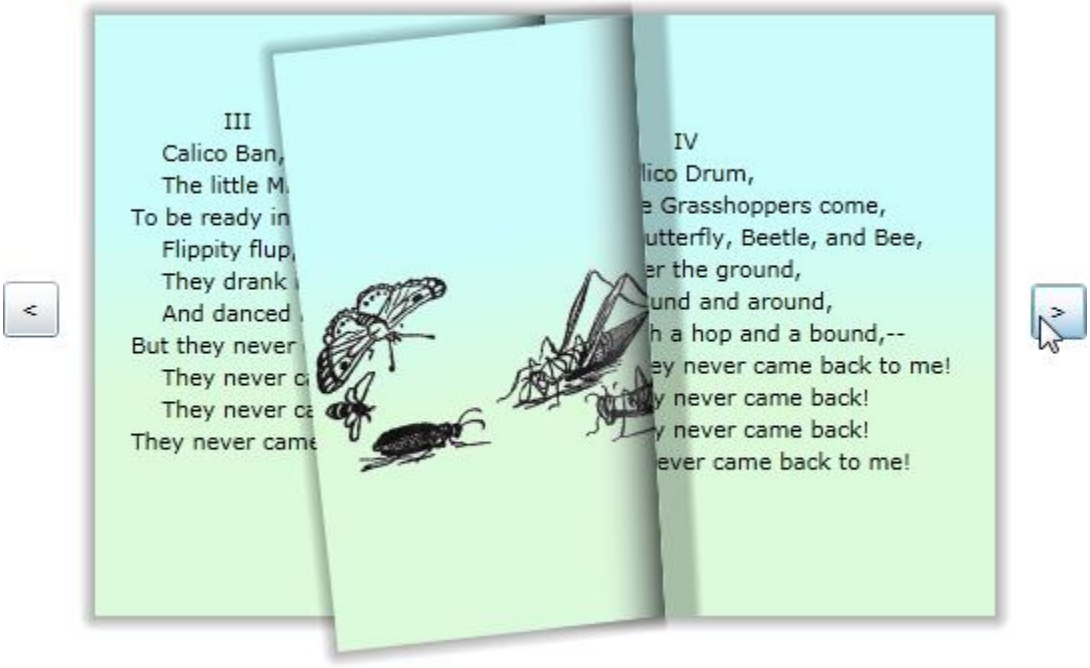
- C#

```
public MainPage()
{
private void btn_next_Click(object sender,
System.Windows.RoutedEventArgs e)
{
    this.c1book1.TurnPage(true);
}
private void btn_last_Click(object sender,
System.Windows.RoutedEventArgs e)
{
    this.c1book1.TurnPage(false);
}}
```

This code will turn the book a page forward or back depending on the button clicked.

What You've Accomplished

You've customized navigation in the book. To view the book's navigation, run the application and click the right button. Notice that the page turns to the next page with a page turning animation:



Click the left button and observe that the book returns to the previous page.

ColorPicker

ComponentOne ColorPicker™ for Silverlight is a color input editor that provides a rich, interactive, color selection interface. Users can select colors from professionally designed palettes or custom colors that you have chosen.

You can choose to include a basic color palette with preselected and standard colors, an advanced palette that users can use to completely customize their color selection, or both! **ColorPicker for Silverlight** even includes transparency, hexadecimal color, and RGB and HLS color model support to provide a rich visual color input interface.

Getting Started

Get started with the following topics:

- Key Features (page 115)

- Quick Start (page 116)

- Task-Based Help (page 135)

ColorPicker for Silverlight Key Features

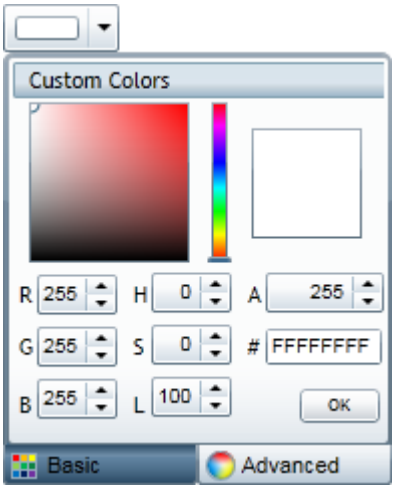
ComponentOne ColorPicker for Silverlight allows you to create customized, rich applications. Make the most of **ColorPicker for Silverlight** by taking advantage of the following key features:

- **Select from 20+ Predefined Professionally Designed Palettes**

  ColorPicker contains over 20 predefined color palettes that match the themes used in Microsoft Office. The colors in each palette go well together and can be used to create applications with a polished, professional appearance.

- **Built-in Color Editor for Custom Colors**

  ColorPicker includes a color editor. This editor allows end-users to create colors that are not on the current palette using the RGB or HLS color models and including support for transparency.



- **Different Views**

  C1ColorPicker supports both simple and advanced views for color selection.

- **Composable Parts**

  Each of the parts of the control can be used independently of the **C1ColorPicker** to create your custom controls. The **C1SpectrumColorPicker** control allows access to just the advanced color picking functionality of the **C1ColorPicker** control, the **C1HexColorBox** control provides data validation for hexadecimal code entries, and the **C1CheckeredBorder** provides a simple way to display colors with transparencies.

- **Create Your Own Custom Palette**

  If the available color palettes so not work for your application, you can create your own custom color palette. At run time, users can even be limited to selecting colors only from your chosen color palette.

- **Silverlight Toolkit Themes Support**

  Add style to your UI with built-in support for the most popular Microsoft Silverlight Toolkit themes, including ExpressionDark, ExpressionLight, WhistlerBlue, RainerOrange, ShinyBlue, and BureauBlack.

ColorPicker for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **ColorPicker for Silverlight**. In this quick start you'll start in Expression Blend and create a new project, add a C1ColorPicker control to your application, customize the appearance and behavior of the control, and observe some of the run-time interactions possible with the control.

This example uses Microsoft Expression Blend 3 to create and customize a Silverlight application, but you can also complete the following steps in Visual Studio 2008. You will create a simple project using two C1ColorPicker controls and a standard **Rectangle** control. The C1ColorPicker controls will control a gradient that is applied to the **Rectangle**, so that choosing colors at run time will change the colors of the gradient – letting you explore the possibilities of using **ColorPicker for Silverlight**.

Step 1 of 4: Setting up the Application

In this step you'll create a Silverlight application in Microsoft Expression Blend using **ColorPicker for Silverlight**. When you add a C1ColorPicker control to your application, you'll have a complete, functional color input selector. To set up your project and add C1ColorPicker controls to your application, complete the following steps:

1. In Expression Blend, select **File | New Project**.

2. In the **New Project** dialog box, select the Silverlight project type in the left pane and in the right-pane select **Silverlight Application + Website**. Enter a **Name** and **Location** for your project, select a **Language** in the drop-down box, and click **OK**.

   A new application will be created and should open with the MainPage.xaml file displayed in Design view.

3. Navigate to the Project window and right-click the **References** folder in the project files list. In the **Add Reference** dialog box choose **Add Reference** and locate and select the **C1.Silverlight.dll** and **C1.Silverlight.Extended.dll** assemblies and click **Open**. The dialog box will close and the references will be added to your project.

4. Click once on the design area of the **UserControl** to select it. Unlike in Visual Studio, in Blend you can add Silverlight controls directly to the design surface as in the next steps.

5. Navigate to the Toolbox and double-click the **Rectangle** icon to add the standard control to the **Grid**.

6. Add `x:Name="rc1" Width="Auto" Height="Auto"` to the `<Rectangle>` tag so that it appears similar to the following:

   ```
   <Rectangle x:Name="rc1" Width="Auto" Height="Auto"></Rectangle>
   ```

   The rectangle will now fill the **Grid**.

7. Switch to XAML view and add a `Fill` to the `<Rectangle>` tag so it appears similar to the following:

   ```
   <Rectangle x:Name="rc1" Height="Auto" Width="Auto">
       <Rectangle.Fill>
           <LinearGradientBrush x:Name="colors">
               <GradientStop x:Name="col1" Color="Black" Offset="0" />
               <GradientStop x:Name="col2" Color="White" Offset="1" />
           </LinearGradientBrush>
       </Rectangle.Fill>
   </Rectangle>
   ```

   This will add a black and while linear gradient fill to the rectangle. The design view of the page should now look similar to the following image:

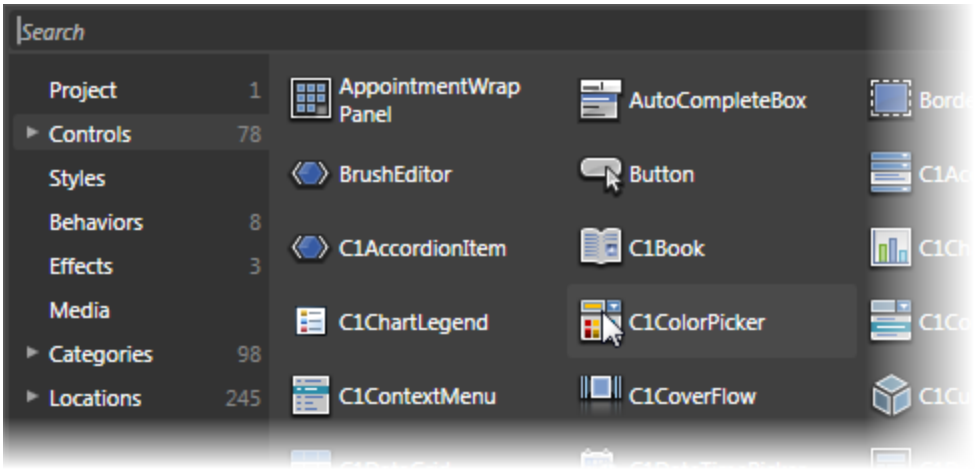
You've successfully created a Silverlight application and customized the **Rectangle** control. In the next step you'll add and customize C1ColorPicker controls.

Step 2 of 4: Adding C1ColorPicker Controls

In the previous step you created a new Silverlight project and added a **Rectangle** control with a gradient to the application. In this step you'll continue by adding C1ColorPicker controls that will control the gradient fill in the **Rectangle**.

Complete the following steps:

1. In XAML view, add the following markup just under the `</Rectangle>` tag to add a **StackPanel** to the page:
   ```
   <StackPanel Height="Auto" Width="Auto" HorizontalAlignment="Center"
   VerticalAlignment="Center" Orientation="Horizontal"></StackPanel>
   ```
2. In the ToolBox click on the **Assets** button (the double chevron icon) to open the **Assets** dialog box.
3. In the **Asset Library** dialog box, choose the **Controls** item in the left pane, and then click on the **C1ColorPicker** icon in the right pane:



   The **C1ColorPicker** icon will appear in the Toolbox under the **Assets** button.
4. Click within the `<StackPanel></StackPanel>` tags and double-click the **C1ColorPicker** icon to add the control to the panel.
5. Click once on the **C1ColorPicker** control in design view, navigate to the Properties window and set the following properties:
   - Set **Name** to "c1cp1" to give the control a name so it is accessible in code.
   - Set **Width** to "65" and **Height** to "50".
   - Set **HorizontalAlignment** and **VerticalAlignment** to **Center** to center the control in the panel.
   - Set the **Left**, **Right**, **Top**, and **Bottom Margin** properties to "5".
   - Set **DropDownDirection** to **AboveOrBelow** to control how the control opens.
   - Set the **Mode** to **Advanced** so only the advanced color picker appears.
   - Set the **SelectedColor** to **Black** (or "#FF000000").

   The XAML will appear similar to the following:

   ```
   <c1:C1ColorPicker x:Name="c1cp1" Width="65" Height="50"
   HorizontalAlignment="Center" VerticalAlignment="Center" Margin="5"
   DropDownDirection="AboveOrBelow" Mode="Advanced" SelectedColor="Black"/>
   ```

6. Add a second **C1ColorPicker** control to the StackPanel, by adding the following XAML just after the `<c1:C1ColorPicker x:Name="c1cp1"/>` tag:

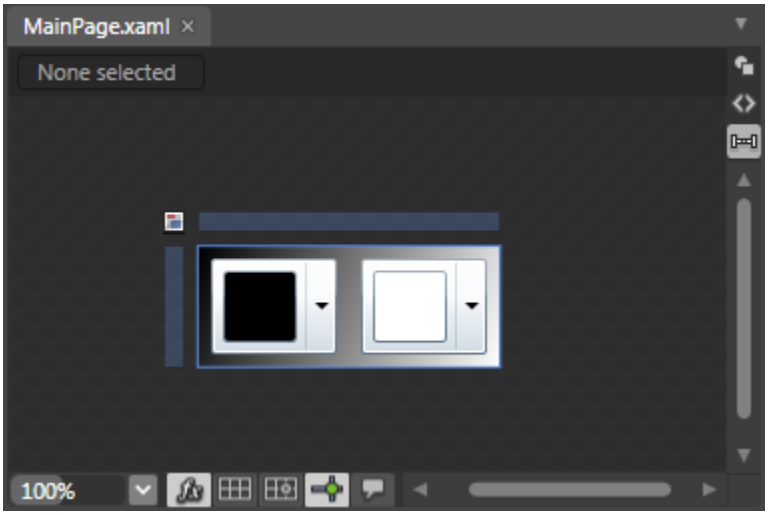
```
<c1:C1ColorPicker x:Name="c1cp2" Width="65" Height="50"
HorizontalAlignment="Center" VerticalAlignment="Center" Margin="5"
DropDownDirection="BelowOrAbove" Mode="Both" SelectedColor="White"/>
```

   Note that some properties have been set to different values, including the **Mode** and **DropDownDirection** which have been set to their default values.

7. In XAML view, replace the `Width="640" Height="480"` in the `<UserControl>` tag with `Width="Auto" Height="Auto"` so it appears similar to the following:

```
<UserControl
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:c1="clr-
namespace:C1.Silverlight.Extended;assembly=C1.Silverlight.Extended"
x:Class="C1ColorPicker.MainPage" Width="Auto" Height="Auto">
```

   The page's Design view should now look similar to the following image:



You've successfully set up your application's user interface, but if you run your application right now the color pickers will do nothing if you select a color. In the next step you'll add code to your application to add functionality to the controls.

Step 3 of 4: Adding Code to the Application

In the previous steps you set up the application's user interface and added controls to your application. In this step you'll add code to your application to finalize it.

Complete the following steps:

1. Click once on the (left-side) **c1cp1 C1ColorPicker** control to select it.
2. In the Properties window, click the lightning bolt **Events** icon to view control events.
3. Double-click in the text box next to the **SelectedColorChanged** event to switch to Code view and create the event handler.
4. In Code view, add the following import statements to the top of the page:
   - Visual Basic

```
Imports C1.Silverlight
Imports C1.Silverlight.Extended
```

- C#

```
using C1.Silverlight;
using C1.Silverlight.Extended;
```

5. Add the following code just after the page's constructor to update the gradient values:

- Visual Basic

```
Private Sub UpdateGradient()
    If c1cp1 IsNot Nothing And c1cp2 IsNot Nothing Then
        Me.col1.Color = Me.c1cp1.SelectedColor
        Me.col2.Color = Me.c1cp1.SelectedColor
    End If
End Sub
```

- C#

```
void UpdateGradient()
{
    if (c1cp1 != null & c1cp2 != null)
    {
        this.col1.Color = this.c1cp1.SelectedColor;
        this.col2.Color = this.c1cp2.SelectedColor;
    }
}
```

6. Add code to the **c1cp1_SelectedColorChanged** event handler so that it appears like the following:

- Visual Basic

```
Private Sub c1cp1_SelectedColorChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles c1cp1.SelectedColorChanged
    UpdateGradient()
End Sub
```

- C#

```
private void c1cp1_SelectedColorChanged(object sender,
C1.Silverlight.PropertyChangedEventArgs<System.Windows.Media.Color> e)
{
    UpdateGradient();
}
```

7. Return to Design view.
8. Click once on the (right-side) **c1cp2 C1ColorPicker** control to select it.
9. In the Properties window, double-click in the text box next to the **SelectedColorChanged** event to switch to Code view and create the event handler (you may need to click the lightning bolt **Events** icon to view control events if events are not listed).
10. Add code to the **c1cp2_SelectedColorChanged** event handler so that it appears like the following:

- Visual Basic

```
Private Sub c1cp2_SelectedColorChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles c1cp1.SelectedColorChanged
    UpdateGradient()
End Sub
```

- C#

```
private void c1cp2_SelectedColorChanged(object sender,
C1.Silverlight.PropertyChangedEventArgs<System.Windows.Media.Color> e)
{
```

```
    UpdateGradient();
}
```

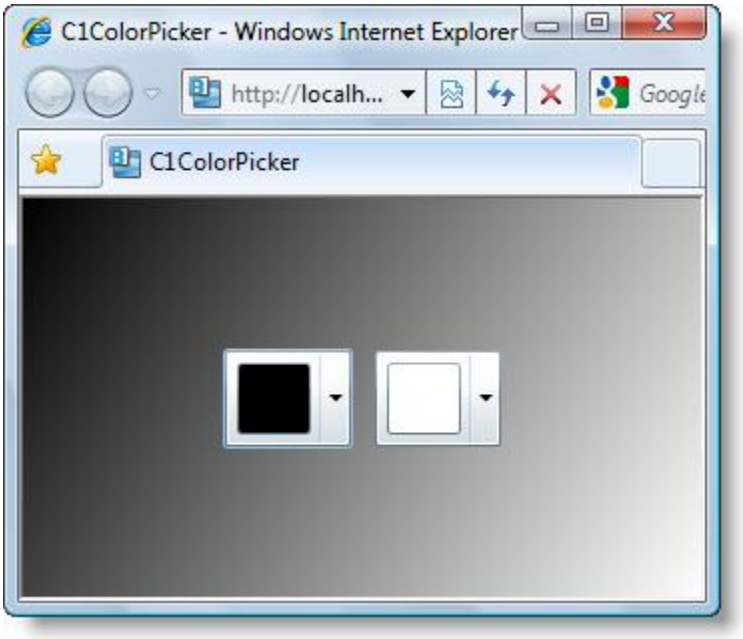
In this step you completed adding code to your application. In the next step you'll run the application and observe run-time interactions.

Step 4 of 4: Running the Application

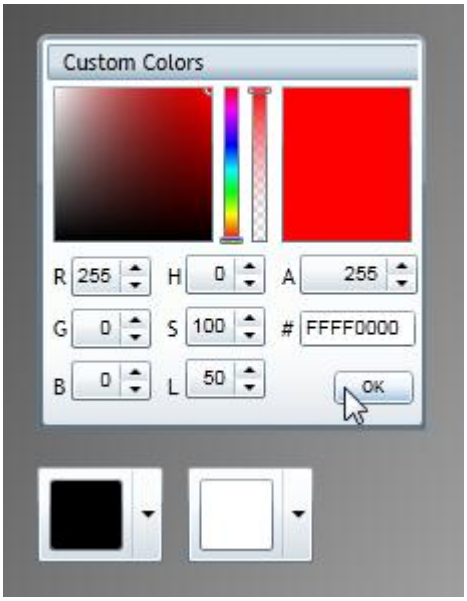
Now that you've created a Silverlight application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **ColorPicker for Silverlight**'s run-time behavior, complete the following steps:

1. From the **Project** menu, select **Run Project** to view how your application will appear at run time.

   The application will appear similar to the following:

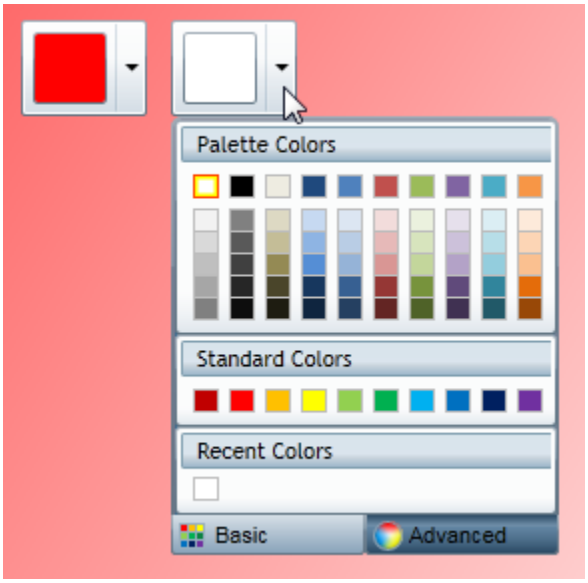


2. Click the drop-down arrow in the left color picker. Notice that the window opens above the drop-down box and that only the advanced mode is visible – this reflects the changes you made to the control. In advanced mode, users can specify any color and can use multiple methods of selecting a color.
3. Choose a color, for example **Red**, and click **OK**:



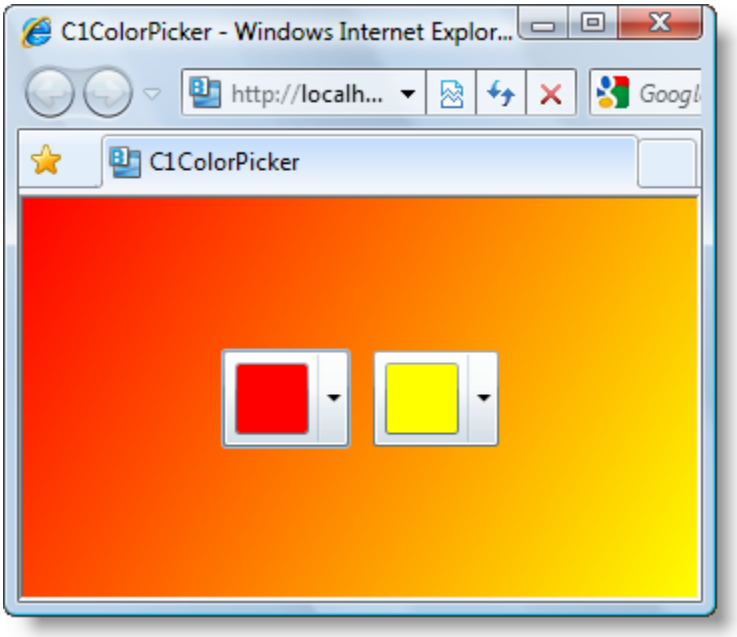
Notice that the control's selected color and the rectangle's gradient changes to reflect your color choice.

4. Click the drop-down arrow in the right color picker.



Notice that the **Basic** tab is visible (default). This tab displays **Palette Colors**, **Standard Colors**, and **Recent Colors**. You can pick any color and can also switch to the **Advanced** tab to pick a custom color. Note that the currently selected color is highlighted with a red border.

5. Pick a color, for example **Yellow**. The selected color will change and the background gradient of the rectangle will change to match your selection:



Congratulations! You've completed the **ColorPicker for Silverlight** quick start and created a simple Silverlight application, added and customized **ColorPicker for Silverlight** controls, and viewed some of the run-time capabilities of the controls.

ColorPicker XAML Quick Reference

This topic is dedicated to providing a quick overview of the XAML used to create a C1ColorPicker control. For more information, see the ColorPicker for Silverlight Task-Based Help (page 135) section.

```
<c1:C1ColorPicker Name="C1ColorPicker1" Margin="296,98,273,0" Height="45"
VerticalAlignment="Top" ShowRecentColors="False" DropDownDirection="ForceAbove"
/>
```

Working with ColorPicker for Silverlight

ComponentOne ColorPicker for Silverlight includes the C1ColorPicker control, a simple color selection control that lets users select colors from professionally designed palettes or your own custom colors. When you add the C1ColorPicker control to a XAML window, it exists as a complete color selection control, which you can further customize.

The control's default interface looks similar to the following image:

Basic Properties

ComponentOne ColorPicker for Silverlight includes several properties that allow you to set the functionality of the control. Some of the more important properties are listed below. Note that you can see ColorPicker Appearance Properties (page 131) for more information about properties that control appearance.

The following properties let you customize the C1ColorPicker control:

Property	Description
DropDownDirection	Specifies the expand direction of the control drop-down arrow.
IsDropDownOpen	Opens or closes the control drop-down box.
Mode	Indicates the mode of the color picker. Options include **Basic**, **Advanced**, and **Both** (default).
Palette	Gets/sets the palette to be used.
SelectedBrush	Gets the currently selected color as a Brush.
SelectedColor	Gets/sets the currently selected color.
ShowAlphaChannel	Gets/sets whether the user can change the alpha channel (transparency value).
ShowRecentColors	Indicates if recently picked colors should be shown.

Basic Events

ComponentOne ColorPicker for Silverlight includes several events that allow you to set interaction and customize the control. Some of the more important events are listed below.

The following events let you customize the C1ColorPicker control:

Event	Description
IsDropDownOpenChanged	Event raised when the IsDropDownOpen property has changed.
IsMouseOverChanged	Event raised when the IsMouseOver property has changed.
SelectedColorChange	Event raised when the SelectedColor property has changed.

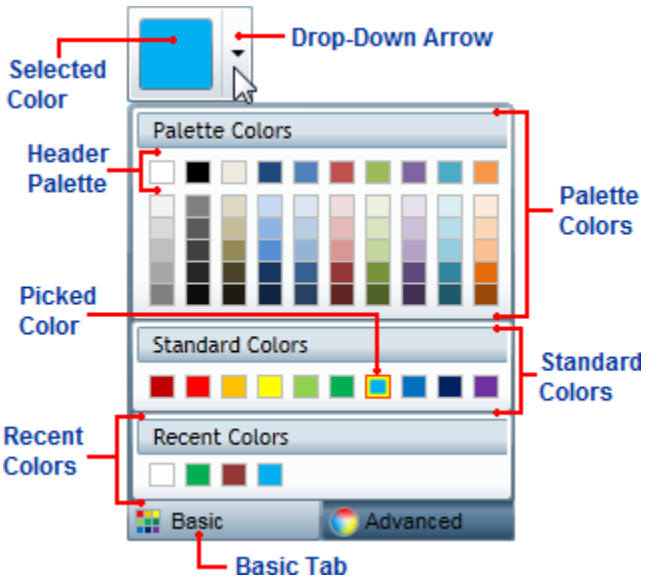
d	

ColorPicker Mode

The Mode property indicates if users should choose from a pre-selected palette of colors and/or if they can pick their own. Options include **Basic**, **Advanced**, and **Both**. By default, Mode is set to **Both** and both the **Basic** and **Advanced** tabs are visible. The following topics describe the two available tabs.

Basic ColorPicker Mode

By default, the **C1ColorPicker** control will open will the **Basic** tab open when the control's drop-down arrow is clicked. The **Basic** tab appears similar to the following image:

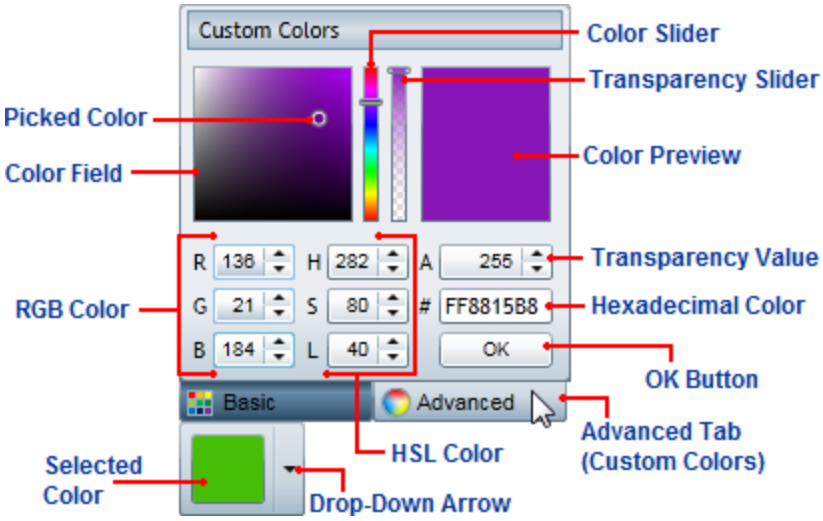


The **Basic** tab includes the following options/sections:

- **Drop-Down Arrow:** Click the drop-down arrow to open the C1ColorPicker control's window. See Drop-Down Direction (page 131) for information about setting where the drop-down window appears.
- **Basic Tab:** Click the **Basic** tab to access pre-selected colors at run time. Click the **Advanced** tab to choose a custom color. The Mode property must be set to **Basic** or **Both** for the **Basic** tab to be visible.
- **Selected Color:** The currently selected color will appear in the color picker's window.
- **Picked Color:** The currently picked color will appear with a red border in the list of colors.
- **Palette Colors:** Palette colors reflect the currently selected color palette. You can choose a palette by setting the Palette property.
- **Header Palette:** These colors are the basic colors of the palette – the expanded list of palette colors are typically variations of these basic colors.
- **Standard Colors:** Lists ten standard colors. These colors include a dark brick red, red, orange, yellow, light green, green, sky blue, blue, navy blue, and purple.
- **Recent Colors:** Lists up to ten recently selected colors. By default this section is visible, but you can choose to hide recent colors by setting the ShowRecentColors property to **False**. See Recent Colors (page 130) for more information.

Advanced ColorPicker Mode

By default, the **C1ColorPicker** control will open with the **Advanced** tab available when the control's drop-down arrow is clicked. The **Basic** tab appears by default, but the **Advanced** view can be selected by clicking the **Advanced** at the bottom of the control. The **Advanced** view appears similar to the following image:



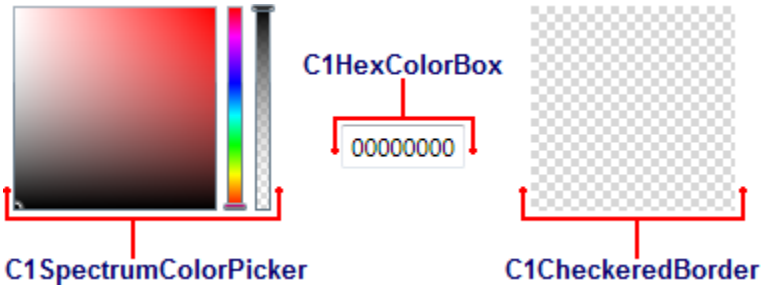
The **Advanced** tab includes the following options/sections:

- **Color Field/Picked Color:** The **Color Field** lets you choose a tone in a color's range. The **Picked Color** indicated the currently selected color. Move the **Color Slider** to pick a general color family and then fine tune the color selection in the **Color Field**.
- **Color Slider:** This slider lets you choose from the color spectrum. Move the **Color Slider** to pick a general color and then fine tune the color selection in the **Color Field**.
- **Transparency Slider:** This slider lets you lets you set the color's transparency. You can choose to make the color opaque or partially or completely transparent. Move the **Transparency Slider** to pick a transparency and note that the number in the **Transparency Value** box changes as well. This slider is only visible when the ShowAlphaChannel property is set to **True** (default)
- **Color Preview:** Preview the color you are currently choosing. Once you are satisfied with the color choice, click the **OK** button to close the drop-down box and set the color as the **Selected Color**.
- **Transparency Value:** This box lets you set the color's transparency. You can set the Transparency to a number between **0**, which is completely transparent, and **255**, which is completely opaque (default). When the ShowAlphaChannel property is set to **False** this box appears grayed out.
- **RGB Color:** These three numeric boxes let you choose a color using the Red Green Blue (RGB) color model.
- **HSL Color:** These three numeric boxes let you choose a color using the Hue Saturation Lightness (HSL) color model.
- **Hexadecimal Color:** If eight digits are visible, the first two digits represent the color's transparency ranging from FF (opaque) to 00 (transparent) and the last six digits represent standard hexadecimal color selection. Note that if the ShowAlphaChannel property is set to **False**, only the last six digits will be visible (no transparency value). For more information about hexadecimal color selection, see w3schools.
- **OK Button:** Once you are satisfied with the color choice, click the **OK** button to close the drop-down box and set the color as the **Selected Color**.

- **Selected Color:** The currently selected color will appear in the color picker's window.
- **Drop-Down Arrow:** Click the drop-down arrow to open the C1ColorPicker control's window. See Drop-Down Direction (page 131) for information about setting where the drop-down window appears.
- **Advanced Tab:** Click the **Advanced** tab to choose a custom color at run time. Click the **Basic** tab to view pre-selected colors. The Mode property must be set to **Advanced** or **Both** for the **Advanced** tab to be visible.

Additional Controls

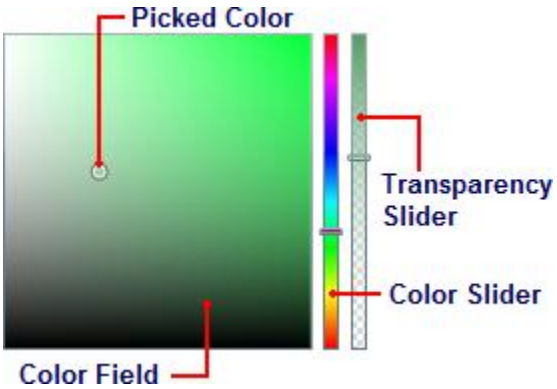
In addition to the full-featured C1ColorPicker control, **ColorPicker for WPF** includes parts of the C1ColorPicker control, that allow you to customize color picking in your application:



The C1SpectrumColorPicker control allows access to just the advanced color picking functionality of the C1ColorPicker control, the C1HexColorBox control provides data validation for hexadecimal code entries, and the C1CheckeredBorder provides a simple way to display colors with transparencies. The following topics describe these parts.

C1SpectrumColorPicker

The C1SpectrumColorPicker control allows access to just the advanced color picking functionality of the C1ColorPicker control. The C1SpectrumColorPicker control appears similar to the following image:



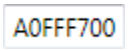
The C1SpectrumColorPicker control includes the following options/sections:

- **Color Field/Picked Color:** The **Color Field** lets you choose a tone in a color's range. The **Picked Color** indicated the currently selected color. Move the **Color Slider** to pick a general color family and then fine tune the color selection in the **Color Field**.

- **Color Slider:** This slider lets you choose from the color spectrum. Move the **Color Slider** to pick a general color and then fine tune the color selection in the **Color Field**.
- **Transparency Slider:** This slider lets you lets you set the color's transparency. You can choose to make the color opaque or partially or completely transparent. Move the **Transparency Slider** to pick a transparency. This slider is only visible when the ShowAlphaChannel property is set to **True** (default)

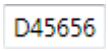
C1HexColorBox

The C1HexColorBox control provides data validation for hexadecimal code entries. For example the basic C1HexColorBox control appears similar to the following image:



The C1HexColorBox control appears similar to a regular text box; the example above represents a semi-transparent bright yellow color. By default, the C1HexColorBox control appears with eight digits. If eight digits are visible, the first two digits represent the color's transparency ranging from FF (opaque) to 00 (transparent) and the last six digits represent standard hexadecimal color selection. For more information about hexadecimal color selection, see w3schools.

Note that if the ShowAlphaChannel property is set to **False**, only the last six digits will be visible (no transparency value will be included):



You can also, if you choose, choose to display a '#' symbol to the start of the C1HexColorBox control by setting the ShowSharpPrefix property to **True**:

C1CheckeredBorder

The C1CheckeredBorder control provides a simple way to display colors with set Alpha values – so colors with varying transparencies. By default, the control appears similar to the following:

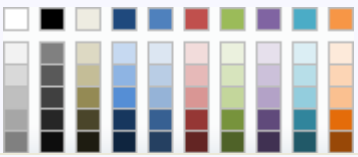
The C1CheckeredBorder control supports both transparent and opaque color values:

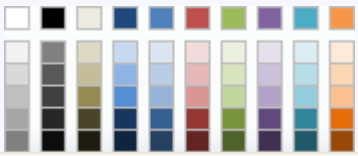
Transparent	Opaque
	

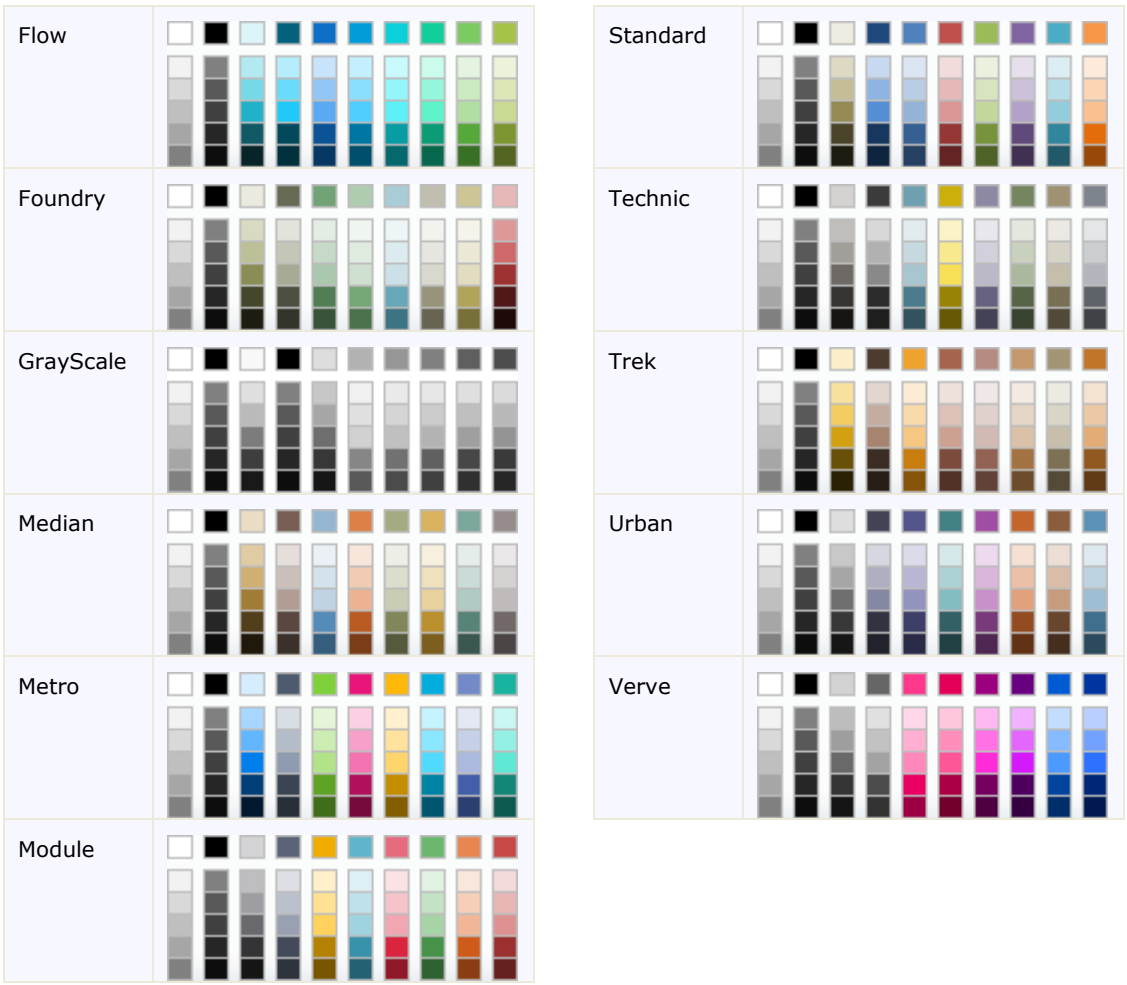
Available ColorPicker Palettes

ColorPicker for Silverlight includes over 20 predefined color palettes that match the themes used in Microsoft Office. The colors in each palette go well together and can be used to create applications with a polished, professional appearance. To change the color palette, you can set the Palette property. For more information, see Setting the Palette (page 135).

The following built-in palettes are available:

Name	Palette
Apex	
Aspect	
Civic	
Concourse	
Default	
Equity	

Name	Palette
Office	
Opulent	
Oriel	
Origin	
Paper	
Solstice	



Recent Colors

By default, when the user views the C1ColorPicker control's **Basic** tab at run time, along with the selected color palette and the standard color palette, the tab lists recently picked colors:

If you choose, you can turn the display of recent colors off. The ShowRecentColors property sets whether or not these colors are displayed. For more information and an example, see Hiding Recent Colors (page 141).

Drop-Down Direction

By default, when the user clicks the C1ColorPicker control's drop-down arrow at run-time the color picker will appear below the control, and, if that is not possible, above the control. However, you can customize where you would like the color picker to appear by setting the DropDownDirection property.

You can set the DropDownDirection property to one of the following options:

Event	Description
BelowOrAbove (default)	Tries to open the drop-down **C1ComboBox** below the header. If it is not possible tries to open above it.
AboveOrBelow	Tries to open the drop-down **C1ComboBox** above the header. If it is not possible tries to open below it.
ForceBelow	Forces the drop-down **C1ComboBox** to open below the header.
ForceAbove	Forces the **C1ComboBox** content to open above the header.

For more information and an example, see Changing the Drop-Down Window Direction (page 140).

ColorPicker Layout and Appearance

The following topics detail how to customize the C1ColorPicker control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the grid and take advantage of Silverlight's XAML-based styling. You can also use templates to format and lay out the control and to customize the control's actions.

ColorPicker Appearance Properties

ComponentOne ColorPicker for Silverlight includes several properties that allow you to customize the appearance of the control. You can change the color, border, and height of the control. The following topics describe some of these appearance properties.

Color Properties

The following properties let you customize the colors used in the control itself:

Property	Description
Background	Gets or sets a brush that describes the background of a control. This is a dependency property.
Foreground	Gets or sets a brush that describes the foreground color. This is a dependency property.

Border Properties

The following properties let you customize the control's border:

Property	Description
BorderBrush	Gets or sets a brush that describes the border background of a control. This is a dependency property.
BorderThickness	Gets or sets the border thickness of a control. This is a dependency property.

Size Properties

The following properties let you customize the size of the **C1ColorPicker** control:

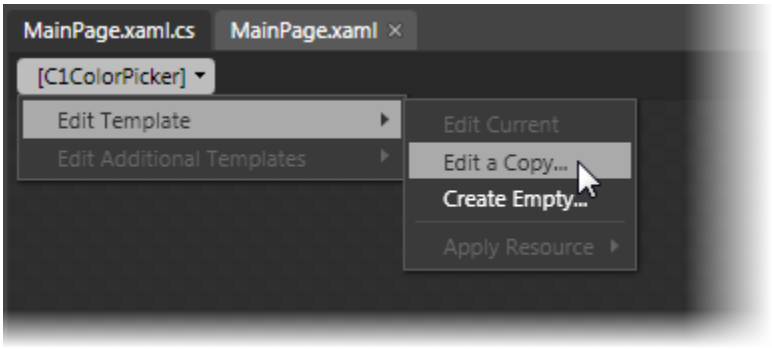
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
MinHeight	Gets or sets the minimum height constraint of the element. This is a dependency property.
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

ColorPicker Templates

One of the main advantages to using a Silverlight control is that controls are "lookless" with a fully customizable user interface. Just as you design your own user interface (UI), or look and feel, for Silverlight applications, you can provide your own UI for data managed by **ComponentOne ColorPicker for Silverlight**. Extensible Application Markup Language (XAML; pronounced "Zammel"), an XML-based declarative language, offers a simple approach to designing your UI without having to write code.

Accessing Templates

You can access templates in Microsoft Expression Blend by selecting the C1ColorPicker control and, in the menu, selecting **Edit Control Parts (Templates)**. Select **Edit a Copy** to create an editable copy of the current template or **Create Empty**, to create a new blank template.



Note: If you create a new template through the menu, the template will automatically be linked to that template's property. If you manually create a template in XAML you will have to link the appropriate template property to the template you've created.

Note that you can use the Template property to customize the template.

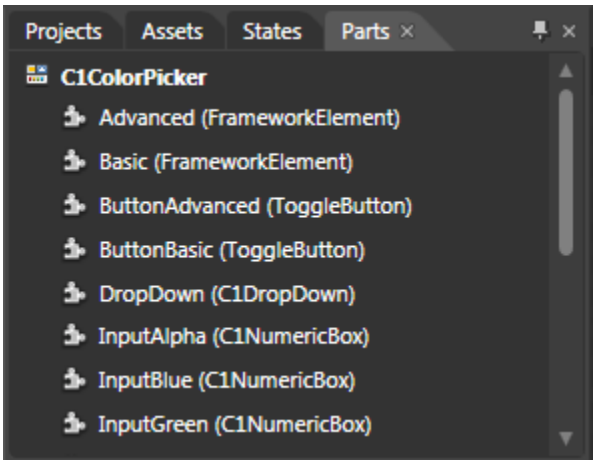
ColorPicker Styles

ComponentOne ColorPicker for Silverlight's **C1ColorPicker** control provides several style properties that you can use to change the appearance of the control. Some of the included styles are described in the table below:

Style	Description
ColorContainerStyle	Sets/gets the style of the **ItemsControl** used to show a section of colors (that is recent colors).
FontStyle	Gets or sets the font style. This is a dependency property.
Style	Gets or sets the style used by this element when it is rendered. This is a dependency property.

ColorPicker Template Parts

In Microsoft Expression Blend, you can view and edit template parts by creating a new template (for example, click the **C1ColorPicker** control to select it and choose **Object | Edit Template | Edit a Copy**). Once you've created a new template, the parts of the template will appear in the **Parts** window:



Note that you may have to select the **ControlTemplate** for its parts to be visible in the **Parts** window.

In the Parts window, you can double-click any element to create that part in the template. Once you have done so, the part will appear in the template and the element's icon in the **Parts** pane will change to indicate selection:

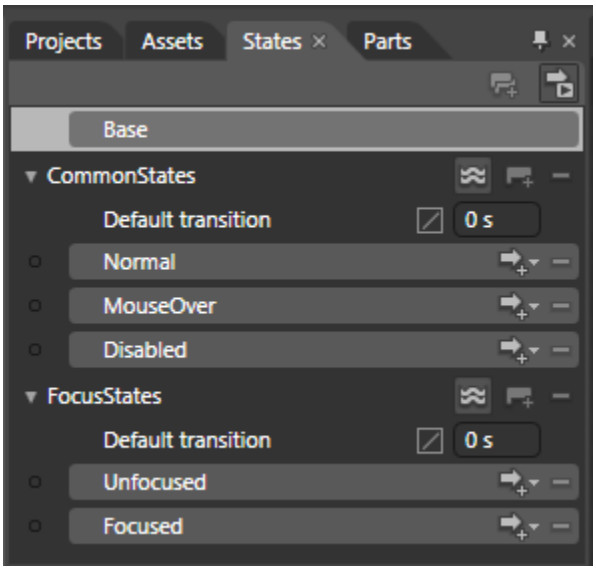
Template parts available in the **C1ColorPicker** control include:

Name	Type	Description
Advanced	FrameworkElement	Provides a framework of common APIs for objects that participate in Silverlight layout. Also defines APIs related to data binding, object tree, and object lifetime feature areas in Silverlight.
Basic	FrameworkElement	Provides a framework of common APIs for objects that participate in Silverlight layout. Also defines APIs related to data binding, object tree, and object lifetime feature areas in Silverlight.
ButtonAdvanced	ToggleButton	Base class for controls that can switch states, such as CheckBox and RadioButton.
ButtonBasic	ToggleButton	Base class for controls that can switch states, such as CheckBox and RadioButton.
DropDown	Grid	Defines a flexible grid area that consists of columns and rows.
InputAlpha	**C1NumericBox**	The **C1NumericBox** control is a numeric editor that allows you to display and edit numeric values in many formats.
InputBlue	**C1NumericBox**	The **C1NumericBox** control is a numeric editor that allows you to display and edit numeric values in many formats.
InputGreen	**C1NumericBox**	The **C1NumericBox** control is a numeric editor that allows you to display and edit numeric values in many formats.
InputHue	**C1NumericBox**	The **C1NumericBox** control is a numeric editor that allows you to display and edit numeric values in many formats.
InputLuminance	**C1NumericBox**	The **C1NumericBox** control is a numeric editor that allows you to display and edit numeric values in many formats.
InputRed	**C1NumericBox**	The **C1NumericBox** control is a numeric editor that allows you to display and edit numeric values in many formats.
InputSaturation	**C1NumericBox**	The **C1NumericBox** control is a numeric editor that allows you to display and edit numeric values in many formats.
InputWeb	TextBox	Represents a control that can be used to display single-format, multi-line text.
OkButton	Button	Represents a button control.
Preview	Rectangle	Draws a rectangle shape, which can have a stroke and a fill.
RecentColors	Grid	Defines a flexible grid area that consists of columns and rows.
RecentColorsHeader	FrameworkElement	Provides a framework of common APIs for objects that participate in Silverlight layout. Also defines APIs related to data binding, object tree, and object lifetime feature areas in Silverlight.
Root	FrameworkElement	Provides a framework of common APIs for objects that participate in Silverlight layout. Also defines APIs related to data binding, object tree, and object lifetime feature areas in Silverlight.

Spectrum	C1SpectrumColorPicker	Represents a sliding color picker.
StandardColors	Grid	Defines a flexible grid area that consists of columns and rows.
ThemeColorsHeader	Grid	Defines a flexible grid area that consists of columns and rows.
ThemeColorsValues	Grid	Defines a flexible grid area that consists of columns and rows.

ColorPicker Visual States

In Microsoft Expression Blend, you can add custom states and state groups to define a different appearance for each state of your user control – for example, the visual state of the control could change on mouse over. You can view and edit visual states by creating a new template and adding a new template part (page 133). Once you've done so the available visual states for that part will be visible in the **Visual States** window:



Common states include **Normal** for the normal appearance of the item, **MouseOver** for the item on mouse over, and **Disabled** for when the item is not enabled. Focus states include **Unfocused** for when the item is not in focus and **Focused** when the item is in focus.

ColorPicker for Silverlight Task-Based Help

The task-based help assumes that you are familiar with Visual Studio and Expression Blend and know how to use the C1ColorPicker control in general. If you are unfamiliar with the **ComponentOne ColorPicker for Silverlight** product, please see the ColorPicker for Silverlight Quick Start (page 116) first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne ColorPicker for Silverlight** product. Each task-based help topic also assumes that you have created a new Silverlight project and added a C1ColorPicker control to the project.

Setting the Palette

ColorPicker for Silverlight includes over 20 predefined color palettes that match the themes used in Microsoft Office. For more information about palette choices, see Available ColorPicker Palettes (page 129). To change the color palette, you can set the Palette property.

To set the Palette property, complete the following steps:

1. Click once within the **UserControl** to select it.
2. Navigate to the ToolBox and double-click the Button control to add it to the project.
3. Resize and reposition the Button on the page.
4. Change the Button's default content by replacing the Content markup with `Content="Change Palette"` in the button's XAML tag.
5. Give the control a name and add an event handler, by adding `Name="btn1" Click="btn1_Click"` to the button's XAML tag.
6. Switch to Code view and add code for the **Button_Click** event handler:

   - Visual Basic

     ```
     Private Sub btn1_Click(ByVal sender as Object, ByVal e as
     System.Windows.RoutedEventArgs)

         ' Set the color palette.

         c1cp1.Palette =
     ColorPalette.GetColorPalette(Office2007ColorTheme.GrayScale)

     End Sub
     ```

   - C#

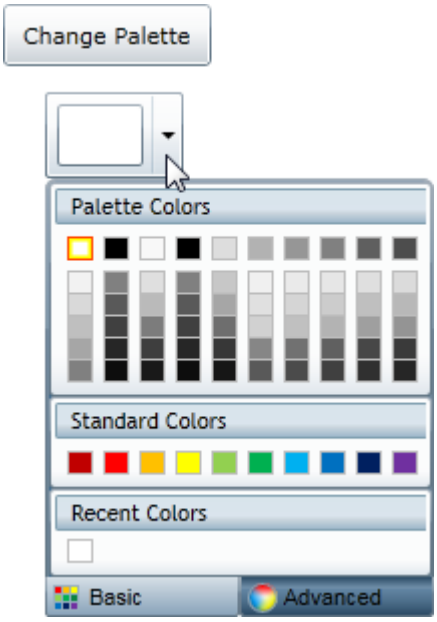
     ```
     private void btn1_Click(object sender, System.Windows.RoutedEventArgs
     e)

     {

         // Change color palette.

         this.c1cp1.Palette =
     ColorPalette.GetColorPalette(Office2007ColorTheme.GrayScale);

     }
     ```

   The **ColorPicker**'s color palette will now change to grayscale when the button is clicked.

Run the application and observe:

Observe the following:

1. Click the C1ColorPicker control's drop-down arrow and notice that the default palette appears.
2. Click the **Change Palette** button and click the C1ColorPicker control's drop-down arrow once again. Notice that a grayscale palette appears:



Creating a Custom Palette

ColorPicker for Silverlight includes over 20 predefined color palettes that match the themes used in Microsoft Office, but if you choose you can create your own custom color palette rather than using a predefined one. In the following steps you'll create a custom palette, and when a button is pressed, apply that palette to the C1ColorPicker control.

To create a custom palette, complete the following steps:

1. Click once within the **UserControl** to select it.
2. Navigate to the ToolBox and double-click the Button control to add it to the project.
3. Resize and reposition the Button on the page.
4. Change the Button's default content by replacing the Content markup with `Content="Change Palette"` in the button's XAML tag.
5. Give the control a name and add an event handler, by adding `Name="btn1" Click="btn1_Click"` to the button's XAML tag.
6. Switch to Code view and add code for the **Button_Click** event handler:

   - Visual Basic

```
Private Sub btn1_Click(ByVal sender as Object, ByVal e as
System.Windows.RoutedEventArgs)

    ' Set the color palette.

    Dim cp1 as New ColorPalette("Pittsburgh")

    cp1.Clear()

    cp1.Add(Color.FromArgb(255, 0, 0, 0))

    cp1.Add(Color.FromArgb(255, 99, 107, 112))

    cp1.Add(Color.FromArgb(255, 255, 255, 255))

    cp1.Add(Color.FromArgb(255, 247, 181, 18))

    cp1.Add(Color.FromArgb(255, 253, 200, 47))
```

```
        cp1.Add(Color.FromArgb(255, 43, 41, 38))
        cp1.Add(Color.FromArgb(255, 149, 123, 77))
        cp1.Add(Color.FromArgb(255, 209, 201, 157))
        cp1.Add(Color.FromArgb(255, 0, 33, 71))
        cp1.Add(Color.FromArgb(255, 99, 177, 229))
        c1cp1.Palette = cp1
    End Sub
```

- C#

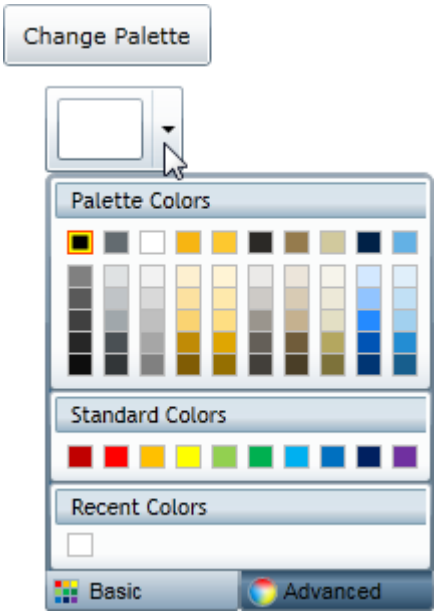
```
    private void btn1_Click(object sender, System.Windows.RoutedEventArgs
    e)
    {
        // Set the color palette.
        ColorPalette cp1 = new ColorPalette("Pittsburgh");
        cp1.Clear();
        cp1.Add(Color.FromArgb(255, 0, 0, 0));
        cp1.Add(Color.FromArgb(255, 99, 107, 112));
        cp1.Add(Color.FromArgb(255, 255, 255, 255));
        cp1.Add(Color.FromArgb(255, 247, 181, 18));
        cp1.Add(Color.FromArgb(255, 253, 200, 47));
        cp1.Add(Color.FromArgb(255, 43, 41, 38));
        cp1.Add(Color.FromArgb(255, 149, 123, 77));
        cp1.Add(Color.FromArgb(255, 209, 201, 157));
        cp1.Add(Color.FromArgb(255, 0, 33, 71));
        cp1.Add(Color.FromArgb(255, 99, 177, 229));
        c1cp1.Palette = cp1;
    }
```

The **ColorPicker**'s color palette will now change to a custom palette when the button is clicked.

Run the application and observe:

Observe the following:

1. Click the C1ColorPicker control's drop-down arrow, and notice that the default palette appears.
2. Click the **Change Palette** button and click the C1ColorPicker control's drop-down arrow once again. Notice that the custom palette appears:



Changing the Background Color

The **Background** property gets or sets the value of the C1ColorPicker control's background color. By default the C1ColorPicker control starts with the **Background** property unset, but you can customize this at design time in Microsoft Expression Blend, in XAML, and in code.

At Design Time in Blend

To set the **Background** property at run time, complete the following steps:

1. Click the C1ColorPicker control once to select it.
2. Navigate to the tab window and click the **Background** item.
3. Click the **Solid Color** brush tab, and choose **Red** or another color in the color picker.

   This will set the **Background** property to the color you chose

In XAML

For example, to set the **Background** property to **Red** add `Background="Red"` to the `<c1: C1ColorPicker>` tag so that it appears similar to the following:

```
<c1:C1ColorPicker Name="C1ColorPicker1" Margin="296,98,273,0" Height="45"
VerticalAlignment="Top" Background="Red"/>
```

In Code

For example, to set the **Background** property to **Red**, add the following code to your project:

- Visual Basic
  ```
  Me.C1ColorPicker1.Background = System.Windows.Media.Brushes.Red
  ```
- C#
  ```
  this.c1ColorPicker1.Background = System.Windows.Media.Brushes.Red;
  ```

Run the application and observe:

The background of the C1ColorPicker control will appear red:

Changing the Drop-Down Window Direction

By default, when the user clicks the C1ColorPicker control's drop-down arrow at run-time the color picker will appear below the control, and if that is not possible, above the control. However, you can customize where you would like the color picker to appear. For more information about the drop-down arrow direction, see Drop-Down Direction (page 131).

At Design Time in Blend

To change the drop-down window direction at run time, complete the following steps:

1. Click the C1ColorPicker control once to select it.
2. Navigate to the Properties window and click the DropDownDirection drop-down arrow.
3. Choose an option, for example **ForceAbove**.

   This will set the DropDownDirection property to the option you chose.

In XAML

For example, change the drop-down window direction add `DropDownDirection="ForceAbove"` to the `<c1:C1ColorPicker>` tag so that it appears similar to the following:

```
<c1:C1ColorPicker Name="C1ColorPicker1" Margin="296,98,273,0" Height="45"
VerticalAlignment="Top" DropDownDirection="ForceAbove"/>
```

In Code

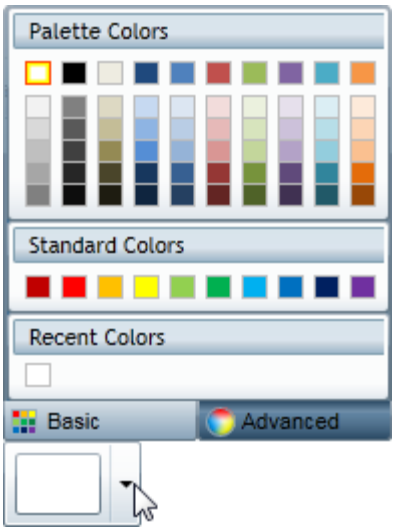
For example, to change the drop-down window direction, add the following code to your project:

- Visual Basic
  ```
  Me.C1ColorPicker1.DropDownDirection = DropDownDirection.ForceAbove
  ```
- C#
  ```
  this.c1ColorPicker1.DropDownDirection = DropDownDirection.ForceAbove;
  ```

This will set the DropDownDirection property to **ForceAbove**.

Run the application and observe:

When you click the C1ColorPicker control's drop-down arrow, the drop down window will appear above the control:



Hiding Recent Colors

By default the C1ColorPicker control displays recent colors in the Basic tab of the color picker window. For more information, see Recent Colors (page 130). If you choose, you prevent Recent Colors from being displayed at design time in Microsoft Expression Blend, in XAML, and in code.

At Design Time in Blend

To prevent recent colors from being displayed at run time, complete the following steps:

1. Click the C1ColorPicker control once to select it.
2. Navigate to the Properties window.
3. Locate and uncheck the **ShowRecentColors** check box.

   This will prevent recent colors from being displayed at run time.

In XAML

To prevent recent colors from being displayed add `ShowRecentColors="False"` to the `<c1:C1ColorPicker>` tag so that it appears similar to the following:

```
<c1:C1ColorPicker Name="C1ColorPicker1" Margin="296,98,273,0" Height="45"
VerticalAlignment="Top" ShowRecentColors="False" />
```

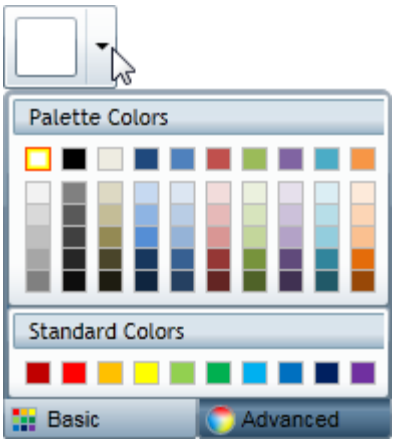
In Code

To prevent recent colors from being displayed, add the following code to your project:

- Visual Basic
  ```
  Me.C1ColorPicker1.ShowRecentColors = False
  ```
- C#
  ```
  this.c1ColorPicker1.ShowRecentColors = false;
  ```

Run the application and observe:

When you click the C1ColorPicker control's drop-down arrow, observe that recent colors are not displayed on the Basic tab:


Palette Colors
Standard Colors
Basic
Advanced

CoverFlow

Visually navigate through items in an animated, three-dimensional graphical UI with **ComponentOne CoverFlow™ for Windows Phone**. Browse the Cover Flow by sliding your finger across the touch screen or tapping adjacent items.

Getting Started

- C1CoverFlow Control Basics (page 152)
- Quick Start (page 143)
- Task-Based Help (page 169)

CoverFlow for Silverlight Key Features

ComponentOne CoverFlow for Silverlight allows you to create customized, rich applications. Make the most of **CoverFlow for Silverlight** by taking advantage of the following key features:

- **Configure the Angle of Items**

  The unselected images appear at an angle of perspective creating the illusion that the albums are stacked. You can easily change the angle perspective by setting one property. See Item Angle (page 155) for more information.

- **Configure the Distance of Items**

  Alter the spacing between the items as well as the eye distance. See Eye Distance (page 153), Selected Item Distance (page 158) and Item Distance (page 159) for more information.

- **Set the Speed of Items**
- Control the speed when the user selects an item or scrolls through the control. You can also control the speed at which the items' angles change and the speed at which the camera position changes. See Speed Settings (page 160) for more information.
- **Set the Size of the Items**

  Easily configure the size of the covers.

- **Virtualization**

  Easily show a large amount of items in your C1CoverFlow; it will load images on demand.

- **Silverlight Toolkit Themes Support**

  Add style to your UI with built-in support for the most popular Microsoft Silverlight Toolkit themes, including ExpressionDark, ExpressionLight, WhistlerBlue, RainierOrange, ShinyBlue, and BureauBlack.

CoverFlow for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **CoverFlow for Silverlight**. In this quick start, you'll start in Expression Blend to create a new project with a C1CoverFlow control. Once the control is added to your project, you'll customize it by setting a few properties and adding content to its content panel. You will then run the project to see the results of the quick start.

Step 1 of 5: Creating the Project

In this step, you'll begin in Expression Blend to create a Silverlight application using **CoverFlow for Silverlight**. You will also add a **StackPanel** control, a **TextBlock** control, and a folder containing three album cover images to the project.

Complete the following steps:

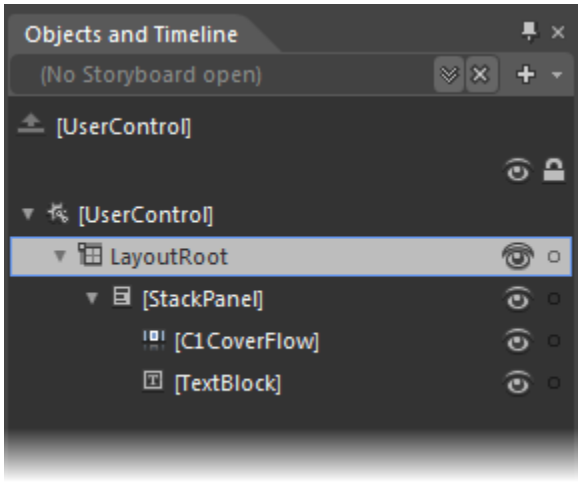
1. In Expression Blend, select **File | New Project**.
2. In the **New Project** dialog box, select the Silverlight project type in the left pane and, in the right-pane, select **Silverlight Application + Website**.
3. Enter a **Name** and **Location** for your project, select a **Language** in the drop-down box, and click **OK**. Blend creates a new application, which opens with the **MainPage.xaml** file displayed in Design view.
4. Add a **StackPanel** control to your project by completing the following steps:
   a. On the menu, select **Window | Assets** to open the Assets tab.
   b. In the **Assets** panel, enter "StackPanel" into the search bar.

      The **StackPanel** control's icon appears.
   c. Double-click the **C1CoverFlow** icon to add the control to your project.
5. Add a C1CoverFlow control to the **StackPanel** control by completing the following steps:
   a. In the **Objects and Timeline** panel, select **[StackPanel]**.
   b. In the **Assets** panel, enter "C1CoverFlow" into the search bar.

      The C1CoverFlow control's icon appears.
   c. Double-click the C1CoverFlow icon to add the control to the **StackPanel**.
6. Add a **TextBlock** control to the **StackPanel** control by completing the following steps:
   a. In the **Objects and Timeline** panel, select **[StackPanel]**.
   b. In the **Assets** panel, enter "TextBlock" into the search bar.

      The **TextBlock** control's icon appears.
   c. Double-click the **TextBlock** icon to add the control to the **StackPanel**.

      In the **Objects and Timeline** tab, the layout hierarchy looks as follows:



7. Add album covers to the project by completing the following steps:
    a. In the **Projects** panel, right-click the project to open its context menu and select **Add New Folder**; name the new folder "Images".
    b. Right-click the **Images** folder and select **Add Existing Item**.

       The **Add Existing Item** dialog box opens.
    c. Navigate to the following location:
        - In XP

          C:\Documents and Settings\<username>\My Documents\ComponentOne Samples\Studio for Silverlight\QuickStart\QuickStart
        - Windows 7/Vista

          C:\Users\<username>\Documents\ComponentOne Samples\Studio for Silverlight\QuickStart\QuickStart
    d. Select **cover1.jpg**, **cover2.jpg**, and **cover3.jpg**.
    e. Click **Open** to close the **Add Existing Item** dialog box and to add the images to the folder.

       The images are added to the **Images** folder.

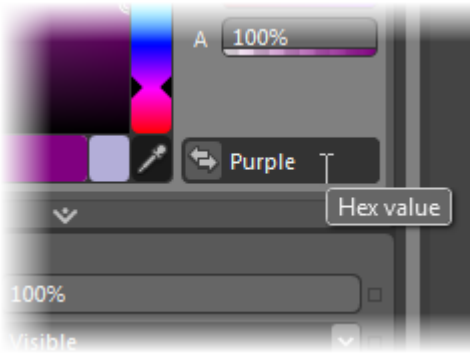
In this step, you created a project and added a C1CoverFlow control, a **StackPanel** control, and a **TextBlock** control to it; in addition, you added a folder with three album images to the project. In the next step, you'll customize the controls by setting a few properties.

Step 2 of 5: Customizing the Controls

In the last step, you created a Silverlight project in Blend with three controls – a C1CoverFlow control, a **StackPanel** control, and a **TextBlock** control - and a folder of images. In this step, you'll customize the three controls you added to your project by setting a few of their properties.

Complete the following steps:

1. In the **Objects and Timeline** panel, select **[StackPanel]** and set the following properties in the **Properties** panel:
    - Set the **Width** property to "Auto".
    - Set the **Height** property to "Auto".
2. In the **Objects and Timeline** panel, select **[C1CoverFlow]** and set the following properties in the **Properties** panel:
    - Set the **Name** property to "C1CoverFlow1".
    - Set the **Width** property to "470".
    - Set the **Height** property to "250".
3. In the **Objects and Timeline** panel, select **[TextBlock]** and set the following properties:
    - Set the **Name** property to "TextBlock1".
    - Set the **Text** property to "Album information will display here."
    - Set the **Foreground** property to purple by typing "Purple" into the **Hex value** text box and pressing ENTER.



In this step, you customized the **StackPanel**, C1CoverFlow, and **TextBlock** controls. In the next step, you'll add items to the C1CoverFlow control.

Step 3 of 5: Adding Items to the C1CoverFlow Control

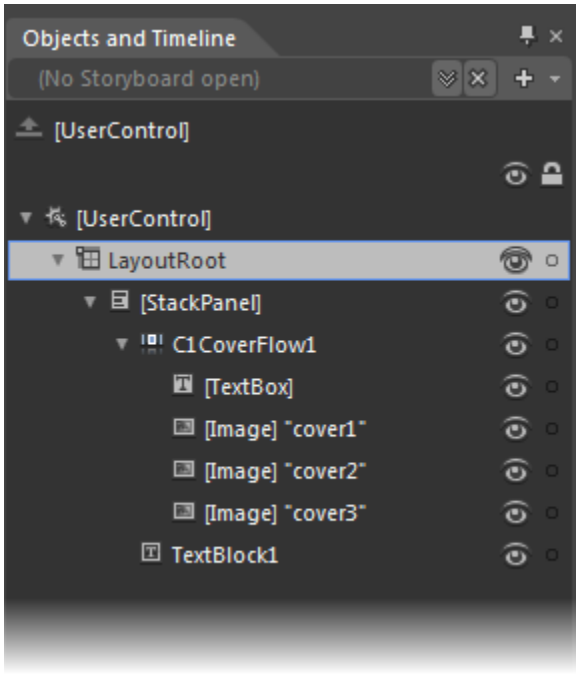
In the last step, you customized the **StackPanel**, C1CoverFlow, and **TextBlock** controls by setting a few properties. In this step, you'll add four items – a **TextBox** control and three images – to the C1CoverFlow control. You will also customize each item by settings its properties.

Complete the following steps:

1. Add a **TextBox** control to the C1CoverFlow control by completing the following steps:
   a. In the **Objects and Timeline** panel, select **C1CoverFlow1.**
   b. In the Assets panel, enter "TextBox" into the search bar.

      The **TextBox** control's icon appears.
   c. Double-click the **TextBox** icon to add the control to the **C1CoverFlow** control.
2. Add the first image to the control by completing the following steps:
   a. In the **Objects and Timeline** panel, select **C1CoverFlow1**.
   b. In the **Projects** panel, expand the **Images** folder if it's not already expanded.
   c. Double-click **cover1.jpg** to add the image to the C1CoverFlow control.
   d. Double-click **cover2.jpg** to add the image to the C1CoverFlow control.
   e. Double-click **cover3.jpg** to add the image to the C1CoverFlow control.

      In the **Objects and Timeline** tab, the layout looks as follows:



3. In the **Objects and Timeline** panel, select **[TextBox]** and set the following properties in the **Properties** panel:
    - Set the **Background** property to a light purple by entering "#FFB3AED8" into the **Hex value** text box.
    - Set the **Width** property to "150".
    - Set the **Height** property to "150".
    - Set the **Text** property to the following string: "As you scroll through the list of album covers, the TextBlock control will display the title of the currently selected item."
4. In the **Objects and Timeline** panel, select **[Image] "cover1"** and set the following properties in the **Properties** panel:
    - Set the **Width** property to "150".
    - Set the **Height** property to "150".
5. In the **Objects and Timeline** panel, select **[Image] "cover2"** and set the following properties in the **Properties** panel:
    - Set the **Width** property to "150".
    - Set the **Height** property to "150".
6. In the **Objects and Timeline** panel, select **[Image] "cover3"** and set the following properties in the **Properties** panel:
    - Set the **Width** property to "150".
    - Set the **Height** property to "150".

In this step, you added four items to the C1CoverFlow control and then set the items' properties. In the next step, you'll add code to the project.

Step 4 of 5: Adding Code to the Project

In the last step, you added four items to the C1CoverFlow control and set the items' properties. In this step, you're going to create a SelectedIndexChanged event handler and add code to the event handler that checks for the value of the SelectedIndex property. The **TextBlock** control, which you added in Step 1 of 5: Creating the Project (page 144), will change its **Text** property string value when the user selects an album cover.

Complete the following steps:

1. In the **Objects and Timeline** panel, select **C1CoverFlow1**.
2. In the **Properties** panel, click the **Events** button  .
3. Locate the SelectedIndexChanged event handler and double-click inside its text box.

   The **C1CoverFlow1_SelectedIndexChanged** event handler is added to your project.
4. Replace the code comment with the following code:
   - Visual Basic

```
If C1CoverFlow1.SelectedIndex = 1 Then
   TextBlock1.Text = "Deep Purple: Machine Head"

ElseIf C1CoverFlow1.SelectedIndex = 2 Then
   TextBlock1.Text = "Deep Purple: Made in Japan"

ElseIf C1CoverFlow1.SelectedIndex = 3 Then
   TextBlock1.Text = "Deep Purple: Perfect Strangers"
Else
TextBlock1.Text = Nothing
End If
```

   - C#

```
if (C1CoverFlow1.SelectedIndex == 1)
{
TextBlock1.Text = "Deep Purple: Machine Head";
}

else if (C1CoverFlow1.SelectedIndex == 2)
{
TextBlock1.Text = "Deep Purple: Made in Japan";
}

else if (C1CoverFlow1.SelectedIndex == 3)
{
TextBlock1.Text = "Deep Purple: Perfect Strangers";
}
```

```
else
{
    TextBlock1.Text = null;
}
```

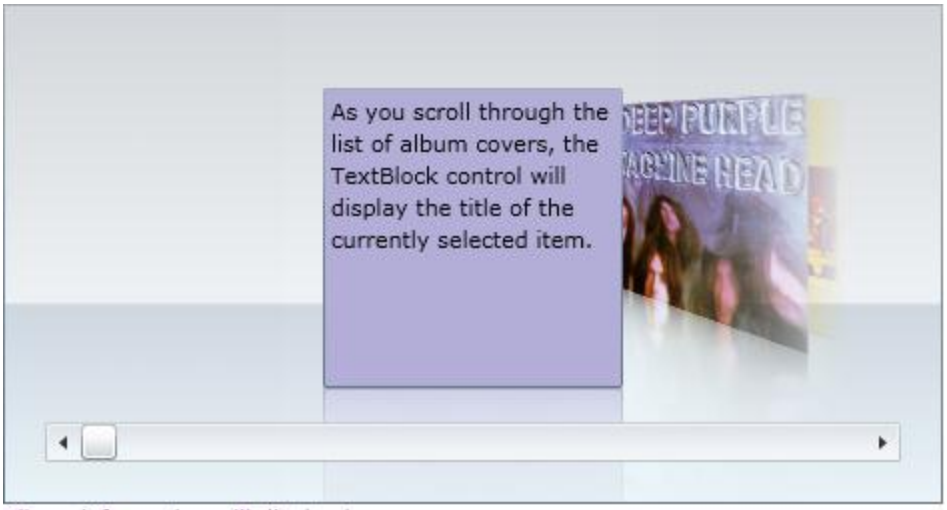
In this step, you added an event handler and code to your project. In the next step, you'll run program and see the results of this quick start topic.

Step 5 of 5: Running the Project

In the previous four steps, you created a project with a **StackPanel**, C1CoverFlow, and **TextBlock** controls, set the controls' properties, added items to the C1CoverFlow control, and added code to your project. All that's left for you to do now is run the project.

Complete the following steps:

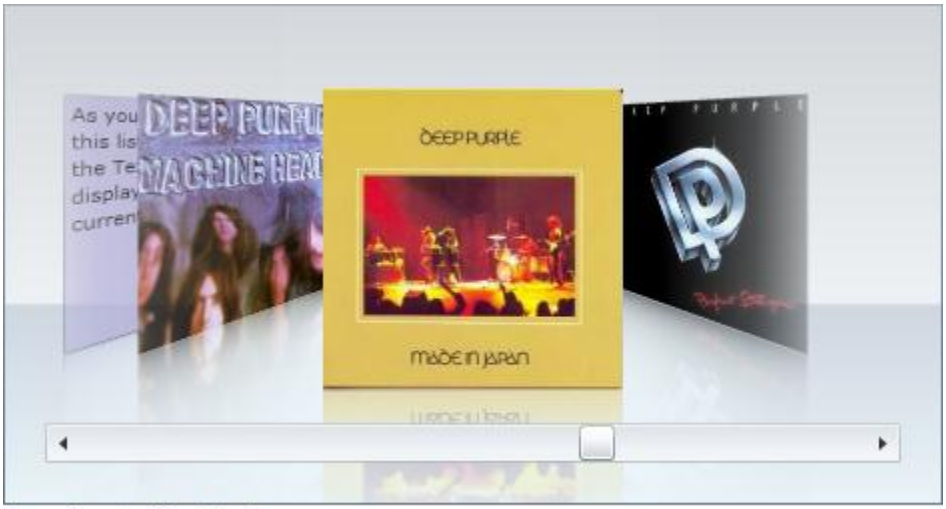
1. Press F5 to run the project and observe that the project looks as follows:



2. Click the scrollbar's next button once to change the selected item and observe that the selected album's name appears in the **TextBlock**.

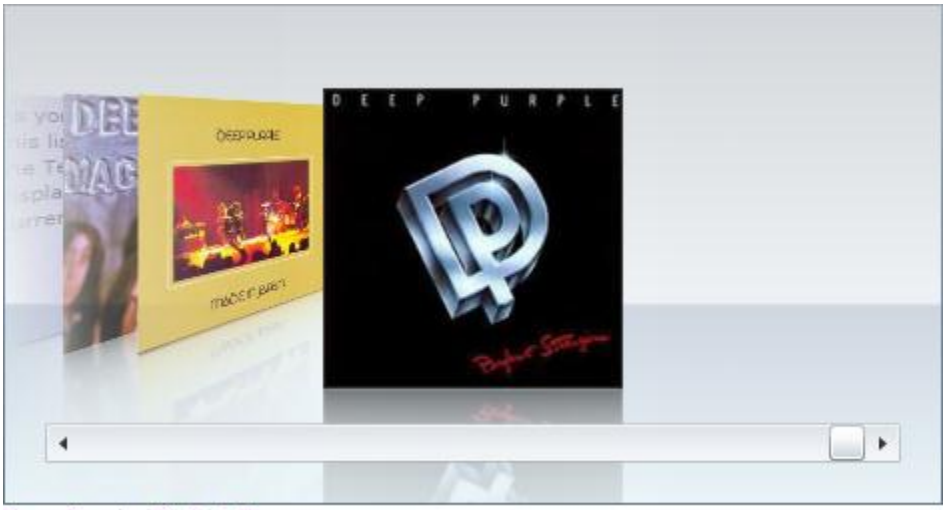
Deep Purple: Machine Head

3. Click the scrollbar's next button once again to change the selected item and observe that the selected album's name appears in the **TextBlock**.



Deep Purple: Made in Japan

4. Click the scrollbar's next button once again to change the selected item and observe that the selected album's name appears in the **TextBlock**.



Deep Purple: Perfect Strangers

Congratulations! You have completed the **CoverFlow for Silverlight** quick start. To learn more about the control, visit the C1CoverFlow Control Basics (page 152) and CoverFlow for Silverlight Task-Based Help (page 169) topics.

CoverFlow XAML Quick Reference

This topic is dedicated to providing a quick overview of the XAML used to complete various C1CoverFlow tasks. For more information, see the CoverFlow for Silverlight Task-Based Help (page 169) section.

C1CoverFlow with C1CoverFlowItems

The following XAML creates a C1CoverFlow control with two C1CoverFlowItems nested in it. The C1CoverFlowItems then have arbitrary controls nested within them.

```
        <c1:C1CoverFlow Height="278" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="c1CoverFlow1" VerticalAlignment="Top"
Width="378">
            <c1:C1CoverFlowItem Height="auto"
HorizontalAlignment="Left" Name="c1CoverFlowItem1"
VerticalAlignment="Top" Width="auto">
              <sdk:Calendar Height="172" Width="249" />
            </c1:C1CoverFlowItem>
                <Image
Source="/XAMLReference1;component/Images/Cropped.jpg" />
            <c1:C1CoverFlowItem />
        </c1:C1CoverFlow>
```

C1CoverFlow with Theme

The following XAML applies the **ShinyBlue** theme to the C1CoverFlow control.

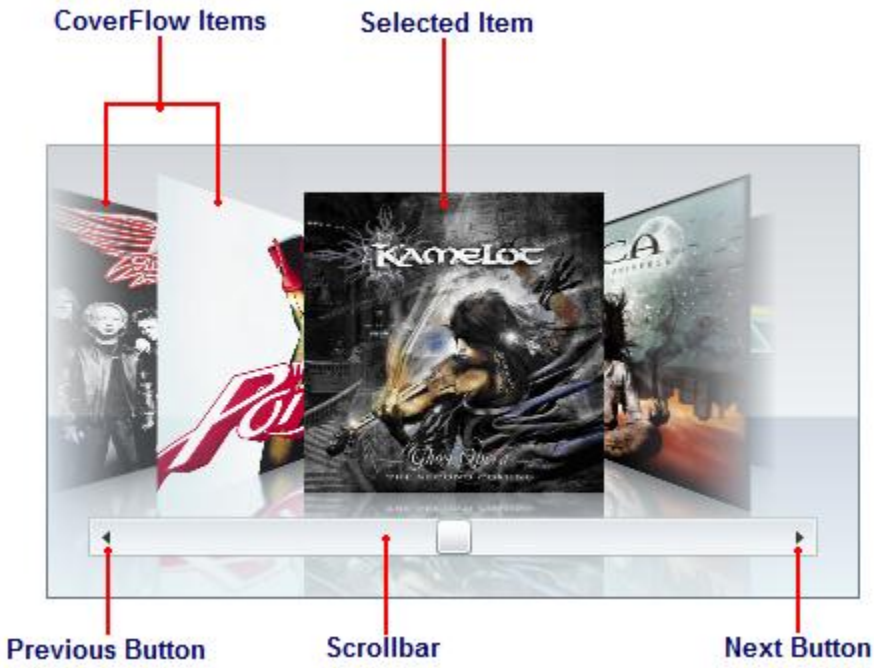
```
        <c1:C1ThemeShinyBlue Height="290" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="c1ThemeShinyBlue1" VerticalAlignment="Top"
Width="auto">
            <c1:C1CoverFlow Height="278" HorizontalAlignment="Left"
Margin="10,10,0,0" Name="c1CoverFlow1" VerticalAlignment="Top"
Width="378">
            </c1:C1CoverFlow>
        </c1:C1ThemeShinyBlue>
```

C1CoverFlow Control Basics

The C1CoverFlow is an animated, three-dimensional user interface. It is an Items control and, as such, can hold text, images, and controls.

Users can browse through the C1CoverFlow control using the scrollbar, or they can simply click an item in the list. When a user clicks an item on the list, that item will take focus by sliding into the center of the control, which will also cause other items to come into view.

The following image diagrams the elements of the C1CoverFlow control:



- **CoverFlow Items**

  The CoverFlow items can be anything from text to images to controls. Creating a CoverFlow item is simple: Just add a child item to the C1CoverFlow control.

- **Selected Item**

  The selected item always appears at the center of the control. As users scroll through the items, the selected item will change. Users can also select items by clicking on them. You can change the selected item by setting the **SelectedIndex** property.

- **Scrollbar**

  The scrollbar allows users to scroll through a list of CoverFlow items. You can remove the scrollbar through the C1CoverFlow template (see Removing the Scrollbar (page 172)).

- **Previous Button**

  The next button allows users to scroll to the right one item at a time. The next button is a part of the scrollbar item, meaning that if you remove the scrollbar, the previous button will also be removed.

- **Next Button**

The next button allows users to scroll to the right one item at a time. The next button is a part of the scrollbar item, meaning that if you remove the scrollbar, the next button will also be removed.

The following topics provide an overview of several of the C1CoverFlow control's features.

Eye Distance

The eye distance is the distance between the eye and the C1CoverFlow items. The eye distance is what determines how much perspective the items have.

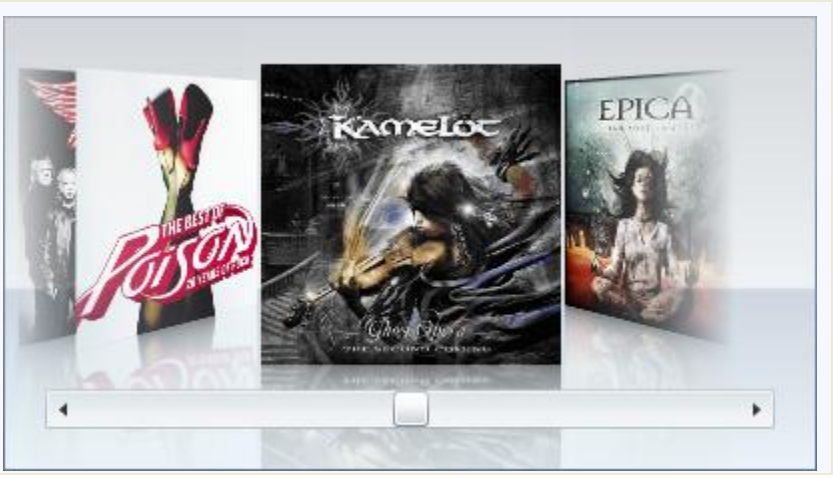
You can modify the eye distance by setting the EyeDistance property. The value of this property is relative to the current cover size, meaning that setting it to a value of 0.5 means that the distance will be half the size of the CoverFlow item. By default, this property is set to a value of 1.5.

Here are a few examples of EyeDistance property settings:

EyeDistance = 0.5

EyeDistance = 1

EyeDistance = 3



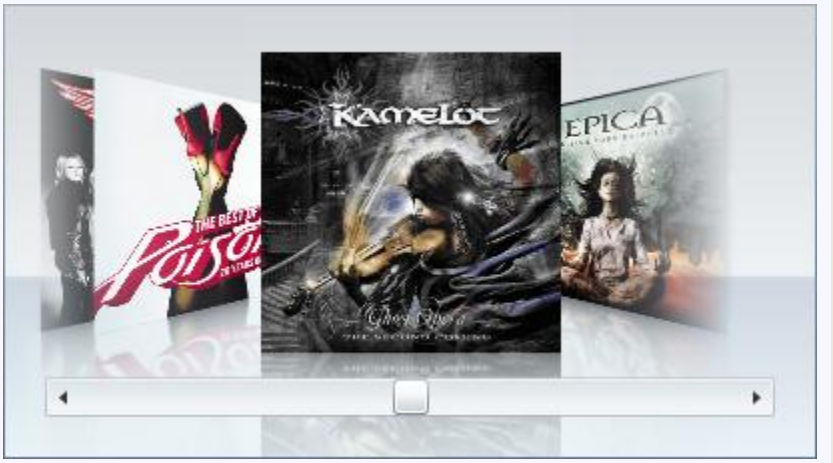
Eye Height

The EyeHeight property sets the level at which the camera sits and adjusts the view of the items accordingly. The value of this property is relative to the item size, meaning that a value of 0.5 will center the camera vertically. A value of 1 will place the camera at the bottom, and a value of 0 will place the camera overhead. The default value of this property is 0.25.

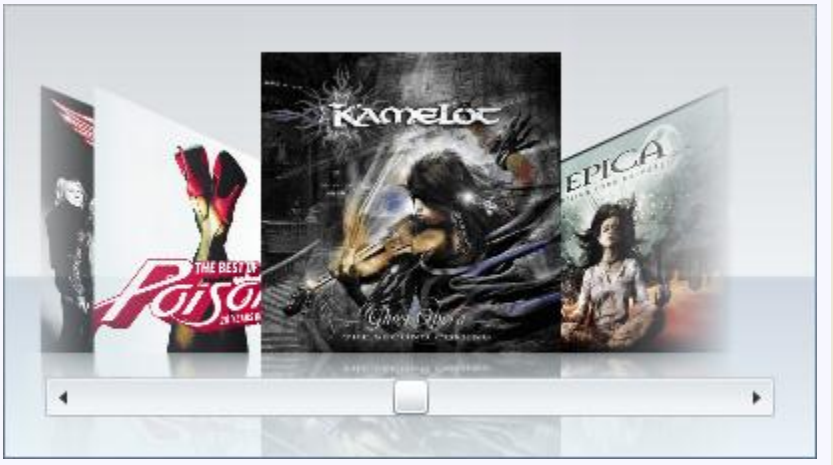
Here are a few examples of EyeHeight property settings:

EyeHeight = 0

EyeHeight = 0.5




```
EyeHeight= 1
```

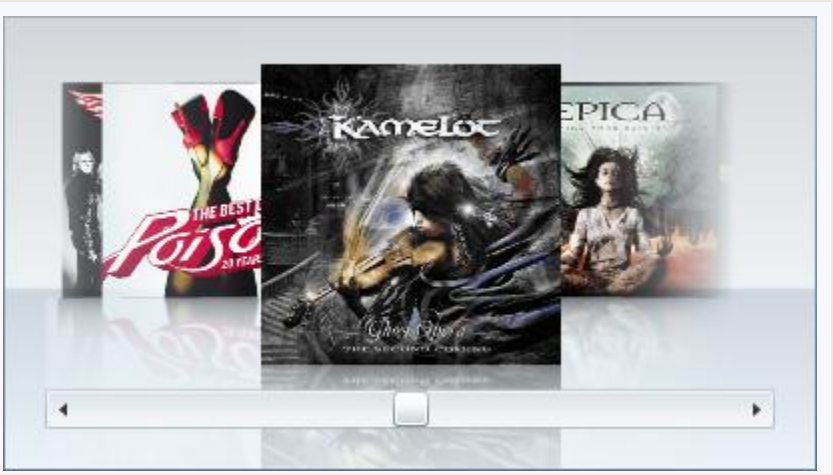


Item Angle

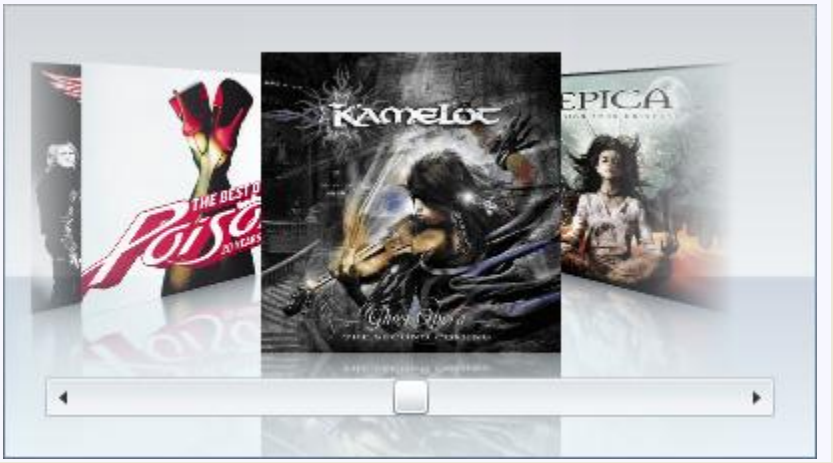
The ItemAngle property sets the angle of the control's unselected items. The value of the ItemAngle property is interpreted as a measurement of degree, so setting the ItemAngle property to 180 would flip the image horizontally. The default item angle is 60, which turns the unselected items at a 60-degree angle.

Here are a few examples of ItemAngle property settings:

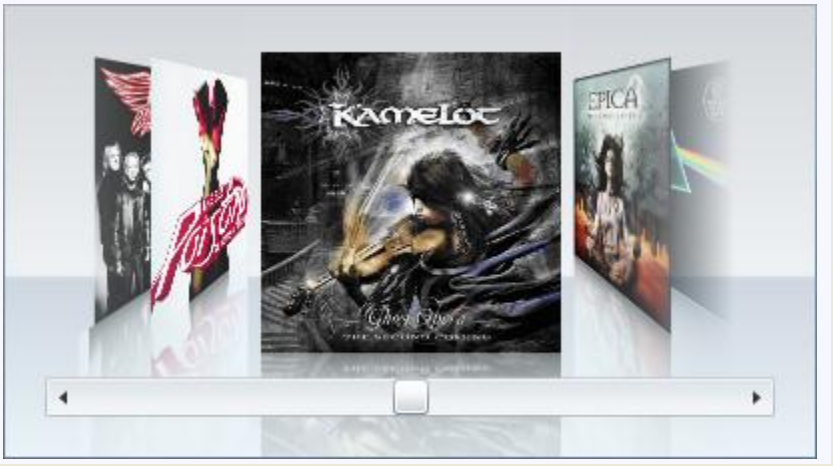
```
ItemAngle = 1
```



ItemAngle = 45



ItemAngle = 90

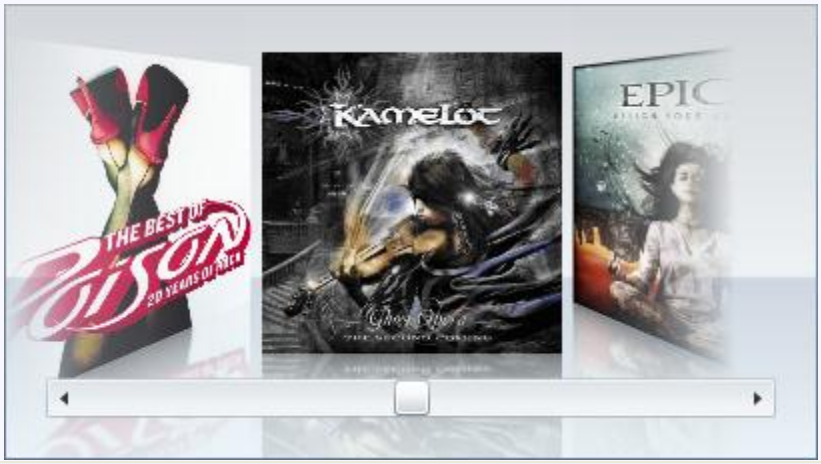


SelectedItem Offset

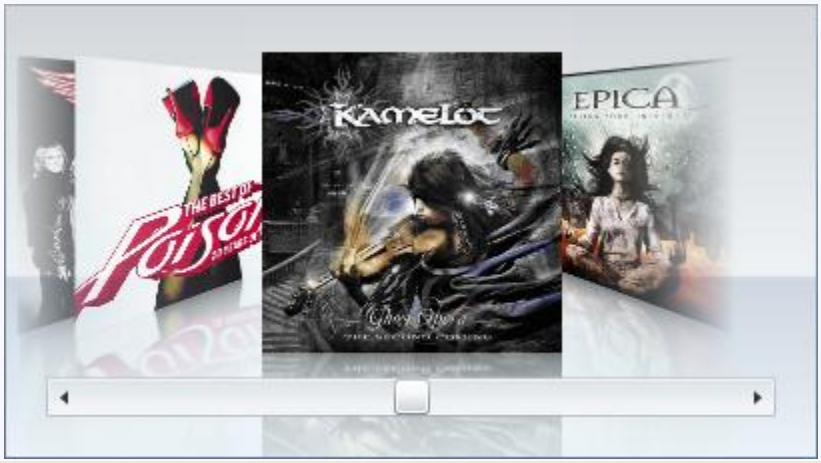
The SelectedItemOffset property determines how much closer the selected item is to the viewer than the unselected items. The value of this property is relative to the current item size, meaning that a value of 0.5 offsets the item to half of its own size.

Here are a few examples of SelectedItemOffset property settings:

SelectedItemOffset = 0



SelectedItemOffset = .5



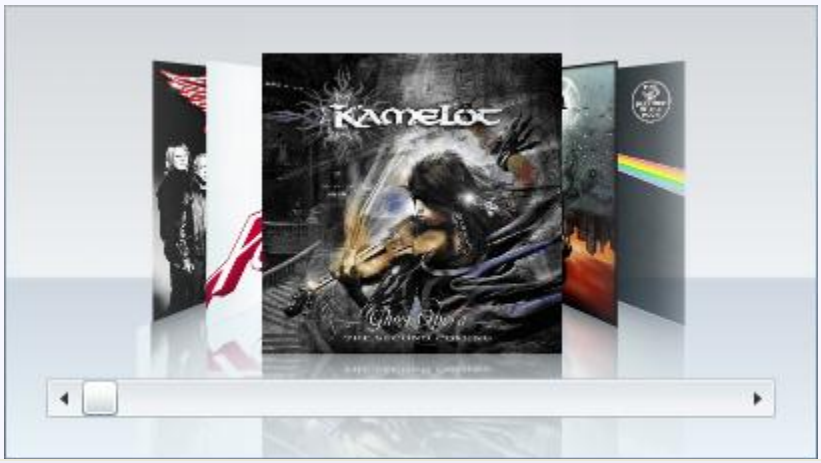
SelectedItemOffset = 1

Selected Item Distance

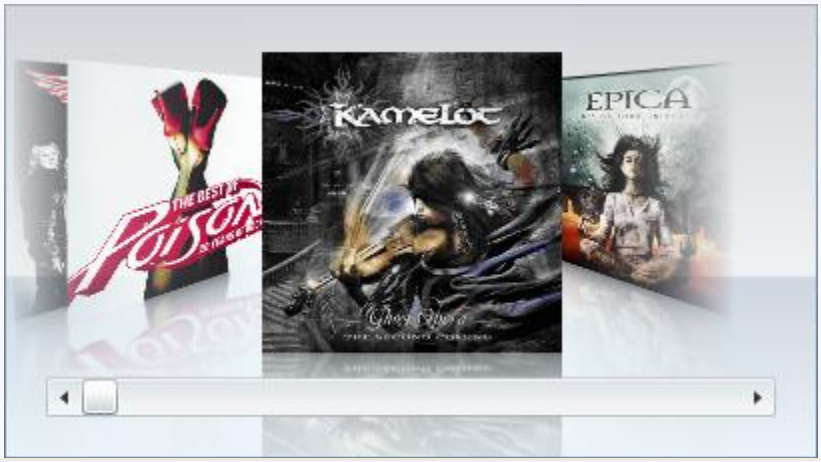
The SelectedItemDistance property sets the distance between the selected item and items directly next to it. The value for this property is relative to the current item size, meaning that a value of 0.5 places the distance between the selected item and the items next to it at half the size of the selected item.

Here are a few examples of SelectedItemDistance property settings:

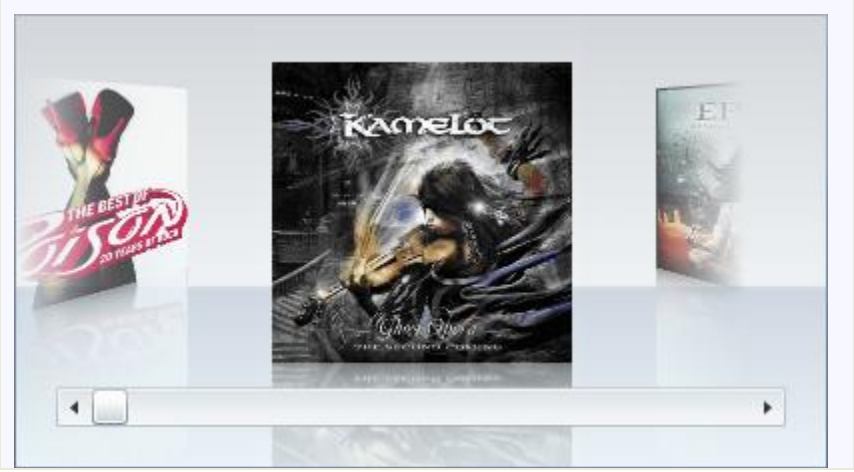
SelectedItemDistance = .5



SelectedItemDistance = 1



SelectedItemDistance = 1.5



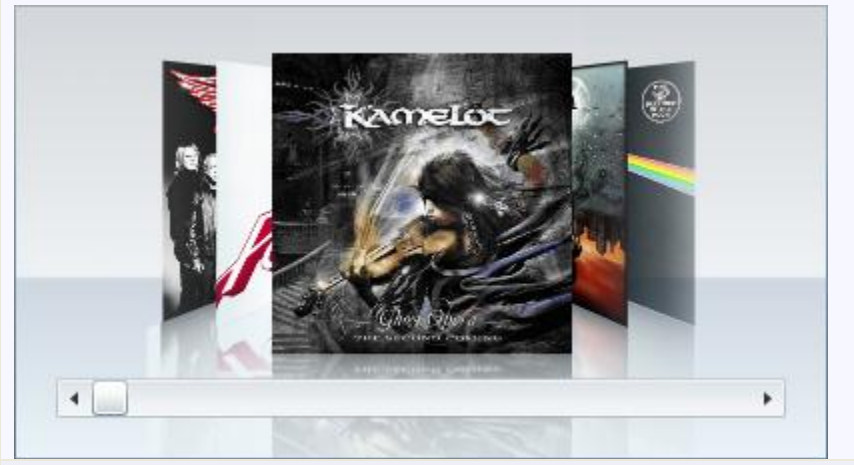
Item Distance

The ItemDistance property sets the distance between the unselected items. The value for this property is relative to the current item size, meaning that the distance between the items will be half of the item's size.

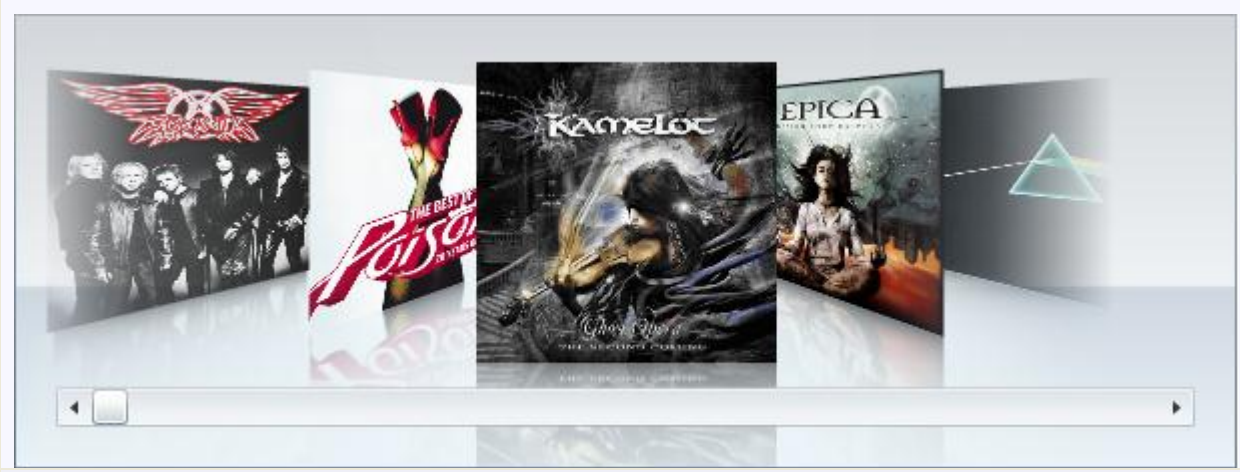
> **Note**: The ItemDistance property doesn't change the distance between the selected item and the items beside it. For more information, see the Selected Item Distance (page 158) topic.

Here are a few examples of ItemDistance property settings:

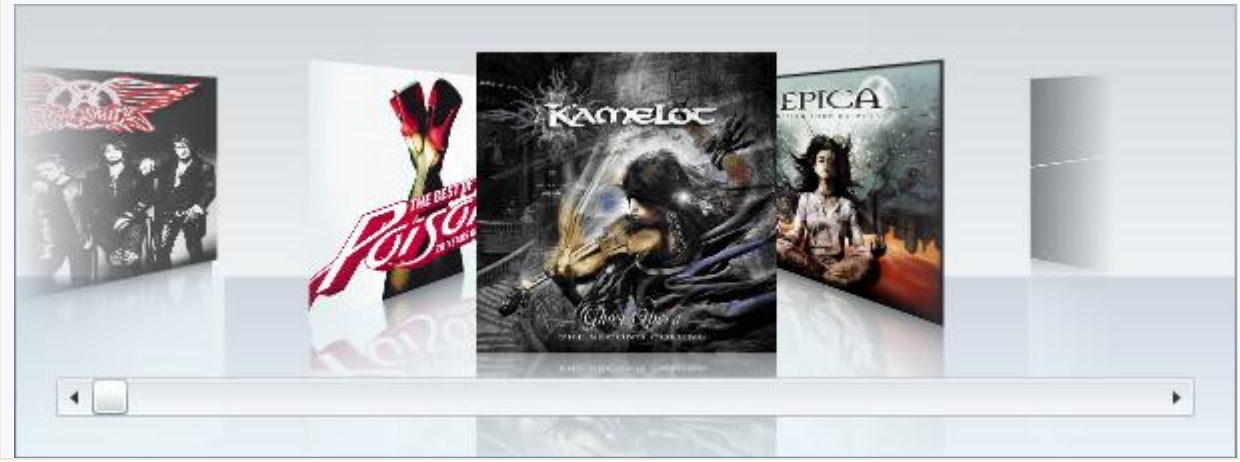
ItemDistance = 0.5



ItemDistance = 1



ItemDistance = 1.5



Speed Settings

You can adjust the three types of speed on the C1CoverFlow control: ItemSpeed, ItemAngleSpeed, and EyeSpeed. The table below provides an overview of each speed property.

Property	Description
ItemSpeed property	Gets or sets the speed at which the items' positions change when a user selects an item or scrolls through the control. This value should be greater than 0 and less than or equal than 1, with 1 meaning instantaneous changes.
ItemAngleSpeed property	Gets or sets the speed at which the items' angles change when a user selects an item or scrolls through the control.
EyeSpeed property	Gets or sets the speed at which the camera position changes. This value should be greater than 0 and less than or equal than 1, with 1 meaning instantaneous changes.

CoverFlow for Silverlight Layout and Appearance

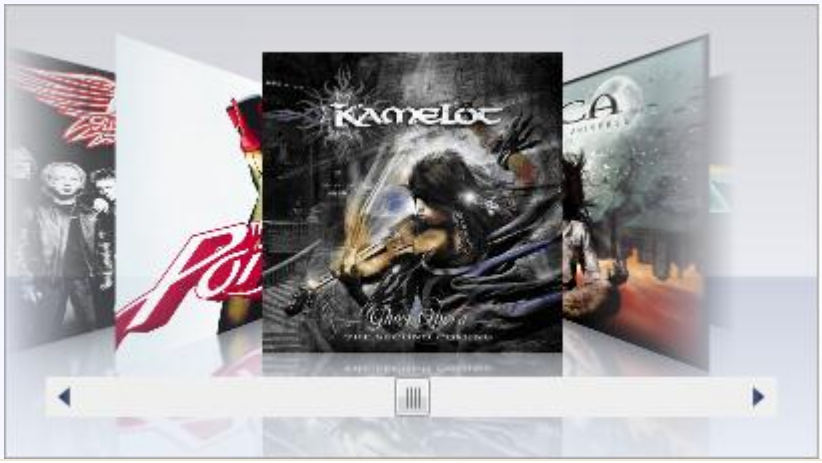
The following topics detail how to customize the C1CoverFlow control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the grid and take advantage of Silverlight's XAML-based styling. You can also use templates to format and layout the control and to customize the control's actions.

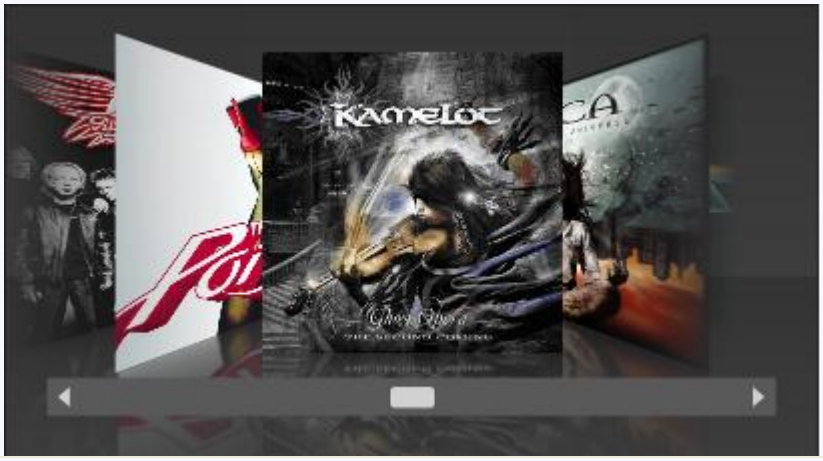
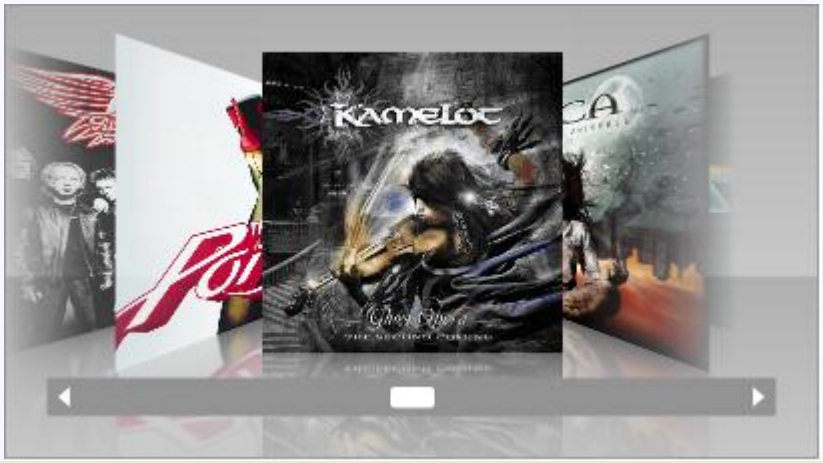
CoverFlow Theming

Silverlight themes are a collection of image settings that define the look of a control or controls. The benefit of using themes is that you can apply the theme across several controls in the application, thus providing consistency without having to repeat styling tasks.

When the C1CoverFlow control is added to your project, it appears with the default blue theme:

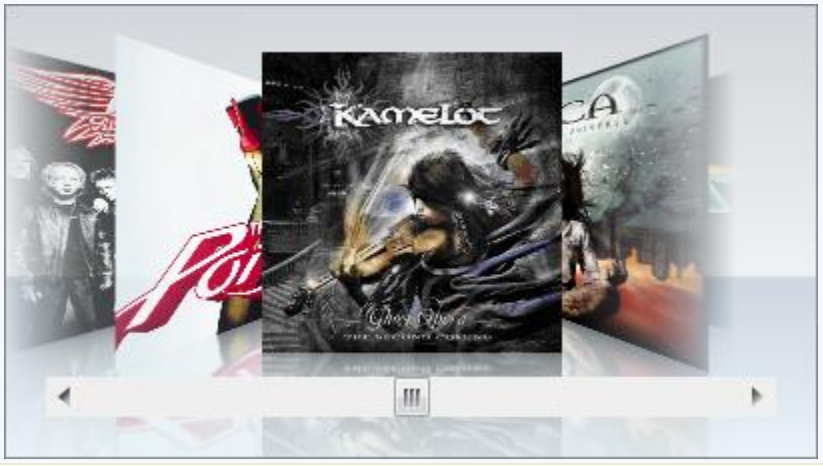
You can also theme the C1CoverFlow control with one of our six included Silverlight themes: BureauBlack, ExpressionDark, ExpressionLight, RainierOrange, ShinyBlue, and WhistlerBlue. The table below provides a sample of each theme.

Full Theme Name	Appearance
C1ThemeBureauBlack	

C1ThemeCosmopolitan	
C1ThemeExpressionDark	
C1ThemeExpressionLight	

C1ThemeOffice2007Black	
C1ThemeOffice2007Blue	
C1ThemeOffice2007Silver	

C1ThemeOffice2010Black	
C1ThemeOffice2010Blue	
C1ThemeOffice2010Silver	

C1ThemeRainierOrange	
C1ThemeShinyBlue	
C1ThemeWhistlerBlue	

For task-based help about adding themes to the C1CoverFlow control, see the Using C1CoverFlow Themes (page 181).

C1CoverFlow ClearStyle Properties

CoverFlow for Silverlight supports ComponentOne's new ClearStyle technology that allows you to easily change control colors without having to change control templates. By just setting a few color properties you can quickly style the entire grid.

The following table outlines the brush properties of the **C1CoverFlow** control:

Brushes	Description
Background	Gets or sets the brush of the control's background.
ButtonForeground	Gets or sets the brush of the buttons' foreground text.
ButtonBackground	Gets or sets the brush of the buttons' background text.
MouseOverBrush	Gets or sets the System.Windows.Media.Brush used to highlight the control when it is hovered over.
PressedBrush	Gets or sets the System.Windows.Media.Brush used to highlight the control when it is pressed on.

You can completely change the appearance of the **C1CoverFlow** control by setting a few properties, such as the **Background** property, which sets the background color of the CoverFlow's header. For example, if you set the **Background** property to "#FF490C0D", the **C1CoverFlow** control would appear similar to the following:

It's that simple with ComponentOne's ClearStyle technology. For more information on ClearStyle, see the ComponentOne ClearStyle Technology topic.

CoverFlow for Silverlight Appearance Properties

ComponentOne CoverFlow for Silverlight includes several properties that allow you to customize the appearance of the control. You can change the appearance of the text displayed in the control and customize graphic elements of the control. The following topics describe some of these appearance properties.

Text Properties

The following properties let you customize the appearance of text in the C1CoverFlow control.

Property	Description

FontFamily	Gets or sets the font family of the control. This is a dependency property.
FontSize	Gets or sets the font size. This is a dependency property.
FontStretch	Gets or sets the degree to which a font is condensed or expanded on the screen. This is a dependency property.
FontStyle	Gets or sets the font style. This is a dependency property.
FontWeight	Gets or sets the weight or thickness of the specified font. This is a dependency property.

Content Positioning Properties

The following properties let you customize the position content of the C1CoverFlow control.

Property	Description
HorizontalContentAlignment	Gets or sets the horizontal alignment of the control's content. This is a dependency property.
VerticalContentAlignment	Gets or sets the vertical alignment of the control's content. This is a dependency property.
VerticalAlignment	Gets or sets the vertical alignment characteristics applied to a FrameworkElement when it is composed within a parent object such as a panel or items control.
HorizontalAlignment	Gets or sets the horizontal alignment characteristics applied to a FrameworkElement when it is composed within a layout parent, such as a panel or items control.

Color Properties

The following properties let you customize the colors used in the C1CoverFlow control.

Property	Description
Background	Gets or sets a brush that describes the background of a control. This is a dependency property.
Foreground	Gets or sets a brush that describes the foreground color. This is a dependency property.

Border Properties

The following properties let you customize the border of the C1CoverFlow control.

Property	Description
BorderBrush	Gets or sets a brush that describes the border background of a control. This is a dependency property.
BorderThickness	Gets or sets the border thickness of a control. This is a dependency property.

Size Properties

The following properties let you customize the size of the C1CoverFlow control.

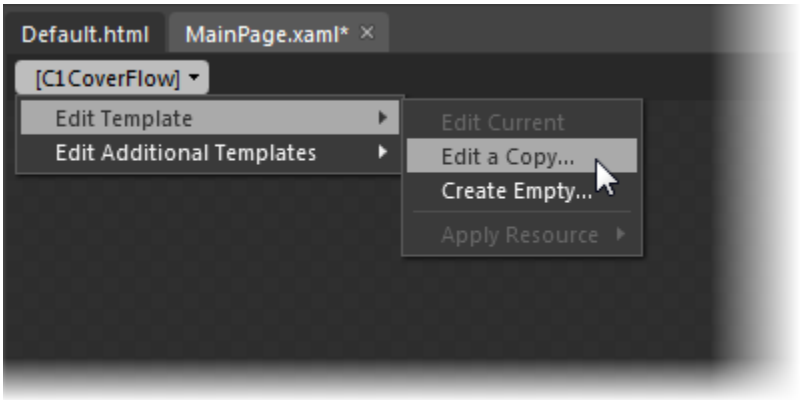
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
MinHeight	Gets or sets the minimum height constraint of the element. This is a dependency property.
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

Templates

One of the main advantages to using a Silverlight control is that controls are "lookless" with a fully customizable user interface. Just as you design your own user interface (UI), or look and feel, for Silverlight applications, you can provide your own UI for data managed by **ComponentOne CoverFlow for Silverlight**. Extensible Application Markup Language (XAML; pronounced "Zammel"), an XML-based declarative language, offers a simple approach to designing your UI without having to write code.

Accessing Templates

You can access templates in Microsoft Expression Blend by selecting the C1CoverFlow control and, in the menu, selecting **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or select **Create Empty** to create a new blank template.



If you want to edit the C1CoverFlowItem template, simply select the C1CoverFlowItem control and, in the menu, select **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or **Create Empty**, to create a new blank template.

Note: If you create a new template through the menu, the template will automatically be linked to that template's property. If you manually create a template in XAML you will have to link the appropriate template property to the template you've created.

Note that you can use the Template property to customize the template.

Additional Templates

In addition templates, the C1CoverFlow control and C1CoverFlowItem control include a few additional templates. These additional templates can also be accessed in Microsoft Expression Blend – in Blend select the C1CoverFlow or C1CoverFlowItem control and, in the menu, select **Edit Additional Templates**. Choose a template, and select **Create Empty.**

CoverFlow for Silverlight Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1CoverFlow control in general. If you are unfamiliar with the **ComponentOne CoverFlow for Silverlight** product, please see the **CoverFlow for Silverlight** Quick Start first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne CoverFlow for Silverlight** product.

Each task-based help topic also assumes that you have created a new Silverlight project.

Working with Item Reflections

The following topics illustrate how to remove, modify, and style the reflection that appears underneath the C1CoverFlow control's items.

Adding a Blur Effect to the Reflection

In this topic, you will use Expression Blend to add a blur effect to the reflections of a C1CoverFlow control's items.

Complete the following steps:

1. Right-click the C1CoverFlow control and select **Edit Additional Templates | Edit Generated Item Container (ItemContainerStyle) | Edit a Copy**.

   The **Create Style Resource** dialog box opens.
2. In the **Name(Key)** field, enter "AddBlurEffect".
3. Click **OK** to create the template and close the **Create Style Resource** dialog box. Observe that the **Objects and Timeline** panel shows a **Template** with a **Reflector** control as its child item.
4. In the **Objects and Timeline** panel, select **Reflector** to reveal its list of properties in the **Properties** panel and then complete the following:
   a. Expand the **Miscellaneous** node.
   b. Click the **ReflectionEffects(Collection)** ellipsis button.

      The **Effect Collection Editor: ReflectionEffects** dialog box opens.
   c. Click **Add another item** to open the **Select Object** dialog box.
   d. Select **BlurEffect** and click **OK** to add the blur effect and to close the **Select Object** dialog box.
   e. Set the **BlurEffect**'s Radius property to "45".
   f. Click **OK** to close the **Effect Collection Editor: ReflectionEffects** dialog box.

This Topic Illustrates the Following:

The following image demonstrates what the reflection of a C1CoverFlow control's items looks like with a heavy blur effect.

Modifying the Reflection

In this topic, you'll use Expression Blend to modify the reflections of a C1CoverFlow control's items.

Complete the following steps:

1. Right-click the C1CoverFlow control and select **Edit Additional Templates | Edit Generated Item Container (ItemContainerStyle) | Edit a Copy**.

   The **Create Style Resource** dialog box opens.
2. In the **Name(Key)** field, enter "ModifyReflection".
3. Click **OK** to create the template and close the **Create Style Resource** dialog box. Observe that the **Objects and Timeline** panel shows a **Template** with a **Reflector** control as its child item.
4. In the **Objects and Timeline** panel, select **Reflector** to reveal its list of properties in the **Properties** panel and then complete the following:
   a. Expand the **Miscellaneous** node.
   b. Click the **ReflectionEffects(Collection)** ellipsis button.

      The **Effect Collection Editor: ReflectionEffects** dialog box opens.
   c. Set the **ReflectionOpacityEffect**'s properties as you wish. For this example, set them as follows:
      - Set the **Coefficient** property to "5".
      - Set the **Offset** property to "0".

This Topic Illustrates the Following:

The following image shows the modified reflections of a C1CoverFlow control's items:

Removing the Reflection Effect

You can remove the items' reflections by creating a copy of the **ItemContainerStyle** template and then replacing the **Reflector** control with a **ContentPresenter** control. This topic assumes you have added a C1CoverFlow control with at least one item to your project.

Complete the following steps:

1. Right-click the C1CoverFlow control and select **Edit Additional Templates | Edit Generated Item Container (ItemContainerStyle) | Edit a Copy**.

   The **Create Style Resource** dialog box opens.
2. In the **Name(Key)** field, enter "Remove Reflection".
3. Click **OK** to create the template and close the **Create Style Resource** dialog box. Observe that the **Objects and Timeline** panel shows a **Template** with a **Reflector** control as its child item.
4. Open the **Assets** panel and, its search box, enter "ContentPresenter".
5. Double-click the **ContentPresenter** icon to replace the **Reflector** control with the **ContentPresenter** control. Observe that the reflection has been removed.
6. With the **ContentPresenter** control selected, navigate to the **Properties** panel and expand the **Transform** node.
7. Click the **Projection** property's **Advanced options** button and select **Custom Expression**.
8. In the **Custom expression** text box, enter "{TemplateBinding ContentProjection}". This ensures that the items in the C1CoverFlow control will project the same way that they did when the control was using the default **Reflector** control.

This Topic Illustrates the Following:

The following image demonstrates what the items of a C1CoverFlow control look like without a reflection.

Working with the Scrollbar

The following topics illustrate how to remove, size, and color the C1CoverFlow control's scrollbar.

Removing the Scrollbar

In this topic, you will use Expression Blend to remove the C1CoverFlow control's scrollbar.

Complete the following steps:

1. Right-click the C1CoverFlow control and select **Edit Template | Edit a Copy**.

   The **Create Style Resource** dialog box opens.
2. In the **Name (Key)** field, enter "RemoveScrollbar".
3. Press **OK** to create the template and close the **Create Style Resource** dialog box.
4. In the **Objects and Timeline** panel, expand the **[Grid]** node.
5. Select **Scroll** and then press DELETE.

   The C1CoverFlow control's scrollbar is removed.

This Topic Illustrates the Following:

The following image demonstrates a C1CoverFlow control with its scrollbar removed.

Resizing the Scrollbar

In this topic, you will use Expression Blend to resize the C1CoverFlow control's scrollbar.

1. Right-click the C1CoverFlow control and select **Edit Template | Edit a Copy**.

   The **Create Style Resource** dialog box opens.
2. In the **Name (Key)** field, enter "ResizeScrollbar".
3. Press **OK** to create the template and close the **Create Style Resource** dialog box.
4. In the **Objects and Timeline** panel, expand the **[Grid]** node.
5. Select **Scroll** to reveal its list of properties in the **Properties** panel and then set the following properties:
   - Set the **Width** property to "150".
   - Set the **Height** property to "35".

This Topic Illustrates the Following:

The following image depicts a C1CoverFlow control with a scrollbar that has a width of 150 pixels and a height of 35 pixels.

Adding Images to the C1CoverFlow Control

In this topic, you'll learn how to add images to the C1CoverFlow control in Blend, in XAML, and in code.

In Blend

Complete the following steps:

1. Add a C1CoverFlow control to your project.
2. In the **Objects and Timeline** panel, select **[C1CoverFlow]**.
3. In the **Assets** panel, enter "Image" into the search field.
4. Double-click the **Image** icon to add the **Image** control to the C1CoverFlow control.
5. In the **Objects and Timeline** panel, select **[Image]**.
6. Under the **Properties** tab, click the **Source** ellipsis button.

   The **Add Existing Item** dialog box opens.

7. Navigate to the location of your image, select the image file, and click **Open** to add the image to the **Image** control.

In XAML

Complete the following steps:

1. Add a closing tag for the C1CoverFlow control so that the XAML appears similar to the following:

```
<c1:C1CoverFlow Margin="0,0,205,200"></c1:C1CoverFlow>
```

2. Place the following XAML between the `<c1:C1CoverFlow>` and `</c1:C1CoverFlow>` tags, replacing "YourImage.png" with the name of your image file:

```
<Image Height="100" Width="100" Source="YourImage.png"/>
```

In Code

Complete the following steps:

1. In XAML view, add `"x:Name="C1CoverFlow1"` to the `<c1:C1CoverFlow>` tag so that the control will have a unique identifier for you to call in code.
2. Open the **MainPage.xaml** code page (either **MainPage.xaml.cs or MainPage.xaml.vb**, depending on which language you've chosen for your project).
3. Import the following namespace:
   - Visual Basic

     ```
     Imports System.Windows.Media.Imaging
     ```

   - C#

     ```
     using System.Windows.Media.Imaging;
     ```

4. Add the following code beneath the **InitializeComponent** method:
   - Visual Basic

     ```
     ' Create the Image control
     Dim Image1 As New Image()

     ' Create a bitmap image and add your image as its source
     Dim BitMapImage1 As New BitmapImage()
     BitMapImage1.UriSource = New Uri("Epica.jpg",
     UriKind.RelativeOrAbsolute)

     ' Add the bitmap image as the Image control's source
     Image1.Source = BitMapImage1

     'Add the Image control to the C1CoverFlow control
     C1CoverFlow1.Items.Add(Image1)
     ```

   - C#

```
// Create the Image control
Image Image1 = new Image();

// Create a bitmap image and add your image as its source
BitmapImage BitMapImage1 = new BitmapImage();
BitMapImage1.UriSource = new Uri("Epica.jpg",
UriKind.RelativeOrAbsolute);

// Add the bitmap image as the Image control's source
Image1.Source = BitMapImage1;

//Add the Image control to the C1CoverFlow control
C1CoverFlow1.Items.Add(Image1);
```

5. Run the program.

Binding to Objects in an Object Collection

The C1CoverFlow control can be bound to a collection of items. In this section, you will bind the control to an **ObservableCollection** of type string. The collection you will bind to will be passed as a data context. This topic assumes you are working in Microsoft Expression Blend and that you have added a C1CoverFlow control to your project.

Complete the following steps:

1. Select the C1CoverFlow control and set the following properties:
   - Set the **Width** property to "400".
   - Set the **Height** property to "200".
   - Set the **SelectedIndex** property to "2".
2. Add images to the project by completing the following steps:
   a. In the **Projects** panel, right-click the project to open its context menu and select **Add New Folder**; name the new folder "Images".
   b. Right-click the **Images** folder and select **Add Existing Item**.

      The **Add Existing Item** dialog box opens.
   c. Navigate to the following location:
      - In XP

        C:\Documents and Settings\<username>\My Documents\ComponentOne Samples\Studio for Silverlight\QuickStart\QuickStart
      - Windows 7/Vista

        C:\Users\<username>\Documents\ComponentOne Samples\Studio for Silverlight\QuickStart\QuickStart
   d. Select **cover1.jpg**, **cover2.jpg**, **cover3.jpg**, **cover4.jpg**, and **cover5.jpg**.
   e. Click **Open** to close the **Add Existing Item** dialog box and to add the images to the folder.

      The images are added to the **Images** folder.

3. Open the **MainPage.xaml** code page (either **MainPage.xaml.cs or MainPage.xaml.vb**).
4. Place the following code beneath the **Initialize ()** method:
   - Visual Basic

```
' Create the ObservableCollection
Dim bluRayCovers As New
System.Collections.ObjectModel.ObservableCollection(Of String)()
bluRayCovers.Add("Images/cover1.jpg")
bluRayCovers.Add("Images/cover2.jpg")
bluRayCovers.Add("Images/cover3.jpg")
bluRayCovers.Add("Images/cover4.jpg")
bluRayCovers.Add("Images/cover5.jpg")
' Pass the collection to the control as a data context
Me.DataContext = bluRayCovers
```

   - C#

```
// Create the ObservableCollection
System.Collections.ObjectModel.ObservableCollection<string>bluRayCovers
= new System.Collections.ObjectModel.ObservableCollection<string>();
bluRayCovers.Add("Images/cover1.jpg");
bluRayCovers.Add("Images/cover2.jpg");
bluRayCovers.Add("Images/cover3.jpg");
bluRayCovers.Add("Images/cover4.jpg");
bluRayCovers.Add("Images/cover5.jpg");
//Pass the collection to the control as a data context
this.DataContext = bluRayCovers;
```

Bind the collection to the control by completing the following steps:

a. Return to Design view.
b. Select the C1CoverFlow control to reveal its list of properties in the **Properties** panel.
c. Next to the **ItemsSource** property, click the **Advanced options** button and select **Custom Expression**.
d. Set the **Custom expression** field to "{Binding}". This sets up the **ItemsSource** to pass the **DataContext** directly to its template, which you will create in the next step.

5. Create a **DataTemplate** with an **Image** control and then bind the **Image** control's **Source** property to the collection by completing the following steps:
   a. Right-click the C1CoverFlow control and select **Edit Additional Templates | Edit Generated Items | Create Empty**.

      The **Create ControlTemplate Resource** dialog box opens.
   b. In the **Name (Key)** field, enter "ImageTemplate".
   c. Click **OK** to close the **Create ControlTemplate Resource** dialog box and create the template.
   d. In the **Assets** panel, enter "Image" into the search bar and then double-click the **Image** icon to add the **Image** control to the template.

e. Select the **Image** control to reveal its list of properties in the **Properties** panel and then complete the following:

- Click the **Width** property's glyph to set the **Width** property to **Auto**.
- Click the **Height** property's glyph to set the **Height** property to **Auto**.
- Click the **Source** property's **Advanced options** button, select **Custom Expression**, and set the **Custom expression** field to "{Binding}".

6. Run the project and observe that the images specified in the collection are now C1CoverFlow control items.

This Topic Illustrates the Following:

The following image demonstrates the results of this topic.

Changing the Angle of Coverflow Side Items

You can change the angle of the items that lie on each side of the C1CoverFlow control's selected item by setting the ItemAngle property to a numeric value (for more information, see the Item Angle (page 155) topic). In this topic, you will set the ItemAngle property to **15** so that the side items will be turned at a 15-degree angle.

In Blend

Complete the following steps:

1. Select the C1CoverFlow control.
2. In the **Properties** panel, set the ItemAngle property to "15".

In XAML

Add `ItemAngle="1"` to the `<c1:C1CoverFlow>` tag so that the markup resembles the following:

```
<c1:C1CoverFlow Margin="0,0,87,233" ItemAngle="15">
```

In Code

Complete the following steps:

1. In XAML view, add `"x:Name="C1CoverFlow"` to the `<c1:C1CoverFlow>` tag so that the control will have a unique identifier for you to call in code.
2. Add the following code beneath the **InitializeComponent** method:

- Visual Basic

```
C1CoverFlow1.ItemAngle = 15
```

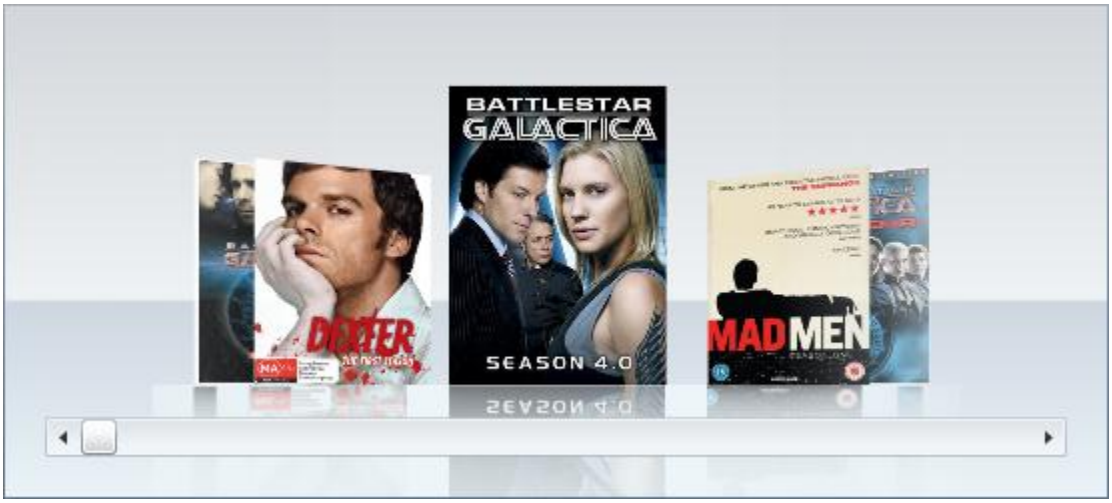
- C#

```
C1CoverFlow1.ItemAngle = 15;
```

3. Run the program.

This Topic Illustrates the Following:

The following image demonstrates a C1CoverFlow control with its ItemAngle property set to **15**.



Changing the Camera's Vertical Position

You can change the vertical position of the camera by setting the EyeHeight property (for more information, see the Eye Height (page 154) topic). In this topic, you'll set the EyeHeight property to **1** so that the camera will view the C1CoverFlow items from the bottom.

In Blend

Complete the following steps:

1. Select the C1CoverFlow control.
2. In the **Properties** panel, set the EyeHeight property to "1".

In XAML

Add `EyeHeight="1"` to the `<c1:C1CoverFlow>` tag so that the markup resembles the following:

```
<c1:C1CoverFlow Margin="0,0,87,233" EyeHeight="1">
```

In Code

Complete the following steps:

1. In XAML view, add `"x:Name="C1CoverFlow1"` to the `<c1:C1CoverFlow>` tag so that the control will have a unique identifier for you to call in code.
2. Add the following code beneath the **InitializeComponent** method:

- Visual Basic

```
C1CoverFlow1.EyeHeight = 1
```

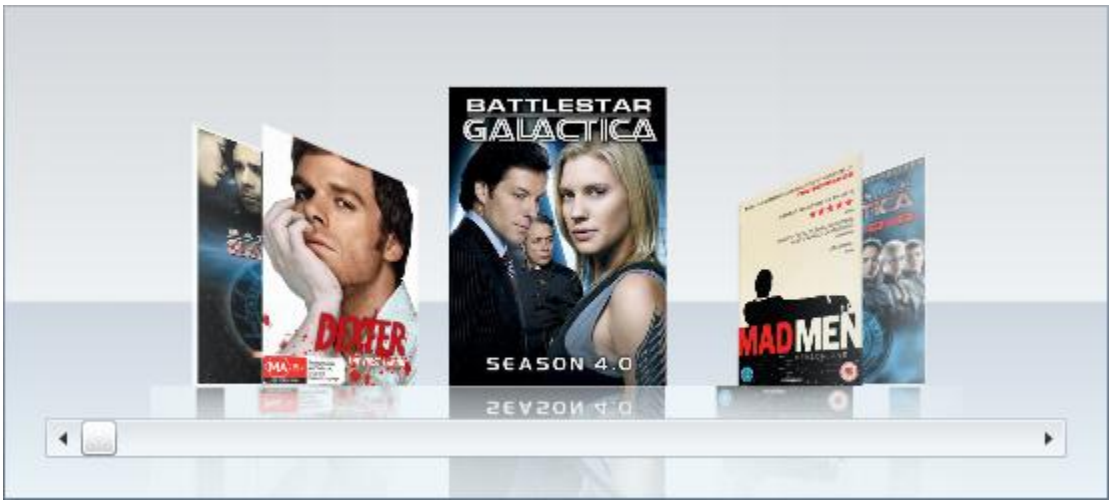
- C#

```
C1CoverFlow1.EyeHeight = 1;
```

3. Run the program.

This Topic Illustrates the Following:

The following image demonstrates a C1CoverFlow control with its EyeHeight property set to **1**.



Setting the Distance Between the Selected Item and the Side Items

The SelectedItemDistance property sets the distance between the selected item, which appears in the center of the control, and the items that are on either side of it (for more information, see the Selected Item Distance (page 158) topic). In this topic, you will set the SelectedItemDistance property to 0.4 so that the distance between the selected item and its side items is 4/10ths the size of the selected item.

In Blend

Complete the following steps:

1. Select the C1CoverFlow control.
2. In the **Properties** panel, set the SelectedItemDistance property to "0.4".

In XAML

Add `SelectedItemDistance="0.4"` to the `<c1:C1CoverFlow>` tag so that the markup resembles the following:

```
<c1:C1CoverFlow Margin="0,0,87,233" SelectedItemDistance="0.4">
```

In Code

Complete the following steps:

1. In XAML view, add `"x:Name="C1CoverFlow"` to the `<c1:C1CoverFlow>` tag so that the control will have a unique identifier for you to call in code.
2. Add the following code beneath the **InitializeComponent** method:
   - Visual Basic

     ```
     C1CoverFlow1.SelectedItemDistance = 0.4
     ```

   - C#

```
C1CoverFlow1.SelectedItemDistance = 0.4;
```

3. Run the program.

This Topic Illustrates the Following:

The following image demonstrates a C1CoverFlow control with its SelectedItemDistance property set to **0.4**.

Setting the Distance Between Items

The ItemDistance property sets the distance between the unselected C1CoverFlow items (for more information, see the Item Distance (page 159) topic). In this topic, you will set the ItemDistance property to 1 so that the distance between items will be equal to the item size.

Note: The ItemDistance property doesn't change the distance between the selected item and the items beside it. For more information, see the Selected Item Distance (page 158) topic.

In Blend

Complete the following steps:

1. Select the C1CoverFlow control.
2. In the **Properties** panel, set the ItemDistance property to "1".

In XAML

Add `ItemDistance="1"` to the `<c1:C1CoverFlow>` tag so that the markup resembles the following:

```
<c1:C1CoverFlow Margin="0,0,87,233" ItemDistance="1">
```

In Code

Complete the following steps:

1. In XAML view, add `"x:Name="C1CoverFlow"` to the `<c1:C1CoverFlow>` tag so that the control will have a unique identifier for you to call in code.
2. Add the following code beneath the **InitializeComponent** method:
   - Visual Basic

     ```
     C1CoverFlow1.ItemDistance = 1
     ```

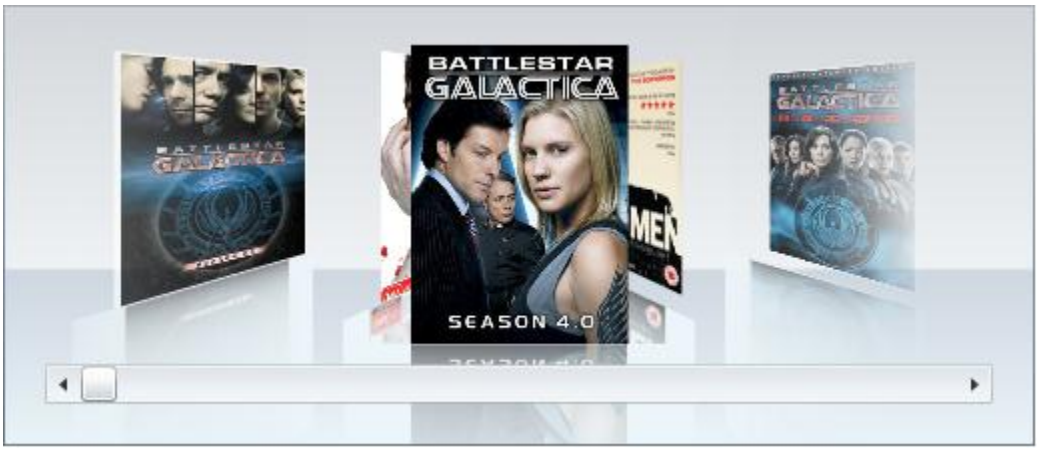
- C#

```
C1CoverFlow1.ItemDistance = 1;
```

3. Run the program.

This Topic Illustrates the Following:

The following image demonstrates a C1CoverFlow control with its ItemDistance property set to **1**.



Using C1CoverFlow Themes

The C1CoverFlow control comes equipped with a light blue default theme, but you can also apply six themes (see CoverFlow Theming (page 161)) to the control. In this topic, you will change the C1CoverFlow control's theme to **C1ThemeRainierOrange**.

In Blend

Complete the Following steps:

1. Click the **Assets** tab.
2. In the search bar, enter "C1ThemeRainierOrange".

   The **C1ThemeRainierOrange** icon appears.
3. Double-click the **C1ThemeRainierOrange** icon to add it to your project.
4. In the search bar, enter "C1CoverFlow" to search for the C1CoverFlow control.
5. Double-click the C1CoverFlow icon to add the C1CoverFlow control to your project.
6. Under the **Objects and Timeline** tab, select **[C1CoverFlow]** and use a drag-and-drop operation to place it under **[C1ThemeRainierOrange]**.
7. Run the project.

In Visual Studio

Complete the following steps:

1. Open the **.xaml** page in Visual Studio.
2. Place your cursor between the `<Grid></Grid>` tags.
3. In the **Tools** panel, double-click the **C1ThemeRainierOrange** icon to declare the theme. Its tags will appear as follows:

```
<my:C1ThemeRainierOrange></my:C1ThemeRainierOrange>
```

4. Place your cursor between the `<my:C1ThemeRainierOrange>` and `</my:C1ThemeRainierOrange>` tags.
5. In the **Tools** panel, double-click the C1CoverFlow icon to add the control to the project. Its tags will appear as children of the `<my:C1ThemeRainierOrange>` tags, causing the markup to resemble the following:

```
<my:C1ThemeRainierOrange>
            <c1:C1CoverFlow x:Name="C1CoverFlow1"></c1:C1CoverFlow>
</my:C1ThemeRainierOrange>
```

6. Run your project.

This Topic Illustrates the Following:

The following image depicts a C1CoverFlow control with the **C1ThemeRainierOrange** theme.

Expander

Save precious screen real estate with **ComponentOneExpander™ for Silverlight**. **Expander for Silverlight** includes one control, **C1Expander**, which allows you create an expandable and collapsible information panel that can include text, images, and controls. Choose from four expand directions and take complete control of the control's style by customizing its appearance in Microsoft Expression Blend.

Getting Started

- Working with the C1Expander control (page 186)
- Quick Start (page 183)
- Task-Based Help (page 196)

Expander for Silverlight Key Features

ComponentOne Expander for Silverlight allows you to create customized, rich applications. Make the most of **Expander for Silverlight** by taking advantage of the following key features:

- **Expand on Page Load**

  You can choose whether or not the **C1Expander** control is expanded upon page load by using the IsExpanded property. By default, the IsExpanded property is set to **True** and the control is initially appears expanded. For more information, see the Expandability (page 190) topic.

- **Expand Direction**

  The **C1Expander** control has the ability to expand in four different directions. The ExpandDirection property indicated in which direction the control expands and can be set to **Top**, **Right**, **Bottom**, or **Left**. For more information, see the Expand Direction (page 189) topic.

- **Custom Header**

  The **C1Expander** control's header can be customized with both text and controls. For more information on the customizable header element, see Expander Header (page 187).

- **Configure Items in an Organized Pattern**

  Expander is designed to maximize space. Configure the size and position of C1Expander to hide items until needed.

Expander for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **Expander for Silverlight**. In this quick start, you'll start in Visual Studio to create a new project, add a C1Expander control to your application, and then add content to the C1Expander control's content area.

Step 1 of 3: Creating an Application with a C1Expander Control

In this step, you'll begin in Visual Studio to create a Silverlight application using **Expander for Silverlight**.

Complete the following steps:

1. In Visual Studio, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Silverlight Application**. Enter a **Name** for your project and click **OK**. The **New Silverlight Application** dialog box will appear.

3. Click **OK** to close the **New Silverlight Application** dialog box and create your project.
4. In the XAML window of the project, resize the **UserControl** by changing `DesignWidth="400" DesignHeight="300"` to `DesignWidth="Auto" DesignHeight="Auto"` in the `<UserControl>` tag so that it appears similar to the following:

```
<UserControl x:Class="SilverlightApplication24.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="Auto" d:DesignHeight="Auto">
```

   The **UserControl** will now resize to accommodate any content placed within it.
5. In the XAML window of the project, place the cursor between the `<Grid>` and `</Grid>` tags and click once. Note that you cannot currently add Silverlight controls directly to the design area in Visual Studio, so you must add them to the XAML window as directed in the next step.
6. Navigate to the **Tools** panel and double-click the C1Expander icon to add the control to the grid. The XAML markup resembles the following:

```
<Grid x:Name="LayoutRoot">
<c1:C1Expander></c1:C1Expander>
</Grid>
</UserControl>
```

You've successfully created a Silverlight application containing a C1Expander control. In the next step, you will customize the appearance and behavior of the C1Expander control.

Step 2 of 3: Customizing the C1Expander Control

In the last step, you created a Silverlight project and added a C1Expander control to it. In this step, you will customize the behavior and appearance of the C1Expander control.

Complete the following steps:

1. Add `Height="75"` to the `<c1:C1Expander>` tag to set the height of the control. The XAML markup appears as follows:

```
<c1:C1Expander Height="75">
```

2. Add `Width="140"` to the `<c1:C1Expander>` tag to set the width of the control. The XAML markup appears as follows:

```
<c1:C1Expander Height="75" Width="140">
```

3. Add `Header="Expander Quick Start"` to the `<c1:C1Expander>` tag to set the header text of the control. The XAML markup appears as follows:

```
<c1:C1Expander Height="75" Width="140" Header="Expander Quick Start">
```

4. Add `Background="Aqua"` to the `<c1:C1Expander>` tag to set the background color of the content area. The XAML markup appears follows:

```
<c1:C1Expander Height="75" Width="140" Header="Expander Quick Start"
Background="Aqua">
```

5. Add `ExpandDirection="Up"` to the `<c1:C1Expander>` tag so that the C1Expander control will expand from the bottom rather than expanding from the top, which is its default. The XAML markup appears as follows:

```
<c1:C1Expander Height="75" Width="140" Header="Expander Quick Start"
Background="Aqua" ExpandDirection="Up">
```

In this step, you customized the appearance and behavior of the C1Expander control. In the next step, you will add content to the control.

Step 3 of 3: Adding Content to the C1Expander Control

In the last step, you customized the appearance and behavior of the C1Expander control. In this step, you will add controls to the C1Expander control's content area and then run the project to observe the run-time features of the application you created in this quick start.

Complete the following steps:

1. Place your cursor between the `<c1:C1Expander>` and `</c1:C1Expander>` tags and press ENTER.
2. Navigate to the **Tools** panel and double-click the **StackPanel** icon to add a **StackPanel** control to the C1Expander control. The XAML markup will resemble the following:

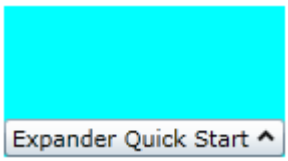
   ```
   <c1:C1Expander Height="200" Width="250" Background="Aqua"
   Header="Expander Quick Start" ExpandDirection="Up">
      <StackPanel></StackPanel>
   </c1:C1Expander>
   ```

   You are adding a **StackPanel** control to the C1Expander control's content area because you'll be adding more than one control to the content area in this quick start. The C1Expander control, as a content control, can only accept one child item at a time; however, you can get around this limitation by adding a panel-based control, which can accept multiple child items, to the C1Expander control.
3. Add `HorizontalAlignment="Center"` to the `<StackPanel>` tag so that all content added to the panel will be centered. The XAML markup will resemble the following:

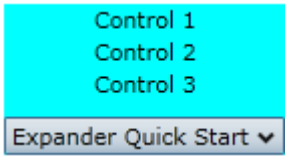
   ```
   <StackPanel HorizontalAlignment="Center">
   ```
4. Place your cursor between the `<StackPanel>` and `</StackPanel>` tags and press ENTER.
5. Navigate to the **Tools** panel and double-click the **TextBlock** icon to add the control to the **StackPanel** control. Repeat this twice so that a total of three **TextBlock** controls are added as the **StackPanel**'s content. The XAML markup will resemble the following:

   ```
   <StackPanel HorizontalAlignment="Center" >
       <TextBlock></TextBlock>
       <TextBlock></TextBlock>
       <TextBlock></TextBlock>
   </StackPanel>
   ```
6. Add `Text="Control 1"` to the first `<TextBlock>` tag, `Text="Control 2"` to the second `<TextBlock>` tag, and `Text="Control 3"` to the third `<TextBlock>` tag so that the XAML markup resembles the following:

   ```
    <TextBlock Text="Control 1"></TextBlock>
    <TextBlock Text="Control 2"></TextBlock>
    <TextBlock Text="Control 3"></TextBlock>
   ```
7. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time. Observe that the expander content isn't visible.



Click the C1Expander control's header to expand the content. Observe that the three **TextBox** controls that you added to the content area appear.



Congratulations! You have successfully completed the **Expander for Silverlight** quick start. In this quick start, you've created and customized an **Expander for Silverlight** application, added content to the control content area, and observed several of the control's run-time features.

Expander XAML Quick Reference

This topic is dedicated to providing a quick overview of the XAML used to create a C1Expander control. For more information, see the Expander for Silverlight Task-Based Help (page 196) section.

The following illustrates how to create an expander with **TextBlock** content:

```
<c1:C1Expander Height="122" HorizontalAlignment="Left"
Name="c1Expander1" VerticalAlignment="Top" Width="179">
   <TextBlock Height="23" HorizontalAlignment="Left" Name="textBlock1"
Text="TextBlock" VerticalAlignment="Top">
       Hello World!
   </TextBlock>
</c1:C1Expander>
```

Working with the C1Expander Control

The C1Expander control is a header content control that provides an expandable and collapsible pane for storing text, images, and controls. When you add the C1Expander control to a project, it exists as a header with a blank content area. By default, the control's interface looks similar to the following image:

After the C1Expander control is added to your project, you can add a header and content to the control. You can also modify behaviors, such as the expandability and the direction, of the control. The following topics provide an overview of the C1Expander control's elements and features.

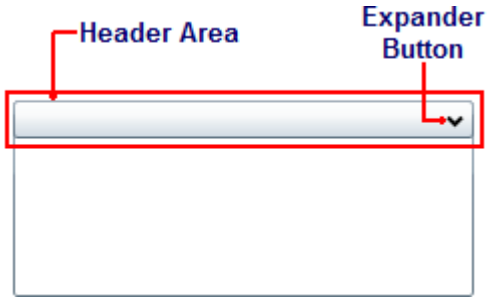
C1Expander Elements

This section provides a visual and descriptive overview of the elements that comprise the C1Expander control. The control is comprised of two elements – the header and the content area – that combine to make the complete C1Expander control.

Expander Header

By default, the header element of the C1Expander control appears at the top of the control and the expander button appears on the right side of the header. When the C1Expander control is first placed on the page, the header element contains no text.

The following image labels the header area of the **C1Expander** control.



To add text to the header element, simply set the **Header** property to a string. Once the text is added, you can style it using several font properties (see Text Properties (page 191))You can also add Silverlight controls to the header. For task-based help about adding content to the header, see Adding Content to the Header Element (page 196).

The placement of the header element and expander button will change depending on the expand direction of the control. For more information on expand directions, see the Expand Direction (page 189) topic.

Attribute Syntax versus Property Element Syntax

When you want to add something simple to the C1Expander header, such as an unformatted string, you can simply use the common XML attributes in your XAML markup, such as in the following:

```
<c1:C1Expander Header="Hello World"/>
```

However, there may be times where you want to add more complex elements, such as grids or panels, to the content area. In this case you would use property element syntax, such as in the following:

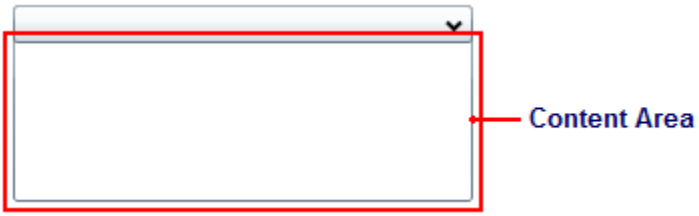
```
<c1:C1Expander ExpandDirection="Down" Width="150" Height="55"
Name="C1Expander1">
            <c1:C1Expander.Header>

      <Grid HorizontalAlignment="Stretch">
         <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="*" />
         </Grid.ColumnDefinitions>
```

```
            <TextBlock Text="C1Expander Header" />
        </Grid>
            </c1:C1Expander.Header>
</c1:C1Expander>
```

Expander Content Area

The C1Expander control's content area initially consists of an empty space. In the content area, you can add grids, text, images, and arbitrary controls. When working in Blend, elements in the content area of the control can be added and moved on the control through a simple drag-and-drop operation.

The following image labels the content area of the C1Expander control.



You can add text to the content area by setting the C1Expander control's **Content** property or by adding a **TextBox** element to the content area. Adding Silverlight elements to the content area at run time is simple: You can use either simple drag-and-drop operations or XAML to add elements. If you'd prefer to add a control at run time, you can use C# or Visual Basic code.

Content controls like C1Expander can only accept one child element at a time. However, you can circumvent this issue by adding a panel-based control as the C1Expander control's child element. Panel-based controls, such as a **StackPanel** control, are able to hold multiple elements. The panel-based control meets the one control limitation of the C1Expander control, but its ability to hold multiple elements will allow you to show several controls in the content area.

For task-based help about adding content to the content area, see Adding Content to the Content Area (page 198).

Attribute Syntax versus Property Element Syntax

When you want to add something simple to the C1Expander content area, such as an unformatted string or a single control, you can simply use the common XML attributes in your XAML markup, such as in the following:

```
<c1:C1Expander Content="Hello World"/>
```

However, there may be times where you want to add more complex elements, such as grids or panels, to the content area. In this case you can use property element syntax, such as in the following:

```
<c1:C1Expander ExpandDirection="Down" Width="150" Height="55"
Name="C1Expander1">
            <c1:C1Expander.Content>
                <StackPanel>
                    <TextBlock Text="Hello"/>
                    <TextBlock Text="World"/>
                </StackPanel>
            </c1:C1Expander.Content>
</c1:C1Expander>
```

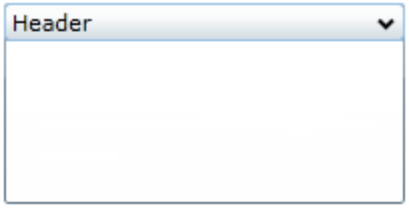
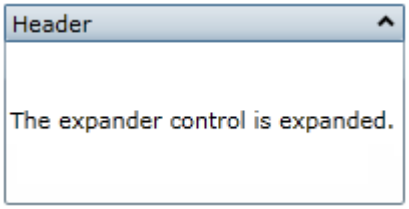
Expanding and Collapsing C1Expander

There are several options for customizing the way that the C1Expander control expands and collapses. The following sections describe how to set the initial expand state, set the direction to which the control expands, and how to prevent a C1Expander control from expanding.

Initial Expand State

By default, the IsExpanded property is set to **False**, which means that the control appears in its collapsed state when the page is loaded. If you want the control to be expanded upon page load, you can set the IsExpanded property to **True** .

The following table illustrates the difference between the two expand states.

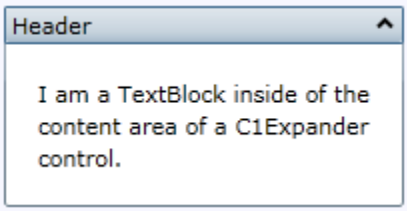
IsExpanded	Result
IsExpanded=False	
IsExpanded=True	

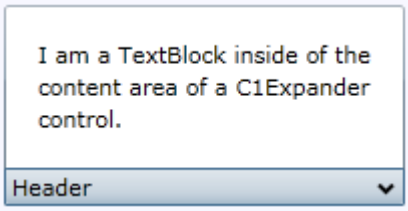
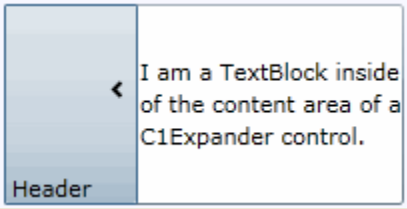
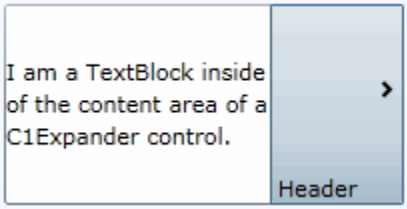
You can set the expand states in Blend, XAML, or code.

Expand Direction

The C1Expander control includes the option to specify the expand direction using the ExpandDirection property. In addition to setting how the direction the control expands, changing the ExpandDirection also changes the header's orientation to the content area of the control. By default the ExpandDirection property is set to **Down** and the control expands from top to bottom.

The following table illustrates each ExpandDirection setting.

ExpandDirection	Result
Down	

Up	
Right	
Left	

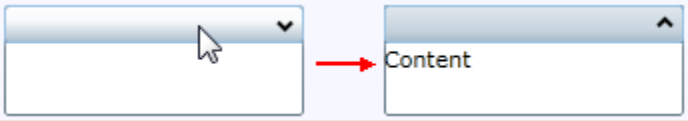
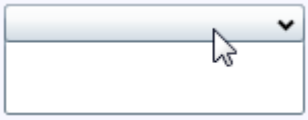
You can set the collapsing and expanding direction in Blend, XAML, or code. For task-based help on changing the expand direction, see Changing the Expand Direction (page 37).

Expandability

By default, the C1Expander control's IsExpandable property is set to **True**, meaning that the content can be expanded by clicking on the header bar. In some instances, such as when you'd like to prevent expansion until a specific event, you may not want to allow for expansion of the control; in this case, you would set the IsExpandable property to **False**.

Once the IsExpandable property is set to **False**, users can no longer interact with the control. If the user hovers over the control when the IsExpandable property is set to **False**, no mouse-over effect will appear; if the user clicks on the header when the IsExpandable property is set to **False**, the control will remain collapsed.

The table below illustrates the behavioral effect of each IsExpandable property setting.

IsExpandable	Result
True	
False	

Expander for Silverlight Layout and Appearance

The following topics detail how to customize the C1Expander control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the

appearance of the grid and take advantage of Silverlight's XAML-based styling. You can also use templates to format and lay out the control and to customize the control's actions.

C1Expander ClearStyle Properties

Expander for Silverlight supports ComponentOne's new ClearStyle technology that allows you to easily change control colors without having to change control templates. By just setting a few color properties you can quickly style the entire grid.

The following table outlines the brush properties of the **C1Expander** control:

Brushes	Description
Background	Gets or sets the brush of the control's background.
ButtonForeground	Gets or sets the brush of the button's foreground.
ExpandedBackground	Gets or sets the brush of the header when the control is expanded.
FocusBrush	Gets or sets the brush of the control when the control is focused.
MouseOverBrush	Gets or sets the System.Windows.Media.Brush used to highlight the buttons when the mouse is hovered over them.
PressedBrush	Gets or sets the System.Windows.Media.Brush used to highlight the buttons when they are clicked on.

You can completely change the appearance of the **C1Expander** control by setting a few properties, such as the **HeaderBackground** property, which sets the background color of the expander's header. For example, if you set the **HeaderBackground** property to "#FFE40005", the **C1Expander** control would appear similar to the following in its unexpanded state:

It's that simple with ComponentOne's ClearStyle technology. For more information on ClearStyle, see the ComponentOne ClearStyle Technology topic.

Expander for Silverlight Appearance Properties

ComponentOne Expander for Silverlight includes several properties that allow you to customize the appearance of the control. You can change the appearance of the text displayed in the control and customize graphic elements of the control. The following topics describe some of these appearance properties.

Text Properties

The following properties let you customize the appearance of text in the C1Expander control.

Property	Description
FontFamily	Gets or sets the font family of the control. This is a dependency property.
FontSize	Gets or sets the font size. This is a dependency property.
FontStretch	Gets or sets the degree to which a font is condensed or expanded on the screen. This is

	a dependency property.
FontStyle	Gets or sets the font style. This is a dependency property.
FontWeight	Gets or sets the weight or thickness of the specified font. This is a dependency property.
Header	Gets or sets the header of an expander control.
HeaderFontFamily	Gets or sets the font family of the header.
HeaderFontStretch	Gets or sets the font stretch of the header.
HeaderFontStyle	Gets or sets the font style of the header.
HeaderFontWeight	Gets or sets the font weight of the header.

Content Positioning Properties

The following properties let you customize the position of header and content area content in the C1Expander control.

Property	Description
HeaderPadding	Gets or sets the padding of the header.
HeaderHorizontalContentAlignment	HorizontalContentAlignment of the header.
HeaderVerticalContentAlignment	Gets or sets the vertical content alignment of the header.
HorizontalContentAlignment	Gets or sets the horizontal alignment of the control's content. This is a dependency property.
VerticalContentAlignment	Gets or sets the vertical alignment of the control's content. This is a dependency property.

Color Properties

The following properties let you customize the colors used in the control itself.

Property	Description
Background	Gets or sets a brush that describes the background of a control. This is a dependency property.
Foreground	Gets or sets a brush that describes the foreground color. This is a dependency property.
HeaderBackground	Gets or sets the background brush of the

	header.
HeaderForeground	Gets or sets the foreground brush of the header.

Border Properties

The following properties let you customize the control's border.

Property	Description
BorderBrush	Gets or sets a brush that describes the border background of a control. This is a dependency property.
BorderThickness	Gets or sets the border thickness of a control. This is a dependency property.

Size PropertiesThe following properties let you customize the size of the **C1Expander** control.

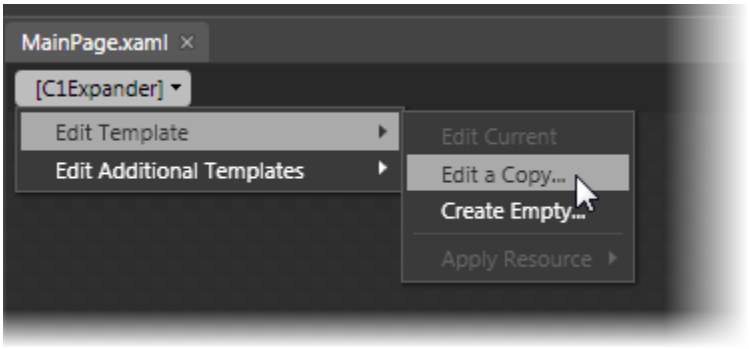
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
MinHeight	Gets or sets the minimum height constraint of the element. This is a dependency property.
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

Templates

One of the main advantages to using a Silverlight control is that controls are "lookless" with a fully customizable user interface. Just as you design your own user interface (UI), or look and feel, for Silverlight applications, you can provide your own UI for data managed by **ComponentOne Expander for Silverlight**. Extensible Application Markup Language (XAML; pronounced "Zammel"), an XML-based declarative language, offers a simple approach to designing your UI without having to write code.

Accessing Templates

You can access templates in Microsoft Expression Blend by selecting the C1Expander control and, in the menu, selecting **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or select **Create Empty** to create a new blank template.



Note: If you create a new template through the menu, the template will automatically be linked to that template's property. If you manually create a template in XAML you will have to link the appropriate template property to the template you've created.

Note that you can use the Template property to customize the template.

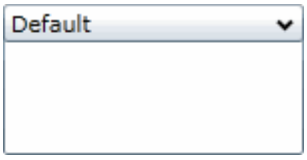
Additional Templates

In addition templates, the C1Expander control includes a few additional templates. These additional templates can also be accessed in Microsoft Expression Blend – in Blend select the C1Expander control and, in the menu, select **Edit Additional Templates**. Choose a template, and select **Create Empty.**

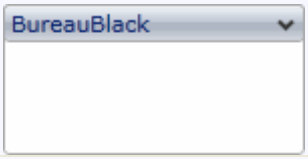
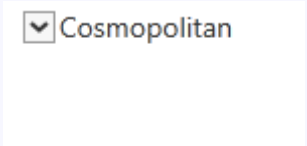
Expander Theming

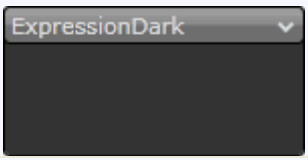
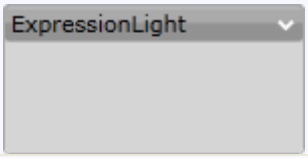
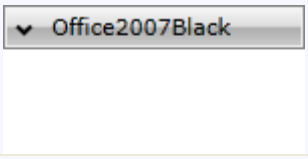
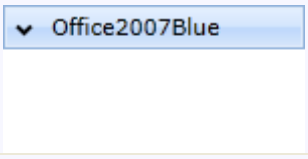
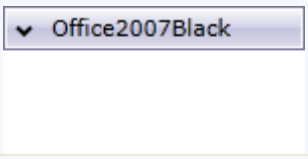
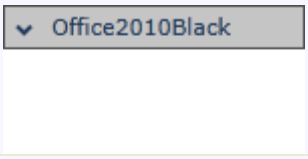
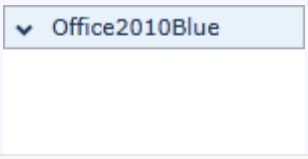
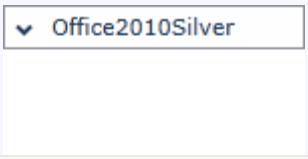
Silverlight themes are a collection of image settings that define the look of a control or controls. The benefit of using themes is that you can apply the theme across several controls in the application, thus providing consistency without having to repeat styling tasks.

When you add the C1Expander control your project, it appears with the default blue theme:



You can also theme the C1Expander control with one of our twelve included Silverlight themes: BureauBlack, ExpressionDark, ExpressionLight, Office2007Black, Office2007Blue, Office2007Silver, Office2010Black, Office2010Blue, Office2010Silver, RainierOrange, ShinyBlue, and WhistlerBlue. The table below provides a sample of each theme.

Full Theme Name	Appearance
C1ThemeBureauBlack	
C1ThemeCosmopolitan	

C1ThemeExpressionDark	
C1ThemeExpressionLight	
C1ThemeOffice2007Black	
C1ThemeOffice2007Blue	
C1ThemeOffice2007Silver	
C1ThemeOffice2010Black	
C1ThemeOffice2010Blue	
C1ThemeOffice2010Silver	
C1ThemeRainierOrange	
C1ThemeShinyBlue	

C1ThemeWhistlerBlue	

You can add any of these themes to a C1Expander control by declaring the theme around the control in markup. For task-based help about adding themes to the C1Expander control , see the Using C1Expander Themes (page 207).

Expander for Silverlight Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1Expander control in general. If you are unfamiliar with the **ComponentOne Expander for Silverlight** product, please see the **Expander for Silverlight** Quick Start first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne Expander for Silverlight** product.

Each task-based help topic also assumes that you have created a new Silverlight project.

Adding Content to the Header Element

You can easily add both simple text and Silverlight controls to the C1Expander control's header. The topics in this section will provide systematic instructions about adding text content and controls to the header.

For more information on the header element, you can also visit the Expander Header (page 187) topic.

Adding Text to the Header

By default, the C1Expander control's header is empty. You can add text to the control's header by setting the **Header** property to a string in Blend, in XAML, or in code.

At Design Time in Blend

To set the **Header** property in Blend, complete the following steps:

1. Click the C1Expander control once to select it.
2. Under the **Properties** tab, set the **Header** property to a string (for example, "Hello World").

In XAML

To set the **Header** property in XAML, add `Header="Hello World"` to the `<c1:C1Expander>` tag so that it appears similar to the following:

```
<c1:C1Expander Header="Hello World" Width="150" Height="55">
```

In Code

To set the **Header** property in code, complete the following steps:

1. Add `x:Name="C1Expander1"` to the `<c1:C1Expander>` tag so that the control will have a unique identifier for you to call in code.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic
     ```
     C1Expander1.Header = "Hello World"
     ```
   - C#

```
C1Expander1.Header = "Hello World";
```

3. Run the program.

This Topic Illustrates the Following:

The header of the C1Expander control now reads "Hello World". The end result of this topic should resemble the following:

Adding a Control to the Header

The C1Expander control's header element is able to accept a Silverlight control. In this topic, you will add a **Button** control to the header in XAML and in code.

In XAML

To add a **Button** control to the header in XAML, place the following XAML markup between the `<c1:C1Expander>` and `</c1:C1Expander>` tags:

```
        <c1:C1Expander.Header>
    <Button Content="Button" Height="Auto" Width="50"/>
        </c1:C1Expander.Header>
```

In Code

To add a **Button** control to the header in code, complete the following steps:

1. Add `x:Name="C1Expander1"` to the `<c1:C1Expander>` tag so that the control will have a unique identifier for you to call in code.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

```
'Create the Button control
Dim NewButton As New Button()
NewButton.Content = "Button"
'Set the Button Control's Width and Height properties
NewButton.Width = 50
NewButton.Height = Double.NaN
'Add the Button to the header
C1Expander1.Header = (NewButton)
```

   - C#

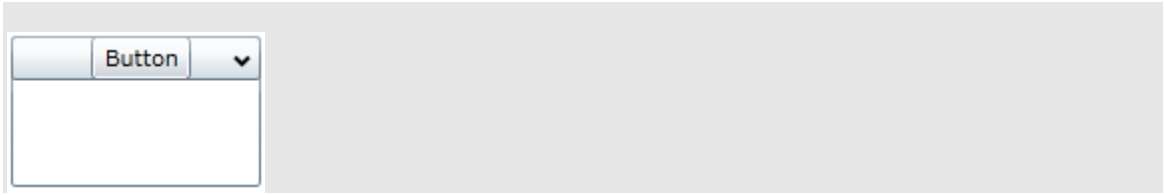
```
//Create the Button control
Button NewButton = new Button();
NewButton.Content = "Button";
//Set the Button Control's Width and Height properties
NewButton.Width = 50;
```

```
NewButton.Height = Double.NaN;
//Add the Button to the header
C1Expander1.Header = (NewButton);
```

3. Run the program.

This Topic Illustrates the Following:

As a result of this topic, a **Button** control will appear in the header. The final result will resemble the following image:



Adding Content to the Content Area

You can easily add both simple text and Silverlight controls to the C1Expander control's content area. The topics in this section will provide systematic instructions about how to add text content and controls to the header.

For more information on the header element, you can also visit the Expander Content Area (page 188) topic.

Adding Text to the Content Area

You can easily add a simple line of text to the content area of the C1Expander control by setting the **Content** property to a string in Blend, in XAML, or in code.

> **Note**: You can also add text to the content area by adding a **TextBox** control to the content area and then setting the **TextBox** control's **Text** property. To learn how to add a control to the content area, see Adding a Control to the Content Area (page 35).

At Design Time in Blend

To set the **Content** property in Blend, complete the following steps:

1. Click the C1Expander control once to select it.
2. Under the **Properties** tab, set the **Content** property to a string (for example, "Hello World").
3. Run the program and then expand the C1Expander control.

In XAML

To set the **Content** property in XAML, complete the following steps:

1. Add `Content="Hello World"` to the `<c1:C1Expander>` tag so that it appears similar to the following:

```
<c1:C1Expander Content="Hello World" Width="150" Height="55">
```

2. Run the program and then expand the C1Expander control.

In Code

To set the **Content** property in code, complete the following steps:

1. Add `x:Name="C1Expander1"` to the `<c1:C1Expander>` tag so that the control will have a unique identifier for you to call in code.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic
     ```
     C1Expander1.Content = "Hello World"
     ```
   - C#
     ```
     C1Expander1.Content = "Hello World";
     ```
3. Run the program and then expand the C1Expander control.

This Topic Illustrates the Following:

When the C1Expander control is expanded, it reads "Hello World". The end result of this topic should resemble the following:

Adding a Control to the Content Area

The C1Expander control will accept one child control in its content area. In this topic, you will learn how to add a Silverlight button control in Blend, in XAML, and in code.

At Design Time in Blend

To add a control to the content area, complete the following steps:

1. Navigate to the **Assets** tab and expand the **Controls** node.
2. Select **All** to open a list of all available Silverlight controls.
3. Select the **Button** icon and use a drag-and-drop operation to add it to the content area of the C1Expander control.
4. Under the **Objects and Timeline** tab, select **[Button]** so that the **Button** control's properties take focus in the **Properties** tab.
5. Next to the **Width** property, click the **Set to Auto** button. This will ensure that the height of the button control is the same height as the C1Expander control's content area.
6. Next to the **Height** property, click the **Set to Auto** button. This will ensure that the height of the button control is the same height as the C1Expander control's content area.
7. Run the program and then expand the C1Expander control.

In XAML

To add a button control to the C1Expander control's content area in XAML, complete the following steps:

1. Place the following markup between the `<c1:C1Expander>` and `</c1:C1Expander>` tags:

   ```
   <Button Content="Button" Height="Auto" Width="Auto"/>
   ```

2. Run the program and then expand the C1Expander control.

In Code

To add a button control to the C1Expander control's content area in code, complete the following:

1. Add `x:Name="C1Expander1"` to the `<c1:C1Expander>` tag so that the control will have a unique identifier for you to call in code.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

```
'Create the Button control
Dim NewButton As New Button()
NewButton.Content = "Button"
'Set the Button Control's Width and Height properties
NewButton.Width = Double.NaN
NewButton.Height = Double.NaN
'Add the Button to the content area
C1Expander1.Content = (NewButton)
```

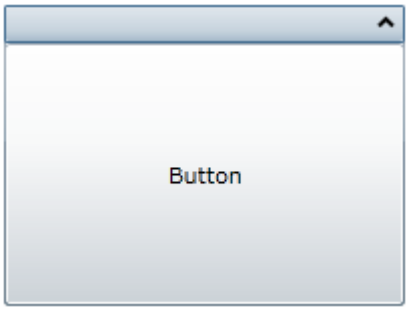
   - C#

```
//Create the Button control
Button NewButton = new Button();
NewButton.Content = "Button";
//Set the Button Control's Width and Height properties
NewButton.Width = double.NaN;
NewButton.Height = double.NaN;
//Add the Button to the content area
C1Expander1.Content = (NewButton);
```

3. Run the program and then expand the C1Expander control.

This Topic Illustrates the Following:

When the C1Expander control is expanded, the button control appears in its content area and resembles the following image:



Adding Multiple Controls to the Content Area

You cannot set the **Content** property to more than one control at a time. However, you can circumvent this issue by adding a panel-based control that can accept more than one control, such as a **StackPanel** control, to the

content area of the C1Expander control. When you add multiple controls to the panel-based control, each one will appear within the C1Expander control's content area.

At Design Time in Blend

To add a control to the content area, complete the following steps:

1. Navigate to the **Assets** tab and expand the **Controls** node.
2. Select **All** to open a list of all available Silverlight controls.
3. Select the **StackPanel** icon and use a drag-and-drop operation to add it to the content area of the C1Expander control.
4. Under the **Objects and Timeline** tab, select **StackPanel**.
5. Under the **Assets** tab, double-click the **TextBlock** icon to add a **TextBlock** control to the **StackPanel**. Repeat this step twice to add a total of three **TextBlock** controls to the **StackPanel**.
6. Under the **Objects and Timeline** tab, select the first TextBlock control to reveal its properties in the **Properties** tab and then set its **Text** property to "1st TextBlock".
7. Under the **Objects and Timeline** tab, select the secondTextBlock control to reveal its properties in the **Properties** tab and then set its **Text** property to "2nd TextBlock".
8. Under the **Objects and Timeline** tab, select the third **TextBlock** control to reveal its properties in the **Properties** tab and then set its **Text** property to "1st TextBlock"
9. Run the program.
10. Expand the C1Expander control and observe that each of the three **TextBlock** controls appear in the content area.

In XAML

To add multiple controls to the content area, complete these steps:

1. Place the following XAML markup between the `<c1:C1Expander>` and `</c1:C1Expander>` tags:

```
<c1:C1Expander.Content>
    <StackPanel>
        <TextBlock Text="1st TextBlock"/>
        <TextBlock Text="2nd TextBlock"/>
        <TextBlock Text="3rd TextBlock"/>
    </StackPanel>
</c1:C1Expander.Content>
```

2. Run the program.
3. Expand the C1Expander control and observe that each of the three **TextBlock** controls appear in the content area.

In Code

To add multiple controls to the content area, complete these steps:

1. Enter Code view and add the following code beneath the **InitializeComponent()** method:

- Visual Basic

```
'Create a stack panel and add it to C1Expander
Dim StackPanel1 As New StackPanel()
c1Expander1.Content = (StackPanel1)
'Create three TextBlock control and set their Text properties
Dim TextBlock1 As New TextBlock()
```

```
Dim TextBlock2 As New TextBlock()
Dim TextBlock3 As New TextBlock()
TextBlock1.Text = "1st TextBlock"
TextBlock2.Text = "2nd TextBlock"
TextBlock3.Text = "3rd TextBlock"
'Add TextBlock controls to StackPanel
StackPanel1.Children.Add(TextBlock1)
StackPanel1.Children.Add(TextBlock2)
StackPanel1.Children.Add(TextBlock3)
```

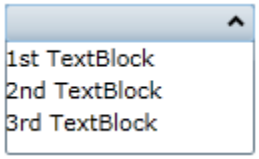
- C#

```
//Create a stack panel and add it to C1Expander
StackPanel StackPanel1 = new StackPanel();
c1Expander1.Content = (StackPanel1);
//Create three TextBlock control and set their Text properties
TextBlock TextBlock1 = new TextBlock();
TextBlock TextBlock2 = new TextBlock();
TextBlock TextBlock3 = new TextBlock();
TextBlock1.Text = "1st TextBlock";
TextBlock2.Text = "2nd TextBlock";
TextBlock3.Text = "3rd TextBlock";
//Add TextBlock controls to StackPanel
StackPanel1.Children.Add(TextBlock1);
StackPanel1.Children.Add(TextBlock2);
StackPanel1.Children.Add(TextBlock3);
```

2. Run the program.
3. Expand the C1Expander control and observe that each of the three **TextBlock** controls appear in the content area.

This Topic Illustrates the Following:

When the C1Expander control is expanded, three **TextBlock** controls will appear in the content area as follows:



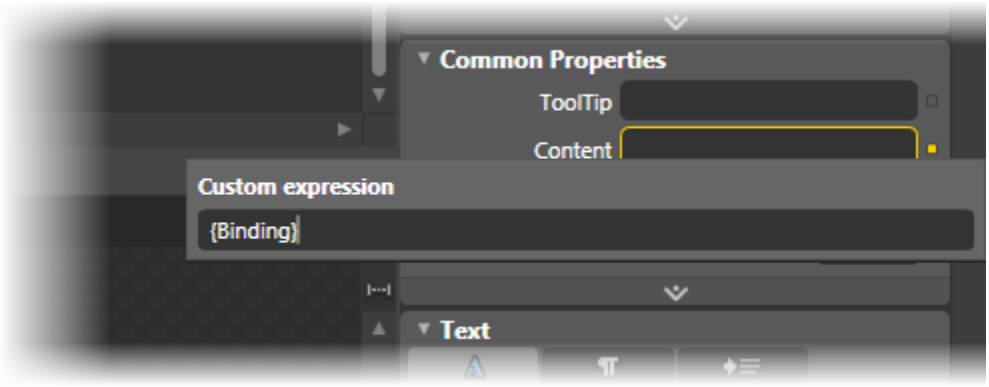
Binding Data to the Header and Content Panel Using Templates

In this topic, you will learn how to bind data to the C1Expander control's heading and content panel using the **ContentTemplate** template and **HeaderTemplate** template. This topic assumes that you are working in Microsoft Expression Blend.

Step 1: Add the C1Expander Control to the Project and Prepare it for Data Binding

Complete the following steps:

1. Add a C1Expander control to your Silverlight project.
2. Select the C1Expander control to expose its properties in the **Properties** tab and complete the following:
3. Set the **Name** property to "NameAgeHolder1".
4. Next to the **Content** property, click the **Advanced options** button and select **Custom Expression**. Set the **Custom expression** field to "{Binding}". This sets up the **Content** property to pass the DataContext directly to its template, which you will create in a later step.



5. Next to the **Header** property, click the **Advanced options** button and select **Custom Expression**. Set the **Custom expression** field to "{Binding}". This sets up the **Header** to pass the DataContext directly to its template, which you will create in a later step.

Step 2: Create Templates, Add a Control to Each Template, and Bind Each Control to a Data Source Property

Complete the following steps:

1. Create the first template, the **HeaderTemplate**, by completing the following steps:
   a. Click the **NameAgeHolder1** breadcrumb and select **Edit Additional Templates | Edit HeaderTemplate | Create Empty**.

      The **Create DataTemplate Resource** dialog box opens.

   b. In the **Name (Key)** field, enter "NameTemplate".

   c. Click **OK** to close the **Create DataTemplate Resource** dialog box to create the new template.

   d. Click the **Assets** tab and then, in the search field, enter "Label" to find the **Label** control.

   e. Double-click the **Label** icon to add the **Label** control to your template.

   f. Select the **Label** control to expose its properties in the **Properties** tab.

   g. Next to the **Content** property, click the **Advanced options** button and select **Custom Expression**. Set the **Custom expression** field to "{Binding Name}". This sets the **Label** control's **Content** property to the value of the **Name** property, which is a property you'll create in code later.

2. Create the second template, the **ContentTemplate**, by completing the following steps:
    a. Click the **NameAgeHolder1** breadcrumb and select **Edit Additional Templates | Edit Generated Content (ContentTemplate) | Create Empty**.

       The **Create DataTemplate Resource** dialog box opens.
    b. In the **Name (Key)** field, enter "AgeTemplate".
    c. Click **OK** to close the **Create DataTemplate Resource** dialog box to create the new template.
    d. Click the **Assets** tab and then, in the search field, enter "Label" to find the **Label** control.
    e. Double click the **Label** icon to add the **Label** control to your template.
    f. Select the **Label** control to expose its properties in the **Properties** tab.
    g. Next to the **Content** property, click the **Advanced options** button and select **Custom Expression**. Set the **Custom expression** field to "{Binding Age}". This sets the Label control's Content property to the value of the **Age** property, which is a property you'll create in code later.

Step 3: Create the Data Source

Complete the following steps:

1. Open the MainPage.xaml code page (this will be either MainPage.xaml.cs or MainPage.xaml.vb depending on which language you've chosen for your project).
2. Add the following class to your project, placing it beneath the namespace declaration:
    - Visual Basic

```
Public Class NameAndAge
   Public Sub New(name As String, age As Integer)
         Name = name
         Age = age
   End Sub
   Public Property Name() As String
         Get
         End Get
         Set
         End Set
   End Property
   Public Property Age() As Integer
         Get
         End Get
         Set
         End Set
   End Property
End Class
```

    - C#

```
public class NameAndAge
    {
```

```
        public NameAndAge(string name, int age)
        {
            Name = name;
            Age = age;
        }

        public string Name { get; set;  }
        public int Age { get; set;  }
    }
```

This class creates a class with two properties: a string property named **Name** and a numeric property named **Age**.

3. Add the following code beneath the **InitializeComponent()** method to set the **Name** property and the **Age** property:

   - Visual Basic

     ```
     NameAgeHolder1.DataContext = New NameAndAge("Gaius Baltar", 40)
     ```

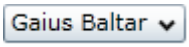
   - C#

     ```
     NameAgeHolder1.DataContext = new NameAndAge("Gaius Baltar", 40);
     ```

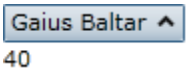
Step 4: Run the Project and Observe the Results

Complete the following steps:

1. Press F5 to run the project and observe that the value of the **Name** property appears on the control's header bar.



2. Click the header bar to expand the control and observe that the value of the **Age** property appears in the content panel:



Changing the Expand Direction

By default, the C1Expander control expands from top-to-bottom because its ExpandDirection property is set to **Down**. You can easily change the expand direction by setting the ExpandDirection property to **Up**, **Right**, or **Left** in Blend, in XAML, or in code.

At Design Time in Blend

To set the ExpandDirection property in Blend, complete the following steps:

1. Click the C1Expander control once to select it.
2. Under the **Properties** tab, click the ExpandDirection drop-down arrow and select one of the options from the list. For this example, select **Right**.

In XAML

To set the ExpandDirection property to Right in XAML, add `ExpandDirection="Right"` to the `<c1:C1Expander>` tag so that it appears similar to the following:

```
<c1:C1Expander ExpandDirection="Right" Width="150" Height="55">
```

In Code

To set the ExpandDirection property in code, complete the following steps:

1. Add `x:Name="C1Expander1"` to the `<c1:C1Expander>` tag so that the control will have a unique identifier for you to call in code.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

```
C1Expander1.ExpandDirection =
C1.Silverlight.Extended.ExpandDirection.RightC#
C1Expander1.ExpandDirection =
C1.Silverlight.Extended.ExpandDirection.Right;Run the program.
```

This Topic Illustrates the Following:

By following the instructions in this topic, you have learned how to set the ExpandDirection property. In this topic, you set the ExpandDirection property to **Right**, which will make the C1Expander control resemble the following:

Changing the Initial Expand State

By default, the C1Expander control's IsExpanded property is set to **False**, meaning that the control will appear in its collapsed state upon page load. If you'd prefer that the C1Expander control be expanded upon page load, you can set the IsExpanded property to **True** in Blend, in XAML, or in code.

At Design Time in Blend

To set the IsExpanded property to **True** in Blend, complete the following steps:

1. Click the C1Expander control once to select it.
2. Under the **Properties** tab, locate the IsExpanded check box and then select it.

In XAML

To set the IsExpanded property to **True** in XAML, add `IsExpanded="True"` to the `<c1:C1Expander>` tag so that it appears similar to the following:

```
<c1:C1Expander IsExpanded="True" Width="150" Height="55">
```

In Code

To set the IsExpanded property to **True** in code, complete the following steps:

1. Add `x:Name="C1Expander1"` to the `<c1:C1Expander>` tag so that the control will have a unique identifier for you to call in code.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

```
C1Expander1.IsExpanded = TrueC#
C1Expander1.IsExpanded = true;Run the program.
```

Preventing Expansion

You can prevent a C1Expander control from being expanded by setting the IsExpandable property to **False** in Blend, in XAML, or in code.

At Design Time in Blend

To set the IsExpandable property to **False** in Blend, complete the following steps:

1. Click the C1Expander control once to select it.
2. Under the **Properties** tab, locate the IsExpandable check box and then deselect it.

In XAML

To set the IsExpandable property to **False** in XAML, add `IsExpandable="False"` to the `<c1:C1Expander>` tag so that it appears similar to the following:

```
<c1:C1Expander IsExpandable="False" Width="150" Height="55">
```

In Code

To set the IsExpanded property to **True** in code, complete the following steps:

1. Add `x:Name="C1Expander1"` to the `<c1:C1Expander>` tag so that the control will have a unique identifier for you to call in code.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

```
C1Expander1.IsExpandable = FalseC#
C1Expander1.IsExpandable = false;Run the program.
```

Using C1Expander Themes

The C1Expander control comes equipped with a light blue default theme, but you can also apply six themes (see Expander Theming (page 194)) to the control. In this topic, you will change the C1Expander control's theme to **C1ThemeRainierOrange** in Blend and in Visual Studio.

In Blend

Complete the following steps:

1. Click the **Assets** tab.
2. In the search bar, enter "C1ThemeRainierOrange".

   The **C1ThemeRainierOrange** icon appears.
3. Double-click the **C1ThemeRainierOrange** icon to add it to your project.
4. In the search bar, enter "C1Expander" to search for the C1Expander control.
5. Double-click the C1Expander icon to add the C1Expander control to your project.
6. Under the **Objects and Timeline** tab, select **[C1Expander]** and use a drag-and-drop operation to place it under **[C1ThemeRainierOrange]**.
7. Run the project.

In Visual Studio

Complete the following steps:

1. Open the **.xaml** page in Visual Studio.
2. Place your cursor between the <Grid></Grid> tags.
3. In the **Tools** panel, double-click the **C1ThemeRainierOrange** icon to declare the theme. Its tags will appear as follows:

```
<my:C1ThemeRainierOrange></my:C1ThemeRainierOrange>
```

4. Place your cursor between the <my:C1ThemeRainierOrange> and </my:C1ThemeRainierOrange> tags.
5. In the **Tools** panel, double-click the C1Expander icon to add the control to the project. Its tags will appear as children of the <my:C1ThemeRainierOrange> tags, causing the markup to resemble the following:

```
<my:C1ThemeRainierOrange>
    <c1:C1Expander></c1:C1Expander>
</my:C1ThemeRainierOrange>
```

6. Run your project.

This Topic Illustrates the Following:

The following image depicts a C1Expander control with the **C1ThemeRainierOrang**e theme.

HtmlHost

Render HTML and arbitrary URI content from within Silverlight using **ComponentOne HtmlHost™ for Silverlight**. The HTML host control (**C1HtmlHost**) provides a frame that can host arbitrary HTML content, and display content from arbitrary URIs or HTML text.

Getting Started

Get started with the following topics:

- Key Features (page 209)

- Quick Start (page 209)

- Task-Based Help (page 217)

HtmlHost for Silverlight Key Features

ComponentOne HtmlHost for Silverlight includes several key features, such as:

- **Inside Browser Support**

  You can see HTML content inside your browser.

- **Display HTML Content**

  Display existing HTML content within the Silverlight plug-in using the browser itself to provide accurate rendering and interaction.

- **Load HTML from URI**

  HtmlHost for Silverlight can load HTML content from any URI available, not just the application server.

- **Load HTML from Text**

  Display HTML content contained in a string.

- **Access the HTML Content**

  HtmlHost for Silverlight fires an event when the content is fully loaded and allows the developer to access the content through Silverlight's browser object model.

- **Silverlight Layout System Support**

  HtmlHost for Silverlight arranges its content following Silverlight's layout for the control.

- **Silverlight Toolkit Themes Support**

  Add style to your UI with built-in support for the most popular Microsoft Silverlight Toolkit themes, including ExpressionDark, ExpressionLight, WhistlerBlue, RainerOrange, ShinyBlue, and BureauBlack.

HtmlHost for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **ComponentOne HtmlHost for Silverlight**. In this quick start you'll create a Silverlight application that allows you to view a Web site in a Silverlight application using the **C1HtmlHost** control. You'll create a new project in Visual Studio, add and customize controls, and view the run-time interactions possible with the **C1HtmlHost** control.

Step 1 of 3: Creating a Silverlight Application

In this step you'll create a Silverlight application in Visual Studio which will use **ComponentOne HtmlHost for Silverlight** to display a Web site. You'll create a new Silverlight project and add controls to your application.

To set up and add controls to your application, complete the following steps:

1. In Visual Studio, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Silverlight Application**. Enter a **Name** for your project and click **OK**. The **New Silverlight Application** dialog box will appear.
3. Click **OK** to accept default settings, close the **New Silverlight Application** dialog box, and create your project. The **MainPage.xaml** file should open.
4. In the XAML window of the project, place the cursor between the `<Grid>` and `</Grid>` tags and click once.
5. Navigate to the Visual Studio Toolbox and double-click the **StackPanel** icon to add the panel to the page.
6. Update the `<StackPanel/>` tag in the **MainPage.xaml** file so it appears like the following:

```
<StackPanel Name="StackPanel1" Margin="10"
Orientation="Vertical"></StackPanel>
```

   This markup adds a margin and sets the panel's orientation.

7. In the XAML window of the project, place the cursor between the `<StackPanel>` and `</StackPanel>` tags and click once – you will add controls to the **StackPanel**.
8. Navigate to the Visual Studio Toolbox and double-click the **C1HtmlHost** icon to add the control to the panel.
9. Update the `<c1:C1HtmlHost/>` tag in the **MainPage.xaml** file so it appears like the following:

```
<c1:C1HtmlHost Name="C1HtmlHost1" Margin="5"
SourceUri="http://www.componentone.com" />
```

   This markup names the control, adds a margin, and sets the SourceUri property which sets the Web site or page that will initially appear in the C1HtmlHost control when the application is loaded.

10. Add the following markup below the `<c1:C1HtmlHost/>` tag in XAML view:

```
<StackPanel Name="StackPanel2" Orientation="Horizontal">
    <TextBlock Height="23" HorizontalAlignment="Left" Margin="5"
Name="TextBlock1" Text="Source URI:" VerticalAlignment="Top" />
    <TextBox Height="23" Name="TextBox1" Width="200" Margin="5"
Text="http://www.componentone.com" />
    <Button Content="Set Source URI" Height="23" Name="Button1"
Margin="5" />
</StackPanel>
```

    The markup above adds a **StackPanel** that contains a **TextBlock**, **TextBox**, and **Button**. When the user enters a URL in the text box at run time and presses the button, the C1HtmlHost control will display the entered Web site. Your markup will now appear similar to the following:

```
<Grid x:Name="LayoutRoot" Background="White">
    <StackPanel Margin="10" Name="StackPanel1" Orientation="Vertical">
        <c1:C1HtmlHost Name="C1HtmlHost1" Margin="5"
SourceUri="http://www.componentone.com" c1:C1NagScreen.Nag="True" />
        <StackPanel Name="StackPanel2" Orientation="Horizontal">
            <TextBlock Height="23" HorizontalAlignment="Left" Margin="5"
Name="TextBlock1" Text="Source URI:" VerticalAlignment="Top" />
            <TextBox Height="23" Name="TextBox1" Width="200" Margin="5"
Text="http://www.componentone.com" />
            <Button Content="Set Source URI" Height="23" Name="Button1"
Margin="5" />
        </StackPanel>
    </StackPanel>
</Grid>
```

11. Navigate to the Solution Explorer, and expand the **YourProject.Web** node (where *YourProject* is the name of the project) and double-click the **YourProjectTestPage.aspx** file (where *YourProject* is the name of the project) to open it.
12. In the .aspx file scroll down to the `<div id="silverlightControlHost">` tag and add the following parameter in the list of parameters between the `<object></object>` tags:
    ```
    <param name="windowless" value="true" />
    ```
13. Save all your changes and return to the **MainPage.xaml** page.

What You've Accomplished

You've successfully created and set up a Silverlight application and added controls, including a C1HtmlHost control, to the page. In the next step you'll add code to add functionality to your application.

Step 2 of 3: Adding Code to the Application

In the last step you set up a Silverlight application, but if you run your application the button and text box currently do nothing. In this step you'll continue by adding code to add functionality to the application.

Complete the following steps:

1. Navigate to the Solution Explorer, right-click **MainPage.xaml** file, and select **View Code** to switch to Code view.
2. In Code view, add the following import statement to the top of the code page:
   - Visual Basic
     ```
     Imports C1.Silverlight.Extended
     ```
   - C#
     ```
     using C1.Silverlight.Extended;
     ```
3. Add the following event handler to the **MainPage.xaml.cs** file, below all the other methods in the **MainPage** class:
   - Visual Basic
     ```
     Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
     System.Windows.RoutedEventArgs) Handles Button1.Click
         Me.C1HtmlHost1.SourceUri = New Uri(String.Format(TextBox1.Text))
     End Sub
     ```
   - C#
     ```
     private void Button1_Click(object sender, RoutedEventArgs e)
     {
         Cc1HtmlHost1.SourceUri = new Uri(string.Format(TextBox1.Text));
     }
     ```

   This code handles the button's **Click** event and customizes the C1HtmlHost control.

What You've Accomplished

In this step you added code to add functionality to your application – now when a user enters a URL in the text box and clicks the button at run time, the C1HtmlHost control will display the selected Web site. In the next step you'll run your application and observe some of the run-time interactions possible with **ComponentOne HtmlHost for Silverlight**.

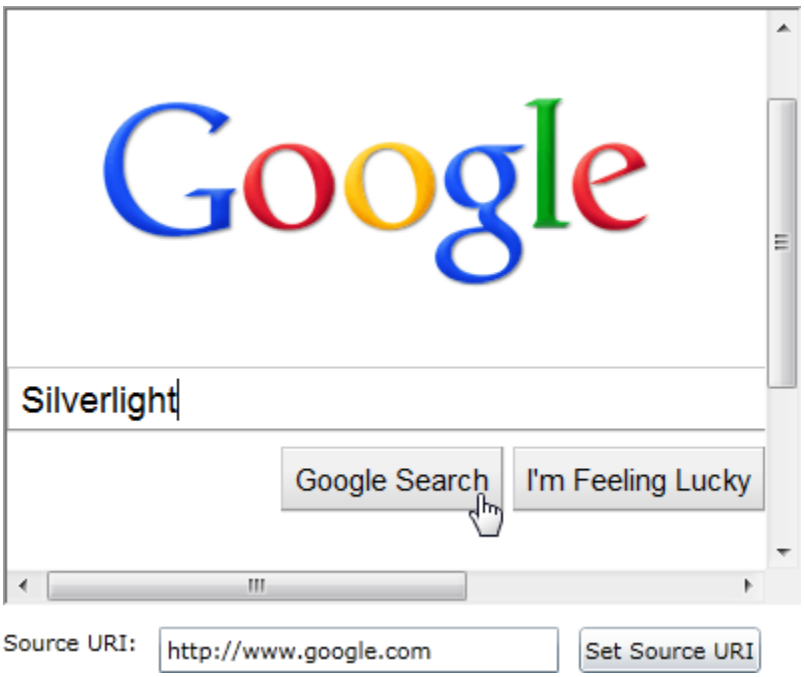
Step 3 of 3: Running the Application

Now that you've created a Silverlight application and, set up the application, and added code to add functionality to the application, the only thing left to do is run your application. To observe your application's run-time interactions, complete the following steps:

1. Choose **Debug | Start Debugging** from the menu to run your application. The application will appear similar to the following image:

   The application appears with the ComponentOne Web site loaded in the **C1HtmlHost** control.

2. Enter http://www.google.com in the text box and click the **Set Source URI** button. The **C1HtmlHost** control will display's Google's Web site.
3. Enter "Silverlight" in the text box and click the **Google Search** button:



A page will appear displaying search results.

4. Click a search result link. Notice that the control displays the page you choose:

5. Continue to click links and surf the Internet through the **C1HtmlControl** – notice that the page it displays functions as a page in a Web browser would.

What You've Accomplished

Congratulations, you've completed the **HtmlHost for Silverlight** quick start! You've created a simple application that uses **HtmlHost for Silverlight** to display a selected Web site.

To learn more about the features and functionality of **ComponentOne HtmlHost for Silverlight**, see the Working with HtmlHost for Silverlight (page 214) topic. For examples of specific customizations, see the HtmlHost for Silverlight Task-Based Help (page 217) topic.

Working with HtmlHost for Silverlight

ComponentOne HtmlHost™ for Silverlight provides a simple and reliable way to display a Website or HTML content within a Silverlight application. The following topics detail information useful in working with **HtmlHost for Silverlight**.

Showing HTML Content

The **C1HtmlHost** control behaves like an HTML iFrame tag, and allows you to show HTML content within your Silverlight pages. This is a reversal of the typical scenario where Silverlight plug-ins are islands in the HTML that makes up the page. The **C1HtmlHost** control allows you to create HTML islands in your Silverlight pages.

This concept is useful because in many cases your Silverlight applications may want to display existing HTML content within the Silverlight plug-in itself (as opposed to controlling elements in the host page's DOM, which Silverlight also allows).

ComponentOne HtmlHost for Silverlight works using an iFrame. The C1HtmlHost control adds an iFrame through JavaScript and locates it over the Silverlight plugin in the exact location where the C1HtmlHost control is located. Then it listens to changes in the layout and updates the iFrame continually.

Note that some pages that you may want to display may not be viewable in the C1HtmlHost control. For example, the page you wish to be display may be customized so that is not displayable with in an iFrame object. At its essence the C1HtmlHost control is uses an iFrame object located on top of the Silverlight plugin, so this behavior is expected.

Populating C1HtmlHost

The **C1HtmlHost** can be populated in two ways:

- Use the **SourceHtml** property to specify an HTML string that should be displayed in the control. This option is useful if your application builds or loads the HTML content.
- Use the **SourceUri** property to specify the URL that will be displayed in the control. This option is useful if your application needs to display content that is already available at a given URL.

The **C1HtmlHost** control has an important requirement. In order to display the HTML content within the Silverlight plug-in, the plug-in must have its **Windowless** property set to **True**. If you run the application now, you will see an error message that notes:

> **Note:** To use this control the Silverlight plug-in must have its **Windowless** parameter set to **True**.

The control cannot modify the properties of the plug-in, the page author must do that instead. To make the necessary change, open the page that creates the plug-in and add a line to set the **Windowless** property to **True**. For details see Windowless Mode (page 214).

Windowless Mode

ComponentOne HtmlHost for Silverlight requires that the Silverlight plug-in's **Windowless** parameter be set to **True**. In windowless mode, the Silverlight plug-in does not have its own rendering window. Instead, the plug-in content is displayed directly by the browser window. This enables Silverlight content to visually overlap and blend with HTML content if the plug-in and its content both specify background transparency.

To set the **Windowless** parameter, you would complete the following steps:

1. Create a Silverlight application that includes a **C1HtmlHost** control.
2. Navigate to the Solution Explorer, and expand the **YourProject.Web** node (where *YourProject* is the name of the project).
3. In the Solution Explorer, double-click the page in which the Silverlight plugin is declared. For example, double-click **YourProjectTestPage.aspx** or **YourProjectTestPage.html** file (where *YourProject* is the name of the project) to open the page.
4. In the page scroll down to the `<div id="silverlightControlHost">` tag and add the following parameter in the list of parameters between the `<object></object>` tags:
   ```
   <param name="windowless" value="true" />
   ```
5. Save your change and return to the **MainPage.xaml** page.

When the **Windowless** parameter is not set to **True**, you may receive a warning when running the application.

Frame Borders

The C1HtmlHost control is displayed within a frame by default. By default, borders are displayed around the C1HtmlHost control. For example, in the following image a frame appears in the top and left borders of the C1HtmlHost control:

You may want to hide this frame border – for example, if you want the control to blend more seamlessly with your application. When the FrameBorder property is set to **False**, a frame will not appear around the C1HtmlHost control. For example, in the image below the frame is not visible:

You'll notice that when the frame is not visible, the C1HtmlHost control blends into the background of the application more. For an example, see Hiding Frame Borders (page 219).

Basic Properties

ComponentOne HtmlHost for Silverlight includes several properties that allow you to set the functionality of the C1HtmlHost control. Some of the more important properties are listed below.

The following properties let you customize the C1HtmlHost control:

Property	Description
AllowTransparency	Gets or sets a value indicating whether transparency is allowed (only required for IE).
FrameBorder	Gets or sets whether the default browser frame border is displayed when using SourceUri.
HtmlElement	Gets the **HtmlElement** hosted by this C1HtmlHost control.
SourceHtml	Gets or set the HTML content as a string.
SourceUri	Gets or set a URI that will provide the content for the control.

Basic Events

ComponentOne HtmlHost for Silverlight includes several events that allow you to set interaction and customize the control. Some of the more important events are listed below.

The following events let you customize the C1HtmlHost control:

Event	Description
UriLoaded	Fired when the content from a URI has been loaded.

HtmlHost for Silverlight Appearance Properties

ComponentOne HtmlHost for Silverlight includes several properties that allow you to customize the appearance of the control. You can change the color, border, and height of the control. The following topics describe some of these appearance properties.

Color Properties

The following properties let you customize the colors used in the control itself:

Property	Description
Background	Gets or sets a brush that describes the background of a control. This is a dependency property.

Alignment Properties

The following properties let you customize the control's alignment:

Property	Description
HorizontalAlignment	Gets or sets the horizontal alignment characteristics applied to this element when it is composed within a parent element, such as a panel or items control. This is a dependency property.
VerticalAlignment	Gets or sets the vertical alignment characteristics

	applied to this element when it is composed within a parent element such as a panel or items control. This is a dependency property.

Size Properties

The following properties let you customize the size of the **C1HtmlHost** control:

Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
MinHeight	Gets or sets the minimum height constraint of the element. This is a dependency property.
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

HtmlHost Styles

ComponentOne HtmlHost for Silverlight's **C1HtmlHost** control provides several style properties that you can use to change the appearance of the control. Some of the included styles are described in the table below:

Style	Description
Style	Gets or sets the style used by this element when it is rendered. This is a dependency property.

HtmlHost for Silverlight Samples

ComponentOne HtmlHost for Silverlight includes C# samples. By default samples are installed in the **Documents** or **My Documents** folder in the **ComponentOne Samples\Studio for Silverlight** folder.

The following sample is included:

Sample	Description
ControlExplorer	The **ControlExplorer** includes a page that demonstrates using the **C1HtmlHost** control to display an external Web site. This sample and is installed by default in the **ControlExplorer** folder in the samples directory.

HtmlHost for Silverlight Task-Based Help

The following task-based help topics assume that you are familiar with Visual Studio and Expression Blend and know how to use the C1HtmlHost control in general. If you are unfamiliar with the **ComponentOne HtmlHost for Silverlight** product, please see the HtmlHost for Silverlight Quick Start (page 209) first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne HtmlHost for Silverlight** product. Most task-based help topics also assume that you have created a new Silverlight project and added the C1HtmlHost control to the application.

Displaying an External Web Site

Displaying an external Web site in **ComponentOne HtmlHost for Silverlight** is as simple as setting a single property. The SourceUri property determines the URI of the site displayed in the control. You can set the SourceUri property to any Web site – even an external site – to display.

At Design Time

To set the SourceUri property in Expression Blend, complete the following steps:

1. Click the C1HtmlHost control once to select it.
2. Navigate to the Properties window and locate the **SourceUri** item.
3. Click in the text box next to the **SourceUri** item and enter "http://www.componentone.com".

In XAML

For example, to set the SourceUri property add `SourceUri=""` to the `<c1:C1HtmlHost>` tag so that it appears similar to the following:

```
<c1:C1HtmlHost Name="c1HtmlHost1" SourceUri="http://www.componentone.com"
/>
```

In Code

For example, to set the SourceUri property, add the following code to your project:

- Visual Basic

```
Me.C1HtmlHost1.SourceUri = "http://www.componentone.com"
```

- C#

```
this.C1HtmlHost1.SourceUri = "http://www.componentone.com";
```

What You've Accomplished

You've set the SourceUri property and customized the C1HtmlHost control so that it displays the ComponentOne Web site at run time. Run the application, and observe the ComponentOne Web site appears in the C1HtmlHost control at run time.

Displaying HTML Markup

HtmlHost for Silverlight supports displaying HTML markup. You set the SourceHtml property to HTML markup that will be translated at run time; for example see the steps below.

At Design Time

To set the SourceHtml property, complete the following steps:

1. Click the C1HtmlHost control once to select it.
2. Navigate to the Properties window and locate the **SourceHtml** item.
3. Enter text in the text box next to the **SourceHtml** item; for example, enter "This is HTML hosted in a Browser!".

In XAML

For example, to set the SourceHtml property add `SourceHtml` to the `<c1:C1HtmlHost>` tag so that it appears similar to the following:

```
<c1:C1HtmlHost Name="c1HtmlHost1" SourceHtml="This is
&amp;lt;b&amp;gt;HTML&amp;lt;/b&amp;gt; hosted in a Browser!"/>
```

In Code

For example, to set the SourceHtml property, add the following code to your project:

- Visual Basic

```
Me.C1HtmlHost1.SourceHtml="This is &lt;b&gt;HTML&lt;/b&gt; hosted in a
Browser!"
```

- C#

```
this.C1HtmlHost1.SourceHtml="This is &lt;b&gt;HTML&lt;/b&gt; hosted in a
Browser!";
```

What You've Accomplished

The C1HtmlHost control will now display HTML text at run time. Note that in the markup above the "<" and ">" were used for the "<" and ">" symbols. So, for example a tag to make the text bold, which would normally appear like "<b>" is written as "<b>".

Hiding Frame Borders

By default, borders are displayed around the C1HtmlHost control. You may want to hide this frame border – for example, if you want the control to blend more seamlessly with your application. Hiding frame borders in **ComponentOne HtmlHost for Silverlight** is as simple as setting a single property. The FrameBorder property determines if a frame is displayed around the control.

At Design Time

To set the FrameBorder property, complete the following steps:

1. Click the C1HtmlHost control once to select it.
2. Navigate to the Properties window and locate the **FrameBorder** item.
3. Clear the check box next to the **FrameBorder** item.

In XAML

For example, to set the FrameBorder property add `FrameBorder="False"` to the `<c1:C1HtmlHost>` tag so that it appears similar to the following:

```
<c1:C1HtmlHost Name="c1HtmlHost1" FrameBorder="False" />
```

In Code

For example, to set the FrameBorder property, add the following code to your project:

- Visual Basic

```
Me.C1HtmlHost1.FrameBorder = False
```

- C#

```
this.C1HtmlHost1.FrameBorder = false;
```

What You've Accomplished

You've set the FrameBorder property so a frame does not appear around the control at run time. Run the application, and observe the C1HtmlHost control appears without a surrounding frame.

Accessing a Silverlight Function from an ASPX Page Within C1HtmlHost

C1HtmlHost gives you the ability to host an ASPX page within the control. This allows you to call a Silverlight function through the Web page. This topic will take you through adding the **C1HtmlHost** control, an ASPX page, and a Silverlight function to your project using both XAML and Code Views.

Complete these steps:

1. Add a new WebForm to your web project and call it WebForm1.aspx. Locate the `<div>` tags on the page and add the following markup to create the input button that will call the Silverlight function:

```
<input type="button" id="btn1" onclick="CallSilverlightFunction()"
value="Call Silverlight Function" />
```

2. Locate the `<head>` tags on the WebForm1.aspx page. Insert the following script between them to call the Silverlight function:

```
<script type="text/javascript">


        function CallSilverlightFunction(sender, args) {
            var parent = window.parent;
            var sl =
parent.document.getElementById("silverlightControl");
            sl.Content.slObject.SilverlightFunction();
        }
    </script>
```

3. Switch to the XAML view of your project and add the following markup to create the **C1HtmlHost** control:

```
<c1:C1HtmlHost Height="100" HorizontalAlignment="Left"
Margin="92,64,0,0" Name="c1HtmlHost1" VerticalAlignment="Top"
Width="200" />
```

4. Now navigate to the Code View by right-clicking on the page and selecting **View Code** from the menu. Add the following statements to the top of the page:

   - Visual Basic

     ```
     Imports System.Windows.Browser
     Imports C1.Silverlight.Extended
     ```

   - C#

     ```
     using System.Windows.Browser;
     using C1.Silverlight.Extended;
     ```

5. Add the following code below the **InitializeComponent()** method. This will set the WebForm you added as the source for the **C1HtmlHost** control.

   - Visual Basic

     ```
     c1HtmlHost1.SourceUri = New Uri("WebForm1.aspx", UriKind.Relative)
     ```

   - C#

     ```
     c1HtmlHost1.SourceUri = new Uri("WebForm1.aspx", UriKind.Relative);
     ```

6. Next, we will declare the Silverlight Function we wish to call as a scriptable member to make it accessible outside Silverlight.

   - Visual Basic

     ```
     <ScriptableMember()>  _
         Public Sub SilverlightFunction()
             MessageBox.Show("Message from Silverlight Function")
         End Sub
     ```

   - C#

```
[ScriptableMember]
        public void SilverlightFunction()
        {
            MessageBox.Show("Message from Silverlight Function");
```

7. From the Solution Explorer, open your App.xaml.cs file to add the following code in place of the Application Startup event:

   - Visual Studio

   ```
   Private Sub Application_Startup(ByVal sender As Object, ByVal e As
   StartupEventArgs)
            Dim newMainPage As MainPage = New MainPage
            Me.RootVisual = newMainPage
            HtmlPage.RegisterScriptableObject("slObject", newMainPage)
       End Sub
   ```

   - C#

   ```
   private void Application_Startup(object sender, StartupEventArgs e)
        {
            MainPage newMainPage = new MainPage();
            this.RootVisual = newMainPage;
            HtmlPage.RegisterScriptableObject("slObject", newMainPage);
        }
   ```

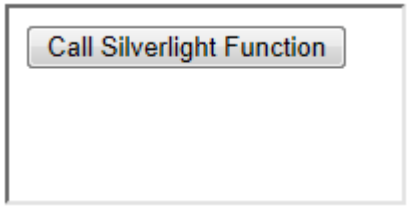
8. Locate the TestPage.aspx in the Solution Explorer in the web portion of your project. It will be labeled *YourProjectName*TestPage.aspx. Locate the `<object>` tags on the page. Insert `id="silverlightControl"` in the tag so that the markup resembles the following:

   ```
   <object id= "silverlightControl" data="data:application/x-silverlight-
   2," type="application/x-silverlight-2" width="100%" height="100%">
   ```

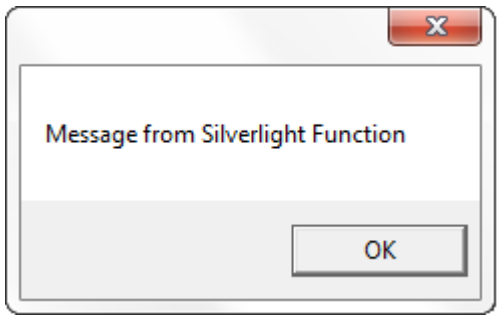
9. Check your `<object>` parameters to ensure that the `windowless` parameter is set to true as in the following markup:

   ```
   <param name="windowless" value="true" />
   ```

10. Press F5 to run your application. The application should resemble the following image:

    When you press the button, the following dialog box will appear:



What You've Accomplished

You've added a **C1HtmlHost** control to the page, hosted an ASPX page within the **C1HtmlHost** control, and called a Silverlight function from within the control.

PropertyGrid

ComponentOne PropertyGrid™ for Silverlight is a Silverlight version of the popular **PropertyGrid** control that ships as part of the .NET **WinForms** platform. It allows you to easily edit any class and includes more than 10 built-in editors. Using **ComponentOne PropertyGrid™ for Silverlight**, users can browse and edit properties on any .NET object.

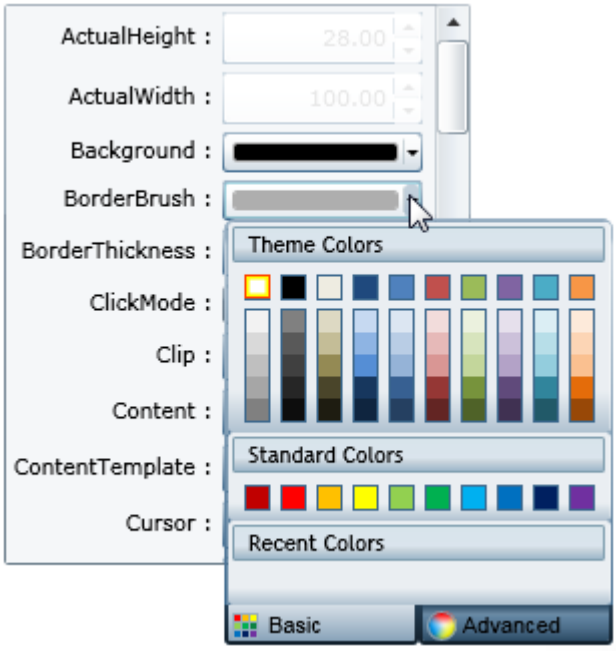
The **C1PropertyGrid** control is part of the **C1.Silverlight.Extended** assembly.

PropertyGrid Key Features

ComponentOne PropertyGrid for Silverlight allows you to create customized, rich applications. Make the most of **PropertyGrid for Silverlight** by taking advantage of the following key features:

- **Run-time Property Editing**

  You can add the **C1PropertyGrid** control to your Silverlight application as part of your design and use it to allow end-users to edit properties directly.



- **Dynamic Display**

  It's as simple as adding the **C1PropertyGrid** control to your form, setting one property, and the UI is automatically built for editing objects.

- **Familiar Properties Window**

  You can add the **C1PropertyGrid** control to your Silverlight application as part of your design and use it to allow end-users to edit properties directly.

- **Select from the Most Popular Silverlight Toolkit Themes**

Add style to your UI with built-in support for the most popular Microsoft Silverlight Toolkit themes, including **ExpressionDark**, **ExpressionLight**, **WhistlerBlue**, **RainerOrange**, **ShinyBlue**, and **BureauBlack**.

PropertyGrid for Silverlight Quick Start

Like the standard Microsoft **PropertyGrid** control, the C1PropertyGrid control works based on a **SelectedObject** property. Once this property is set, the control displays the object's public properties and allows the user to edit them. In this quick start guide, you will define a Silverlight application, and C1PropertyGrid to create a user interface to display and edit information in a control.

Step 1 of 3: Creating the C1PropertyGrid Application

In this step you'll create a Silverlight application in Microsoft Expression Blend using **PropertyGrid for Silverlight**. When you add a C1PropertyGrid control to your application, you'll have a complete, functional standard Properties window-like interface that users can use to browse and edit properties and/or methods on any .NET object.

To set up your project and add a C1PropertyGrid control to your application, complete the following steps:

1. In Expression Blend, select **File | New Project**.
2. In the **New Project** dialog box, select the **Silverlight** project type in the left pane and in the right-pane select **Silverlight Application + Website**. Enter a **Name** and **Location** for your project, select a **Language** in the drop-down box, and click **OK**.

   A new application will be created and should open with the **MainPage.xaml** file displayed in Design view.
3. Navigate to the **Projects** window and right-click the **References** folder in the project files list. In the context menu choose **Add Reference**, locate and select the **C1.Silverlight.dll** and **C1.Silverlight.Extended.dll** assemblies, and click **Open**. The dialog box will close and the references will be added to your project.
4. In the Toolbox click on the **Assets** button (the double chevron icon) to open the **Assets** dialog box.
5. In the **Asset Library** dialog box, choose the **Controls** item in the left pane, and then click on the **C1PropertyGrid** icon in the right pane.

   The **C1PropertyGrid** icon will appear in the Toolbox under the **Assets** button.
6. Click once on the design area of the **UserControl** to select it. Unlike in Visual Studio, in Blend you can add Silverlight controls directly to the design surface as in the next steps.
7. Double-click the **StackPanel** button in the Toolbox to add it to the page. If the **StackPanel** button is not visible, you may need to click once on the **Grid** panels button and select **StackPanel**.
8. In the Properties window, set the following properties for the **StackPanel**:
   - Set **Height** and **Width** properties to "Auto".

- Set the **Orientation** property to **Horizontal**.
- Set **HorizontalAlignment** and **VerticalAlignment** to **Center** to center controls in the panel.

9. With the **StackPanel** still selected in the **Objects and Timeline** pane, double-click on the standard **Button** control in the Toolbox to add it to the panel. You'll set properties on this control using the **C1PropertyGrid** control.
10. Select the **Button** control and in the Properties window set its **Name** property to "button1", its **Width** to **75**, and its **Height** to **290**.
11. Select the **StackPanel** in the **Objects and Timeline** pane and then double-click the **C1PropertyGrid** icon in the Toolbox to add the control to the panel.
12. Click once on the **C1PropertyGrid** control in the **Objects and Timeline** pane, navigate to the Properties window and set the following properties:
    - Set **Name** to "c1propertygrid1" to give the control a name so it is accessible in code.
    - Set **Width** to "250" and **Height** to "290".
    - Set **HorizontalAlignment** and **VerticalAlignment** to **Center** to center the control in the panel.

    The XAML will appear similar to the following:

```
<StackPanel HorizontalAlignment="Center" VerticalAlignment="Center"
Orientation="Horizontal">
    <Button Width="75" Height="290" Content="Button"/>
    <c1:C1PropertyGrid Height="250" Width="290" HorizontalAlignment="Center"
VerticalAlignment="Center"/>
</StackPanel>
```

You've successfully set up your application's user interface, but the C1PropertyGrid control contains no content. In the next step you'll set the C1PropertyGrid control to display certain properties of the **Button** control, and then you'll add code to your application to add functionality to the control.

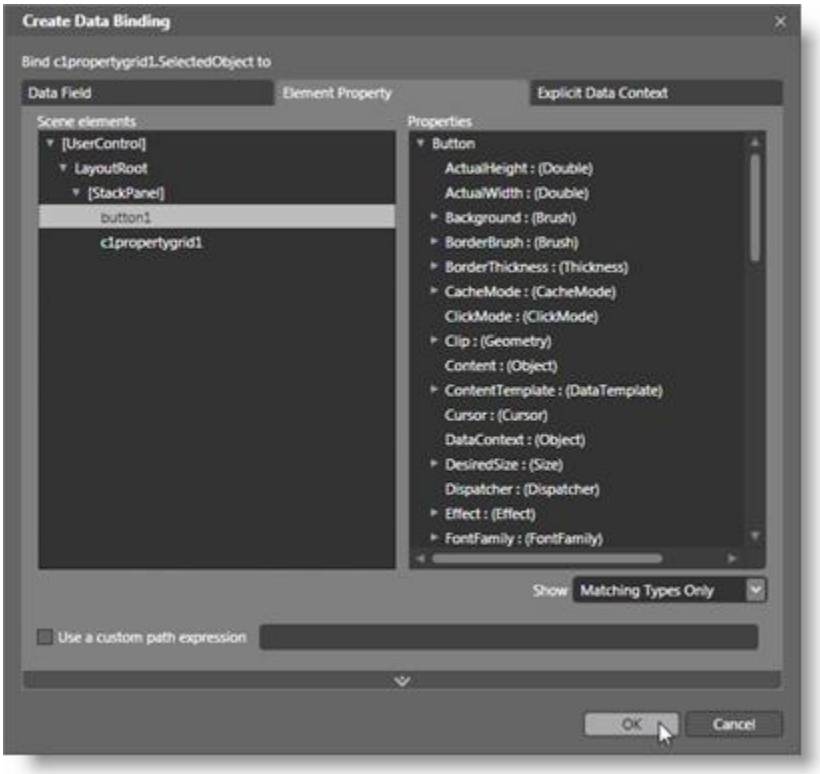
Step 2 of 3: Customizing the C1PropertyGrid Application

In the last step you created a Silverlight application and added a **Button** and the C1PropertyGrid control to the application. In this step you'll customize the C1PropertyGrid control to display specific properties of the **Button** control.

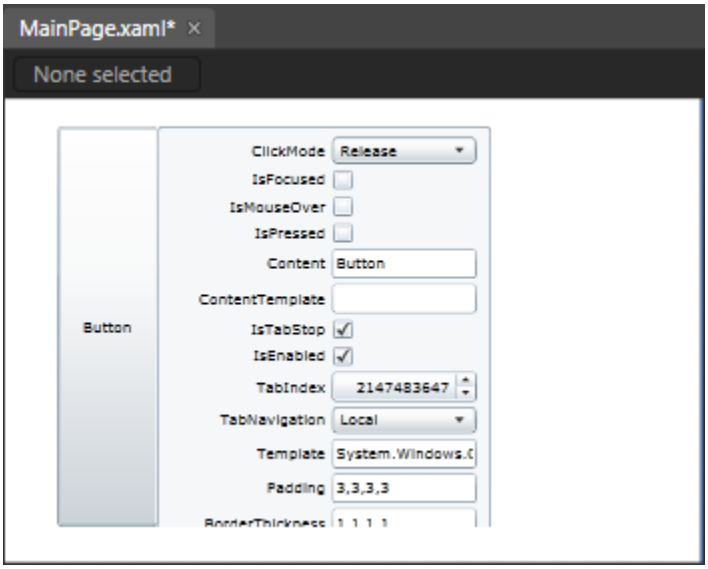
To customize and connect the C1PropertyGrid control to the **Button** control, complete the following steps:

1. Select **c1propertygrid1** in the **Objects and Timeline** pane and navigate to the Properties window.
2. In the Properties window, locate the SelectedObject property, click the small square next to it to access advanced property options, and select the **Data Binding** option.

   The **Create Data Binding** dialog box will appear.
3. In the **Create Data Binding** dialog box, click the **Element Property** tab.
4. In the left side **Scene elements** window select **button1** and click **OK**:



Notice that all the button's properties are now displayed in the C1PropertyGrid control:



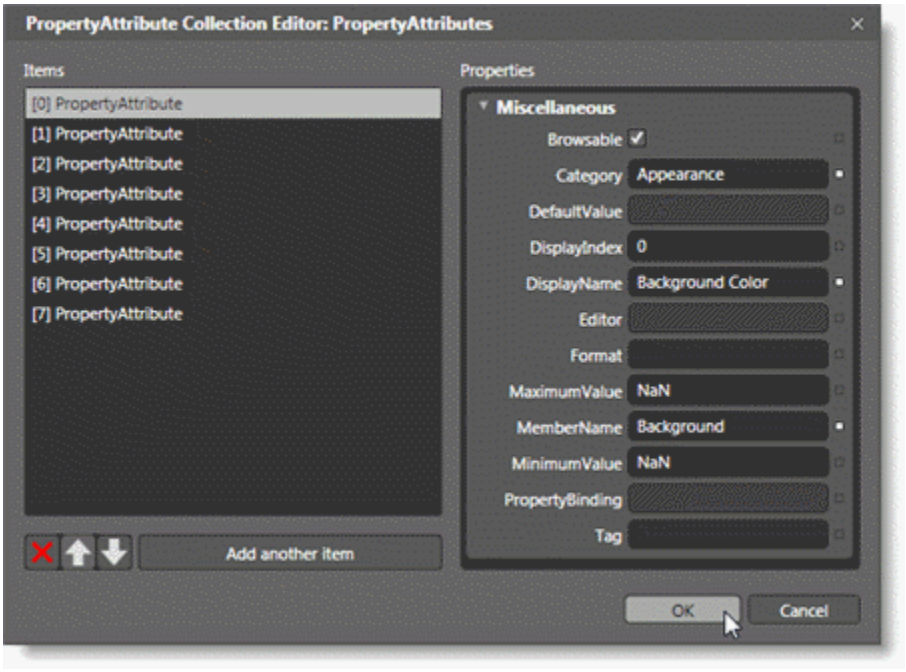
In the next steps you'll customize the C1PropertyGrid control so that only certain properties are displayed.

5. In the Properties window uncheck the AutoGenerateProperties check box. Now every property will no longer be displayed – only those that you specify.
6. Locate the PropertyAttributes collection in the Properties window, and click the ellipsis button next to the item. The **Property Attribute Collection Editor** dialog box will appear.

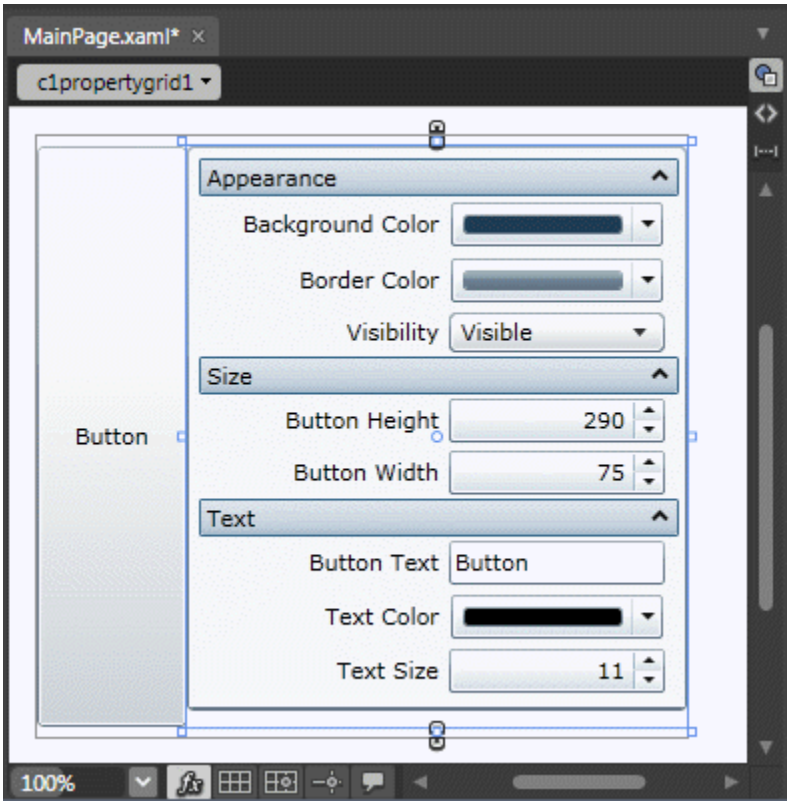
7. In the **Property Attribute Collection Editor** dialog box, click the **Add another item** button. Repeat this step seven more times to create a total of eight **PropertyAttribute** items, numbered 0 to 7.
8. Set the following Properties in the right-side Properties pane for the items you just added:

PropertyAttribute	Category	DisplayName	MemberName
[0] PropertyAttribute	Appearance	Background Color	Background
[1] PropertyAttribute	Appearance	Border Color	BorderBrush
[2] PropertyAttribute	Appearance	Visibility	Visibility
[3] PropertyAttribute	Size	Button Height	Height
[4] PropertyAttribute	Size	Button Width	Width
[5] PropertyAttribute	Text	Button Text	Content
[6] PropertyAttribute	Text	Text Color	Foreground
[7] PropertyAttribute	Text	Text Size	FontSize

The **Category** identifies what section the item appears in. The **DisplayName** indicates the name displayed for the item. The **MemberName** indicates the actual name of the member.



9. Click the **OK** button to close the **Property Attribute Collection Editor** dialog box and change the settings. The page should now look similar to the following image at design time:



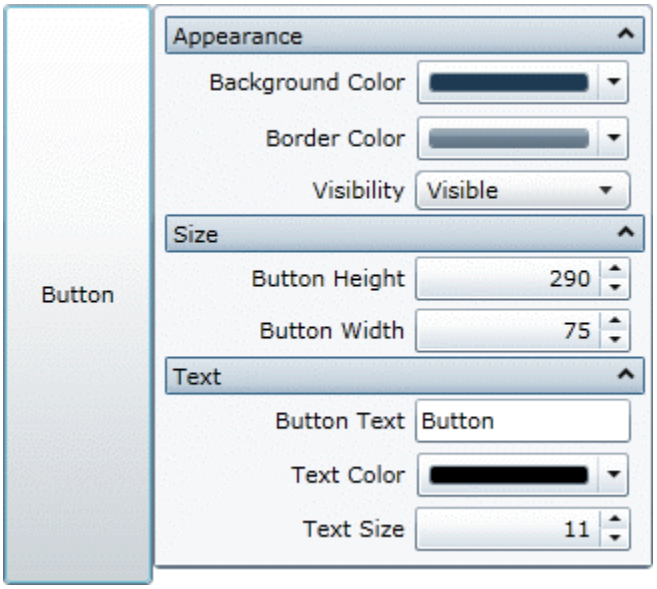
In this step you'll customized the C1PropertyGrid control to display specific properties of the **Button** control. In the next step, you'll run the application and view some of the possible run-time interactions.

Step 3 of 3: Running the PropertyGrid Application

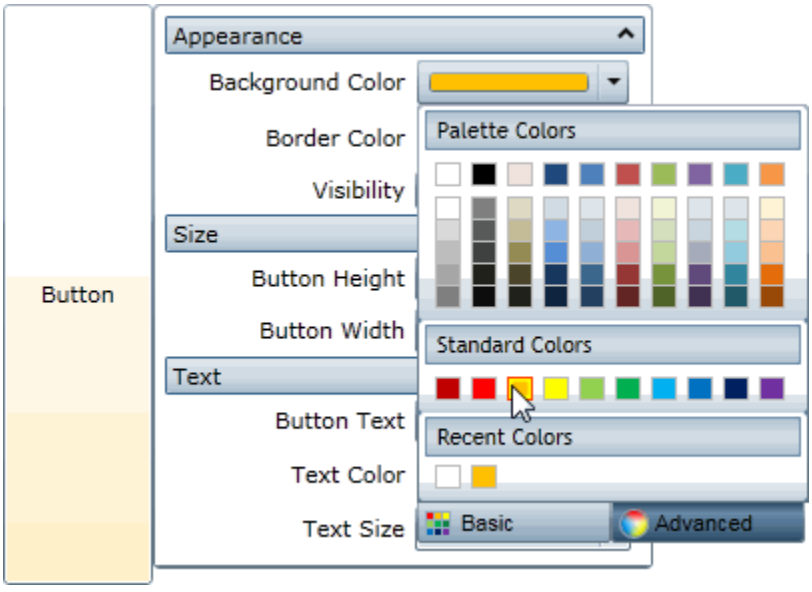
In the previous steps you created a Silverlight application in Microsoft Expression Blend using **PropertyGrid for Silverlight**, set it to view properties of a **Button** control, and customized the displayed properties. In this step you'll run your application and view some of the run time interactions possible with **PropertyGrid for Silverlight**.

Complete the following steps:

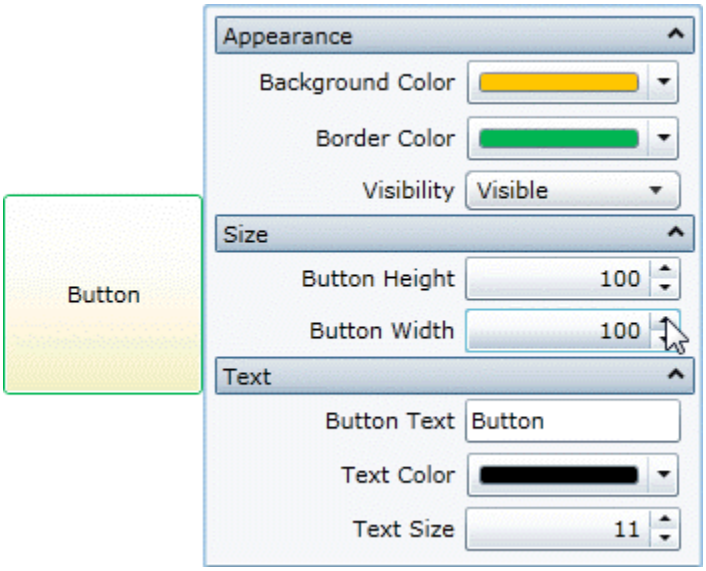
1. In Expression Blend, select **Project | Run Project**. The application will open in your default Web browser. Observe that the application appears with the properties you specified and the display names you entered:



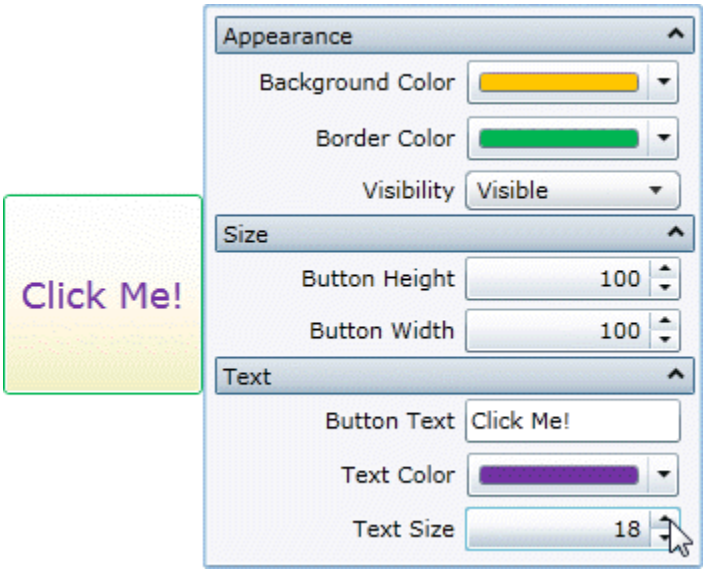
2. Click the **Background Color** drop-down arrow and pick a color, for example orange, from the color picker that appears. The button's background color will change to your selection:



3. Click the **Border Color** drop-down arrow and pick a color, for example green, from the color picker that appears.
4. Change the size of the button by entering values in the **Button Height** and **Button Width** numeric boxes. For example enter **100** for both values. The application will appear similar to the following:



5. Enter a string, for example "Click Me!" in the **Button Text** box.
6. Click the **Text Color** drop-down arrow and pick a color, for example purple, from the color picker that appears.
7. Click the **Up** or **Down** arrow next to the **Text Size** value to change the size of the text that appears on the button control. For example, set the value to **18**. The application will appear similar to the following:

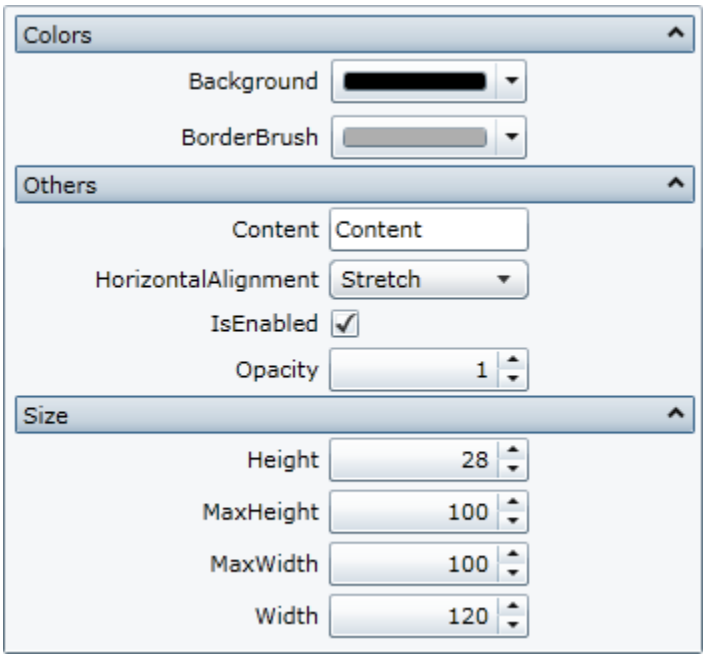


Congratulations! You've completed the **ComponentOne PropertyGrid for Silverlight** quick start. In this quick start you added the C1PropertyGrid and **Button** controls to a page, linked the C1PropertyGrid control to the **Button**, customized the controls, and view the run-time interactions possible with **PropertyGrid for Silverlight**.

Working with PropertyGrid for Silverlight

ComponentOne PropertyGrid for Silverlight includes the C1PropertyGrid control, a simple control that lets users easily edit any class with more than 10 built-in editors. When you add the C1PropertyGrid control to a XAML window, it exists as a complete control which you can further customize.

The Basic C1PropertyGrid control appears similar to the Microsoft **PropertyGrid** control. It appears as a window in which properties and methods of an object can be edited at run time:



You can customize what appears in the control or have the content be automatically generated from an object's properties. You can customize this grid by added headers to organize content. You can also customize the appearance of the control as you could any other Silverlight control.

Basic Properties

ComponentOne PropertyGrid for Silverlight includes several properties that allow you to set the functionality of the control. Some of the more important properties are listed below. Note that you can see PropertyGrid Appearance Properties (page 237) for more information about properties that control appearance.

The following properties let you customize the C1PropertyGrid control:

Property	Description
AutoGenerateMethods	Gets or sets a value indicating whether the **C1PropertyGrid** should try to get the methods of the **SelectedObject** using reflection.
AutoGenerateProperties	Gets or sets a value indicating whether the **C1PropertyGrid** should try to get the properties of the **SelectedObject** using reflection.
AvailableEditors	The list of currently available editors to use in the **C1PropertyGrid**. The default list includes editors for: bool, Color, Brush, ColorPalette, Enum, Numeric, String, Image and Uri values.

CategoryContainers	The list of category containers (if defined) created to show each category
CategoryContainerStyle	Gets or sets the **Style** applied to all the generated category containers.
DefaultCategoryName	The name used for the default category in which the properties are placed.
DisabledCuesVisibility	Gets a value indicating whether the disabled visuals of the control are visible.
EditorWidth	Gets or sets the width of generated editors for the **SelectedObject**'s properties.
FocusCuesVisibility	Gets a value indicating whether the focus visuals of the control are visible.
IsFocused	Returns true if the control has the focus.
IsMouseOver	Returns true if the mouse is over the control.
LabelStyle	Gets or sets the **Style** applied to all the generated labels for the **SelectedObject**'s properties.
LabelWidth	Gets or sets the width of generated labels for the **SelectedObject**'s properties.
MethodAttributes	Gets or sets the list of method attributes used to configure the visualization of the **SelectedObject**'s methods.
MethodBoxes	Gets the **MethodBoxes** generated to show the **SelectedObject**'s methods on the **C1PropertyGrid**.
MethodsPanel	Gets or sets the template that defines the panel that controls the layout of methods.
PropertiesPanel	Gets or sets the template that defines the panel that controls the layout of properties.
PropertyAttributes	Gets or sets the list of property attributes used to configure the visualization of the **SelectedObject**'s properties.
PropertyBoxes	Gets the **PropertyBoxes** generated to show the **SelectedObject**'s properties on the **C1PropertyGrid**.
PropertySort	Gets or sets the type of sorting the **C1PropertyGrid** uses to display properties.
SelectedObject	Gets or sets the object for which the **C1PropertyGrid** displays properties. See The Selected Object (page 233) for more information.

Basic Events

ComponentOne PropertyGrid for Silverlight includes several events that allow you to set interaction and customize the control. Some of the more important events are listed below.

The following events let you customize the C1PropertyGrid control:

Event	Description
CategoryContainerAdded	Fired when a category container is added to the **C1PropertyGrid**.
CategoryContainerRemoved	Fired when a category container is removed from the **C1PropertyGrid**.
IsMouseOverChanged	Event raised when the IsMouseOver property has changed.

MethodBoxChanged	Fired when the generated list of method editors for the **SelectedObject** changes.
MethodBoxesLoaded	Fired when the all the method boxes in **C1PropertyGrid** were loaded.
PropertyBoxAdded	Fired when a property box is added to the **C1PropertyGrid**.
PropertyBoxChanged	Fired when the generated list of property editors for the **SelectedObject** changes.
PropertyBoxesLoaded	Fired when the all the property boxes in **C1PropertyGrid** were loaded.
PropertyBoxRemoved	Fired when a property box is removed from the **C1PropertyGrid**.
ValueChanged	Fired when the value of a property in the current **SelectedObject** changes.

The Selected Object

The SelectedObject property determines the object for which the C1PropertyGrid control displays properties to edit. You can set the SelectedObject property to any object. For example, you can connect the C1PropertyGrid control to a control as in the PropertyGrid for Silverlight Quick Start (page 224) or you can bind the control to a class as in the Binding C1PropertyGrid to a Class (page 240) topic.

In XAML, you would use a binding statement to connect the C1PropertyGrid control to an object. For example, the following C1PropertyGrid control is linked to a button object:

```
<c1:C1PropertyGrid Margin="244,152,186,168" SelectedObject="{Binding
ElementName=button, Mode=OneWay}"/>
```

You can also set the SelectedObject property in Design view in Blend by selecting the square Advanced Properties icon next to the SelectedObject item in the Properties window and selecting **Data Binding**. The **Create Data Binding** dialog box will appear allowing you to choose an object to bind to.

Automatically Generating Properties and Methods

By default when you set the SelectedObject property to bind to an object, the members listed in the C1PropertyGrid control will be automatically generated. This is because the AutoGenerateProperties property is set to **True** by default. You can set the SelectedObject property to **False** if you do not want properties to be automatically generated. You might do so, for example, if you want only a few properties to be editable or you want to customize the way properties appear in the C1PropertyGrid window.

While properties are visible by default, however, methods are not. By default the AutoGenerateMethods property is set to **False** and methods are not automatically displayed. If you choose, you can automatically generate methods by setting the AutoGenerateMethods property to **True**.

Sorting Members in C1PropertyGrid

By default properties and methods are listed alphabetically in the C1PropertyGrid control, similar to the **Alphabetic** view in the Visual Studio Properties window. However, you can customize the way members are listed by setting the PropertySort property. The C1PropertyGrid control can sort the properties in any of the following ways:

- **Alphabetical**: the properties are sorted in an alphabetical list. This is the default setting and appears similar to the **Alphabetic** view in the Visual Studio Properties window.
- **Categorized**: the categories are displayed in an alphabetical list, the properties in each category are displayed in no particular order (the order in which are retrieved from the **SelectedObject**).

- **CategorizedAlphabetical**: the categories are displayed in an alphabetical list; the properties in each category are displayed in alphabetical order. This appears similar to the **Categorized** view in the Visual Studio Properties window.
- **CategorizedCustom**: the categories are displayed in an alphabetical list; the properties inside each category are displayed in a custom order defined by the user using the **Display.Order** attribute.
- **Custom**: the properties are displayed in custom order defined by the user using the **Display.Order** attribute.
- **NoSort**: properties are displayed in the order in which they are retrieved from the **SelectedObject**.

Set the PropertySort property to one of the above options to customize the way the property grid is sorted.

Built-in Editors

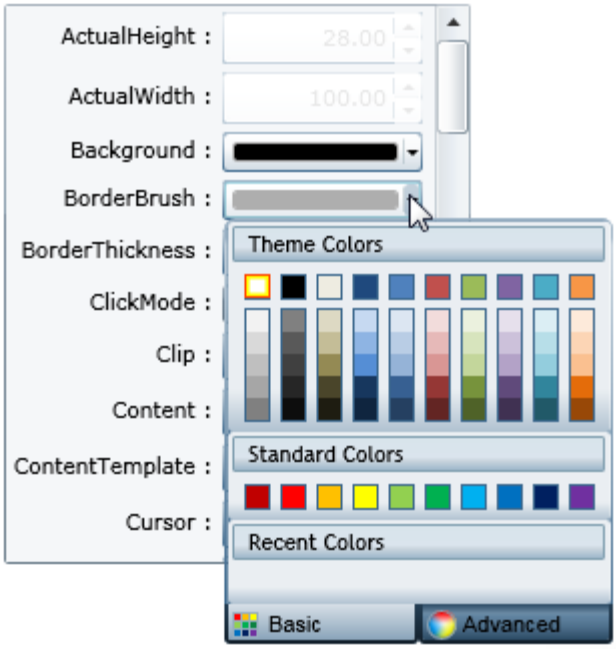
The C1PropertyGrid control includes several built-in editors. If you do not specify an editor, C1PropertyGrid will display each member with a default editor. You can also specify editors, and if needed, create your own custom editor. The AvailableEditors property controls what editors are available in the C1PropertyGrid control.

Because the C1PropertyGrid control does not have built-in editors for complex objects, complex object will be shown in the default editor (StringEditor) and the string representation (tostring()) of the complex object will be shown. If you need to show complex objects, you may want to create a custom editor. For more information about creating a custom editor, see Creating Custom Editors (page 250).

Built-in editors include:

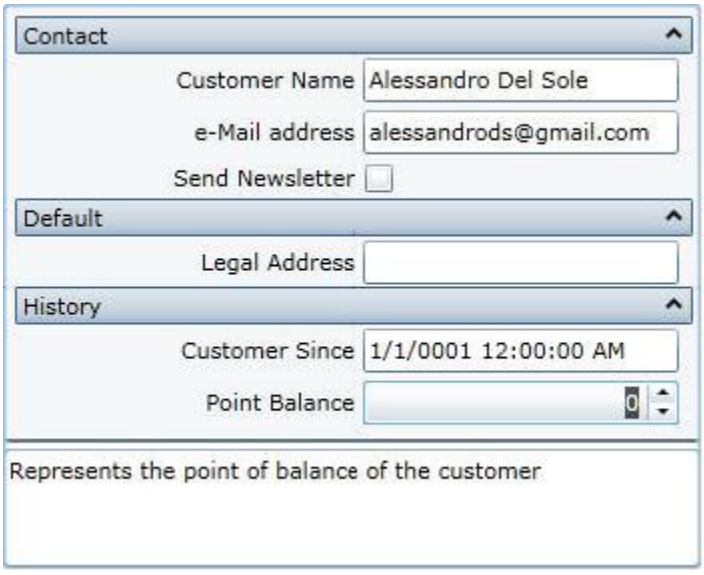
Editor	Description
BoolEditor	Default editor used by C1PropertyGrid to edit bool values.
BrushEditor	Default editor used by C1PropertyGrid to edit Brush values.
ColorEditor	Default editor used by C1PropertyGrid to edit color values.
ColorPaletteEditor	Default editor used by C1PropertyGrid to edit color palette values.
EnumEditor	Default editor used by C1PropertyGrid to edit Enum values.
ImageSourceEditor	Default editor used by C1PropertyGrid to edit image values.
NumericEditor	Default editor used by C1PropertyGrid to edit numeric values.
StringEditor	Default editor used by C1PropertyGrid to edit String values.
UriEditor	Default editor used by C1PropertyGrid to edit Uri values.

For example, color values are editable at run time in the following image:



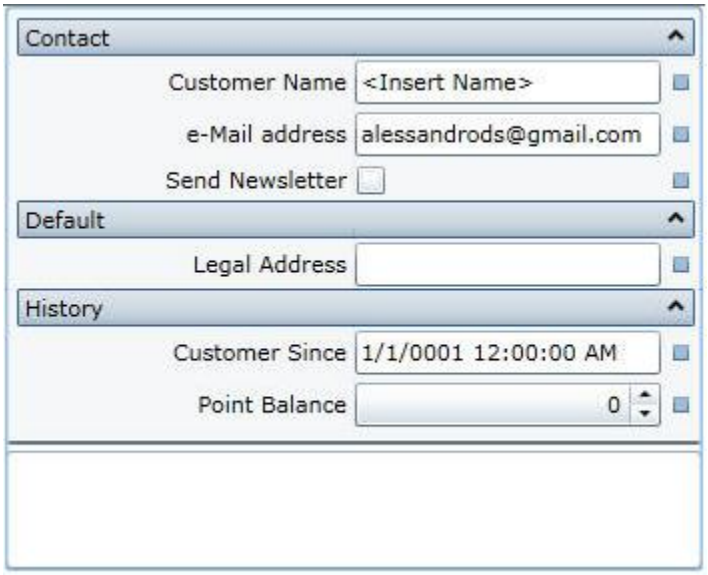
Showing Property Descriptions

By default the C1PropertyGrid control does not display descriptions for properties users can edit at run time. However, in some case, you may want to display additional information about the properties being edited. The C1PropertyGrid control allows you to show a description for each property, similar to the Description pane in the Visual Studio Properties window. You can make property descriptions visible by setting the ShowDescription property to **True**. This will add a description area at the bottom of the C1PropertyGrid control, showing the description for the property currently focused. The description, which can be added using the **Display.Description** attribute, appears similar to the following image:



Resetting Property Values

C1PropertyGrid provides a reset button that allows users to reset a changed property to its default value (specified using the **DefaultValue** attribute). The reset button is small rectangle next to the property editor:



By default, the **Reset** button is not shown, but you can make the **Reset** button visible by setting the ShowResetButton property to **True**. It's important to note that the **SelectedObject** should implement the **INotifyPropertyChanged** interface so it can reflect the updated value; this is because bound properties must notify value changes to the editor so it can refresh values.

Supported PropertyGrid Attributes

You can use attributes to customize the display and content of items displayed in the property grid. The C1PropertyGrid control supports several attributes, including the following:

- **Display.Name**: sets the text label displayed for each property
- **Display.Order**: used to specify the order in which properties are shown (when using custom sort)
- **Display.Description**: used to set a description for the property; this description will be shown at the bottom of the C1PropertyGrid, when the property that is being edited gets the focus.
- **DefaultValue**: used to set a default value for the property; the default value will be applied when the property has no other value (it was not initialized or has a null value) or when the **Reset** button is pressed.
- **Editor**: used to set a custom editor for the current property (override the editor assigned by default).
- **MaximumValue**: used to set a maximum value for a property, the MaximumValue will be taken into consideration only for those editors where it makes sense (the numeric editor, for example).
- **MinimumValue**: used to set a minimum value for a property, the MinimumValue will be taken into consideration only for those editors where it makes sense (the numeric editor, for example).
- **Browsable**: if set to false, the property will not be shown in the C1PropertyGrid.
- **Category**: used to set the category in which the property will be shown.

- **PropertyBinding**: used to override the default binding between the property and its editor; this is useful, for example, if you want to set your own converter or if you don't want the default validation for properties.
- **DisplayFormat**: used to specify a format to show the property value, the **DisplayFormat** will be taken into consideration only for those editors where it makes sense (the numeric editor, for example).
- **ReadOnly**: if set to **True**, the property will be shown in the C1PropertyGrid but it won't be possible to change its value.

PropertyGrid Layout and Appearance

The following topics detail how to customize the C1PropertyGrid control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the grid and take advantage of Silverlight's XAML-based styling. You can also use templates to format and lay out the control and to customize the control's actions.

PropertyGrid Appearance Properties

ComponentOne PropertyGrid for Silverlight includes several properties that allow you to customize the appearance of the control. You can change the color, border, and height of the control. The following topics describe some of these appearance properties.

Color Properties

The following properties let you customize the colors used in the control itself:

Property	Description
Background	Gets or sets a brush that describes the background of a control. This is a dependency property.
Foreground	Gets or sets a brush that describes the foreground color. This is a dependency property.

Alignment Properties

The following properties let you customize the control's alignment:

Property	Description
HorizontalAlignment	Gets or sets the horizontal alignment characteristics applied to this element when it is composed within a parent element, such as a panel or items control. This is a dependency property.
VerticalAlignment	Gets or sets the vertical alignment characteristics applied to this element when it is composed within a parent element such as a panel or items control. This is a dependency property.

Border Properties

The following properties let you customize the control's border:

Property	Description
BorderBrush	Gets or sets a brush that describes the border background of a control. This is a dependency property.
BorderThickness	Gets or sets the border thickness of a control. This is a dependency property.

Size Properties

The following properties let you customize the size of the **C1PropertyGrid** control:

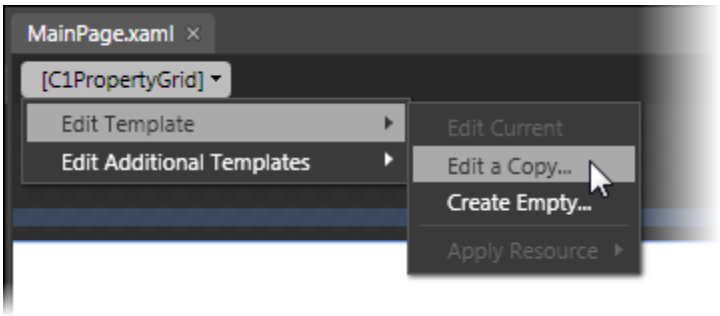
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
MinHeight	Gets or sets the minimum height constraint of the element. This is a dependency property.
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

PropertyGrid Templates

One of the main advantages to using a Silverlight control is that controls are "lookless" with a fully customizable user interface. Just as you design your own user interface (UI), or look and feel, for Silverlight applications, you can provide your own UI for data managed by **ComponentOne PropertyGrid for Silverlight**. Extensible Application Markup Language (XAML; pronounced "Zammel"), an XML-based declarative language, offers a simple approach to designing your UI without having to write code.

Accessing Templates

You can access templates in Microsoft Expression Blend by selecting the C1PropertyGrid control and, in the menu, selecting **Edit Control Parts (Templates)**. Select **Edit a Copy** to create an editable copy of the current template or **Create Empty**, to create a new blank template.



Note: If you create a new template through the menu, the template will automatically be linked to that template's property. If you manually create a template in XAML you will have to link the appropriate template property to the template you've created.

Note that you can use the Template property to customize the template.

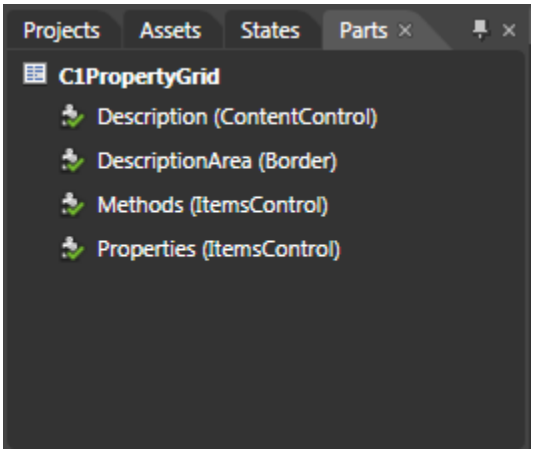
PropertyGrid Styles

ComponentOne PropertyGrid for Silverlight's **C1PropertyGrid** control provides several style properties that you can use to change the appearance of the control. Some of the included styles are described in the table below:

Style	Description
CategoryContainerStyle	Gets or sets the **Style** applied to all the generated category containers.
FontStyle	Gets or sets the font style. This is a dependency property.
LabelStyle	Gets or sets the Style applied to all the generated labels for the **SelectedObject**'s properties.
Style	Gets or sets the style used by this element when it is rendered. This is a dependency property.

PropertyGrid Template Parts

In Microsoft Expression Blend, you can view and edit template parts by creating a new template (for example, click the **C1PropertyGrid** control to select it and choose **Object | Edit Template | Edit a Copy**). Once you've created a new template, the parts of the template will appear in the **Parts** window:



Note that you may have to select the **ControlTemplate** for its parts to be visible in the **Parts** window. In the Parts window, you can double-click any element to create that part in the template.

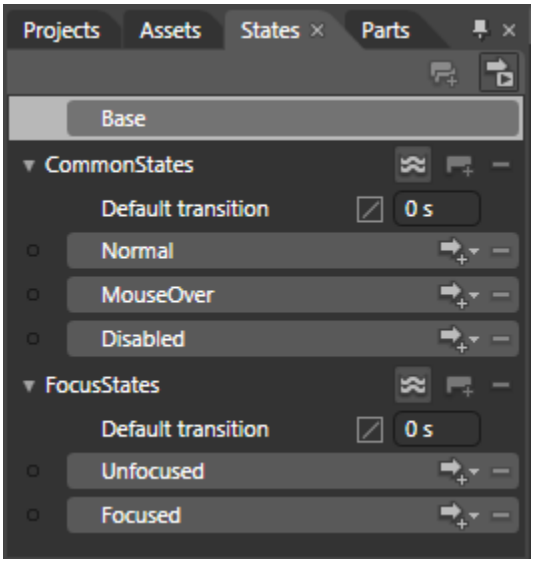
Template parts available in the **C1PropertyGrid** control include:

Name	Type	Description
Description	ContentControl	Represents a control with a single piece of content. Here, it represents the description area at the bottom of the control.
DescriptionArea	Border	Draws a border, background, or both around another element. Here the border surrounds the description area at the bottom of the control.
Methods	ItemsControl	Represents a control that can be used to present a

		collection of items. Here it represents the collection of methods.
Properties	ItemsControl	Represents a control that can be used to present a collection of items. Here it represents the collection of properties.

PropertyGrid Visual States

In Microsoft Expression Blend, you can add custom states and state groups to define a different appearance for each state of your user control – for example, the visual state of the control could change on mouse over. You can view and edit visual states by creating a new template and adding a new template part (page 239). Once you've done so the available visual states for that part will be visible in the **Visual States** window:



Common states include **Normal** for the normal appearance of the item, **MouseOver** for the item on mouse over, and **Disabled** for when the item is not enabled. Focus states include **Unfocused** for when the item is not in focus and **Focused** when the item is in focus.

PropertyGrid for Silverlight Task-Based Help

The task-based help assumes that you are familiar with Visual Studio and Expression Blend and know how to use the C1PropertyGrid control in general. If you are unfamiliar with the **ComponentOne PropertyGrid for Silverlight** product, please see the PropertyGrid for Silverlight Quick Start (page 224) first.

Although it can be used by setting a single property, the **C1PropertyGrid** provides extensive customization support. Each topic in this section provides a solution for specific tasks using the **ComponentOne PropertyGrid for Silverlight** product. Each task-based help topic also assumes that you have created a new Silverlight project and added a C1PropertyGrid control to the project.

Binding C1PropertyGrid to a Class

PropertyGrid for Silverlight allows you to easily bind the control to a class. At run time items in the class can be browsed and edited using the C1PropertyGrid control. For example, assuming you have a simple **Customer** class defined as follows:

- Visual Basic

```
Private _Name As String
```

```
    Public Property Name() As String
        Get
            Return _Name
        End Get
        Set(ByVal value As String)
            _Name = value
        End Set
    End Property
Private _EMail As String
    Public Property EMail() As String
        Get
            Return _EMail
        End Get
        Set(ByVal value As String)
            _EMail = value
        End Set
    End Property
Private _Address As String
    Public Property Address() As String
        Get
            Return _Address
        End Get
        Set(ByVal value As String)
            _Address = value
        End Set
    End Property
Private _CustomerSince As DateTime
    Public Property CustomerSince() As DateTime
        Get
            Return _CustomerSince
        End Get
        Set(ByVal value As DateTime)
            _CustomerSince = value
        End Set
    End Property
Private _SendNewsletter As Boolean
    Public Property SendNewsletter() As Boolean
        Get
            Return _SendNewsletter
        End Get
        Set(ByVal value As Boolean)
            _SendNewsletter = value
        End Set
    End Property
Private _PointBalance As System.Nullable(Of Integer)
    Public Property PointBalance() As System.Nullable(Of Integer)
        Get
            Return _PointBalance
        End Get
        Set(ByVal value As System.Nullable(Of Integer))
            _PointBalance = value
        End Set
    End Property
End Class
```

- C#

```
public class Customer
```

```
{
  public string Name { get; set; }
  public string EMail { get; set; }
  public string Address { get; set; }
  public DateTime CustomerSince { get; set; }
  public bool SendNewsletter { get; set; }
  public int? PointBalance { get; set; }
}
```

You could build a user interface to display and edit customers using the following code:

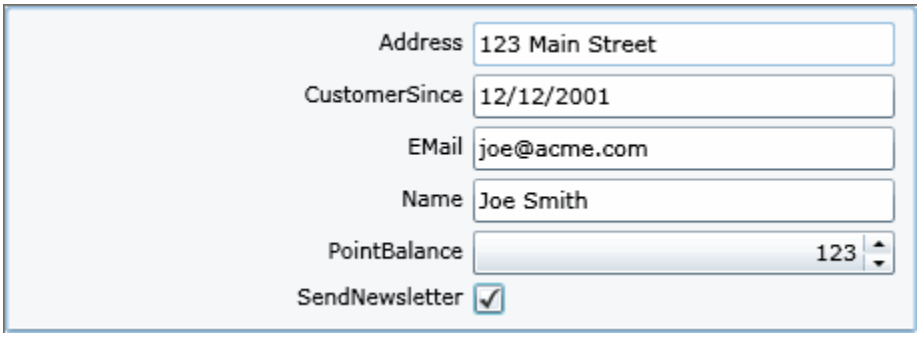
- Visual Basic

```
Public Sub New()
    InitializeComponent()

    ' Create object to browse
    Dim customer = New Customer()

    ' Create C1PropertyGrid
    Dim pg = New C1PropertyGrid()
    LayoutRoot.Children.Add(pg)

    ' Show customer properties
    pg.SelectedObject = customer
End Sub
```

- C#

```
public Page()
{
  InitializeComponent();

  // Create object to browse
  var customer = new Customer();

  // Create C1PropertyGrid
  var pg = new C1PropertyGrid();
  LayoutRoot.Children.Add(pg);

  // Show customer properties
  pg.SelectedObject = customer;
}
```

Run the application and observe that the resulting application would look similar to the following:



This simple UI allows users to edit all the properties in the **Customer** objects. It was built automatically based on the object's properties and will be automatically updated if you add or modify the properties in the **Customer** class.

Notice that properties are shown in alphabetical order by default. You can change this by setting the PropertySort property; for more information see Sorting Members in C1PropertyGrid (page 233).

Customizing the Control Layout

The first aspect of the control that you may want to customize is the layout. The control presents two columns, one with labels and one with editors. The columns have the same size by default, but you can change that by changing the value of the LabelWidth and EditorWidth properties.

For example, you could make the label column narrower in the example above by adding one line of code:

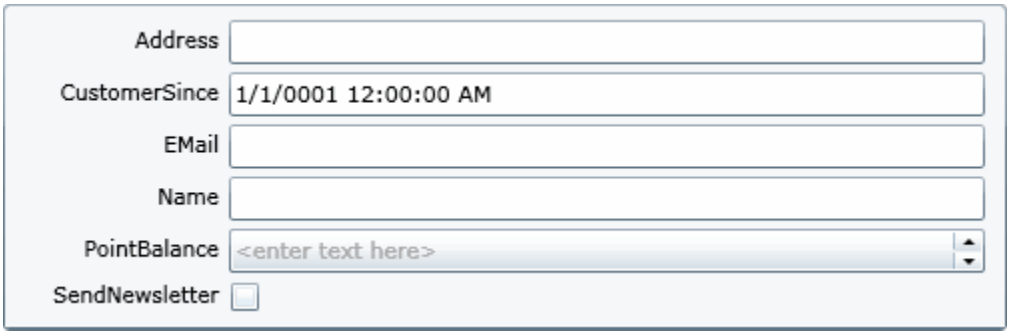
- Visual Basic

```
Public Sub New()
    InitializeComponent()

    ' Create object to browse
    Dim customer = New Customer()

    ' Create C1PropertyGrid
    Dim pg = New C1PropertyGrid()
    LayoutRoot.Children.Add(pg)

    ' Customize the PropertyGrid layout
    pg.LabelWidth = 100

    ' Show customer properties
    pg.SelectedObject = customer
End Sub
```

- C#

```
public Page()
{
  InitializeComponent();

  // Create object to browse
  var customer = new Customer();

  // Create C1PropertyGrid
  var pg = new C1PropertyGrid();
  LayoutRoot.Children.Add(pg);

  // Customize the PropertyGrid layout
  pg.LabelWidth = 100;

  // Show customer properties
  pg.SelectedObject = customer;
}
```

The result would be as shown below:



As you can see, the label column is now narrower and more room is left for the editor part. If you resize the form, you will notice that the width of the label column remains constant.

Customizing Display Names

By default, the labels shown next to each property display the property name. This works fine in many cases, but you may want to customize the display to provide more descriptive names. The easiest way to achieve this is to decorate the properties on the object with custom attributes and by setting the **Name** property in the **Display** attribute (note that the Display attribute is defined in the System.ComponentModel.DataAnnotations namespace, in the System.ComponentModel,DataAnnotations assembly).

For example, you could define the **Display** attribute in the class itself and set the value for the **Name** property as in the following code:

- Visual Basic

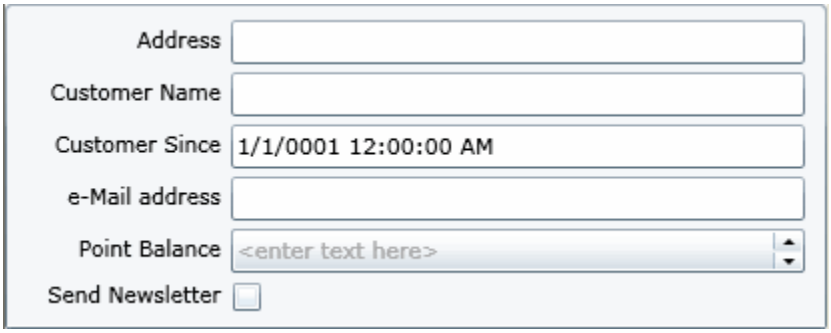
```
Public Class Customer
Private _Name As String
    <Display(Name:="Customer Name")> _
    Public Property Name() As String
        Get
            Return _Name
        End Get
        Set(ByVal value As String)
            _Name = value
        End Set
    End Property

Private _EMail As String
    <Display(Name:="e-Mail address")> _
    Public Property EMail() As String
        Get
            Return _EMail
        End Get
        Set(ByVal value As String)
            _EMail = value
        End Set
    End Property

Private _Address As String
    Public Property Address() As String
        Get
            Return _Address
        End Get
        Set(ByVal value As String)
            _Address = value
```

```
        End Set
    End Property

Private _CustomerSince As DateTime
    <Display(Name:="Customer Since")> _
    Public Property CustomerSince() As DateTime
        Get
            Return _CustomerSince
        End Get
        Set(ByVal value As DateTime)
            _CustomerSince = value
        End Set
    End Property

Private _SendNewsletter As Boolean
    <Display(Name:="Send Newsletter")> _
    Public Property SendNewsletter() As Boolean
        Get
            Return _SendNewsletter
        End Get
        Set(ByVal value As Boolean)
            _SendNewsletter = value
        End Set
    End Property

Private _PointBalance As System.Nullable(Of Integer)
    <Display(Name:="Point Balance")> _
    Public Property PointBalance() As System.Nullable(Of Integer)
        Get
            Return _PointBalance
        End Get
        Set(ByVal value As System.Nullable(Of Integer))
            _PointBalance = value
        End Set
    End Property
End Class
```

- C#

```
public class Customer
{
  [Display(Name = "Customer Name")]
  public string Name { get; set; }

  [Display(Name = "e-Mail address")]
  public string EMail { get; set; }

  public string Address { get; set; }

  [Display(Name = "Customer Since")]
  public DateTime CustomerSince { get; set; }

  [Display(Name ="Send Newsletter")]
  public bool SendNewsletter { get; set; }

  [Display(Name ="Point Balance")]
  public int? PointBalance { get; set; }
}
```

The **C1PropertyGrid** uses this additional information and displays the customer as shown below:



This method requires that you have access to the class being displayed in the C1PropertyGrid. If you want to change the display strings but cannot modify the class being shown, then you would have to use the PropertyAttributes property to provide explicit information about each property you want to show on the C1PropertyGrid.

Categorizing Properties

You can group properties by category by adding a **Category** attribute to each property on the object being browsed (note that the **Category** attribute is defined in the System.ComponentModel namespace, in the System.Windows assembly). By default the members in the list will be listed alphabetically and uncategorized. By adding categories you can better organize members by listing related members together.

Continuing with our example, here's a revised version of the **Customer** class which includes categories:

- Visual Basic

```
Public Class Customer
Private _Name As String
    <Category("Contact")> _
    <DisplayName("Customer Name")> _
    Public Property Name() As String
        Get
            Return _Name
        End Get
        Set(ByVal value As String)
            _Name = value
        End Set
    End Property

Private _EMail As String
    <Category("Contact")> _
    <DisplayName("e-Mail address")> _
    Public Property EMail() As String
        Get
            Return _EMail
        End Get
        Set(ByVal value As String)
            _EMail = value
        End Set
    End Property

Private _Address As String
    <Category("Contact")> _
```

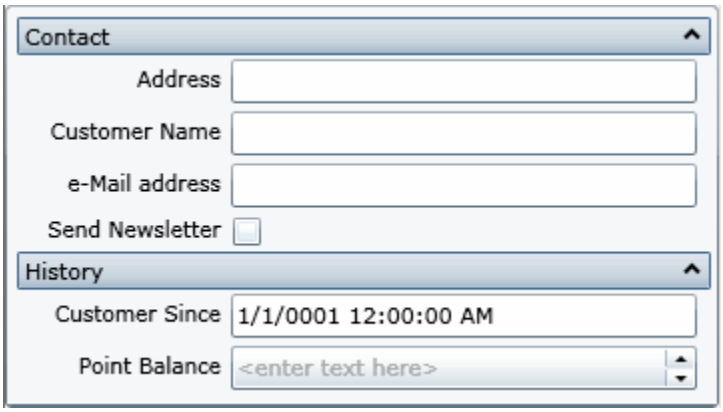
```
    Public Property Address() As String
        Get
            Return _Address
        End Get
        Set(ByVal value As String)
            _Address = value
        End Set
    End Property

Private _CustomerSince As DateTime
    <Category("History")> _
    <DisplayName("Customer Since")> _
    Public Property CustomerSince() As DateTime
        Get
            Return _CustomerSince
        End Get
        Set(ByVal value As DateTime)
            _CustomerSince = value
        End Set
    End Property

Private _SendNewsletter As Boolean
    <Category("Contact")> _
    <DisplayName("Send Newsletter")> _
    Public Property SendNewsletter() As Boolean
        Get
            Return _SendNewsletter
        End Get
        Set(ByVal value As Boolean)
            _SendNewsletter = value
        End Set
    End Property

Private _PointBalance As System.Nullable(Of Integer)
    <Category("History")> _
    <DisplayName("Point Balance")> _
    Public Property PointBalance() As System.Nullable(Of Integer)
        Get
            Return _PointBalance
        End Get
        Set(ByVal value As System.Nullable(Of Integer))
            _PointBalance = value
        End Set
    End Property
End Class
```

- C#

```
public class Customer
{
  [Category("Contact")]
  [DisplayName("Customer Name")]
  public string Name { get; set; }

  [Category("Contact")]
  [DisplayName("e-Mail address")]
  public string EMail { get; set; }
```

```
  [Category("Contact")]
  public string Address { get; set; }

  [Category("History")]
  [DisplayName("Customer Since")]
  public DateTime CustomerSince { get; set; }

  [Category("Contact")]
  [DisplayName("Send Newsletter")]
  public bool SendNewsletter { get; set; }

  [Category("History")]
  [DisplayName("Point Balance")]
  public int? PointBalance { get; set; }
}
```

And here is the result of this change:



Notice how properties are neatly grouped by category. Each group can be expanded and collapsed, making it easier for the user to find specific properties.

You can also use the DefaultCategoryName property to set the name of a default category which will contain all the properties that have no other category defined.

Displaying Methods and Properties

When the value of the **SelectedObject** property changes, the **C1PropertyGrid** updates itself based on the settings of three properties:

Property	Setting
PropertyAttributes	If this property is set to a non-empty collection, then the collection is used to generate the UI. Only the properties or methods included in the collection are displayed as long as the AutoGenerateProperties property is set to **False**. Otherwise all the properties will be shown and this collection will be used to override the default display of the properties or methods.
AutoGenerateProperties	If the **PropertyAttributes** collection is empty and this property is set to **True** (the default), then the C1PropertyGrid automatically shows all public properties of the **SelectedObject**.
AutoGenerateMethods	If the **PropertyAttributes** collection is empty and this property is set to **True**, then the **C1PropertyGrid** automatically shows all public methods of the **SelectedObject** that take no parameters. Methods are shown as buttons which can be clicked to invoke the method.

For example, the following code would cause the **C1PropertyGrid** control to show one group with two entries in it for customer name and e-mail address:

- Visual Basic

```
' Create C1PropertyGrid
Dim pg = New C1PropertyGrid()
LayoutRoot.Children.Add(pg)

' Customize the C1PropertyGrid layout
pg.LabelWidth = 100

' Show customer properties
pg.SelectedObject = customer

' Customize what is shown to the user
pg.AutoGenerateProperties = False
pg.PropertyAttributes.Add(New PropertyAttribute())
pg.PropertyAttributes.Add(New PropertyAttribute())
```

- C#

```
// Create C1PropertyGrid
var pg = new C1PropertyGrid();
LayoutRoot.Children.Add(pg);

// Customize the C1PropertyGrid layout
pg.LabelWidth = 100;

// Show customer properties
pg.SelectedObject = customer;

// Customize what is shown to the user
pg.AutoGenerateProperties = false;
pg.PropertyAttributes.Add(new PropertyAttribute()
  {
    MemberName = "Name",
    DisplayName = "Customer Name",
    Category = "Contact"
  }
);
pg.PropertyAttributes.Add(new PropertyAttribute()
  {
    MemberName = "EMail",
    DisplayName = "e-Mail Address",
    Category = "Contact"
  }
);
```

You can use this method to customize the display of objects that you have no access to. For example, if your **Customer** class were defined in a third-party assembly, you would not be able to add attributes to its members but would still be able to customize how the object is shown in the **C1PropertyGrid**.

Note that the **MemberName** must match the exact name of a property or method; otherwise the entry will be ignored.

Customizing the Editors

The **C1PropertyGrid** control has a set of built-in editors which support all common data types: **string**, **numeric**, **bool**, **Enum**, **Color**, **Brush**, **Image**, and so on.

The most suitable editor is automatically selected depending on the type of the property. When no suitable editor is found for the current property type, the **string** editor is used by default.

The list of editors is exposed by the **AvailableEditors** property of the **C1PropertyGrid** class. You can add your own custom editors to this list. The next section, Creating Custom Editors (page 250), explains how you can implement your own custom editors.

For each property, the **C1PropertyGrid** control checks the list of available editors and selects the first one that supports the current property type. This allows you to specify your own editor for all properties of a given type. Simply add your editor to the start of the **AvailableEditors** list and it will be used for all suitable properties. For example:

- Visual Basic
```
Dim pg = New C1PropertyGrid()
pg.AvailableEditors.Insert(0, New DateTimeEditor())
```
- C#
```
var pg = new C1PropertyGrid();
pg.AvailableEditors.Insert(0, new DateTimeEditor());
```

If you want to use a custom editor only for specific properties and not for all properties of a certain type you can either add an "Editor" attribute to the object being edited or use the **PropertyAttributes** property described above.

Creating Custom Editors

If the built-in editors do not fit your needs, you can easily create your own editors and use them with the **C1PropertyGrid**.

The following simple steps are required:

1. Create a class that implements the **ITypeEditorControl** interface.
2. Add an instance of this class to the **AvailableEditors** collection on the **C1PropertyGrid** control.

   OR

   Specify this class as the editor for a specific property by adding an **EditorAttribute** to the property definition.

The **ITypeEditorControl** interface contains the following members:

- **bool Supports(PropertyAttribute Property)**
  This method is used by the **C1PropertyGrid** to determine whether the editor supports the type of a given property.
- **void Attach(PropertyAttribute property)**
  This method is called when initializing the editor with the property it will manage. This typically consists of initializing the editor content based on the current property value.
- **void Detach(PropertyAttribute property)**
  This method is called when the editor instance is being released.
- **ITypeEditorControl Create()**
  This method is called when the **C1PropertyGrid** needs a new instance of the editor.
- **event PropertyChangedEventHandler ValueChanged**
  This event fires when the value of the property changes. This is used by editors that perform validation.

For a complete implementation of a custom editor, please refer to the **ControlExplorer** samples installed with **ComponentOne Studio for Silverlight**.

Reflector

ComponentOne Reflector™ for Silverlight allows you to display a 2D or 3D reflection of text or a UI element. Use the included opacity effect or any of the standard Silverlight effects, such as blur and drop shadow, to alter the look of your reflection.

Getting Started

- Working with the C1Reflector Control (page 255)- Quick Start (page 251)

- Task-Based Help (page 260)

Reflector for Silverlight Key Features

ComponentOne Reflector for Silverlight allows you to create customized, rich applications. Make the most of **Reflector for Silverlight** by taking advantage of the following key features:

- **3D Reflection**

  Reflector for Silverlight supports Silverlight 3 plane projections. To generate a reflection, add an effect to the **ReflectionEffects** collection. **C1Reflector** ships with an opacity effect, but you can also use the standard Silverlight effects such as blur and drop-shadow to apply to the reflection.

- **Use Any UI Element**

  Reflector for Silverlight supports any UIElement as content. To add multiple UIElements you would add one outer container element such as a grid and fill it with many sub elements.

- **Customize the Look of Reflections**

  Apply and configure opacity and blur effects. With opacity, for instance, you can configure how fast the opacity grows across the content and its starting value.

- **Auto-Update**

  The reflection is automatically updated when the content changes.

Reflector for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **Reflector for Silverlight**. In this quick start, you'll start in Visual Studio and create a new project, add a C1Reflector control to your application, add content to the C1Reflector control, and then customize the appearance of the C1Reflector control and its contents.

Step 1 of 4: Creating an Application with a C1Reflector Control

In this step, you'll begin in Visual Studio to create a Silverlight application using **Reflector for Silverlight**.

Complete the following steps:

1. In Visual Studio 2008, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **Silverlight Application**. Enter a **Name** for your project and click **OK**. The **New Silverlight Application** dialog box will appear.
3. Click **OK** to close the **New Silverlight Application** dialog box and create your project.

4. In the XAML window of the project, resize the **UserControl** by changing `DesignWidth="400" DesignHeight="300"` to `DesignWidth="Auto" DesignHeight="Auto"` in the `<UserControl>` tag so that it appears similar to the following:

```
<UserControl x:Class="SilverlightApplication24.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:d=http://schemas.microsoft.com/expression/blend/2008
   xmlns:mc="http://schemas.openxmlformats.org/markup-
   compatibility/2006"
   mc:Ignorable="d" d:DesignWidth="Auto" d:DesignHeight="Auto">
```

   The **UserControl** will now resize to accommodate any content placed within it.

5. In the XAML window of the project, place the cursor between the `<Grid>` and `</Grid>` tags and click once. Note that you cannot currently add Silverlight controls directly to the design area in Visual Studio, so you must add them to the XAML window as directed in the next step.
6. Navigate to the Toolbox and double-click the C1Reflector icon to add the control to the grid. The XAML resembles the following:

```
<Grid x:Name="LayoutRoot">
<c1:C1Reflector></c1:C1Reflector>
</Grid>
```

   If you ran the project now, you'd see a blank page because you still need to add content to the C1Reflector control. You'll learn how to do this in the next step.

You've successfully created a Silverlight application containing a C1Reflector control. In the next step, you will add two controls to the C1Reflector control.

Step 2 of 4: Adding Content to the C1Reflector Control

In this section of the quick start tutorial, you will add two controls to the C1Reflector control; however, the **Content** property can only accept one control at a time. You can circumvent this issue by adding a panel-based control as the C1Reflector control's content and adding then stacking multiple controls in the panel. The C1Reflector control will create a reflection for all controls added to a panel.

Complete the following steps:

1. Place your cursor between the `<c1:C1Reflector>` and `</c1:C1Reflector>` tags and press ENTER.
2. Navigate to the Toolbox and double-click the **StackPanel** icon to add the panel to **MainPage.xaml**.
3. Place your cursor between the `<StackPanel>` and `</StackPanel>` tags and press ENTER.
4. Navigate to the Toolbox and double-click the **C1DateTimePicker** icon to add the **C1DateTimePicker** control to the **StackPanel** control. Since the **C1DateTimePicker** control is interactive, it will allow you to see the C1Reflector control's auto-update feature.
5. Add `Width="190"` to the `</c1datetime:C1DateTimePicker>` tag to set the width of the control.
6. Place your cursor after the `</c1datetime:C1DateTimePicker>` tag and press ENTER.
7. Navigate to the Toolbox and double-click the **TextBox** icon to add a **TextBox** control beneath the **C1DateTimePicker** control.
8. Add `Text="Change the date/time and watch the reflection change"` to the `<TextBox>` tag to add text to the control

9. Add `FontSize=18` to the `<TextBox>` tag to change the size of the font.

You have successfully added custom content to the C1Reflector control. In the next step, you will utilize Blend to modify the appearance of the C1Reflector control.

Step 3 of 4: Customizing the C1Reflector Control

In the first two steps of this quick start, you created a Silverlight project, added a C1Reflector control to it, and then added content to the C1Reflector control. At this point, you have a functional project; if you run it, you'll see a two-dimensional, solid reflection. But you're not done yet: there are still properties you can set to make the project look even better. In this step, you'll use Expression Blend to add blur and opacity effects to the reflection and then place the control on a three-dimensional plane.

Complete the following steps:

1. In Solution Explorer, right click **MainPage.xaml** to open its context menu and then select **Open in Expression Blend**.

   The project opens in Expression Blend.
2. Under the **Objects and Timeline** tab, select **C1Reflector** to open its properties in the **Properties** panel.
3. Add the opacity and blur effects to the reflection by completing the following steps:
   a. Click the ReflectionEffects ellipsis button to open the **Effect Collection Editor: ReflectionEffects** dialog box.
   b. Click **Add another item**.

      The **Select Object** dialog box opens.
   c. Select **ReflectionOpacityEffect** from the list and then click **OK to** add the effect to the control and return to the **Effect Collection Editor: ReflectionEffects** dialog box.
   d. In the **Properties** grid, set the following properties:
      - Set the Coefficient property to "1".
      - Set the Offset property to "0.5".
   e. Click Add another item.

      The **Select Object** dialog box opens.
   f. Select **BlurEffect** from the list and then click **OK to** add the effect to the control and return to the **Effect Collection Editor: ReflectionEffects** dialog box.
   g. In the **Properties** grid, set the **Radius** property to "2". This low number will provide a soft blur around the edges of the control.
   h. Press **OK** to close the **Effect Collection Editor: ReflectionEffects** dialog box.
4. To alter the C1Reflector control's plane, locate the **ContentProjection** property and, under the **Rotation** tab, complete the following:
   a. In the X text box, enter "25".
   b. In the Y text box, enter "45".
   c. In the Z text box, enter "1".

In this step, you set properties that altered the appearance of the C1Reflector control and its contents. In the next step, you will run the project.

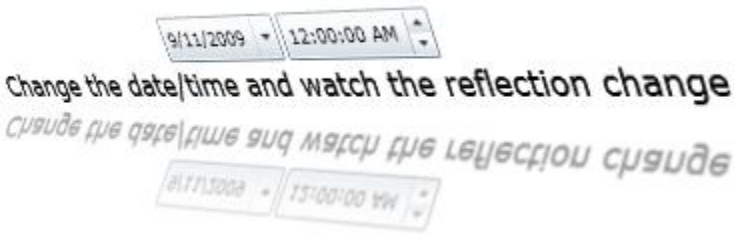
Step 4 of 4: Running the Project

In first three steps of this quick start, you created a Silverlight application containing a C1Reflector control, added content to the C1Reflector control, and modified the appearance of the C1Reflector control. In this step, you will run the project and observe the results of each step.

Complete the following steps:

1. Press F5 to run your Silverlight project.

   The application appears similar to the following:



2. Click the date drop down and select a different date from the calendar. For this example, select September 22, 2009. Observe that the reflection of the date changes to 9/22/09. This is the result of the C1Reflector control's automatic updating feature.
3. Use the time picker's arrow keys to change the time. As you scroll through the minutes and hours, observe that there is a real-time reflection of each change. This is another result of the C1Reflector control's automatic updating feature.

Congratulations! You have successfully completed the **Reflector for Silverlight** quick start. In this quick start, you've created a **Reflector for Silverlight** application, added content to the C1Reflector control, customized the C1Reflector control's appearance, and viewed some of the run-time capabilities of the C1Reflector control.

Reflector XAML Quick Reference

This topic is dedicated to providing a quick overview of the XAML used to create a C1Reflector control. For more information, see the Reflector for Silverlight Task-Based Help (page 260) section.

The XAML markup below creates a reflector with one item, a button control, and applies a blur effect and a drop-shadow effect to the control.

```
<c1:C1Reflector Width="400">

   <Button Content="Button"></Button>
<c1:C1Reflector.ReflectionEffects>
   <BlurEffect Radius="3"/>
   <DropShadowEffect BlurRadius="7 Opacity="0.95" ShadowDepth="8"
   Direction="180"/>
   </c1:C1Reflector.ReflectionEffects>
</c1:C1Reflector>
```

Working with the C1Reflector Control

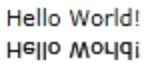
ComponentOne Reflector for Silverlight includes one content control, C1Reflector, that can be used to create 2D and 3D reflections of text and Silverlight controls. When it is added to the project, you will have to add the content that you want to have reflected to it. You can do this in Blend using a simple drag-and-drop operation, in XAML, or in code. This section covers the basics of the C1Reflector control.

Reflector Content

The C1Reflector control can hold text content, Silverlight controls, and DataTemplates as its content. The C1Reflector control will cast a reflection of the content.

Text Content

If you just wanted to reflect a short phrase, you would just set the **Content** property to a string. It would look similar to this:



The example above is a little plain, but you can spruce up the text by setting just a few properties. You can change the font to a built-in Silverlight font (or add your own font to the project), change the font color, and customize the font size from the **Properties** panel in Blend. The image below uses a 36 point Curlz MT font in red.

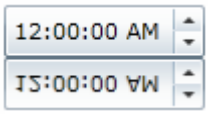
For task-based help about how to add text content, see Adding Simple Text Content to the Reflector (page 260).

Control Content

Adding control content is just as simple. You can add controls to the C1Reflector control in Blend by selecting a control icon and then using a drag-and-drop operation to add it to the C1Reflector control 's container. If you prefer to use XAML, you just wrap the `<c1:C1Reflector>` tags around the control so that your markup looks as follows:

```
<c1:C1Reflector Width="400">
      <c1datetime:C1TimeEditor Width="400"/>
</c1:C1Reflector>
```

The above XAML markup creates the following:



For task-based help about how to use controls with the C1Reflector control, see Adding a Control to the Reflector (page 261).

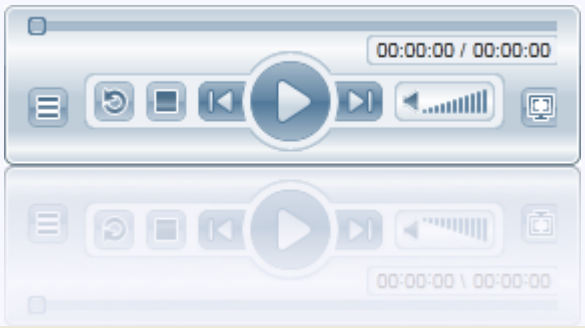
Adding Multiple Controls

Content property can only accept one control at a time. However, you can get around this limitation by adding a panel-based control as the C1Reflector control's child element. Panel-based controls, such as a **StackPanel** control, are able to hold multiple elements. The panel-based control meets the one control limitation of the C1Reflector control, but its ability to hold multiple elements will allow you to show several controls in the content area at the same time.

Reflection Effects

The C1Reflector control comes with a ReflectionOpacityEffect effect that you can use to alter the reflection of an item. Setting the Coefficient property will set the coefficient applied to the function that computes the effect's opacity. Setting the Offset property sets the offset applied to the function that computes opacity.

You can also use standard Silverlight effects, such as the **BlurEffect** effect and the **DropShadowEffect** effect, to alter the look of the reflection.The table below shows a sample of each effect.

Effect	Sample Image
ReflectionOpacityEffect	
BlurEffect	
DropShadowEffect	

The ReflectionEffects property can get a collection of effects, which allows you to use more than one effect at a time. If you, for example, wanted to give the illusion that your content was throwing a reflection onto glazed glass rather than clear glass, you could experiment with using both the **BlurEffect** and the ReflectionOpacityEffect.

For task-based help on adding reflection effects, see Using the Drop Shadow Effect (page 262), Using the Blur Effect (page 264), and Using the Opacity Effect (page 265).

Automatic Updating

The C1Reflector control will, by default, automatically update the reflection each time the control's content is updated. This is useful when you're using controls with interactive parts, such as the **C1Reflector** control or the **C1Accordion** control. The automatic update ensures that every movement of these controls will be shown in the reflection, so that if a user expands an accordion pane, for example, the accordion pane in the reflection will also be expanded.

If you'd prefer not to have your content automatically updated, you can turn the AutoUpdate property to **False**.

Note: AutoUpdate can slow down the application, especially if you intend to use multiple reflectors.

Plane Projection

Reflector for Silverlight supports plane projections, which allows a two-dimensional control to be drawn on a three-dimensional plane. All parts of the control will still function as they would on a two-dimensional plane.

The three-dimensional effect is created by rotating the control along three separate planes – X, Y, and Z. You can set the rotation, center of rotation, global offset, and local offset of each of the three planes; this means that, in total, there are twelve properties that can be set to alter the projection of the C1Reflector control. Each property is described in the table below.

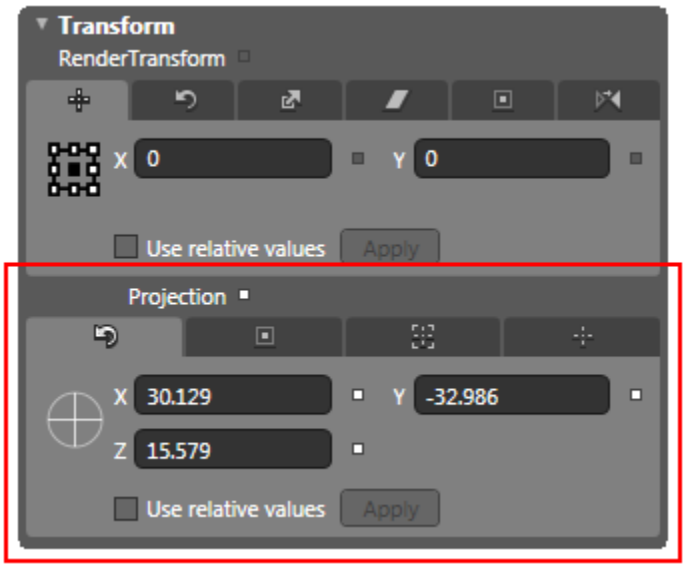
Property	Description
RotationX	Gets or sets the number of degrees to rotate the object around the x-axis of rotation.
RotationY	Gets or sets the number of degrees to rotate the object around the y-axis of rotation.
RotationZ	Gets or sets the number of degrees to rotate the object around the z-axis of rotation.
CenterOfRotationX	Gets or sets the x-coordinate of the center of rotation of the object you rotate.
CenterOfRotationY	Gets or sets the y-coordinate of the center of rotation of the object you rotate.
CenterOfRotationZ	Gets or sets the z-coordinate of the center of rotation of the object you rotate.
GlobalOffsetX	Gets or sets the distance the object is translated along the x-axis of the screen.
GlobalOffsetY	Gets or sets the distance the object is translated along the y-axis of the screen.
GlobalOffsetZ	Gets or sets the distance the object is translated along the z-axis of the screen.
LocalOffsetX	Gets or sets the distance the object is translated along the x-axis of the plane of the object.
LocalOffsetY	Gets or sets the distance the object is translated along the y-axis of the plane of the object.
LocalOffsetZ	Gets or sets the distance the object is translated along the z-axis of the plane of the object.

You can alter the projection of the entire C1Reflector control or you can just alter the projection of the control's content. The results will be slightly different. If you alter the projection of the entire control, the project of both the content of the reflection will change. If you alter the projection of the content, only the projection of the content

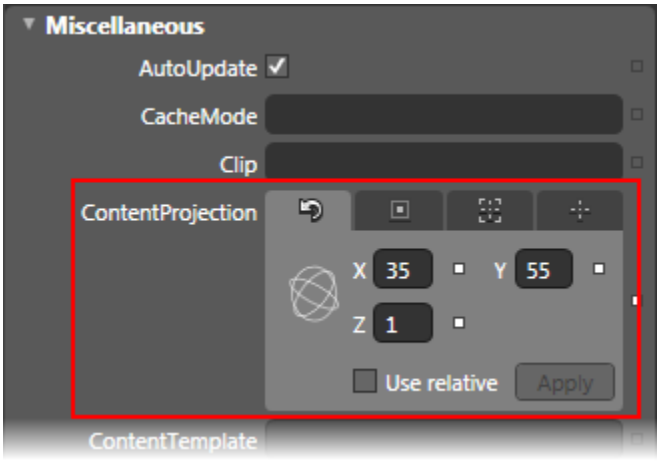
will change and then the reflection will mirror that change. The table below shows the difference between control projection and content projection.

Projection Settings	Control Projection	Content Projection
RotationX = 35 RotationY = 55 RotationZ = 1		

You can set the control or content projection in XAML or in code, but it's easier to adjust the settings in Blend until you have achieved the look that you want. The control projection properties are located under the **Transform** section underneath the **Projection** separator.



The control projection properties are located under the **Miscellaneous** section next to the **ContentProjection** property.



Reflector for Silverlight Layout and Appearance

The following topics detail how to customize the C1Reflector control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the grid and take advantage of Silverlight's XAML-based styling. You can also use templates to format and layout the control and to customize the control's actions.

Text Properties

The following properties let you customize the appearance of text in the C1Reflector control.

Property	Description
FontFamily	Gets or sets the font family of the control. This is a dependency property.
FontSize	Gets or sets the font size. This is a dependency property.
FontStretch	Gets or sets the degree to which a font is condensed or expanded on the screen. This is a dependency property.
FontStyle	Gets or sets the font style. This is a dependency property.
FontWeight	Gets or sets the weight or thickness of the specified font. This is a dependency property.

Color Properties

The following properties let you customize the colors used in the control itself.

Property	Description
Background	Gets or sets a brush that describes the background of a control. This is a dependency property.
Foreground	Gets or sets a brush that describes the foreground color. This is a dependency property.

Border Properties

The following properties let you customize the control's border.

Property	Description
BorderBrush	Gets or sets a brush that describes the border background of a control. This is a dependency property.
BorderThickness	Gets or sets the border thickness of a control. This is a dependency property.

Size Properties

The following properties let you customize the size of the **C1Reflector** control.

Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
MinHeight	Gets or sets the minimum height constraint of the element. This is a dependency property.
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

Reflector for Silverlight Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1Reflector control in general. If you are unfamiliar with the **ComponentOne Reflector for Silverlight** product, please see the **Reflector for Silverlight** Quick Start first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne Reflector for Silverlight** product.

Each task-based help topic also assumes that you have created a new Silverlight project.

Adding Simple Text Content to the Reflector

You can add simple text to the C1Reflector control by setting the **Content** property to a string. Once the text is added, you will see the text and its reflection and the C1Reflector control.

At Design Time in Blend

To add simple text to the control, complete the following steps:

1. Select the C1Reflector control once to select it.
2. Under the **Properties** panel, set the **Content** property to "Hello World!".

In XAML

To add simple text to the control, add `Content ="Hello World"` to the `<c1:C1Reflector>` tag so that the markup resembles the following:

```
<c1:C1Reflector Content="Hello World!">
```

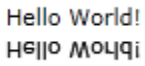
In Code

To add simple text to the control, complete the following steps:

1. Add `x:Name=C1Reflector1` to the `<c1:C1Reflector>` tag. This will give the control a unique identifier that you can use to call it in code.
2. Switch to Code view and add the following beneath the **InitializeComponent()** method:
   - Visual Basic
     ```
     C1Reflector1.Content = "Hello World!"
     ```
   - C#
     ```
     C1Reflector1.Content = "Hello World!";
     ```
3. Run the program

This Topic Illustrates the Following:

The result of this topic will resemble the following image:



Adding a Control to the Reflector

The C1Reflector control can accept Silverlight UI controls. To illustrate this feature, this topic will have you add a standard Silverlight Button to the reflector. Once it is added to the control, you will see the control and its reflection in the C1Reflector control.

At Design Time in Blend

To add a control, complete the following steps:

1. Under the **Objects and Timeline** tab, select **All** for a list of all available controls.
2. In the list of controls, select the **Button** icon and then use a drag-and-drop operation to add it to the C1Reflector container.

In XAML

To add a control, place the following markup between the `<c1:C1Reflector>` and `</c1:C1Reflector>` tags:

```
<Button Content="Button"></Button>
```

In Code

To add a control, complete the following steps:

1. Add `x:Name=C1Reflector1` to the `<c1:C1Reflector>` tag. This will give the control a unique identifier that you can use to call it in code.
2. Switch to Code view and add the following beneath the **InitializeComponent()** method:
   - Visual Basic

```
Dim Button1 As New Button()
Button1.Content = "Button"
C1Reflector1.Content = Button1
```

- C#

```
Button Button1 = new Button();
Button1.Content = "Button";
C1Reflector1.Content = Button1;
```

3. Run the program

This Topic Illustrates the Following:

The result of this topic will resemble the following image:

Using the Drop Shadow Effect

You can add a drop shadow to a reflection using the standard Silverlight drop shadow effect. In this topic, you will add the drop shadow effect in Blend, in XAML, and in code.

At Design Time in Blend

To use the drop shadow effect, complete the following steps:

1. Add a C1Reflector control to your Blend project.
2. Select the C1Reflector control once to select it.
3. Under the **Properties** panel, click the ReflectionEffects ellipsis button to open the **Effect Collection Editor: ReflectionEffects** dialog box.
4. Click **Add another item**.

   The **Select Object** dialog box opens.
5. Select **DropShadowEffect** from the list and then click **OK to** add the effect to the control and return to the **Effect Collection Editor: ReflectionEffects** dialog box.
6. In the **Properties** grid, set the following properties:
   - Set the **BlurRadius** property to "7".
   - Set the **Direction** property to "180".
   - Set the **Opacity** property to "95%".
   - Set the **ShadowDepth** to "8".
7. Press **OK** to close the **Effect Collection Editor: ReflectionEffects** dialog box.

At Design Time in Blend

To use the drop shadow effect, complete the following steps:

1. Add a C1Reflector control to your Blend project.
2. Add `Content="C1Reflector"` to the `<c1:C1Reflector>` tag to set string content.
3. Add the drop shadow effect and set its properties by placing the following XAML between the `<c1:C1Reflector>` and `</c1:C1Reflector>` tags:

```
<c1:C1Reflector.ReflectionEffects>
      <DropShadowEffect BlurRadius="7 Opacity="0.95" ShadowDepth="8"
      Direction="180"/>
</c1:C1Reflector.ReflectionEffects>
```

In Code

To use the drop shadow effect, complete the following steps:

1. Add a C1Reflector control to your Blend project.
2. Add `Content="C1Reflector"` to the `<c1:C1Reflector>` tag to set string content.
3. Add `x:Name=C1Reflector1` to the `<c1:C1Reflector>` tag. This will give the control a unique identifier that you can use to call it in code.
4. Switch to Code view and import the following namespace:
   - Visual Basic

     ```
     Imports System.Windows.Media.Effects;
     ```

   - C#

     ```
     using System.Windows.Media.Effects;
     ```

5. Add the following beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     'Create the DropShadowEffect object and set its properties
     Dim newDropShadowEffect As New DropShadowEffect()
     newDropShadowEffect.BlurRadius = 7
     newDropShadowEffect.Direction = 180
     newDropShadowEffect.Opacity = 95
     newDropShadowEffect.ShadowDepth = 8
     'Add the DropShadowEffect object to the C1Reflector control
     C1Reflector1.ReflectionEffects.Add(newDropShadowEffect)
     ```

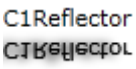
   - C#

     ```
     //Create the DropShadowEffect object and set its properties
     DropShadowEffect newDropShadowEffect = new DropShadowEffect();
     newDropShadowEffect.BlurRadius = 7;
     newDropShadowEffect.Direction = 180;
     newDropShadowEffect.Opacity = 95;
     newDropShadowEffect.ShadowDepth = 8;
     //Add the DropShadowEffect object to the C1Reflector control
     C1Reflector1.ReflectionEffects.Add(newDropShadowEffect);
     ```

6. Run the program

This Topic Illustrates the Following:

When you run the project, the C1Reflector control and its content will resemble the following image:



Using the Blur Effect

You can add a blur effect to a reflection using the standard Silverlight drop shadow effect. In this topic, you will add the drop shadow effect in Blend, in XAML, and in code.

At Design Time in Blend

To use the blur effect, complete the following steps:

1. Add a C1Reflector control to your Blend project.
2. Select the C1Reflector control once to select it.
3. Under the **Properties** panel, click the ReflectionEffects ellipsis button to open the **Effect Collection Editor: ReflectionEffects** dialog box.
4. Click **Add another item**.

   The **Select Object** dialog box opens.
5. Select **BlurEffect** from the list and then click **OK to** add the effect to the control and return to the **Effect Collection Editor: ReflectionEffects** dialog box.
6. In the **Properties** grid, set the **Radius** property to "3".
7. Press **OK** to close the **Effect Collection Editor: ReflectionEffects** dialog box.

In XAML

To use the blur effect, complete the following steps:

1. Add a C1Reflector control to your Blend project.
2. Add `Content="C1Reflector"` to the `<c1:C1Reflector>` tag to set string content.
3. Add the blur effect and set its properties by placing the following XAML between the `<c1:C1Reflector>` and `</c1:C1Reflector>` tags:

```
<c1:C1Reflector.ReflectionEffects>
      <BlurEffect Radius="3"/>
</c1:C1Reflector.ReflectionEffects>
```

In Code

To use the blur effect, complete the following steps:

1. Add a C1Reflector control to your Blend project.
2. Add `Content="C1Reflector"` to the `<c1:C1Reflector>` tag to set string content.
3. Add `x:Name=C1Reflector1` to the `<c1:C1Reflector>` tag. This will give the control a unique identifier that you can use to call it in code.
4. Switch to Code view and import the following namespace:
   - Visual Basic

```
Imports System.Windows.Media.Effects;
```

- C#

```
using System.Windows.Media.Effects;
```

5. Add the following beneath the **InitializeComponent()** method:

- Visual Basic

```
'Create the BlurEffect object and set its Radius property
Dim newBlurEffect As New BlurEffect() newBlurEffect.Radius = 3
'Add the BlurEffect object to the C1Reflector control
C1Reflector1.ReflectionEffects.Add(newBlur
```

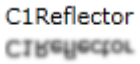
- C#

```
//Create the BlurEffect object and set its Radius property
BlurEffect newBlurEffect = new BlurEffect();
newBlurEffect.Radius = 3;
//Add the BlurEffect object to the C1Reflector control
C1Reflector1.ReflectionEffects.Add(newBlurEffect);
```

6. Run the program

This Topic Illustrates the Following:

When you run the project, the C1Reflector control and its content will resemble the following image:



Using the Opacity Effect

You can add an opacity effect to a reflection using the standard Silverlight drop shadow effect. In this topic, you will add the drop shadow effect in Blend, in XAML, and in code.

At Design Time in Blend

To use the opacity effect, complete the following steps:

1. Add a C1Reflector control to your Blend project.
2. Select the C1Reflector control once to select it.
3. Under the **Properties** panel, click the ReflectionEffects ellipsis button to open the **Effect Collection Editor: ReflectionEffects** dialog box.
4. Click **Add another item**.

   The **Select Object** dialog box opens.
5. Select **ReflectionOpacityEffect** from the list and then click **OK to** add the effect to the control and return to the **Effect Collection Editor: ReflectionEffects** dialog box.
6. In the **Properties** grid, set the following properties:

   - Set the Coefficient property to "1".

- Set the Offset property to "0.5".

7. Press **OK** to close the **Effect Collection Editor: ReflectionEffects** dialog box.

At Design Time in Blend

To use the opacity effect, complete the following steps:

1. Add a C1Reflector control to your Blend project.
2. Add `Content="C1Reflector"` to the `<c1:C1Reflector>` tag to set string content.
3. Add the opacity effect and set its properties by placing the following XAML between the `<c1:C1Reflector>` and `</c1:C1Reflector>` tags:

```
<c1:C1Reflector.ReflectionEffects>
      <c1:ReflectionOpacityEffect Coefficient="1" Offset="0.5"/>
</c1:C1Reflector.ReflectionEffects>
```

In Code

To use the opacity effect, complete the following steps:

1. Add a C1Reflector control to your Blend project.
2. Add `Content="C1Reflector"` to the `<c1:C1Reflector>` tag to set string content.
3. Add `x:Name=C1Reflector1` to the `<c1:C1Reflector>` tag. This will give the control a unique identifier that you can use to call it in code.
4. Switch to Code view and import the following namespace:
   - Visual Basic

     ```
     Imports C1.Silverlight.Extended
     ```

   - C#

     ```
     using C1.Silverlight.Extended;
     ```

5. Add the following beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     'Create the ReflectionOpacityEffect object and set its properties
     Dim newOpacity As New ReflectionOpacityEffect()
     newOpacity.Coefficient = 1
     newOpacity.Offset = 0.5
     'Add the ReflectionOpacityEffect object to the C1Reflector1 control
     C1Reflector1.ReflectionEffects.Add(newOpacity)
     ```

   - C#

     ```
     //Create the ReflectionOpacityEffect object and set its properties
     ReflectionOpacityEffect newOpacity = new ReflectionOpacityEffect();
     newOpacity.Coefficient = 1;
     newOpacity.Offset = 0.5;
     //Add the ReflectionOpacityEffect object to the C1Reflector1 control
     ```

```
C1Reflector1.ReflectionEffects.Add(newOpacity);
```

6. Run the program

This Topic Illustrates the Following:

When you run the project, the C1Reflector control and its content will resemble the following image: