ComponentOne

Studio for Silverlight Basic Library

ComponentOne, a division of GrapeCity

201 South Highland Avenue, Third 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 Basic Library Overview

The future of Web development tools is here! **ComponentOne Studio for Silverlight Basic Library** features industrial strength Silverlight controls you cannot find anywhere else**. C1.Silverlight** is the main **Studio for Silverlight** assembly and it contains several basic controls and utility classes that can be used by themselves and are often used by other **Studio for Silverlight** controls.

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

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.

ComboBox

ComponentOne ComboBox™ for Silverlight is a full-featured combo box control that combines an editable text box with an auto-searchable drop-down list.

Getting Started

- Working with the C1ComboBox Control
- Quick Start
- Task-Based Help

ComboBox for Silverlight Key Features

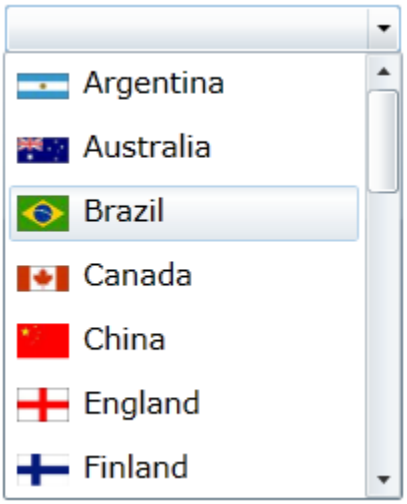
ComponentOne ComboBox for Silverlight allows you to create customized, rich applications. Make the most of **ComboBox for Silverlight** by taking advantage of the following key features:

- **Auto-searchable Drop-down List**

  Locate items quickly by typing the first few characters. ComboBox will automatically search the list and select the items for you as you type.

- **Populate the Drop-down List with Data Templates**

  ComboBox fully supports data templates, making it easy to add any visual elements to the list items. This includes text, images, and any other controls. The control uses element virtualization, so it always loads quickly, even when populated with hundreds of items.



- **Time-tested, Familiar Object Model**

ComboBox has a rich object model based on the WPF ComboBox control. You can easily specify whether the end user is able to enter items that are not on the drop-down list, get or set the index of the selected item, the height of the drop-down list, and more.

ComboBox for Silverlight Quick Start

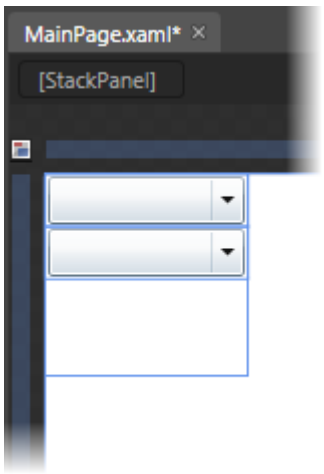
The following quick start guide is intended to get you up and running with **ComboBox for Silverlight**. In this quick start, you'll start in Expression Blend to create a new project with two C1ComboBox controls. The first control will be populated with a list of three items that, when clicked, will determine the list that appears in the second combo box.

Step 1 of 4: Creating an Application with a C1ComboBox Control

In this step, you'll begin in Expression Blend to create a Silverlight application using **ComboBox for Silverlight**.

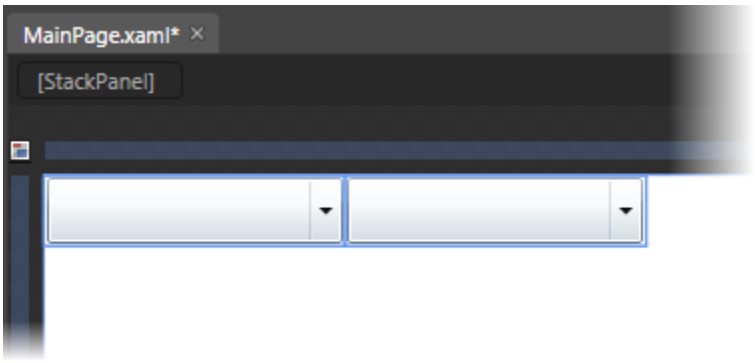
Complete the following steps:

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 two C1ComboBox controls to the project by completing the following steps:
   a. Click the **Assets** tab.
   b. In the search box, enter "StackPanel" to bring up the **StackPanel** icon.
   c. Double-click the **StackPanel** icon to add it to the project.
   d. Under the **Objects and Timeline** tab, select **StackPanel**.
   e. Navigate to the **Assets** tab and, in the search bar, enter "C1ComboBox". Double-click the C1ComboBox icon to add the control to the **StackPanel**.
   f. Repeat steps 4-d and 4-e to add another C1ComboBox to the **StackPanel.** The project resembles the following:



5. Under the **Objects and Timeline** tab, select **StackPanel** to reveal its list of properties under the **Properties** panel and then complete the following:
   - Set the **Width** property to "300".
   - Set the **Height** property to "35".
   - Set the **Orientation** property to **Horizontal**.
6. Under the **Objects and Timeline** tab, select the first **[C1ComboBox]** to reveal its list of properties under the **Properties** panel and then complete the following:
   - Set the **Width** property to "150".
   - Set the **Height** property to "35".
7. Under the **Objects and Timeline** tab, select the second **[C1ComboBox]** to reveal its list of properties under the **Properties** panel and then complete the following:
   - Set the **Width** property to "150".
   - Set the **Height** property to "35".

   The project resembles the following:



You have completed the first step of the quick start by creating a Silverlight project and adding two C1ComboBox controls to it. In the next step, you'll add items to the first C1ComboBox control.

Step 2 of 4: Adding Items to the First C1ComboBox Control

In the last step, you created a project and added two C1ComboBox controls to it. In this step, you will add three items to the first combo box.

Complete the following steps:

1. Under the **Objects and Timeline** tab, select the first **[C1ComboBox]**.
2. Click the **Assets** tab and, in the search bar, type "C1ComboBoxItem" to bring up the C1ComboBoxItem icon.
3. Double-click the C1ComboBoxItem icon to add one item to the drop-down list.
4. Repeat step 3 twice to add a total of three C1ComboBoxItem items to the control.
5. Under the **Objects and Timeline** tab, select the first **[C1ComboBoxItem]** to reveal its properties under the **Properties** panel and then set the following properties:
   - Set the **Content** property to "Comedy".
   - Set the **Height** property to "25".

- Set the **Width** property to **Auto** by clicking on the glyph.

6. Under the **Objects and Timeline** tab, select the second **[C1ComboBoxItem]** to reveal its properties under the **Properties** panel and then set the following properties:
   - Set the **Content** property to "Drama".
   - Set the **Height** property to "25".
   - Set the **Width** property to **Auto** by clicking on the glyph.
7. Under the **Objects and Timeline** tab, select the third **[C1ComboBoxItem]** to reveal its properties under the **Properties** panel and then set the following properties:
   - Set the **Content** property to "Science Fiction".
   - Set the **Height** property to "25".
   - Set the **Width** property to **Auto** by clicking on the glyph.

In this step, you added items to the first combo box. In the next step, you will add code to the project that will populate the second combo box with items when a user selects an item in the first combo box.

Step 3 of 4: Adding Code to the Control

In the last step, you added items to the first combo box. In this step, you will add code to the project that will populate the second combo box according to the option the user selects in the first combo box.

1. Switch to XAML view and complete the following:
   - Add `x:Name="Category"` to the first `<c1:C1ComboBox>` tag.
   - Add `x:Name="Shows"` to the second `<c1:C1ComboBox>` tag.
2. Select the first **C1ComboBox** ("Category").
3. In the **Properties** window, click the **Events** button.
4. Double-click the inside the **SelectedIndexChanged** text box to add the **C1ComboBox1_SelectedIndexChanged** event handler.

   The **MainPage.xaml.cs** page opens.
5. Import the following namespace into your project:
   - Visual Basic

     ```
     Imports System.Collections.Generic
     ```
   - C#

     ```
     using System.Collections.Generic;
     ```
6. Add the following code to the **C1ComboBox1_SelectedIndexChanged** event handler:
   - Visual Basic

     ```
     'Create List for Comedy selection
     Dim dropDownList_Comedy As New List(Of String)()
     dropDownList_Comedy.Add("Absolutely Fabulous")
     dropDownList_Comedy.Add("The Colbert Report")
     dropDownList_Comedy.Add("The Daily Show")
     dropDownList_Comedy.Add("The Office")
     ```

```
'Create List for Drama selection
Dim dropDownList_Drama As New List(Of String)()
dropDownList_Drama.Add("Breaking Bad")
dropDownList_Drama.Add("Desperate Housewives")
dropDownList_Drama.Add("Mad Men")
dropDownList_Drama.Add("The Sopranos")

'Create List for Science Fiction selection
Dim dropDownList_SciFi As New List(Of String)()
dropDownList_SciFi.Add("Battlestar Galactica")
dropDownList_SciFi.Add("Caprica")
dropDownList_SciFi.Add("Stargate")
dropDownList_SciFi.Add("Star Trek")

'Check for SelectedIndex value and assign appropriate list to 2nd combo
box
If Category.SelectedIndex = 0 Then
   Shows.ItemsSource = dropDownList_Comedy
ElseIf Category.SelectedIndex = 1 Then
   Shows.ItemsSource = dropDownList_Drama
ElseIf Category.SelectedIndex = 2 Then
   Shows.ItemsSource = dropDownList_SciFi
End If
```

- C#

```
//Create List for Comedy selection
List<string> dropDownList_Comedy = new List<string>();
dropDownList_Comedy.Add("Absolutely Fabulous");
dropDownList_Comedy.Add("The Colbert Report");
dropDownList_Comedy.Add("The Daily Show");
dropDownList_Comedy.Add("The Office");

//Create List for Drama selection
List<string> dropDownList_Drama = new List<string>();
dropDownList_Drama.Add("Breaking Bad");
dropDownList_Drama.Add("Desperate Housewives");
dropDownList_Drama.Add("Mad Men");
```

```
        dropDownList_Drama.Add("The Sopranos");

        //Create List for Science Fiction selection
        List<string> dropDownList_SciFi = new List<string>();
        dropDownList_SciFi.Add("Battlestar Galactica");
        dropDownList_SciFi.Add("Caprica");
        dropDownList_SciFi.Add("Stargate");
        dropDownList_SciFi.Add("Star Trek");

        //Check for SelectedIndex value and assign appropriate list to 2nd
combo box
        if (Category.SelectedIndex == 0)
        {
             Shows.ItemsSource = dropDownList_Comedy;
        }
        else if (Category.SelectedIndex == 1)
        {
             Shows.ItemsSource = dropDownList_Drama;
        }
        else if (Category.SelectedIndex ==2)
        {
             Shows.ItemsSource = dropDownList_SciFi;
        }
```

In the next step, you will run the project and observe the results of this quick start.

Step 4 of 4: Running the Project

In the previous three steps, you created a Silverlight project with two combo boxes, added items to the first combo box, and wrote code that will populate the second combo box with items once an item is selected in the first combo box. In this step, you will run the project and observe the results of this quick start.

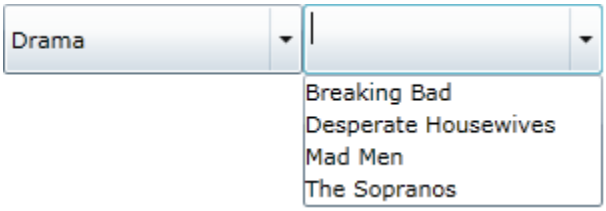
Complete the following steps:

1. Select **Project | Run Project** to run the project. The project loads with two blank combo boxes:

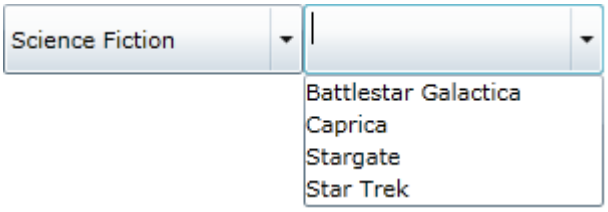
2. Click the second combo box's drop-down arrow and observe that the drop-down list is empty:

3. Click the first combo box's drop-down arrow and select **Comedy**.
4. Click the second combo box's drop-down arrow and observe that the drop-down list features the following items:

5. Click the first combo box's drop-down arrow and select **Drama**.
6. Click the second combo box's drop-down arrow and observe that the drop-down list features the following items:



7. Click the first combo box's drop-down arrow and select **Science Fiction**.
8. Click the second combo box's drop-down arrow and observe that the drop-down list features the following items:



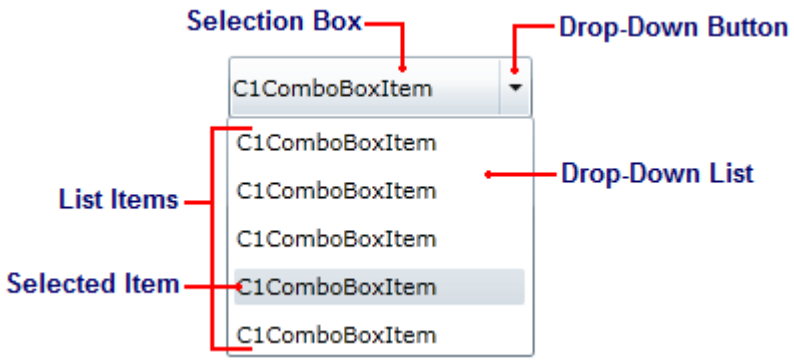
Congratulations! You have completed the **ComboBox for Silverlight** quick start.

Working with the C1ComboBox Control

This section provides an overview of C1ComboBox control basics. If you haven't used the control, we recommend starting with the ComboBox for Silverlight Quick Start topic.

C1ComboBox Elements

The C1ComboBox control is a flexible control used to display data in a drop-down list. It is essentially the combination of two controls: a text box that allows users to enter a selection, and a list box that allows users to select from a series of list options. The following image diagrams the C1ComboBox control.



See below for a description of each C1ComboBox element.

- **Selection Box**

  The selection box serves two purposes: it allows users to enter the list item they're searching for directly into the text box, and it displays the currently selected item. The content of this box is equal to the content of the C1ComboBox control's selected index item.

- **Drop-Down Button**

  The drop-down button reveals the drop-down list when clicked.

- **Drop-Down List**

  The drop-down list consists of a series of list items (see below); it can contain as little or as many list items as you need. If the number of items exceeds the size of the drop-down list, a scrollbar will automatically appear.

- **List Items**

  Each list item in a drop-down list is represented by the C1ComboBoxItem class. List items can contain text, pictures, and even controls.

- **Selected Item**

  The selected item in a list can be fixed by the developer or chosen by a user at run-time. The value of a selected list item's IsSelected property is **True**

C1ComboBox Features

The following topics detail a few of the C1ComboBox control's features. For more information on utilizing these features, see the ComboBox for Silverlight Task-Based Help section.

Drop-Down List Direction

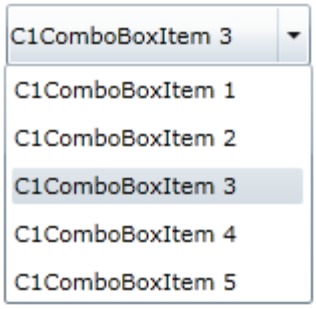
By default, when the user clicks the C1ComboBox control's drop-down arrow at run-time, the drop-down list will appear below the control; if that is not possible, it will appear above the control. You can, however, change the direction in which the drop-down list appears by setting the DropDownDirection property to one of the following four options:

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

For instructions about how to change the drop-down direction, see Changing the Drop-Down List Direction.

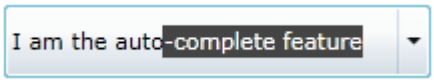
Item Selection

The SelectedIndex property determines which item is selected in a drop-down list. The SelectedIndex is based on a zero-based index, meaning that 0 represents the first C1ComboBoxItem, 1 represents the second C1ComboBoxItem, and so on. In the image below, the SelectedIndex is set to **2**, which selects the third C1ComboBoxItem.



AutoComplete

The C1ComboBox control features an auto-completion feature, which selects a list item based on user input. As the user types, the list item is loaded into the selection box, as seen in the following image:



The user only has to press ENTER to select the list item suggested by the AutoComplete feature.

The AutoComplete feature can be disabled by setting the AutoComplete property to **False**. To learn how to disable the feature at design time, in XAML, and in code, see Disabling AutoComplete.

Drop-Down List Sizing

By default, the size of the drop-down list is determined by the width of the widest C1ComboBoxItem item and the collective height of all of the C1ComboBoxItem items, as the DropDownWidth and DropDownHeight properties are both set to **NaN**.

You can control the maximum width and maximum height of the drop-down list by setting the C1ComboBox control's MaxDropDownWidth and MaxDropDownHeight properties. Setting these properties ensures that the area of the drop-down list can never expand to a larger area than you've specified. If the width or height of the list exceeds the specified maximum height and width, scrollbars will automatically be added to the drop-down list.

For task-based help on drop-down list sizing, see Setting the Maximum Height and Maximum Width of the Drop-Down List.

ComboBox for Silverlight Layout and Appearance

The following topics detail how to customize the C1ComboBox 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.

C1ComboBox and C1ComboBoxItem ClearStyle Properties

ComboBox 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 **C1ComboBox** control:

Brush	Description
Background	Gets or sets the brush of the control's background.
ButtonBackground	Gets or sets the brush of the drop-down button's background.
ButtonForeground	Gets or sets the brush of the drop-down button's foreground.
FocusBrush	Gets or sets the brush for the control when it has focus.
MouseOverBrush	Gets or sets the brush for the control when it is moused over.
PressedBrush	Gets or sets the brush for the control when it is pressed.
SelectedBackground	Gets or sets the brush of the background for the selected C1ComboBoxItem.

The following table outlines the brush properties of the **C1ComboBoxItem** control:

Brush	Description
Background	Gets or sets the brush of the control's background.

You can completely change the appearance of the **C1ComboBox** and **C1ComboBoxItem** controls by setting a few properties, such as the **C1ComboBox** control's **ButtonBackground** property, which sets the background color for the control's drop-down arrow. For example, if you set the **ButtonBackground** property to "#FFC500FF", each header in the **C1ComboBox** control would appear similar to the following:

It's that simple with ComponentOne's ClearStyle technology.

ComboBox for Silverlight Appearance Properties

ComponentOne ComboBox 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 combo box 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.
TextAlignment	Gets or sets how the text should be aligned in the drop-down list. This is a dependency property.

Content Positioning Properties

The following properties let you customize the position of header and content area content in the C1ComboBox 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.

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 **C1ComboBox** control.

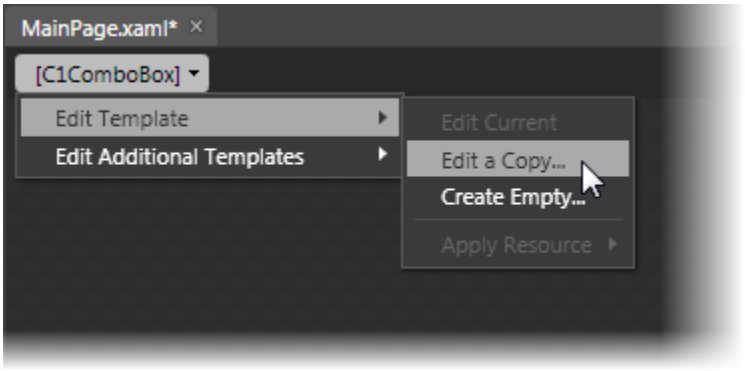
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
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.
DropDownHeight	Gets or sets the height of the dropdown (set to Double.NaN to size automatically).
DropDownWidth	Gets or sets the width of the drop-down list (set to Double.NaN to size automatically).
MaxDropDownHeight	Gets or sets maximum height constraint of the drop-down box.
MaxDropDownWidth	Gets or sets maximum width constraint of the drop-down box.

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 ComboBox 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 C1ComboBox 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 C1ComboBoxItem template, simply select the C1ComboBoxItem 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 ComboBox Templates

In addition to the default templates, the C1ComboBox control includes a few additional templates. These additional templates can also be accessed in Microsoft Expression Blend – in Blend select the C1Menu or C1ComboBox control and, in the menu, select **Edit Additional Templates**. Choose a template, and select **Create Empty.**

Item Templates

ComponentOne ComboBox for Silverlight's combo box 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 combo box. These templates include an **ItemTemplate**, an **ItemsPanel**, and an **ItemContainerStyle** template. You use the **ItemTemplate** to specify the visualization of the data objects, the **ItemsPanel** to define the panel that controls the layout of items, and the **ItemStyleContainer** to set the style of all container items.

Accessing Templates

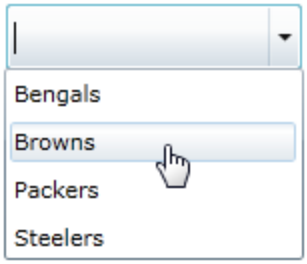
You can access these templates in Microsoft Expression Blend by selecting the C1ComboBox control and, in the menu, selecting **Edit Additional Templates**. Choose **Edit Generated Items (ItemTemplate)**, **Edit Layout of Items (ItemsPanel)**, or **Edit Generated Item Container (ItemStyleContainer)** 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.

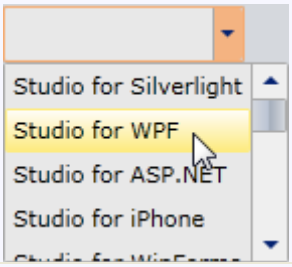
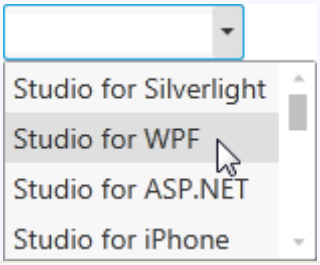
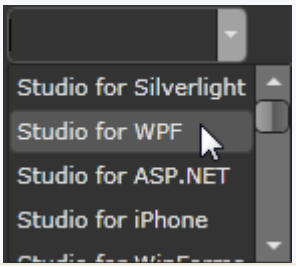
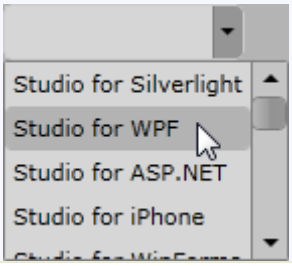
ComboBox Theming

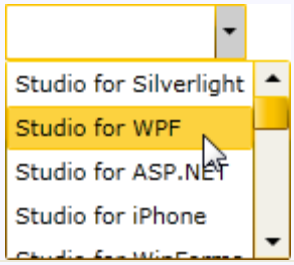
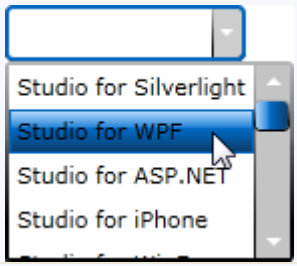
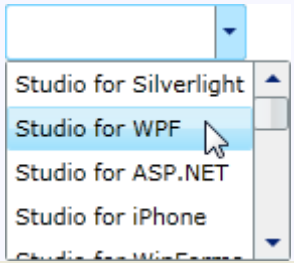
Silverlight themes are a collection of image settings that define the look of a control or controls. The benefit of using themes is that you can apply the theme across several controls in the application, thus providing consistency without having to repeat styling tasks.

When you add a C1ComboBox control to your project, it appears with the default blue theme:



You can also theme the C1ComboBox 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	
C1ThemeCosmopolitian	
C1ThemeExpressionDark	
C1ThemeExpressionLight	

C1ThemeRainierOrange	
C1ThemeShinyBlue	
C1ThemeWhistlerBlue	

You can add any of these themes to a C1ComboBox control by declaring it around the control in markup.

ComboBox 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 C1ComboBox control in general. If you are unfamiliar with the **ComponentOne ComboBox for Silverlight** product, please see the **ComboBox for Silverlight** Quick Start first.

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

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

Working with ComboBox Items

The following topics illustrate several ways to add list items to the C1ComboBox control.

Adding ComboBox Items in Blend's Design View

In this topic, you will learn how to add items to the C1ComboBox control in Expression Blend. This method is useful whenever you're creating a static combo box with just a few items.

Using the Assets Panel

Complete the following steps:

1. Click the C1ComboBox control once to select it.
2. Under the **Assets** tab, enter "C1ComboBoxItem" into the search box.

   The C1ComboBoxItem icon appears in the list.

3. Double-click the C1ComboBoxItem icon.

   A C1ComboBoxItem is added to the C1ComboBox control; its default content is a string, "C1ComboBoxItem".

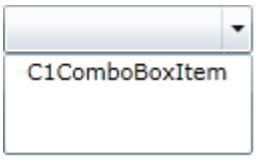
Using the Properties panel

Complete the following steps:

1. Click the C1ComboBox control once to select it.
2. In the **Properties** panel, click the **Items (Collection)** ellipsis button to open **the Object Collection Editor: Items** dialog box.
3. Click **Add Another Item** to open the **Select Object** dialog box.
4. In the search bar, enter “C1ComboBoxItem”.
5. Double-click the C1ComboBoxItem icon.
6. The **Select Object** dialog box closes and the C1ComboBoxItem is added to the **Object Collection Editor: Items** dialog box. If you wish, you can set its properties in the **Properties** grid.
7. Press **OK** to close the **Object Collection Editor: Items** dialog box.
8. The C1ComboBoxItem is added to the project.

This Topic Illustrates the Following:

With the program running, click the drop-down arrow and observe that one item appears in the drop-down list as follows:



Adding ComboBox Items in XAML

In this topic, you will learn how to add items to the C1ComboBox control in XAML markup. This method is useful whenever you're creating a static combo box with just a few items.

Complete the following steps:

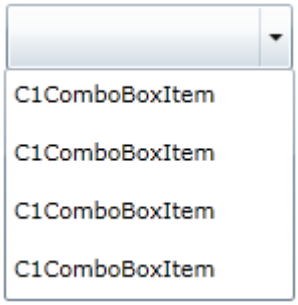
1. To add items to the C1ComboBox control, add the following XAML markup between the `<c1:C1ComboBox>` and `</c1:C1ComboBox>` tags:

```
<c1:C1ComboBoxItem Height="25" Content="C1ComboBoxItem"/>
<c1:C1ComboBoxItem Height="25" Content="C1ComboBoxItem"/>
<c1:C1ComboBoxItem Height="25" Content="C1ComboBoxItem"/>
<c1:C1ComboBoxItem Height="25" Content="C1ComboBoxItem"/>
```

2. Run the program.
3. Click the drop-down arrow and observe that four items appear in the drop-down list. The result resembles the following image:

This Topic Illustrates the Following:

With the program running, click the drop-down arrow and observe that four items appear in the drop-down list. The result resembles the following image:



Adding ComboBox Items in Code

In this topic, you will learn how to add items to the C1ComboBox control in C# and Visual Basic code.

Complete the following steps:

To disable AutoComplete, complete the following:

1. Add `x:Name="C1ComboBox1"` to the `<c1:C1ComboBox>` tag so that the object will have a unique identifier for you to call in code.
2. Open the **MainPage.xaml.cs** page.
3. Import the following namespace into your project:
   - Visual Basic

     ```
     Imports C1.Silverlight
     ```

   - C#

     ```
     Using C1.Silverlight;
     ```

4. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     C1ComboBox1.Items.Add(New C1ComboBoxItem() With {.Content =
     "C1ComboBoxItem1"})

     C1ComboBox1.Items.Add(New C1ComboBoxItem() With {.Content =
     "C1ComboBoxItem2"})

     C1ComboBox1.Items.Add(New C1ComboBoxItem() With {.Content =
     "C1ComboBoxItem3"})

     C1ComboBox1.Items.Add(New C1ComboBoxItem() With {.Content =
     "C1ComboBoxItem4"})
     ```

   - C#

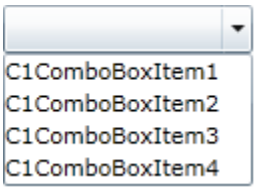
     ```
     C1ComboBox1.Items.Add(new C1ComboBoxItem() { Content =
     "C1ComboBoxItem1" });

     C1ComboBox1.Items.Add(new C1ComboBoxItem() { Content =
     "C1ComboBoxItem2" });

     C1ComboBox1.Items.Add(new C1ComboBoxItem() { Content =
     "C1ComboBoxItem3" });
     ```

```
C1ComboBox1.Items.Add(new C1ComboBoxItem() { Content =
"C1ComboBoxItem4" });
```

5. Run the program.

This Topic Illustrates the Following:

With the project running, click the drop-down arrow and observe that four items appear in the drop-down list. The result resembles the following image:



Adding ComboBox Items from a Collection

In this topic, you will populate a combo box's drop-down list with a collection.

Complete the following steps:

1. Add `x:Name="C1ComboBox1"` to the `<c1:C1ComboBox>` tag so that the object will have a unique identifier for you to call in code.
2. Open the **MainPage.xaml.cs** page.
3. Import the following namespace into the project:
   - Visual Basic

     ```
     Imports System.Collections.Generic
     ```

   - C#

     ```
     using System.Collections.Generic;
     ```

     Create your list by adding the following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     Dim dropDownList As New List(Of String)()
     dropDownList.Add("C1ComboBoxItem1")
     dropDownList.Add("C1ComboBoxItem2")
     dropDownList.Add("C1ComboBoxItem3")
     dropDownList.Add("C1ComboBoxItem4")
     ```

   - C#

     ```
     List<string> dropDownList = new List<string>();
     dropDownList.Add("C1ComboBoxItem1");
     dropDownList.Add("C1ComboBoxItem2");
     dropDownList.Add("C1ComboBoxItem3");
     dropDownList.Add("C1ComboBoxItem4");
     ```

5. Add the list to the combo box by setting the **ItemsSource** property:
   - Visual Basic

     ```
     C1ComboBox1.ItemsSource = dropDownList
     ```

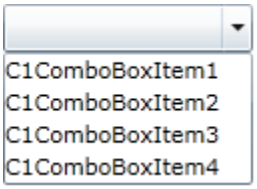
   - C#

     ```
     C1ComboBox1.ItemsSource = dropDownList;
     ```

6. Run the program.

This Topic Illustrates the Following:

With the project running, click the drop-down arrow and observe that four items appear in the drop-down list. The result resembles the following image:



Changing the Drop-Down List Direction

By default, the drop-down list will attempt to open at the bottom of the control; if there is no room at the bottom to display the whole drop-down list, it will appear above the control. You can, however, specify where you would like the drop-down list to open.

In Blend

Complete the following steps:

1. Click the C1ComboBox control once to select it.
2. Under the **Properties** panel, click the DropDownDirection drop-down arrow and select an option. For this example, select **ForceAbove**.
3. Run the program and click the drop-down arrow. Observe that the drop-down list appears above the control.

In XAML

Complete the following steps:

1. Add `DropDownDirection="ForceAbove"` to the `<c1:C1ComboBox>` tags so that the markup resembles the following:

   ```
   <c1:C1ComboBox Width="249" DropDownDirection="ForceAbove">
   ```

2. Run the program and click the drop-down arrow. Observe that the drop-down list appears above the control.

In Code

Complete the following steps:

1. Add `x:Name="C1ComboBox1"` to the `<c1:C1ComboBox>` tag so that the object will have a unique identifier for you to call in code.

2. Open the **MainPage.xaml.cs** page.
3. Add following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     C1ComboBox1.DropDownDirection = ForceAbove
     ```

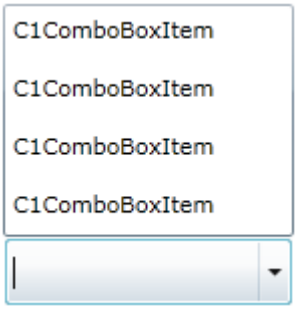
   - C#

     ```
     C1ComboBox1.DropDownDirection = ForceAbove;
     ```

4. Run the program and click the drop-down arrow. Observe that the drop-down list appears above the control.

This Topic Illustrates the Following:

In the following image, a combo box's drop-down list is forced to open above the control.



Disabling AutoComplete

By default, a user can type in the in the combo box's selection box to locate the item they want to select; you can disable this feature by setting the AutoComplete property to **False**.

In Blend

Complete the following steps:

1. Click the C1ComboBox control once to select it.
2. Under the **Properties** panel, clear the AutoComplete check box.

In XAML

To disable AutoComplete, add `AutoComplete="False"` to the `<c1:C1ComboBox>` tag so that the markup resembles the following:

```
<c1:C1ComboBox HorizontalAlignment="Left" Width="249"
AutoComplete="False">
```

In Code

Complete the following steps:

1. Add `x:Name="C1ComboBox1"` to the `<c1:C1ComboBox>` tag so that the object will have a unique identifier for you to call in code.
2. Open the **MainPage.xaml.cs** page.
3. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

```
C1ComboBox1.AutoComplete = False
```

- C#

```
C1ComboBox1.AutoComplete = false;
```

4. Run the program.

This Topic Illustrates the Following:

In this topic, you disabled the AutoComplete feature by setting the AutoComplete property to **False**. If you run the program and try to enter text, the control will not recommend a selection.

Setting the Maximum Height and Maximum Width of the Drop-Down List

You can specify the maximum height and maximum width of a combo box's drop-down list by setting its MaxDropDownHeight and MaxDropDownWidth properties. This topic assumes that the DropDownHeight and DropDownWidth properties are both set to **NaN**. For more information, see Drop-Down List Sizing.

In Blend

Complete the following steps:

1. Click the C1ComboBox control once to select it.
2. Under the **Properties** panel, complete the following:
   - Set the MaxDropDownHeight to a value, such as "150".
   - Set the MaxDropDownWidth to a value, such as "350".
3. Run the program and click the combo box's drop-down arrow to see the result of your settings.

In XAML

Complete the following steps:

1. Add `MaxDropDownHeight="150"` and `MaxDropDownWidth="350"` to the `<c1:C1ComboBox>` tag so that the markup resembles the following:

```
<c1:C1ComboBox HorizontalAlignment="Left" Width="249"
MaxDropDownHeight="150" MaxDropDownWidth="350">
```

2. Run the program and click the combo box's drop-down arrow to see the result of your settings.

In Code

Complete the following steps:

1. Add `x:Name="C1ComboBox1"` to the `<c1:C1ComboBox>` tag so that the object will have a unique identifier for you to call in code.
2. Open the **MainPage.xaml.cs** page.
3. Add the following code beneath the **InitializeComponent()** method to set the DropDownHeight property :

   - Visual Basic

```
C1ComboBox1.MaxDropDownHeight = 150
```

   - C#

```
C1ComboBox1.MaxDropDownHeight = 150;
```

4. Add the following code beneath the **InitializeComponent()** method to set the DropDownWidth property :
   - Visual Basic

     ```
     C1ComboBox1.MaxDropDownWidth = 350
     ```

   - C#

     ```
     C1ComboBox1.MaxDropDownWidth = 350;
     ```

5. Run the program and click the combo box's drop-down arrow to see the result of your settings.

This Topic Illustrates the Following:

In this topic, you set the MaxDropDownWidth property to a value of 350 pixels and the MaxDropDownHeight property to a value of 150 pixels. With these settings, the width of the drop-down list will never be more than 350 pixels and the height will never be more than 150 pixels; however, the height and width can be *less* than 150 pixels by 350 pixels that if the items in the list aren't enough to fill that area.

Launching with the Drop-Down List Open

To launch the C1ComboBox with its drop-down list open, set the IsDropDownOpen property to **True**.

In Blend

Complete the following steps:

1. Click the C1ComboBox control once to select it.
2. Under the **Properties** panel, select the IsDropDownOpen check box.
3. Run the program and observe that the drop-down list is open upon page load.

In XAML

Complete the following steps:

1. Add `IsDropDownOpen="True"` to the `<c1:C1ComboBox>` tag so that the markup resembles the following:

   ```
   <c1:C1ComboBox HorizontalAlignment="Left" Width="249"
   IsDropDownOpen="True">
   ```

2. Run the program and observe that the drop-down list is open upon page load.

In Code

Complete the following steps:

1. Add `x:Name="C1ComboBox1"` to the `<c1:C1ComboBox>` tag so that the object will have a unique identifier for you to call in code.
2. Open the **MainPage.xaml.cs** page.
3. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     C1ComboBox1.IsDropDownOpen = True
     ```

   - C#

     ```
     C1ComboBox1.IsDropDownOpen = true;
     ```

4. Run the program and observe that the drop-down list is open upon page load.

This Topic Illustrates the Following:

In this topic, you set the IsDropDownOpen property to **True** so that the drop-down list would be open at run time. You can also use this property to open the drop-down list when a user mouses over the C1ComboBox control (see Opening the Drop-Down List on MouseOver).

Opening the Drop-Down List on MouseOver

By default, the C1ComboBox control's drop-down list is only revealed when a user clicks the drop-down arrow. In this topic, you will write code that will cause the drop-down list to open whenever a user hovers over the control. This topic assumes that 1) you have already added a C1ComboBox control with at least one item to your project and 2) you are working in Expression Blend.

Complete the following:

1. Switch to XAML view and add `x:Name="C1ComboBox1"` to the `<c1:C1ComboBox>` tag so that the object will have a unique identifier for you to call in code.
2. Switch to Design view.
3. Click the C1ComboBox control to select it.
4. Under the **Properties** panel, click the **Events** button  to reveal the control's list of events.
5. Double-click inside of the **MouseEnter** text box. This will add the **C1ComboBox_MouseEnter** event handler to Code view.
6. Add the following code to the **C1ComboBox1_MouseEnter** event handler:
   - Visual Basic

     ```
     C1ComboBox1.IsDropDownOpen = True
     ```

   - C#

     ```
     C1ComboBox1.IsDropDownOpen = true;
     ```

7. Run the program.

This Topic Illustrates the Following:

With the program running, hover over the C1ComboBox control with your cursor. Observe that the drop-down list appears when you hover over the control. The drop-down list will stay open until you either select an item or click outside of the control.

Selecting an Item

You can select an item at run-time by setting the SelectedIndex property to the position of the item. This topic assumes that your project contains one C1ComboBox control with at least two C1ComboBoxItem items.

In Blend

Complete the following steps:

1. Select the C1ComboBox control.
2. Under the **Properties** panel, set the SelectedIndex property to "1" so that the second C1ComboBoxItem will be selected.

In XAML

To set a selected item, add `SelectedIndex="0"` to the `<c1:C1ComboBoxItem>` tag so that the markup resembles the following:

```
<c1:C1ComboBoxItem Content="C1ComboBoxItem1" SelectedIndex="1">
```

In Code

Complete the following steps:

1. Add `x:Name="C1ComboBoxItem1"` to the `<c1:C1ComboBoxItem>` tag so that the object will have a unique identifier for you to call in code.
2. Open the **MainPage.xaml.cs** page.
3. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     C1ComboBoxItem1.SelectedIndex = 1
     ```

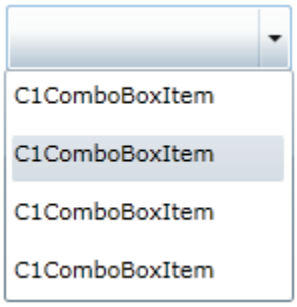
   - C#

     ```
     C1ComboBoxItem1.SelectedIndex = 1;
     ```

4. Run the program.

This Topic Illustrates the Following:

When the drop-down list is revealed at run time, the second item will be selected, such as in the following image.



Using C1ComboBox Themes

The C1ComboBox control comes equipped with a light blue default theme, but you can also apply six themes (see ComboBox Theming) to the control. In this topic, you will change the C1ComboBox 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 "C1ComboBox" to search for the C1ComboBox control.
5. Double-click the C1ComboBox icon to add the C1ComboBox control to your project.

6. Under the **Objects and Timeline** tab, select **[C1ComboBox]** 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>
```

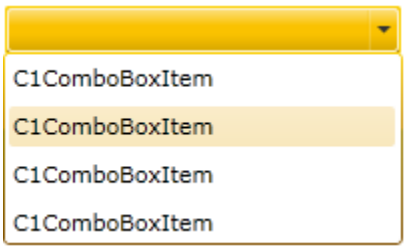
4. Place your cursor between the `<my:C1ThemeRainierOrange>` and `</my:C1ThemeRainierOrange>` tags.
5. In the **Tools** panel, double-click the C1ComboBox icon to add the control to the project. Its tags will appear as children of the `<my:C1ThemeRainierOrange>` tags, causing the markup to resemble the following:

```
<my:C1ThemeRainierOrange>
            <c1:C1ComboBox></c1:C1ComboBox>
</my:C1ThemeRainierOrange>
```

6. Run your project.

This Topic Illustrates the Following:

The following image depicts a C1ComboBox control with the C1ThemeRainierOrange theme.



DragDropManager

Support drag-and-drop operations in your Silverlight apps with **ComponentOne DragDropManager™ for Silverlight**.

Getting Started

Get started with the following topics:

- Key Features
- Quick Start

DragDropManager for Silverlight Key Features

ComponentOne DragDropManager for Silverlight includes several key features, such as:

- **Total Control of the Drag-and-drop Behavior**

  ComponentOne DragDropManager for Silverlight has an extensive set of methods and events that allow you to control the entire drag-and-drop process.

- **Simple Implementation**

  DragDropManager provides helper methods that simplify the process of adding drag-and-drop functionality to your applications. Just register some elements as drag sources and some as drop targets, and then handle the **DragDrop** event to move or copy the elements to their new location.

- **Scrolling Support**

  Dragging objects near the edges of a scrollable target causes it to scroll automatically, allowing end-users to drop elements exactly where they want in a single operation.

- **Customization**

  DragDropManager exposes properties that allow you to completely customize the appearance and behavior of drag-and-drop operations.

DragDropManager for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **ComponentOne DragDropManager for Silverlight**. You'll create a new project in Visual Studio, add a Grid with child elements to the page, and then implement the **C1DragDropManager** to allow users to drag elements from one grid cell to another.

Step 1 of 3: Creating a Silverlight Application

In this step you'll begin in Visual Studio to create a Silverlight application which will use **ComponentOne DragDropManager for Silverlight** to manage user interactions.

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, for example "QuickStart", 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. Right-click the project in the Solution Explorer window and select **Add Reference**.
5. In the **Add Reference** dialog box, locate and select the **C1.Silverlight.dll** assembly and click **OK** to add a reference to your project.
6. In the XAML window of the project, place the cursor between the `<Grid>` and `</Grid>` tags and click once.
7. Add the following XAML markup between the `<Grid>` and `</Grid>` tags in the **MainPage.xaml** file:

```
<Grid x:Name="ddGrid" Background="Lavender" ShowGridLines="True"
Width="400" Height="300" >
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <TextBlock Text="Drag Me" FontSize="14" Grid.Row="1"
Grid.Column="2" />
    <TextBlock Text="Or Me" FontSize="14" Grid.Row="3" Grid.Column="4"
/>
    <Rectangle Fill="Red" Grid.Row="2" Grid.Column="0"/>
    <Rectangle Fill="Blue" Grid.Row="0" Grid.Column="4"/>
</Grid>
```

This markup creates a grid with row definitions in the grid to keep **TextBlock** and **Rectangle** controls in separate areas.

What You've Accomplished

You've successfully created and set up a Silverlight application and added controls 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 you have not added drag-and-drop functionally to the application. 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 statements to the top of the page:
   - Visual Basic

```
Imports C1.Silverlight
```

   - C#

```
using C1.Silverlight;
```

3. Add code to the **MainPage.xaml.cs** (or **.vb**) file in the page constructor so it looks similar to the following:

- Visual Basic

```
Public Sub New()
    InitializeComponent()
    ' Initialize the C1DragDropManager
    Dim dd As New C1DragDropManager()
    dd.RegisterDropTarget(_ddGrid, True)
    For Each e As UIElement In _ddGrid.Children
        dd.RegisterDragSource(e, DragDropEffect.Move,
ModifierKeys.None)
    Next
    AddHandler dd.DragDrop, AddressOf dd_DragDrop
End Sub
```

- C#

```
public MainPage()
{
  InitializeComponent();
  // Initialize the C1DragDropManager
  C1DragDropManager dd = new C1DragDropManager();
  dd.RegisterDropTarget(_ddGrid, true);
  foreach (UIElement e in _ddGrid.Children)
  {
    dd.RegisterDragSource(e, DragDropEffect.Move, ModifierKeys.None);
  }
  dd.DragDrop += dd_DragDrop;
}
```

The code initiates a new instance of a **C1DragDropManager** and then calls the **RegisterDropTarget** method to indicate that the grid should act as a drop target by default. It then calls the **RegisterDragSource** method to indicate that users should be allowed to drag the elements in the grid and finally, the code attaches an event handler to the **DragDrop** event so the application can receive a notification and move the element being dragged into its new position

4. Add the following event handler to the **MainPage.xaml.cs** (or **.vb**) file, below all the other methods in the **MainPage** class:

- Visual Basic

```
Private Sub dd_DragDrop(source As Object, e As DragDropEventArgs)
    ' Get mouse position
    Dim pMouse As Point = e.GetPosition(_ddGrid)

    ' Translate into grid row/col coordinates
    Dim row As Integer, col As Integer
    Dim pGrid As New Point(0, 0)
    For row = 0 To _ddGrid.RowDefinitions.Count - 1
        pGrid.Y += _ddGrid.RowDefinitions(row).ActualHeight
        If pGrid.Y > pMouse.Y Then
        Exit For
        End If
    Next
    For col = 0 To _ddGrid.ColumnDefinitions.Count - 1
        pGrid.X += _ddGrid.ColumnDefinitions(col).ActualWidth
        If pGrid.X > pMouse.X Then
        Exit For
```

```
        End If
    Next

    ' Move the element to the new position
    e.DragSource.SetValue(Grid.RowProperty, row)
    e.DragSource.SetValue(Grid.ColumnProperty, col)
End Sub
```

- C#

```
private void dd_DragDrop(object source, DragDropEventArgs e)
{
  // Get mouse position
  Point pMouse = e.GetPosition(_ddGrid);
  // Translate into grid row/col coordinates
  int row, col;
  Point pGrid = new Point(0, 0);
  for (row = 0; row < _ddGrid.RowDefinitions.Count; row++)
    {
      pGrid.Y += _ddGrid.RowDefinitions[row].ActualHeight;
      if (pGrid.Y > pMouse.Y)
      break;
    }
  for (col = 0; col < _ddGrid.ColumnDefinitions.Count; col++)
  {
    pGrid.X += _ddGrid.ColumnDefinitions[col].ActualWidth;
    if (pGrid.X > pMouse.X)
    break;
  }
  // Move the element to the new position
  e.DragSource.SetValue(Grid.RowProperty, row);
  e.DragSource.SetValue(Grid.ColumnProperty, col);
}
```

The event handler starts by converting the mouse coordinates into row/column values. Then it uses the **SetValue** method to update the **Grid.RowProperty** and **Grid.ColumnProperty** values on the element that was dragged. Similar logic could be used to drag elements within other types of panel or list-type controls, or from one panel to another.

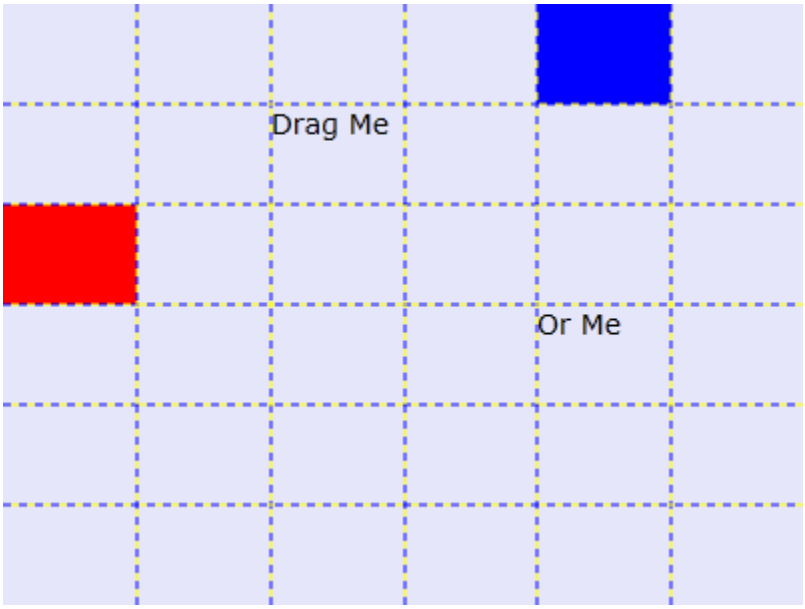
What You've Accomplished

In this step you added code to add functionality to your application. In the next step you'll run your application and observe some of the run-time interactions possible with **ComponentOne DragDropManager for Silverlight**.

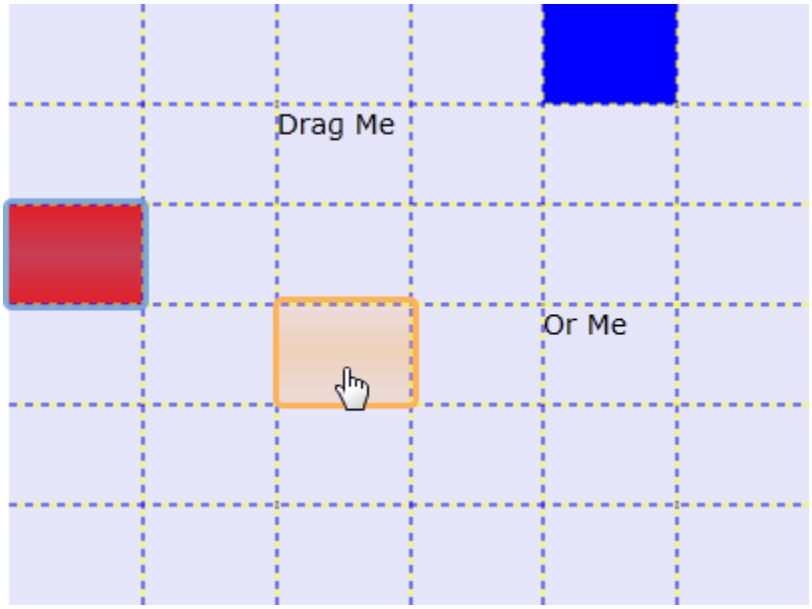
Step 3 of 3: Running the Application

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:



2. Click the red **Rectangle** and drag it to another square in the grid. Notice that as the drag process is in action an extra border appears around the item you are dragging and a transparent rectangle is moved with the mouse to indicate where the item will be dropped:



3. Click another item, such as a **TextBlock**, and move it to a new location.

What You've Accomplished

Congratulations, you've completed the **DragDropManager for Silverlight** quick start! You've created a simple application that uses **DragDropManager for Silverlight** to move items in a grid.

To learn more about the features and functionality of **ComponentOne DragDropManager for Silverlight**, see the Working with DragDropManager for Silverlight topic.

Working with DragDropManager for Silverlight

ComponentOne DragDropManager™ for Silverlight provides a simple way to add drag-and-drop interactions to your Silverlight application. The C1DragDropManager is a class that provides drag-and-drop services on behalf of other elements. It is not a control and has no visual representation at design time.

The C1DragDropManager class follows the patterns found in the WinForms drag-and-drop implementation. It provides a DoDragDrop method that is called to initiate the drag-and-drop operations, and fires events that allow you to customize the whole process (DragStart, DragEnter, DragOver, DragLeave, DragDrop).

The event parameters provide information on what item is being dragged (e.DragSource) and where it is about to be dropped (e.DropTarget). The event handlers are responsible for checking this information and setting the value of the e.Effect parameter to indicate whether the operation is valid or not. The handler for the DragDrop event is responsible for actually moving or copying the source element to the target location.

The C1DragDropManager class also provides two helper methods that simplify the whole process. The methods are called RegisterDragSource and RegisterDropTarget. They provide a simple generic implementation that handles most common drag-and-drop scenarios with very little code. Once you have registered elements as sources and targets with these methods, the C1DragDropManager will monitor the mouse to automatically initiate and manage the drag-and-drop operation. In this case, all you have to do is handle the DragDrop event to perform the move or copy operation.

Basic Properties

ComponentOne DragDropManager for Silverlight includes several properties that allow you to set the functionality of the C1DragDropManager control. Some of the more important properties are listed below.

The following properties let you customize the C1DragDropManager control:

Property	Description
AutoScroll	Gets or sets whether the **C1DragDropManager** should automatically scroll the **ScrollViewer** that contains the drop target.
AutoScrollDelay	Gets or sets the number of milliseconds between auto scroll steps.
AutoScrollEdge	Gets or sets the distance between the mouse and the edges of a drag target element that triggers the auto scroll process.
AutoScrollStep	Gets or sets the number of pixels to scroll in each auto scroll step.
Canvas	Gets a reference to the Canvas being used to show the drag-and-drop process.
DragThreshold	Gets or sets the distance in pixels that the mouse must move before a drag operation starts.
SourceMarker	Gets a reference to the Border used to highlight the drag source.
TargetMarker	Gets the Border used to indicate the drop location.

Basic Methods

ComponentOne DragDropManager for Silverlight includes several methods that allow you to customize the control. Some of the more important methods are listed below.

The following methods let you customize the C1DragDropManager control:

Method	Description
ClearSources	Removes all the registered sources.
ClearTargets	Removes all the registered targets.
DoDragDrop	Initiates a drag drop operation using a **UIElement** as a source, supporting a specified **DragDropEffect**.
RegisterDragSource	Registers a **UIElement** to act as a drag source.
RegisterDropTarget	Registers (or unregisters) an element as a drop target.

Basic Events

ComponentOne DragDropManager 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 C1DragDropManager control:

Event	Description
DragAutoScroll	Fires after the **C1DragDropManager** automatically scrolls a **ScrollViewer** in order to keep the drop location within view.
DragDrop	Fires at the end of a drag drop process, when the user releases the mouse button over a registered drop target.
DragEnter	Fires during a drag drop process, when the cursor enters a registered drop target.
DragLeave	Fires during a drag drop process, when the cursor leaves a registered drop target.
DragOver	Fires during a drag drop process, when the cursor moves over a registered drop target.
DragStart	Fires when a drag drop process starts.

DragDropManager for Silverlight Samples

ComponentOne DragDropManager 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
Checkers	This sample demonstrates how the **C1DragDropManager** control can be used to create a simple checkers game. This page is part of the **C1_Demo** sample and is installed by default at **C1.Silverlight\C1_Demo\C1_Demo\C1DragDropManager\DemoDragDropManager.xaml** in the samples directory.
DataGrid	This sample demonstrates how C1DragDropManager can be used to move content in a **C1DataGrid** control. This page is part of the **C1DataGrid_Demo** sample and is installed by default at **C1.Silverlight.DataGrid\C1DataGrid_Demo\C1DataGrid_Demo2010\Advanced\DragAndDropRows\DragAndDropRows.xaml** in the samples directory.
DemoListBox	The sample shows how to use C1DragDropManager to drag and drop items between list boxes. This page is part of the **C1_Demo** sample and is installed by default at **C1.Silverlight\C1_Demo\C1_Demo\C1DragDropManager\DemoListBox.xaml** in

	the samples directory.
Scrollers	This sample demonstrates how the **C1DragDropManager** component can be used to move items from one **ScrollViewer** to another. This page is part of the **C1_Demo** sample and is installed by default at **C1.Silverlight\C1_Demo\C1_Demo\C1DragDropManager\NestedScrollers.xaml** in the samples directory.

DropDown

Implement single item selection that's simple to setup and powerful to use with **ComponentOne DropDown™ for Silverlight**. **DropDown for Silverlight** provides a simple drop-down box control (like a **ComboBox** or a **DateTimePicker**). The C1DropDown control has a **Header** property that determines what the user sees when the drop-down part of the control is closed, and a **Content** property that determines what goes into the drop-down section. C1DropDown is more flexible than a traditional drop-down box allowing you add controls and even to create a search box.

Getting Started

Get started with the following topics:

- Key Features
- Quick Start
- Task-Based Help

DropDown for Silverlight Key Features

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

- **Create Specialized Drop-down Editors**

  The C1DropDown control gives you complete control to create specialized drop-down editors. Attach your own logic to the spin buttons and your own drop-down form/editor to the drop-down button. For example, you could place a **DataGrid** in the drop-down portion and code its row selection event to display a value from that row in the C1DropDown header portion.

- **Expand Above or Below the Header**

  Configure the drop-down to display above or below the header portion with the DropDownDirection property. Or set the direction preference the control will use if there is allowable space on the form.

- **AutoClose**

  Full control over when and whether the drop-down closes with the AutoClose property.

- **Sizable Drop-down List**

  You can explicitly set the width and height of the drop-down box or have it sized automatically to fit the content.

DropDown for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **DropDown for Silverlight**. In this quick start you'll create a new project, add a C1DropDown 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 C1DropDown control. You'll create a Silverlight application, add a C1DropDown control to the application, customize and add content to the drop-down box, and observe some of the run-time interaction possible with **DropDown for Silverlight**.

Step 1 of 3: Creating the C1DropDown Application

In this step you'll create a Silverlight application in Microsoft Expression Blend using **DropDown for Silverlight**. When you add a C1DropDown control to your application, you'll have a complete, functional DropDown-like interface that you can add images, controls, and other elements to. To set up your project and add a C1DropDown control to your application, complete the following steps:

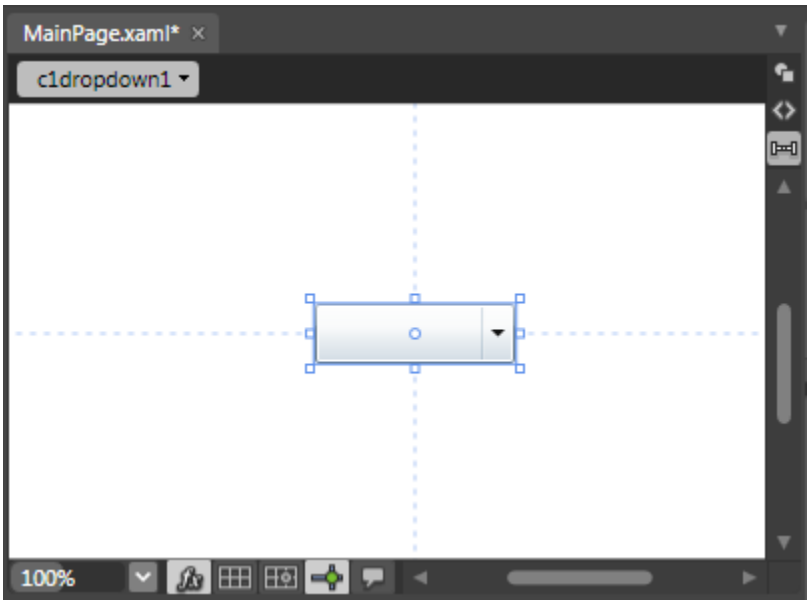
1. In Expression Blend, select **File | New Project**.
2. In the **New Project** dialog box, select the Silverlight project type in the left pane and in the right-pane select **Silverlight Application + Website**. Enter a **Name** and **Location** for your project, select a **Language** in the drop-down 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** assembly, 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 **C1DropDown** icon in the right pane.

   The **C1DropDown** 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 **C1DropDown** icon in the Toolbox to add the control to the panel.
8. Resize the page and reposition the C1DropDown control in the page.
9. Click once on the **C1DropDown** control in the Objects and Timelines pane, navigate to the Properties window and set the following properties:
   - Set **Name** to "c1dropdown1" to give the control a name so it is accessible in code.
   - Set **Height** to "30" and **Width** to "100" to change the control's size.
   - Set **HorizontalAlignment** and **VerticalAlignment** to **Center** to center the control in the panel.

   The XAML will appear similar to the following:

```
<c1:C1DropDown x:Name="c1dropdown1" HorizontalAlignment="Center"
Margin="0" VerticalAlignment="Center" Content="C1DropDown" Width="100"
Height="30"/>
```

   The page's Design view should now look similar to the following image (with the C1DropDown control selected on the form):



You've successfully set up your application's user interface, but if you run your application now you'll see that the C1DropDown control currently contains no content. In the next step you'll add content to the C1DropDown control, and then you'll observe some of the run-time interactions possible with the control.

Step 2 of 3: Adding Content to the C1DropDown Control

In the previous step you created a Silverlight application and added the C1DropDown control to your project. In this step you'll add content to the C1DropDown control. To customize your project and add content to the C1DropDown control in your application, complete the following steps:

1. In the Projects window, right click the **References** item and select **Add Reference**.
2. In the **Add Reference** dialog box, locate and select the **System.Windows.Controls.dll** assembly and click **Open**.
3. Switch to XAML view and add `xmlns:controls="clr-namespace:System.Windows.Controls;assembly=System.Windows.Control` to the initial **UserControl** tag to add the namespace. 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;assembly=C1.Silverlight" xmlns:controls="clr-
   namespace:System.Windows.Controls;assembly=System.Windows.Controls"
   x:Class="C1DropDown.MainPage" Width="640" Height="480">
   ```

   In the next steps you'll add XAML markup to your application to add content to the drop-down box.
4. Delete the default `Content="C1DropDown"` text in the **C1DropDown** tag.
5. Add markup to the C1DropDown control so that it appears similar to the following:

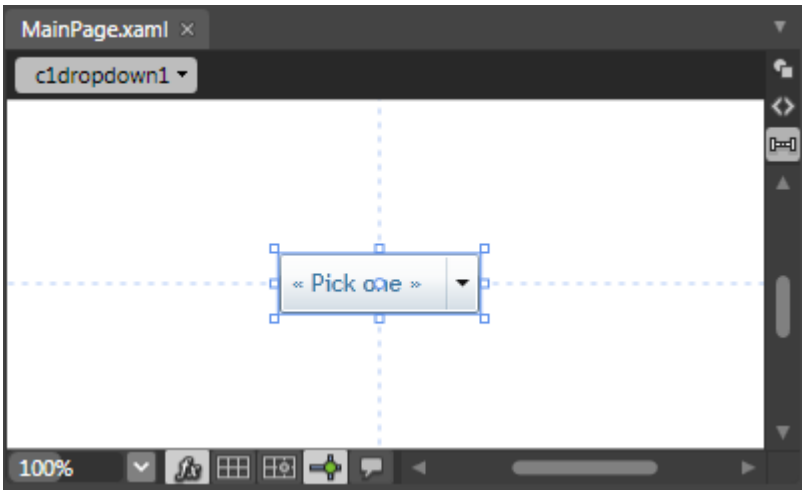
   ```
   <c1:C1DropDown x:Name="c1dropdown1" HorizontalAlignment="Center"
   Margin="0" VerticalAlignment="Center" Width="100" Height="30">
       <c1:C1DropDown.Header>
           <ContentControl VerticalAlignment="Center"
   HorizontalAlignment="Left" Margin="5,0,0,0">
               <TextBlock x:Name="selection" Text="« Pick one »"
   FontSize="12" Foreground="#FF3B76A2" TextAlignment="left" />
           </ContentControl>
       </c1:C1DropDown.Header>
   </c1:C1DropDown>
   ```

   This will add a **TextBlock** to the **Header** of the C1DropDown control. The **Header**'s content will appear in the C1DropDown control at run time when no selection has been made.
6. Add the following markup just after the `</c1:C1DropDown.Header>` tag in the XAML markup you just added:

   ```
   <c1:C1DropDown.Content>
       <controls:TreeView x:Name="treeSelection" Background="Transparent"
   Margin="10">
           <controls:TreeViewItem Header="South America">
               <controls:TreeViewItem Header="Brazil" />
               <controls:TreeViewItem Header="Argentina" />
               <controls:TreeViewItem Header="Uruguay" />
           </controls:TreeViewItem>
           <controls:TreeViewItem Header="Europe">
               <controls:TreeViewItem Header="Italy" />
               <controls:TreeViewItem Header="France" />
               <controls:TreeViewItem Header="England" />
               <controls:TreeViewItem Header="Germany" />
   ```

```
        </controls:TreeViewItem>
    </controls:TreeView>
</c1:C1DropDown.Content>
```

This will add a standard **TreeView** control to the C1DropDown control's content area. Note that the Window should appear similar to the following image in Design view:



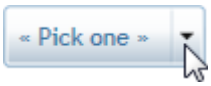
In this step you added content to the C1DropDown control. In the next step you'll further customize the control and run the application to observe run-time interactions.

Step 3 of 3: Running the C1DropDown Application

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

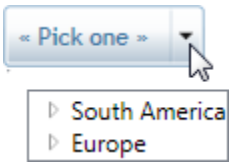
1. From the **Project** menu, select **Run Project** to view how your application will appear at run time.

   The application will appear similar to the following:

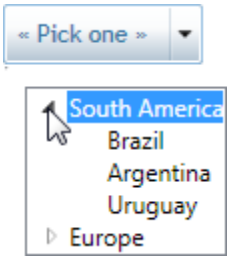


   The C1DropDown control appears as a simple drop-down box. Notice the text that you specified appears in the header of the control.

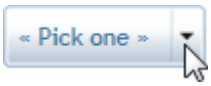
2. Click the C1DropDown control's drop-down arrow. Notice that the **TreeView** control is now visible:



3. Expand an item in the drop-down box – notice that you can interact with the **TreeView** control as you might normally.



4. Click the C1DropDown control's drop-down arrow again:



Observe that the drop-down box closes.

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

Working with DropDown for Silverlight

ComponentOne DropDown for Silverlight includes the C1DropDown control, a simple drop-down box control that acts as a container, allowing you to add controls, images, and more to an interactive input box. When you add the C1DropDown control to a XAML window it exists as a fully function input control that can be customized and include added content.

Basic Properties

ComponentOne DropDown 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 DropDown for Silverlight Appearance Properties for more information about properties that control appearance.

The following properties let you customize the C1DropDown control:

Property	Description
AutoClose	Auto closes the drop-down box when the user clicks outside it.
DropDownDirection	Specifies the expand direction of the control drop-down box.
DropDownHeight	Gets or sets the height of the drop-down box (set to zero to size automatically).
DropDownWidth	Gets or sets the width of the drop-down box (set to zero to size automatically).
IsDropDownOpen	Open or close the control drop-down box.
ShowButton	Gets/sets if the a **ToggleButton** is shown.

Basic Events

ComponentOne DropDown for Silverlight includes 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 C1DropDown control:

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

C1DropDown Elements

The C1DropDown control consists of two parts: a header area and a content area. The header appears visible when the drop-down box is not open; the content is what is visible when the drop-down area is clicked. In the below image, the two sections are identified:

You can add content to neither, either, or both the header and content areas. You can customize the C1DropDown control in XAML by adding content to the header and content areas. For example the following markup creates a drop-down control similar to the one pictured above:

```
<c1:C1DropDown x:Name="c1dropdown1" HorizontalAlignment="Center" Margin="0"
VerticalAlignment="Center" Width="100" Height="30">
    <c1:C1DropDown.Header>
        <ContentControl VerticalAlignment="Center" HorizontalAlignment="Left"
Margin="5,0,0,0">
            <TextBlock x:Name="selection" Text="« Pick one »" FontSize="12"
Foreground="#FF3B76A2" TextAlignment="left" />
        </ContentControl>
    </c1:C1DropDown.Header>
    <c1:C1DropDown.Content>
        <controls:TreeView x:Name="treeSelection" Background="Transparent"
Margin="10">
        <controls:TreeViewItem Header="South America">
            <controls:TreeViewItem Header="Brazil" />
            <controls:TreeViewItem Header="Argentina" />
            <controls:TreeViewItem Header="Uruguay" />
        </controls:TreeViewItem>
        <controls:TreeViewItem Header="Europe">
            <controls:TreeViewItem Header="Italy" />
            <controls:TreeViewItem Header="France" />
```

```
            <controls:TreeViewItem Header="England" />
            <controls:TreeViewItem Header="Germany" />
        </controls:TreeViewItem>
    </controls:TreeView>
    </c1:C1DropDown.Content>
</c1:C1DropDown>
```

Note that the `<c1:C1DropDown.Header>` and `<c1:C1DropDown.Content>` tags are used to define header and content. You can add controls and content within these tags.

DropDown Interaction

Users can interact with items in the drop-down box, or with the C1DropDown control itself at run time. By default users can interact with controls placed within the drop-down box. For example, if you place a button or drop-down box within the C1DropDown control, it will be clickable by users at run time.

You can control the C1DropDown control's drop-down direction using the DropDownDirection property. You can see Drop-Down Box Direction for more information. You can choose if the C1DropDown box appears with the drop-down box automatically open using the IsDropDownOpen property. You can also set whether or not the drop-down box automatically closes when the users click outside of it – this can be set using the AutoClose property.

Drop-Down Box Direction

By default, when the user clicks the C1DropDown 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 box below the header. If it is not possible tries to open above it.
AboveOrBelow	Tries to open the drop-down box above the header. If it is not possible tries to open below it.
ForceBelow	Forces the drop-down box to open below the header.
ForceAbove	Forces the drop-down box to open above the header.

For more information and an example, see Changing the Drop-Down Direction.

Additional Controls

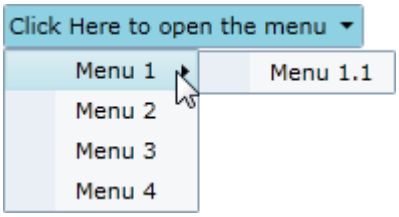
In addition to the full-featured C1DropDown control, **DropDown for Silverlight** includes parts of the C1DropDown control, C1DropDownButton and C1SplitButton that allow you to further customize your application.

C1DropDown is similar to a traditional drop-down control allowing you to choose from a selection of items, C1DropDownButton works like a drop-down control but looks like a button, C1SplitButton has two parts: the header and the arrow which you can customize so that clicking in the arrow will open the drop-down list, and pressing in the header button will typically apply the current selection.

C1DropDownButton Elements

The C1DropDownButton control is similar to the **C1DropDown** control and consists of two parts: a header area and a content area. The header appears visible when the drop-down box is not open; the content is what is visible

when the drop-down area is clicked. For example, in the below image the content area concludes a structured menu:



You can add content to neither, either, or both the header and content areas. You can customize the C1DropDown control in XAML by adding content to the header and content areas. For example the following markup creates a drop-down control similar to the one pictured above:

```
<c1:C1DropDownButton x:Name="ddl" Header="Click Here to open the menu"
HorizontalAlignment="Left">
    <c1:C1MenuList>
        <c1:C1MenuItem Header="Menu 1">
                <c1:C1MenuItem Header="Menu 1.1" />
        </c1:C1MenuItem>
        <c1:C1MenuItem Header="Menu 2" />
        <c1:C1MenuItem Header="Menu 3" />
        <c1:C1MenuItem Header="Menu 4" />
    </c1:C1MenuList>
</c1:C1DropDownButton>
```

Note that the header text is defined in the `<c1:C1DropDownButton>` tag and the content is placed within the `<c1:C1DropDownButton></c1:C1DropDownButton>` tags.

C1SplitButton Elements

The C1SplitButton control combined a button and a drop-down content area, similar to a standard **SplitButton**. The **C1SplitButton** is more dynamic in the type of content that can be included. In the following image, the **C1SplitButton**'s header includes a color picker icon that changes when a color is picked and the **C1SplitButton**'s content area includes a color picker:

You can add content to neither, either, or both the header and content areas. You can customize the C1SplitButton control in XAML by adding content to the header and content areas. For example the following markup creates a drop-down control similar to the one pictured above:

```
<c1:C1SplitButton x:Name="btn" DropDownWidth="100" DropDownHeight="100"
HorizontalAlignment="Left" Click="btn_Click">
    <c1:C1DropDown.Header>
        <StackPanel>
```

```
            <TextBlock FontFamily="Times New Roman" FontSize="12" Text="A"
FontWeight="Bold" VerticalAlignment="Top" Margin="3 0 0 -2" />
            <Border x:Name="selectedColorBorder" Background="Red">
                <Border.Style>
                    <Style TargetType="Border">
                        <Setter Property="BorderThickness" Value="1" />
                        <Setter Property="BorderBrush" Value="#FF9E9DA7" />
                        <Setter Property="Width" Value="14" />
                        <Setter Property="Height" Value="5" />
                        <Setter Property="VerticalAlignment" Value="Bottom"
/>
                        <Setter Property="Margin" Value="1,0,0,1" />
                    </Style>
                </Border.Style>
            </Border>
        </StackPanel>
    </c1:C1DropDown.Header>
    <Border BorderThickness="1" BorderBrush="#FF207A9C"
Background="#FFEFF5FF">
        <Border.Resources>
            <c1:ColorConverter x:Key="colorConverter" />
        </Border.Resources>
        <c1:C1SpectrumColorPicker ShowAlphaChannel="False" Margin="4"
Color="{Binding Background, ElementName=selectedColorBorder, Mode=TwoWay,
Converter={StaticResource colorConverter}}" />
    </Border>
</c1:C1SplitButton>
```

Note that the `<c1:C1DropDown.Header>` defines the header element and the content element is contained within the `<c1:C1SplitButton></c1:C1SplitButton>` tags.

DropDown for Silverlight Layout and Appearance

The following topics detail how to customize the C1DropDown 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 drop-down box 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.

DropDown for Silverlight Appearance Properties

ComponentOne DropDown 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.

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:

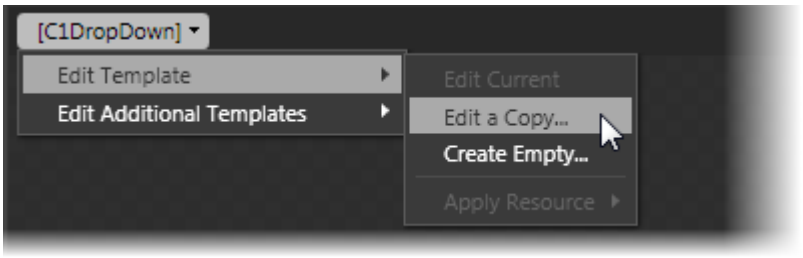
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.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

C1DropDown 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 DropDown 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 C1DropDown control and, in the menu, selecting **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or **Create Empty**, to create a new blank template.

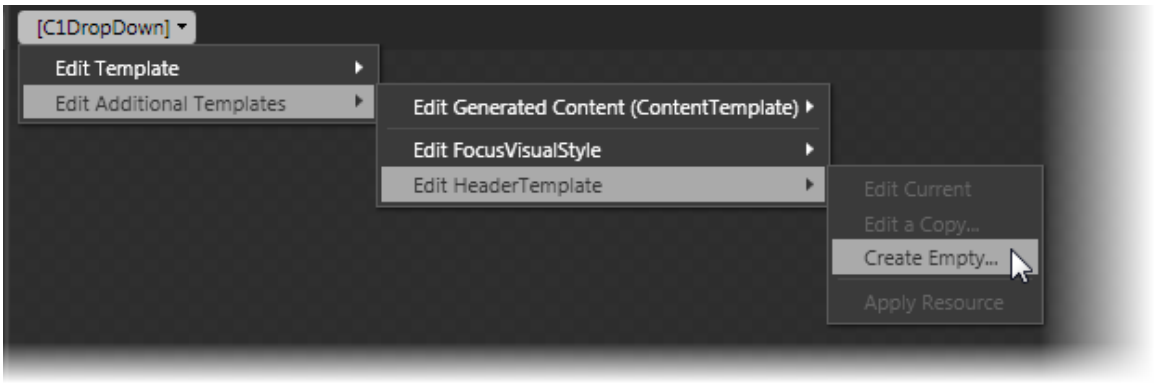


Once you've created a new template, the template will appear in the **Objects and Timeline** window. Note that you can use the Template property to customize the template.

> **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.

Additional Templates

In addition to the default template, the C1DropDown control includes a few additional templates. These additional templates can also be accessed in Microsoft Expression Blend – in Blend select the C1DropDown control and, in the menu, select **Edit Additional Templates**. Choose a template, and select **Create Empty**:



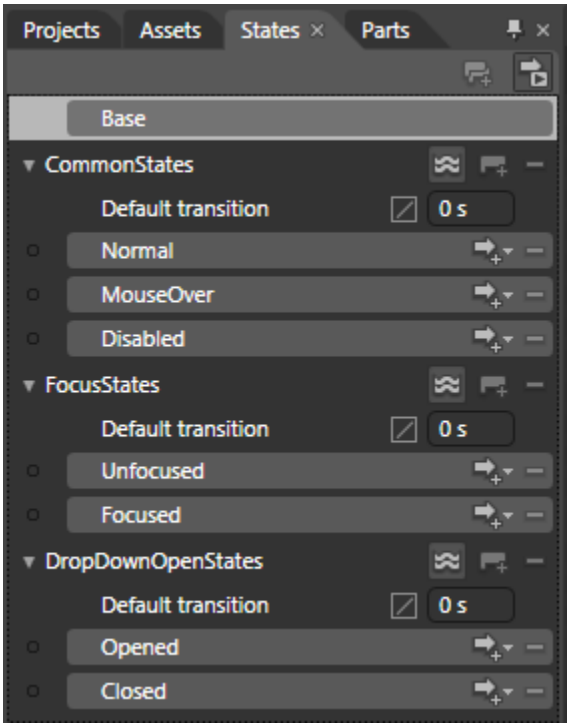
C1DropDown Styles

ComponentOne DropDown for Silverlight's C1DropDown 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.

C1DropDown 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. 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.

DropDown 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 C1DropDown control in general. If you are unfamiliar with the **ComponentOne DropDown for Silverlight** product, please see the DropDown for Silverlight Quick Start first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne DropDown for Silverlight** product. Most task-based help topics also assume that you have created a new Silverlight project and added a C1DropDown control to the project – for information about creating the control, see Creating a DropDown.

Creating a DropDown

You can easily create a C1DropDown control at design time in Expression Blend, in XAML, and in code. Note that if you create a C1DropDown control as in the following steps, it will appear empty. You will need to add items to the control. For an example, see Adding Content to C1DropDown.

At Design Time in Blend

To create a C1DropDown 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** assembly, 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 **C1DropDown** icon in the right pane:

   The **C1DropDown** 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 **C1DropDown** icon in the Toolbox to add the control to the panel. The C1DropDown control will now exist in your application.
6. If you choose, can customize the control by selecting it and setting properties in the Properties window. For example, set the C1DropDown control's **Name** property to "c1dropdown1".

In XAML

To create a C1DropDown 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** assembly, and click **OK**.
2. Add a XAML namespace to your project by adding `xmlns:c1="clr-namespace:C1.Silverlight;assembly=C1.Silverlight"` 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;assembly=C1.Silverlight"
x:Class="C1DropDown.MainPage" Width="640" Height="480">
```

3. Add a `<c1:C1DropDown>` tag to your project within the `<Grid>` tag to create a C1DropDown control. The markup will appear similar to the following:

```
<Grid x:Name="LayoutRoot" Background="White">
    <c1:C1DropDown Height="30" Name="c1dropdown1" Width="100" />
</Grid>
```

   This markup will create an empty C1DropDown control named "c1dropdown1" and set the control's size.

In Code

To create a C1DropDown 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** assembly, 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
```

   - C#

```
using C1.Silverlight;
```

4. Add code to the page's constructor to create the C1DropDown control. It will look similar to the following:

   - Visual Basic

```
Public Sub New()
    InitializeComponent()
    Dim c1dropdown1 as New C1DropDown
    c1dropdown1.Height = 30
    c1dropdown1.Width = 100
    LayoutRoot.Children.Add(c1dropdown1)
```

```
End Sub
```

- C#

```
public MainPage()
{
    InitializeComponent();
    C1DropDown c1dropdown1 = new C1DropDown();
    c1dropdown1.Height = 30;
    c1dropdown1.Width = 100;
    LayoutRoot.Children.Add(c1dropdown1);
}
```

This code will create an empty C1DropDown control named "c1dropdown1", set the control's size, and add the control to the page.

What You've Accomplished

You've created a C1DropDown control. Note that when you create a C1DropDown control as in the above steps, it will appear empty. You can add items to the control that can be interacted with at run time. For an example, see Adding Content to C1DropDown.

Adding Content to C1DropDown

You can add any sort of arbitrary content to a C1DropDown control. This includes text, images, and other standard and 3rd-party controls. In this example, you'll add a **Button** control to a C1DropDown control, but you can customize the steps to add other types of content instead.

At Design Time in Blend

To add a **Button** control to the drop-down box in Blend, complete the following steps:

1. Click the C1DropDown control once to select it.
2. Navigate to the Toolbox, and double-click the **Button** item to add the control to the C1DropDown control.
3. Notice in XAML view that the button appears in the C1DropDown control's tag:

```
<c1:C1DropDown Height="30" Name="c1dropdown1" Width="100">
    <Button Height="23" Name="button1" Width="75">Button</Button>
</c1:C1DropDown>
```

4. If you choose, can customize the C1DropDown and **Button** controls by selecting each control and setting properties in the Properties window. For example, set the **Button**'s **Content** property to "Hello World!".

In XAML

For example, to add a **Button** control to the drop-down add `<Button Height="23" Name="button1" Width="75">Hello World!</Button>` within the `<c1:C1DropDown>` tag so that it appears similar to the following:

```
<c1:C1DropDown Height="30" Name="c1dropdown1" Width="100">
    <Button Height="23" Name="button1" Width="75">Hello World!</Button>
</c1:C1DropDown>
```

In Code

For example, to add a **Button** control to the drop-down box, add code to the page's constructor so it appears like the following:

- Visual Basic

```
Public Sub New()
    InitializeComponent()
    Dim C1Button1 as New Button
    C1Button1.Content = "Hello World!"
```

```
    c1dropdown1.Content = c1button1
End Sub
```

- C#

```
public MainPage()
{
    InitializeComponent();
    Button c1button1 = new Button();
    c1button1.Content = "Hello World!";
    c1dropdown1.Content = c1button1;
}
```

What You've Accomplished

You've added a button control to the C1DropDown control. Run the application and click the drop-down arrow. Observe that the **Button** control has been added to the drop-down box. Note that to add multiple items to the C1DropDown control, add a **Grid** or other panel to the C1DropDown control, and add items to that panel.

Changing the Drop-Down Direction

By default, when the user clicks the C1DropDown control's drop-down arrow at run-time the drop-down box 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 Box Direction.

At Design Time in Blend

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

1. Click the C1DropDown 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:C1DropDown>` tag so that it appears similar to the following:

```
<c1:C1DropDown Height="30" Name="c1dropdown1" Width="100"
DropDownDirection="ForceAbove"/>
```

In Code

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

- Visual Basic

```
Me.c1dropdown1.DropDownDirection = DropDownDirection.ForceAbove
```

- C#

```
this.c1dropdown1.DropDownDirection = DropDownDirection.ForceAbove;
```

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

What You've Accomplished

Run the application. When you click the C1DropDown control's drop-down arrow, the drop down window will appear above the control.

Hiding the Drop-Down Arrow

By default, when you add the C1DropDown control to an application, the drop-down arrow, the **ToggleButton**, is visible. However, if you choose you can hide the drop-down arrow by setting the ShowButton property as in the following steps.

At Design Time in Blend

To hide the drop-down arrow, complete the following steps:

1. Click the C1DropDown control once to select it.
2. Navigate to the Properties window and clear the ShowButton check box.

   This will hide the drop-down arrow on the control.

In XAML

To hide the drop-down arrow, add `ShowButton="False"` to the `<c1:C1DropDown>` tag so that it appears similar to the following:

```
<c1:C1DropDown Height="30" Name="c1dropdown1" Width="100"
ShowButton="False" />
```

In Code

To hide the drop-down arrow, add the following code to your project:

- Visual Basic

```
Me.c1dropdown1.ShowButton = False
```

- C#

```
this.c1dropdown1.ShowButton = false;
```

This will hide the drop-down arrow on the control.

What You've Accomplished

Run the application. Observe that the drop-down arrow no longer appears on the C1DropDown control.

Opening the Drop-Down on MouseOver

By default, the C1DropDown control's drop-down box only opens when users click on the drop-down arrow at run time. In this topic you'll set the drop-down box to open when users mouse over the control at run-time instead. Note that this topic assumes you have already added a C1DropDown control which contains content to the application.

Complete the following steps:

1. Click once on the C1DropDown control to select it.
2. Navigate to the Properties window and click on the lightning bolt **Events** button to view events associated with the control.
3. Double-click the box next to the IsMouseOverChanged item to switch to Code view and create the **C1DropDown_IsMouseOver** event handler.
4. In Code view, add the following import statement to the top of the page:

   - Visual Basic

   ```
   Imports C1.Silverlight
   ```

   - C#

   ```
   using C1.Silverlight;
   ```

5. Add code to the **C1DropDown_IsMouseOver** event handler so it looks like the following:

   - Visual Basic

```
Private Sub c1dropdown1_IsMouseOverChanged(ByVal sender As Object,
ByVal e As PropertyChangedEventArgs(Of Boolean))
    If c1dropdown1.IsMouseOver = True Then
        c1dropdown1.IsDropDownOpen = True
    Else
        c1dropdown1.IsDropDownOpen = False
    End If
End Sub
```

- C#

```
private void c1dropdown1_IsMouseOverChanged(object sender,
C1.Silverlight.PropertyChangedEventArgs<bool> e)
{
    if (c1dropdown1.IsMouseOver == true)
    {
        c1dropdown1.IsDropDownOpen = true;
    }
    else
    {
        c1dropdown1.IsDropDownOpen = false;
    }
}
```

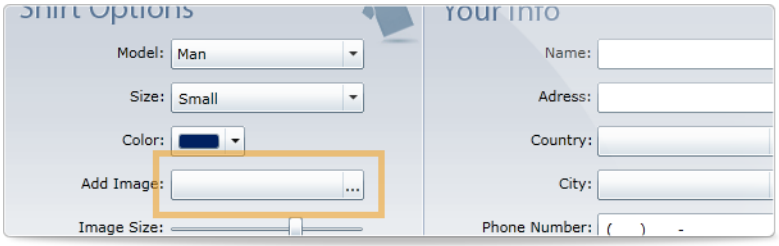
This code sets the C1DropDown control's actions when a user mouses over the control at run time.

What You've Accomplished

In this topic you added code so that the drop-down box opens when moused over at run time using the IsDropDownOpen property. Run the application and move the mouse over the control. Notice that the drop-down box opens. Move the mouse away from the control and observe that the drop-down box closes.

FilePicker

Allow end-users to select files in your Silverlight apps with **ComponentOne FilePicker™ for Silverlight**. The **C1FilePicker** control is similar to a combo box, except instead of showing a drop-down list, it shows a file picker dialog box.



Getting Started

Get started with the following topics:

- Key Features
- Quick Start
- Task-Based Help

FilePicker for Silverlight Key Features

ComponentOne FilePicker for Silverlight includes several key features, such as:

- **Filter File Types**

  Limit the file types you search for based on a file extension or category. For more information, see File Filtering and for an example of adding a filter for image files, see the Adding a File Filter topic.

- **Multi-file Selection**

  Allow end-users to select a single file at a time or multiple files at once to speed up the process. For more information, see Multiple File Selection and for an example of allowing users to select multiple files in the C1FilePicker control, see the Selecting Multiple Files topic.

- **Silverlight Toolkit Themes Support**

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

- **Uploader Integration**

  Use **FilePicker for Silverlight** with **Uploader for Silverlight** to allow users to upload the chosen file. For example, the end-user can select multiple files at a time using the **C1FilePicker** control, and the **Uploader for Silverlight** control loads the files asynchronously.

FilePicker for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **ComponentOne FilePicker for Silverlight**. In this quick start you'll create an application that allows you to select and view thumbnails of images on your computer. You'll create a new project in Visual Studio, add and customize the **C1FilePicker** control, and add controls to view files selected with the **C1FilePicker** control.

Step 1 of 3: Creating a Silverlight Application

In this step you'll begin in Visual Studio to create a Silverlight application which will use **ComponentOne FilePicker for Silverlight** to select image files on your computer to view. You'll create a new Silverlight project and 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, for example "QuickStart", 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. Add the following XAML markup between the `<Grid>` and `</Grid>` tags in the **MainPage.xaml** file:

```
<Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition />
    <RowDefinition Height="Auto" />
</Grid.RowDefinitions>
```

   This markup creates row definitions in the grid to keep controls in separate areas.
6. Navigate to the Toolbox and double-click the **C1FilePicker** icon to add the control to your application.
7. Edit the **C1FilePicker** tag so it appears similar to the following:

```
<c1:C1FilePicker x:Name="C1FilePicker1" Grid.Row="0" Margin="15"
Height="30" SelectedFilesChanged="C1FilePicker_SelectedFilesChanged" />
```

   The markup above gives the control a name, specifies the grid row the control appears in, sets the height of the control and the margin around the control, and indicates an event handler for the SelectedFilesChanged event. You'll add the event handler code in a later step.
8. Add the following XAML markup beneath the `<c1:C1FilePicker>` tag and before the `</Grid>` tag in the **MainPage.xaml** file:

```
<ScrollViewer Grid.Row="1" Margin="15,0,15,0">
    <ListBox x:Name="ListBox" />
</ScrollViewer>
<StackPanel Grid.Row="2" Name="stackPanel1" Orientation="Horizontal"
HorizontalAlignment="Center">
    <Button Content="Clear File Selection" Height="30"
Margin="0,15,15,15" Name="button1" Width="150" Grid.Row="2"
Click="button1_Click" />
    <Button Content="Clear List Items" Height="30" Margin="15,15,0,15"
Name="button2" Width="150" Grid.Row="2" Click="button2_Click" />
</StackPanel>
```

   This markup will add a **ListBox** control which will allow you to view local images you select using the **C1FilePicker** control. The markup also adds two buttons which will let you clear the content of the **C1FilePicker** control and the content of the **ListBox**.

What You've Accomplished

You've successfully created and set up a Silverlight application and added controls 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 it currently does not function. 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 statements to the top of the page if they are not included:
   - Visual Basic
     ```
     Imports System.Windows.Media.Imaging
     Imports C1.Silverlight
     ```
   - C#
     ```
     using System.Windows.Media.Imaging;
     using C1.Silverlight;
     ```
3. Add the following event handler to the **MainPage.xaml.cs** (or **.vb**) file, below all the other methods in the **MainPage** class:
   - Visual Basic
     ```
     Private Sub C1FilePicker_SelectedFilesChanged(sender As Object, e As
     EventArgs)
         If C1FilePicker1.SelectedFile IsNot Nothing Then
             Dim stream = C1FilePicker1.SelectedFile.OpenRead()
             Dim source = New BitmapImage()
             source.SetSource(stream)
             Dim image = New Image()
             image.Source = source
             image.Stretch = Stretch.Uniform
             image.Height = 100
             ListBox.Items.Add(image)
         End If
     End Sub
     ```
   - C#
     ```
     private void C1FilePicker_SelectedFilesChanged(object sender, EventArgs
     e)
     {
         if (C1FilePicker1.SelectedFile != null)
         {
             var stream = C1FilePicker1.SelectedFile.OpenRead();
             var source = new BitmapImage();
             source.SetSource(stream);
             var image = new Image();
             image.Source = source;
             image.Stretch = Stretch.Uniform;
             image.Height = 100;
             ListBox.Items.Add(image);
         }
     }
     ```

   This code handles the SelectedFilesChanged event. When a user selects an image with the **C1FilePicker** control, the image is customized and added to the **ListBox** control.
4. Add the following code to handle the **Click** events of the **Button** controls on the page:
   - Visual Basic
     ```
     Private Sub button1_Click(sender As Object, e As RoutedEventArgs)
         C1FilePicker1.ClearSelection()
     End Sub

     Private Sub button2_Click(sender As Object, e As RoutedEventArgs)
         ListBox.Items.Clear()
     ```

```
End Sub
```

- C#

```
private void button1_Click(object sender, RoutedEventArgs e)
{
    C1FilePicker1.ClearSelection();
}

private void button2_Click(object sender, RoutedEventArgs e)
{
    ListBox.Items.Clear();
}
```

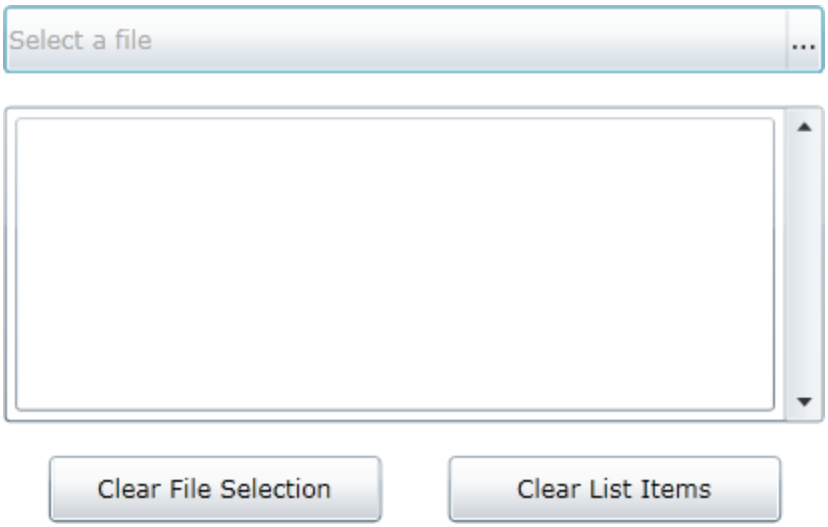
What You've Accomplished

In this step you added code to add functionality to your application. In the next step you'll run your application and observe some of the run-time interactions possible with **ComponentOne FilePicker for Silverlight**.

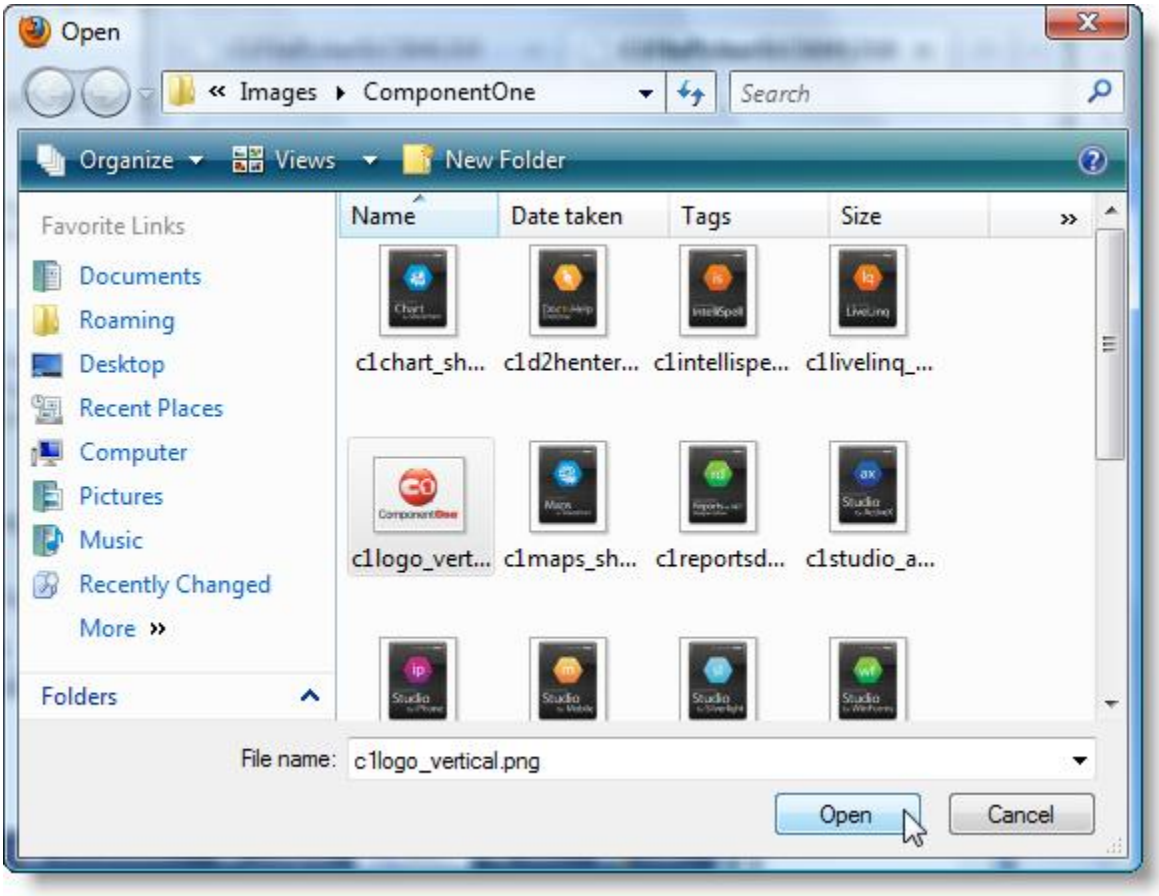
Step 3 of 3: Running the Application

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:

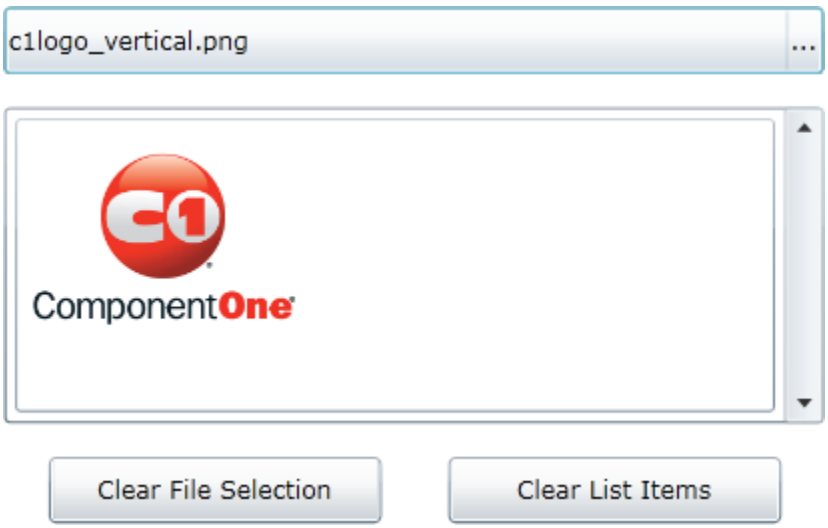
1. Choose **Debug | Start Debugging** from the menu to run your application. The application will appear similar to the following image:



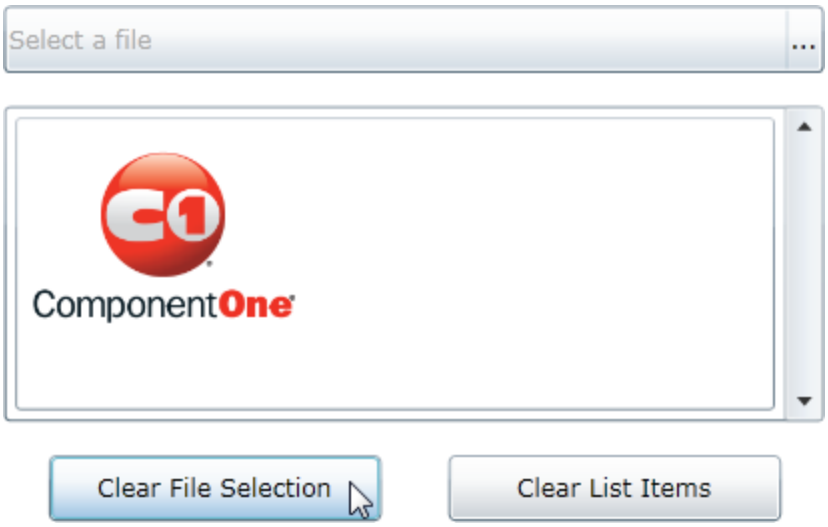
2. Click the ellipses button (**...**) on the **C1FilePicker** control. The **Open** dialog box will appear.
3. In the **Open** dialog box, locate and select an image, and click **Open** to open the selected image:



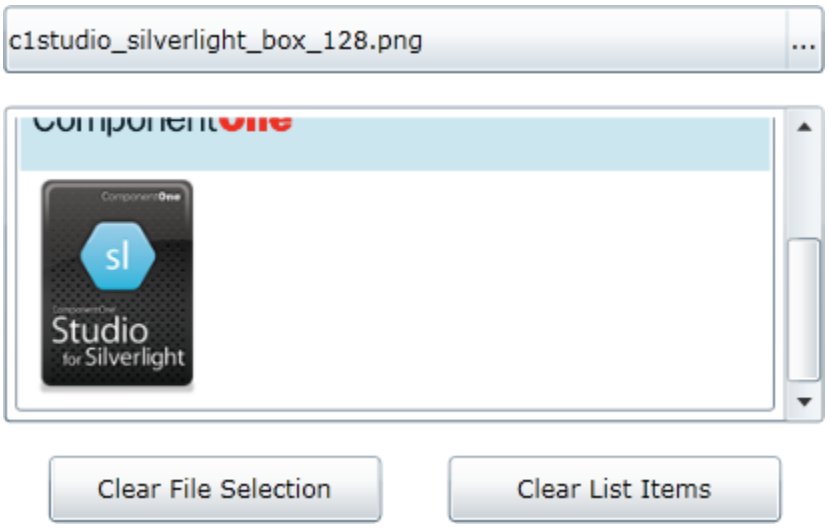
4. The selected image will appear in the list box and the name of the file will appear in the **C1FilePicker** control:



5. Click the **Clear File Selection** button. Notice that the name of the file no longer appears in the **C1FilePicker** control:



6. Click the ellipses button on the **C1FilePicker** control again. The **Open** dialog box will appear.
7. In the **Open** dialog box, try to select more than one image. Notice that you cannot. That is because the Multiselect property is sent to **False** by default. To select multiple files at one, you would need to set the Multiselect property to **True**.
8. Select a file in the **Open** dialog box and click **Open**. Notice that the second file appears listed in the **C1FilePicker** control and has been added to the List Box:



9. Click the **Clear List Items** button; the list of files will be cleared.

What You've Accomplished

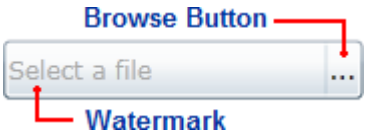
Congratulations, you've completed the **FilePicker for Silverlight** quick start! You've created a simple application that uses **FilePicker for Silverlight** to select image files that are then displayed in another control.

To learn more about the features and functionality of **ComponentOne FilePicker for Silverlight**, see the Working with FilePicker for Silverlight topic. For examples of specific customizations, see the FilePicker for Silverlight Task-Based Help topic.

Elements and Selection

ComponentOne FilePicker for Silverlight provides the ability to select files. Selected files can then be uploaded with **ComponentOne Upload for Silverlight** or used by other controls. The basic C1FilePicker control appears similar to the following image:

The control consists of a box that lists a watermark or the selected file name(s) and a **Browse** button indicated by an **ellipsis**:



Browse Button

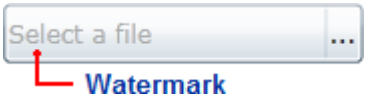
The **Browse** button appears to the right of the C1FilePicker control and appears as an **ellipses** button:

When a user clicks the **Browse** button at run time, the **OpenFileDialog** dialog box will appear. The **OpenFileDialog** allows users to choose one or more files on the local computer or a networked computer that will be added to the SelectedFiles collection. For more information, see the Open File Dialog Box topic.

You can customize the content and appearance of the **Browse** button by setting the BrowseContent property to an object. The object you select will appear as the browse button.

Watermark Text

The watermark appears in the text area of the C1FilePicker control by default. The watermark can give directions or suggestions for users at run time, for example the default watermark text states "Select a file":



Notice that the text appears grayed out. When a file or multiple files are selected the file(s) will appear in this text area instead and appear a darker color to be distinguished from the watermark. If a file is selected and the files are cleared from the C1FilePicker control using the ClearSelection method, the watermark will appear again. See Clearing Selected Files for an example.

You can customize the watermark text by setting the Watermark property to the value you want to appear. For an example, see the Removing the Watermark topic. If you do not want a watermark to appear, you can set the Watermark property to a blank value (for example, a space character).

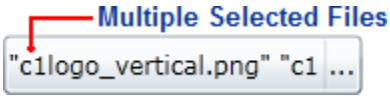
Selected Files

When a file or multiple files are selected the file(s) will appear in this text area instead and appear a darker color to be distinguished from the watermark. For example, when a file is chosen the C1FilePicker control appears similar to the following image:

Note that the name of the selected file appears in the text area of the C1FilePicker control. To get the name of the selected file, you can use the SelectedFile property. To clear selected files, you can use the ClearSelection method. When the selected files are cleared, the watermark will appear displayed in the C1FilePicker control by default. See Clearing Selected Files for an example.

Multiple File Selection

When the Multiselect property is set to **True**, the user can choose multiple files in the **OpenFileDialog** dialog box at run time. When multiple files are selected, they all appear in the text area of the C1FilePicker control, separated by commas, and in quotation marks. For example, two files are selected in the following image:

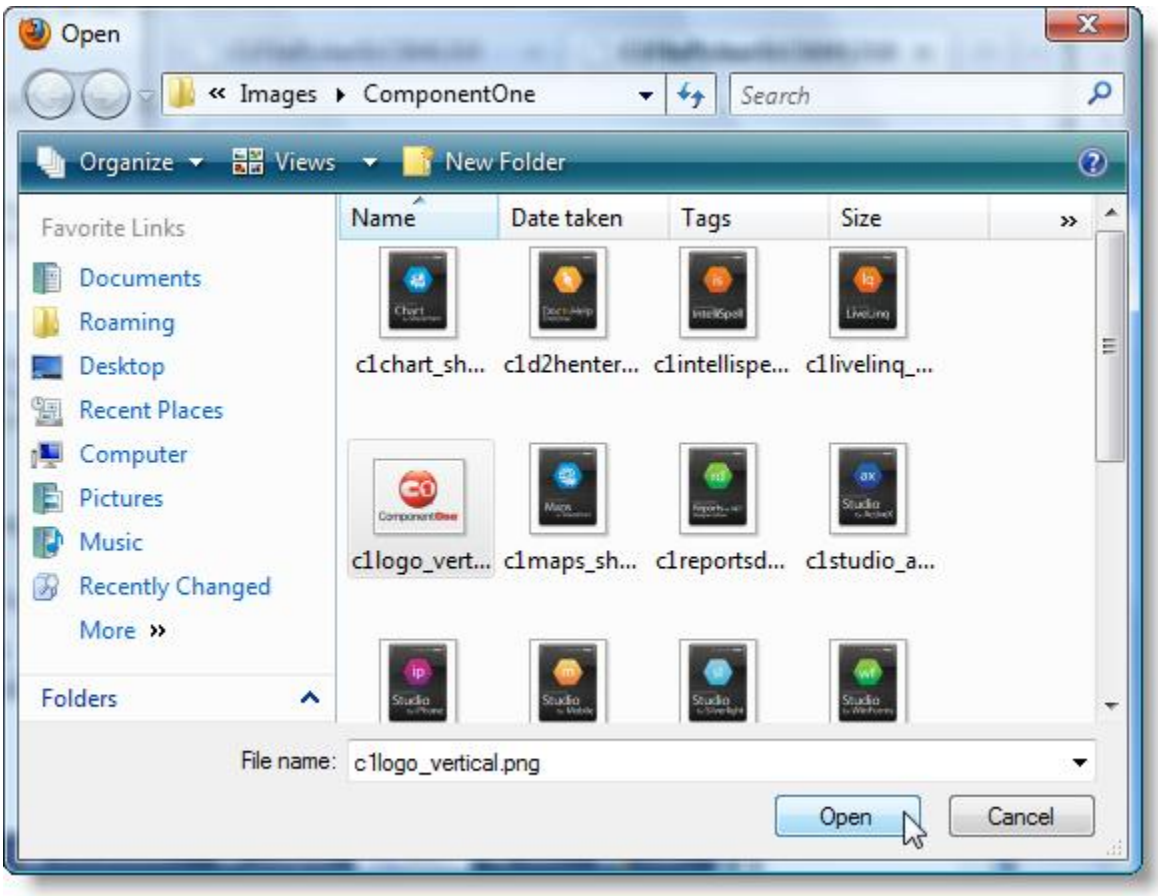


For an example of selecting multiple files, see the Selecting Multiple Files topic. To clear selected files, you can use the ClearSelection method. When the selected files are cleared, the watermark will appear displayed in the C1FilePicker control by default. See Clearing Selected Files for an example.

Open File Dialog Box

ComponentOne FilePicker for Silverlight uses the standard Microsoft **OpenFileDialog** box to enable to users to select files. The **OpenFileDialog** dialog box enables users to open one or more files on a the local computer or a networked computer. Files selected in the **OpenFileDialog** dialog box are then added to the SelectedFiles collection.

The **OpenFileDialog** dialog box appears similar to a traditional dialog box. For example, the dialog box appears similar to the following image:



You can choose to add a files by navigating to the folder where they are located, selecting the files to add, and clicking the **Open** button. To select multiple files in the **OpenFileDialog** dialog box, you must sent the Multiselect property to **True**.

You can apply a filter in the **OpenFileDialog** dialog box so that only files with a certain file extension will appear (so, for example, only image files appear). You can set the Filter property to specify a filter. For more information, see File Filtering, and for an example, see Adding a File Filter.

File Filtering

At run time when users click the **Browse** button and open the **OpenFileDialog** dialog box, they can typically select any type of file to open and no filter options will appear in the **OpenFileDialog** dialog box. You may want to limit the types of files that users can see and select in the **OpenFileDialog** dialog box, for example you might users to only be able to select image files or specific document types. You can customize the file types that users can select by setting the Filter property to a specific filter.

Basically, you can apply a filter in the **OpenFileDialog** dialog box so that only files with a certain file extension will appear. The Filter property functions similarly to the FileDialog.Filter property. You would need to set the Filter property to a filter string. For each filtering option you implement, the filter string contains a description of the filter, followed by the vertical bar (|) and the filter pattern. The strings for different filtering options are separated by the vertical bar.

The following is an example of a filter string:

- Text files (*.txt)|*.txt|All files (*.*)|*.*

You can add several filter patterns to a filter by separating the file types with semicolons, for example:

- Image Files(*.BMP;*.JPG;*.GIF)|*.BMP;*.JPG;*.GIF|All files (*.*)|*.*

You can use the FilterIndex property to set which filtering option is shown first to the user. By default the FilterIndex property is set to "1" and the first filter listed in the filter logic appears first. For example, if you include options for an image filter and for all files, as in the example above, you can show the option for all files first (and have the dialog box appear unfiltered) by setting the FilterIndex property to "2".

For an example of filtering files, see Adding a File Filter.

Working with FilePicker for Silverlight

ComponentOne FilePicker™ for Silverlight provides a simple and reliable way to select files in your Silverlight application. Similar to a combo box in appearance, the **C1FilePicker** control displays a file picker dialog box instead of a drop-down list-down list when the ellipses button is clicked.

Basic Properties

ComponentOne FilePicker for Silverlight includes several properties that allow you to set the functionality of the C1FilePicker control. Some of the more important properties are listed below.

The following properties let you customize the C1FilePicker control:

Property	Description
BrowseContent	Gets or sets the content of the **Browse** button.
DisabledCuesVisibility	Gets a value indicating whether the disabled visuals of the control are visible.
Filter	Gets or sets the filter that will be applied to the **OpenFileDialog** dialog box. See Adding a File Filter for more information.
FilterIndex	Gets or sets the filter index that will be applied to the **OpenFileDialog** dialog box.
FocusCuesVisibility	Gets a value indicating whether the focus visuals of the control are visible.
HasSelectedFiles	True, if files were selected.
Multiselect	Gets or sets whether it's possible to select more than one file. See Selecting Multiple Files for more information.
SelectedFile	Gets the file that the user has selected.
SelectedFiles	Gets the files that the user has selected.
SelectionBackground	Gets or sets the brush that fills the background of the selected text.
SelectionForeground	Gets or sets the brush used for the selected text in the **C1FilePicker**.
TextAlignment	Gets or sets how the text should be aligned in the **C1FilePicker**. See Changing the Text Alignment for an example.
Watermark	Gets or sets the watermark content. See Watermark Text for more information and Removing the Watermark for an example.

Basic Methods

ComponentOne FilePicker for Silverlight includes several methods that allow you to customize the control. Some of the more important methods are listed below.

The following methods let you customize the C1FilePicker control:

Method	Description
ClearSelection	Removes the selected files. See Clearing Selected Files for more information.
OnSelectedFilesChanged	Raises the SelectedFilesChanged event.

Basic Events

ComponentOne FilePicker 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 C1FilePicker control:

Event	Description
SelectedFilesChanged	Fires when the SelectedFile property changes.

FilePicker Layout and Appearance

The following topics detail how to customize the C1FilePicker 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.

FilePicker Appearance Properties

ComponentOne FilePicker 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.

Content Properties

The following properties let you customize the appearance of content in the **C1FilePicker** control:

Property	Description
Watermark	Gets or sets the content of the watermark. See Watermark Text for more information and Removing the Watermark for an example.

Text Properties

The following properties let you customize the appearance of text in the **C1FilePicker** 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.
TextAlignment	Gets or sets how the text should be aligned in the **C1FilePicker** control.

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 **C1FilePicker** 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.

FilePicker Themes

C1FilePicker includes Visual Styles allowing you to easily change the control's appearance. You can easily customize the appearance of a single C1FilePicker control with the themes and you can also change the theme across an application.

FilePicker 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 FilePicker 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 C1FilePicker 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.

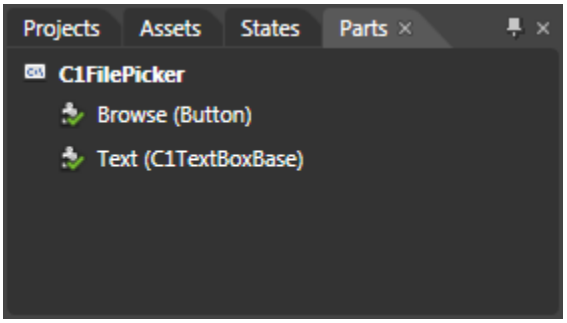
FilePicker Styles

ComponentOne FilePicker for Silverlight's **C1FilePicker** control provides several style properties that you can use to change the appearance of the inner parts 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.
ValidationDecoratorStyle	They style used to display validation errors.

FilePicker Template Parts

In Microsoft Expression Blend, you can view and edit template parts by creating a new template (for example, click the **C1FilePicker** 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.

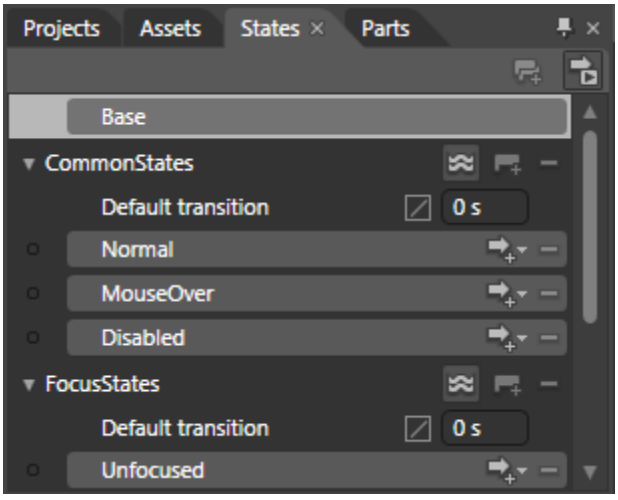
In the Parts window, you can double-click any element to create that part in the template. Once you have done so, the part will appear in the template and the element's icon in the **Parts** pane will change to indicate selection.

Template parts available in the **C1FilePicker** control include:

Name	Type	Description
Browse	Button	Represents a button control used to open the **OpenFileDialog** dialog box.
Text	C1TextBoxBase	Represents the text area of the control, where the selected files are displayed.

FilePicker 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. 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 **Focused** when the control is active and in focus and **Unfocused** for when the control is inactive and not in focus. Validation states include **Valid**, for when

the control's content is valid, **InvalidUnfocused** for when the control's content is not valid and the control is not in focus, and **InvalidFocused** for when the control's content is not valid and the control is in focus.

FilePicker for Silverlight Samples

ComponentOne FilePicker 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
Input Form	This sample demonstrates how to use the **C1FilePicker** component and shows how the **C1FilePicker** control can be used to create an input form. This page is part of the **C1_Demo** sample and is installed by default at **C1.Silverlight\C1_Demo\C1_Demo\C1Inputs\DemoInputs.xaml** in the samples directory.

FilePicker 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 C1FilePicker control in general. If you are unfamiliar with the **ComponentOne FilePicker for Silverlight** product, please see the FilePicker for Silverlight Quick Start first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne FilePicker for Silverlight** product. Most task-based help topics also assume that you have created a new Silverlight project and added the C1FilePicker control to the application.

Removing the Watermark

The watermark appears in the text area of the C1FilePicker control by default and can give directions or suggestions for users at run time. For more information, see the Watermark Text topic. You can customize the watermark text by setting the Watermark property to the value you want to appear.

If you do not want a watermark to appear, you can set the Watermark property to a blank value, for example see the steps below.

At Design Time in Blend

To set the Watermark property in Expression Blend, complete the following steps:

1. Click the C1FilePicker control once to select it.
2. Navigate to the Properties tab and locate the **Watermark** item.
3. Click in the text box next to the **Watermark** item, and delete the current text.

   This will set the Watermark property to a blank value.

In XAML

For example, to set the Watermark property add `Watermark=""` to the `<c1:C1FilePicker>` tag so that it appears similar to the following:

```
<c1:C1FilePicker HorizontalAlignment="Left" Margin="112,36,0,0"
Name="C1FilePicker1" VerticalAlignment="Top" Width="161" Watermark="" />
```

In Code

For example, to set the Watermark property, add the following code to your project:

- Visual Basic

```
Me.C1FilePicker1.Watermark = ""
```

- C#

```
this.c1FilePicker1.Watermark = "";
```

What You've Accomplished

You've removed the default watermark from the C1FilePicker control. Run the application, and observe that the caption bar of the C1FilePicker control will appear without the watermark:

Clearing Selected Files

When a user selects a file at run time using the C1FilePicker control, the file name appears in the text area of the control. If you want to clear all the files selected by the C1FilePicker and remove the selected file names from the control, you can use the ClearSelection method. In this example, you'll add a button to your application to clear the C1FilePicker control.

For example, add a button to your application and in the button's **Click** event handler add the following code:

- Visual Basic

```
C1FilePicker1.ClearSelection()
```

- C#

```
c1FilePicker1.ClearSelection();
```

What You've Accomplished

Run the application, and at run time choose a file in the C1FilePicker control. Click the button that you added and observe that the C1FilePicker control's file selection is cleared.

Selecting Multiple Files

FilePicker for Silverlight allows users to select multiple files at run time, though this option is not selected by default. At run time, users can only select one file at a time by default – you can allow users to select multiple files by setting the Multiselect property to **True**, for example see the steps below.

At Design Time in Blend

To set the Multiselect property in Expression Blend, complete the following steps:

1. Click the C1FilePicker control once to select it.
2. Navigate to the Properties tab and locate the **Multiselect** item.
3. Check the check box next to the **Multiselect** item.

   This will set the Multiselect property to allow users to select multiple files at run time in the **OpenFileDialog** dialog box.

In XAML

For example, to set the Multiselect property add `Multiselect="True"` to the `<c1:C1FilePicker>` tag so that it appears similar to the following:

```
<c1:C1FilePicker HorizontalAlignment="Left" Margin="112,36,0,0"
Name="C1FilePicker1" VerticalAlignment="Top" Width="161"
Multiselect="True" />
```

In Code

For example, to set the Multiselect property, add the following code to your project:

- Visual Basic

```
Me.C1FilePicker1.Multiselect = True
```

- C#

```
this.c1FilePicker1.Multiselect = true;
```

What You've Accomplished

You can now select multiple files using the C1FilePicker control. Run the application, select the **Browse** button in the C1FilePicker control, and observe that you can select multiple files in the dialog box that appears by clicking the CTRL key while choosing files.

Changing the Text Alignment

The C1FilePicker control typically displays the current file selected in the text area of the control. This text is aligned to the left by default, but you can change the text alignment if you choose. Text alignment can be set using the TextAlignment property, and options include **Left** (default), **Center**, or **Right** text alignment. For more information, see the steps below.

At Design Time in Blend

To set the TextAlignment property to **Right** in Expression Blend, complete the following steps:

1. Click the C1FilePicker control once to select it.
2. Navigate to the Properties tab and locate the **TextAlignment** item.
3. Click the drop-down arrow next to the **TextAlignment** item and select **Right**.

In XAML

For example, to set the TextAlignment property to **Right** add `TextAlignment="Right"` to the `<c1:C1FilePicker>` tag so that it appears similar to the following:

```
<c1:C1FilePicker HorizontalAlignment="Left" Margin="112,36,0,0"
Name="C1FilePicker1" VerticalAlignment="Top" Width="161"
TextAlignment="Right" />
```

In Code

For example, to set the TextAlignment property to **Right**, add the following code to your project:

- Visual Basic

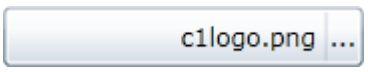
```
Me.C1FilePicker1.TextAlignment = TextAlignment.Right
```

- C#

```
this.c1FilePicker1.TextAlignment = TextAlignment.Right;
```

What You've Accomplished

Text is now aligned to the right. Run the application and select a file. Notice that the name of the file is now aligned to the right in the C1FilePicker control:



Adding a File Filter

At run time when users click the **Browse** button and open the **OpenFileDialog** dialog box, they can typically select any type of file to open. You can apply a filter in the **OpenFileDialog** dialog box so that only files with a certain file extension will appear. So, for example, you can specify a filter so that only image files appear or only text files appear. You can set the Filter property to specify a filter as in the following steps.

At Design Time in Blend

To set the Filter property to filter image files in Expression Blend, complete the following steps:

1. Click the C1FilePicker control once to select it.
2. Navigate to the Properties tab and locate the **Filter** item.
3. In the text box next to the **Filter** item and enter "Image Files(*.PNG;*.JPG;*.GIF)|*.PNG;*.JPG;*.GIF|All files (*.*)|*.*".

In XAML

For example, to set the Filter property to filter image files add `Filter="Image Files(*.PNG;*.JPG;*.GIF)|*.PNG;*.JPG;*.GIF|All files (*.*)|*.*"` to the `<c1:C1FilePicker>` tag so that it appears similar to the following:

```
<c1:C1FilePicker HorizontalAlignment="Left" Margin="112,36,0,0"
Name="C1FilePicker1" VerticalAlignment="Top" Width="161" Filter="Image
Files(*.PNG;*.JPG;*.GIF)|*.PNG;*.JPG;*.GIF|All files (*.*)|*.*" />
```

In Code

For example, to set the Filter property to filter image files, add the following code to your project:

- Visual Basic

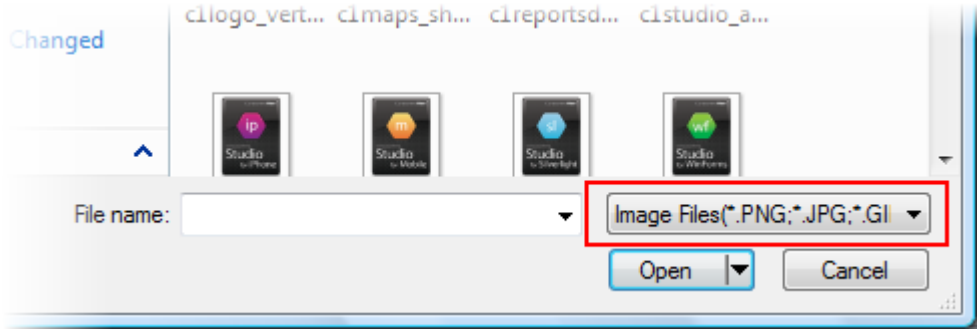
```
Me.C1FilePicker1.Filter = "Image
Files(*.BMP;*.JPG;*.GIF)|*.PNG;*.JPG;*.GIF|All files (*.*)|*.*"
```

- C#

```
this.c1FilePicker1.Filter = "Image
Files(*.BMP;*.JPG;*.GIF)|*.PNG;*.JPG;*.GIF|All files (*.*)|*.*";
```

What You've Accomplished

A filter will be applied to the **OpenFileDialog** dialog box. Run the application and click the **Browse** button in the C1FilePicker control. Notice that a filter has been applied so that only files with the PNG, JPG, and GIF extensions appear in the dialog box:



If you click the filter box's drop-down arrow you can choose to display all files. That is because the "All files" option was included in the filter logic applied in the steps above.

HeaderContent

Easily layout and display blocks of content with ComponentOne **HeaderContent™ for Silverlight**. This control is comprised of two elements: a header bar and a content panel. The header can be horizontal or vertical and the content panel can be located on either side of the header.

Getting Started

- C1HeaderedContentControl Elements
- Quick Start
- Task-Based Help

HeaderContent for Silverlight Key Features

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

- **Implement Frame-like Scenarios**

  With C1HeaderedContentControl you can easily reuse the style of your containers in your application. Just define a new template for the C1HeaderedContentControl and use it everywhere. This saves you time because you don't have to copy and paste all the visual elements composing your custom container again and again.

- **Group Fields**

  Using the C1HeaderedContentControl you can improve usability by categorizing similar fields in Forms.

- **Silverlight Toolkit Themes Support**

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

HeaderContent for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **HeaderContent for Silverlight**. In this quick start, you'll start in Expression Blend to create a new project with a C1HeaderedContentControl 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 3: Creating an Application with a C1HeaderedContentControl Control

In this step, you'll begin in Expression Blend to create a Silverlight application using **HeaderContent for Silverlight**.

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 the C1HeaderedContentControl control to your project by completing the following steps:
    a. On the menu, select **Window | Assets** to open the **Assets** tab.
    b. Under the **Assets** tab, enter "C1HeaderedContentControl" into the search bar.

       The C1HeaderedContentControl control's icon appears.
    c. Double-click the C1HeaderedContentControl icon to add the control to your project.

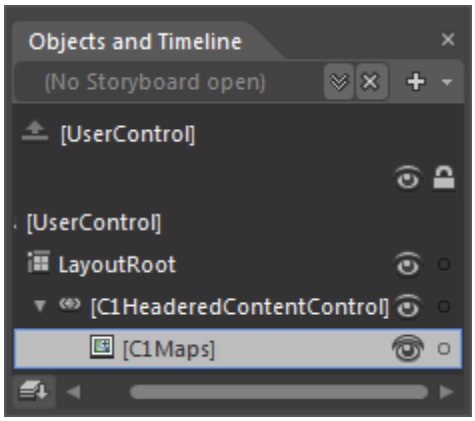
In this step, you created a Silverlight project containing a C1HeaderedContentControl control. In the next step, you'll customize the C1HeaderedContentControl control.

Step 2 of 3: Customizing the C1HeaderedContentControl Control

In the last step, you created a Silverlight project containing a C1HeaderedContentControl control. In this step, you will customize the C1HeaderedContentControl control by setting a few of its properties and adding content to its content panel.

Complete the following steps:

1. Select the C1HeaderedContentControl and complete the following in the **Properties** panel:
    - Set the **BorderBrush** property's hex value to "#FF198315".
    - Set the **HeaderForeground** property's hex value to "#FF198315".
    - Set the **BorderThickness** property to "4, 4, 4, 4".
    - Set the **Header** property to "Map of the World".
    - Set the **HeaderFontFamily** to **Arial**.
    - Set the **HeaderFontSize** to "15".
    - Set the **HeaderHorizontalContentAlignment** property to **Center**.
2. Add a **C1Maps** control to the content area by completing the following steps:
    a. In the Assets panel, enter "C1Maps" into the search bar. The **C1Maps** icon appears.
    b. Double-click the **C1Maps** icon to add the **C1Maps** control to your project.
    c. In the **Objects and Timeline** panel, select **[C1Maps]** and then, using a drag-and-drop operation, place it underneath **[C1HeaderedContentControl]** so it looks as follows:



In this step, you customized the C1HeaderedContentControl control. In the next step, you'll run the project and see the results of the quick start.

Step 3 of 3: Running the Project

In the previous two steps, you created a Silverlight project containing a C1HeaderedContentControl control and then customized the C1HeaderedContentControl control. In this step, you'll run the project and see the result of this quick start.

Complete the following steps:

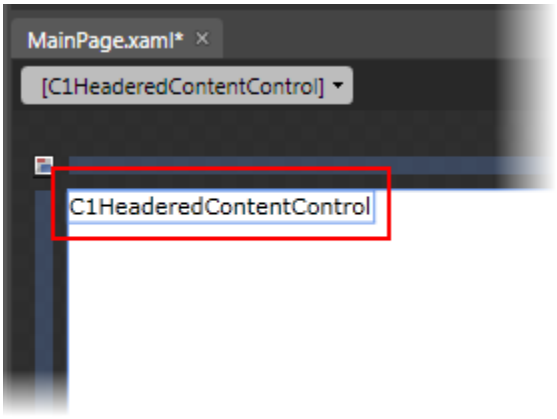
1. Press F5 to run the project and observe that the C1HeaderedContentControl appears as follows:

2. Click the **C1Maps** control's down arrow and observe that the **C1Maps** control moves independent of the header.

Congratulations! You have completed the **HeaderContent for Silverlight** quick start. To learn more about the control, see the C1HeaderedContentControl Elements and the HeaderContent for Silverlight Task-Based Help topics.

C1HeaderedContentControl Elements

The C1HeaderedContentControl control is a **HeaderedContentControl** control that provides an expandable and collapsible pane for storing text, images, and controls. When you add the C1HeaderedContentControl control to a project, the control contains a default Content string and no header. When you add a C1HeaderedContentControl control to an Expression Blend project, it will look as follows:



After the C1HeaderedContentControl control is added to your project, you can add your own header and content to the control. The following topics provide an overview of the C1HeaderedContentControl control's header bar and content panel.

C1HeaderedContentControl Header

By default, the C1HeaderedContentControl control's header bar contains no text. To add text to the header bar, simply set the **Header** property to a string. Once the text is added, you can style it using several font properties (see Text Properties 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 Bar.

You can adjust the padding of the header using the C1HeaderedContentControl control's HeaderPadding property.

Attribute Syntax versus Property Element Syntax

When you want to add something simple to the C1HeaderedContentControl header, such as an unformatted string, you can simply use the common XML attributes in your XAML markup, such as in the following:

```
<c1ext:C1HeaderedContentControl Header="Hello World"/>
```

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

```
<c1ext:C1HeaderedContentControl ExpandDirection="Down" Width="150"
Height="55" Name="C1HeaderedContentControl1">
            <c1ext:C1HeaderedContentControl.Header>
      <Grid HorizontalAlignment="Stretch">
         <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" />
            <ColumnDefinition Width="*" />
         </Grid.ColumnDefinitions>
            <TextBlock Text="C1HeaderedContentControl Header" />
      </Grid>
            </c1ext:C1HeaderedContentControl.Header>
</c1ext:C1HeaderedContentControl>
```

C1HeaderedContentControl Content Panel

The C1HeaderedContentControl control's content panel initially consists of an empty space. In the content panel, you can add grids, text, images, and arbitrary controls. When working in Blend, elements in the content panel of the control can be added and moved on the control through a simple drag-and-drop operation.

You can add text to the content panel by setting the C1HeaderedContentControl control's **Content** property or by adding a **TextBox** element to the content panel. Adding Silverlight elements to the content panel 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 C1HeaderedContentControl can only accept one child element at a time. However, you can circumvent this issue by adding a panel-based control as the C1HeaderedContentControl 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 C1HeaderedContentControl control, but its ability to hold multiple elements will allow you to show several controls in the content panel.

For task-based help about adding content to the content panel, see Adding Content to the content panel.

Attribute Syntax versus Property Element Syntax

When you want to add something simple to the C1HeaderedContentControl content panel, 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:

```
<c1ext:C1HeaderedContentControl Content="Hello World"/>
```

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

```
<c1ext:C1HeaderedContentControl ExpandDirection="Down" Width="150"
Height="55" Name="C1HeaderedContentControl1">
            <c1ext:C1HeaderedContentControl.Content>
                <StackPanel>
                    <TextBlock Text="Hello"/>
                    <TextBlock Text="World"/>
                </StackPanel>
            </c1ext:C1HeaderedContentControl.Content>
</c1ext:C1HeaderedContentControl>
```

HeaderContent for Silverlight Layout and Appearance

The following topics detail how to customize the C1HeaderedContentControl 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 customize the control's actions.

HeaderContent for Silverlight Appearance Properties

ComponentOne HeaderContent 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 C1HeaderedContentControl 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 HeaderedContentControl 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 panel content in the C1HeaderedContentControl 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 Properties

The following properties let you customize the size of the **C1HeaderedContentControl** control.

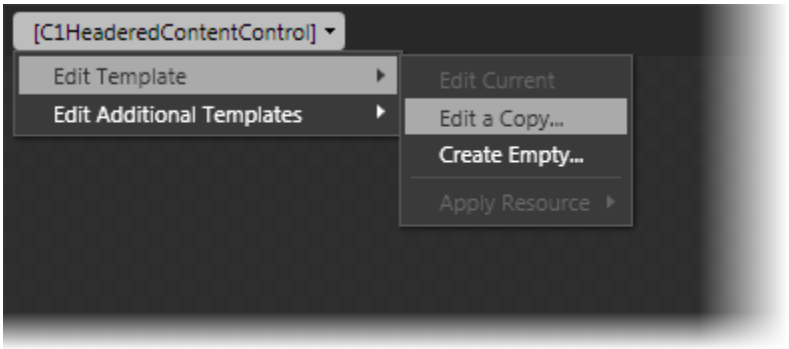
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
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 HeaderContent 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 C1HeaderedContentControl 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 C1HeaderedContentControl control includes a few additional templates. You can access these additional templates in Microsoft Expression Blend. In Blend, select the C1HeaderedContentControl control and, in the menu, select **Edit Additional Templates**. Choose either the **HeaderTemplate** template or the **ContentTemplate** template and select **Create Empty** to create a new blank template.

HeaderContent 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 C1HeaderedContentControl control in general. If you are unfamiliar with the **ComponentOne HeaderContent for Silverlight** product, please see the **HeaderContent for Silverlight** Quick Start first.

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

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

Adding Content to the Header Bar

You can easily add both simple text and Silverlight controls to the C1HeaderedContentControl 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 bar, you can also visit the C1HeaderedContentControl Header topic.

Adding Text to the Header Bar

By default, the C1HeaderedContentControl 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 C1HeaderedContentControl 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 `<c1ext:C1HeaderedContentControl>` tag so that it appears similar to the following:

```
<c1ext:C1HeaderedContentControl Header="Hello World" Width="150"
Height="55">
```

In Code

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

1. Add `x:Name="C1HeaderedContentControl1"` to the `<c1ext:C1HeaderedContentControl>` 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

```
C1HeaderedContentControl1.Header = "Hello World"
```

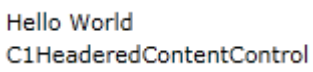
- C#

```
C1HeaderedContentControl1.Header = "Hello World";
```

3. Run the program.

This Topic Illustrates the Following:

The header of the C1HeaderedContentControl control now reads "Hello World". The result of this topic should resemble the following:



Note that the content panel displays the string "C1HeaderedContentControl" by default. To learn how to add different content to the content panel, see Adding Content to the Content Panel.

Adding a Control to the Header

The C1HeaderedContentControl control's header bar 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 `<c1ext:C1HeaderedContentControl>` and `</c1ext:C1HeaderedContentControl>` tags:

```
<c1ext:C1HeaderedContentControl.Header>
    <Button Content="Button" Height="Auto" Width="50"/>
<c1ext:C1HeaderedContentControl.Header>
```

In Code

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

1. Add `x:Name="C1HeaderedContentControl1"` to the `<c1ext:C1HeaderedContentControl>` 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
C1HeaderedContentControl1.Header = (NewButton)
```

   - C#

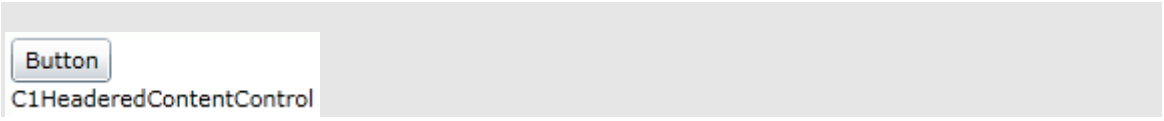
```
//Create the Button control
Button NewButton = new Button();
NewButton.Content = "Button";
```

```
//Set the Button Control's Width and Height properties
NewButton.Width = 50;
NewButton.Height = Double.NaN;
//Add the Button to the header
C1HeaderedContentControl1.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 bar. The final result will resemble the following image:



Adding Content to the Content Panel

You can easily add both simple text and Silverlight controls to the C1HeaderedContentControl control's content panel. 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 bar, you can also visit the C1HeaderedContentControl Content Panel topic.

Adding Text to the Content Panel

You can easily add a simple line of text to the content panel of the C1HeaderedContentControl 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 panel by adding a **TextBox** control to the content panel and then setting the **TextBox** control's **Text** property. To learn how to add a control to the content panel, see Adding a Control to the content panel.

At Design Time in Blend

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

1. Click the C1HeaderedContentControl 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 C1HeaderedContentControl control.

In XAML

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

1. Add `Content="Hello World"` to the `<c1ext:C1HeaderedContentControl>` tag so that it appears similar to the following:

```
      <c1ext:C1HeaderedContentControl Content="Hello World" Width="150"
Height="55">
```

2. Run the program and then expand the C1HeaderedContentControl control.

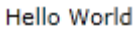
In Code

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

1. Add `x:Name="C1HeaderedContentControl1"` to the `<c1ext:C1HeaderedContentControl>` 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
     ```
     C1HeaderedContentControl1.Content = "Hello World"
     ```
   - C#
     ```
     C1HeaderedContentControl1.Content = "Hello World";
     ```
3. Run the program and then expand the C1HeaderedContentControl control.

This Topic Illustrates the Following:

When the C1HeaderedContentControl control is expanded, it reads "Hello World". The end result of this topic should resemble the following:



Note that there isn't a header; this is because the Header property isn't set by default. To learn how to add text or controls to the header bar, see Adding Content to the Header Bar.

Adding a Control to the Content Panel

The C1HeaderedContentControl control will accept one child control in its content panel. 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 panel, 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 panel of the C1HeaderedContentControl 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 C1HeaderedContentControl control's content panel.
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 C1HeaderedContentControl control's content panel.
7. Run the program and then expand the C1HeaderedContentControl control.

In XAML

To add a button control to the C1HeaderedContentControl control's content panel in XAML, complete the following steps:

1. Place the following markup between the `<c1ext:C1HeaderedContentControl>` and `</c1ext:C1HeaderedContentControl>` tags:
   ```
   <Button Content="Button" Height="Auto" Width="Auto"/>
   ```

2. Run the program and then expand the C1HeaderedContentControl control.

In Code

To add a button control to the C1HeaderedContentControl control's content panel in code, complete the following:

1. Add `x:Name="C1HeaderedContentControl1"` to the `<c1ext:C1HeaderedContentControl>` 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 panel
C1HeaderedContentControl1.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 panel
C1HeaderedContentControl1.Content = (NewButton);
```

3. Run the program and then expand the C1HeaderedContentControl control.

This Topic Illustrates the Following:

When the C1HeaderedContentControl control is expanded, the button control appears in its content panel and resembles the following image:

Note that there isn't a header; this is because the Header property isn't set by default. To learn how to add text or controls to the header bar, see Adding Content to the Header Bar.

Adding Multiple Controls to the Content Panel

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 panel of the C1HeaderedContentControl control. When you add multiple controls to the panel-based control, each one will appear within the C1HeaderedContentControl control's content panel.

At Design Time in Blend

To add a control to the content panel, 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 panel of the C1HeaderedContentControl 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 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 second **TextBlock** 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 C1HeaderedContentControl control and observe that each of the three **TextBlock** controls appear in the content panel.

In XAML

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

1. Place the following XAML markup between the `<c1ext:C1HeaderedContentControl>` and `</c1ext:C1HeaderedContentControl>` tags:

```
<c1ext:C1HeaderedContentControl.Content>
    <StackPanel>
        <TextBlock Text="1st TextBlock"/>
        <TextBlock Text="2nd TextBlock"/>
        <TextBlock Text="3rd TextBlock"/>
    </StackPanel>
</c1ext:C1HeaderedContentControl.Content>
```

2. Run the program.
3. Expand the C1HeaderedContentControl control and observe that each of the three **TextBlock** controls appear in the content panel.

In Code

To add multiple controls to the content panel, 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 C1HeaderedContentControl
Dim StackPanel1 As New StackPanel()
c1HeaderedContentControl1.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)
```

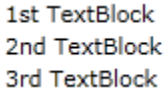
- C#

```
//Create a stack panel and add it to C1HeaderedContentControl
StackPanel StackPanel1 = new StackPanel();
c1HeaderedContentControl1.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 C1HeaderedContentControl control and observe that each of the three **TextBlock** controls appear in the content panel.

This Topic Illustrates the Following:

When the C1HeaderedContentControl control is expanded, three **TextBlock** controls will appear in the content panel as follows:



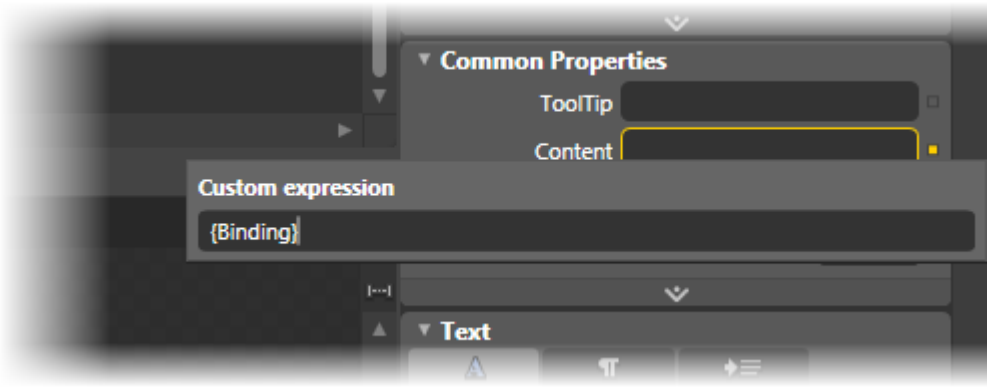
Binding Data to the Header and Content Panel Using Templates

In this topic, you will learn how to bind data to the C1HeaderedContentControl 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 C1HeaderedContentControl Control to the Project and Prepare it for Data Binding

Complete the following steps:

1. Add a C1HeaderedContentControl control to your Silverlight project.
2. Select the C1HeaderedContentControl 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 create the collection that will set the **Name** property and the **Age** property:

- Visual Basic

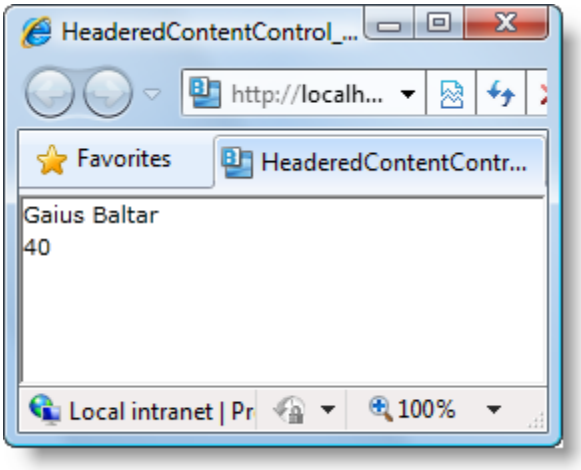
```
NameAgeHolder1.DataContext = New NameAndAge("Gaius Baltar", 40)
```

- C#

```
NameAgeHolder1.DataContext = new NameAndAge("Gaius Baltar", 40);
```

Step 4: Run the Project and Observe the Results

Press F5 to run the project and observe that the values of the **Name** and **Age** properties appear in the control:



HyperPanel

ComponentOne HyperPanel™ for Silverlight is a StackPanel that provides an automatic zoom effect for items near the mouse. You can place any elements in the panel to achieve carousel-like effects and display a large number of elements in a relatively small container, without using scrollbars. The resulting effect is similar to the system toolbar (dock) seen in Mac OS X.

Getting Started

Get started with the following topics:

- Key Features
- Quick Start
- Task-Based Help

HyperPanel for Silverlight Key Features

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

- **Dynamic Zooming**

  ComponentOne HyperPanel™ for Silverlight allows you to display a large number of items in a small space. Items far from the mouse are shrunk and don't take up much space.

- **Control the Zoom Effect**

  Determine how much zooming should be applied when the mouse moves over the panel. Use the Distribution property to control the strength of the zoom effect.

- **Limit the Zoom Effect**

  Use the **MinElementScale** property to prevent items from getting too small.

- **Control Item Opacity**

  Use the **ApplyOpacity** property to make elements near the mouse opaque. Items far from the mouse become more transparent, conveying an idea of distance.

HyperPanel for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **HyperPanel for Silverlight**. In this quick start you'll start in Visual Studio and create a new project, add a **HyperPanel for Silverlight** panel to your application, add controls within C1HyperPanel, and customize the appearance and behavior of the panel.

You will create a simple application using a C1HyperPanel panel that contains several items. You'll then customize the C1HyperPanel panel's appearance and behavior settings to explore the possibilities of using **HyperPanel for Silverlight**.

Step 1 of 3: Setting up the Application

In this step you'll begin in Visual Studio to create a Silverlight application using **HyperPanel for Silverlight**. When you add a C1HyperPanel panel to your application and items to the panel you'll have a completely interactive way of viewing and selecting items.

To set up your project, 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 `Width="400" Height="300"` to `Width="Auto" Height="Auto"` in the `<UserControl>` tag so that it appears similar to the following:

```
<UserControl x:Class="C1HyperPanel.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="Auto"
Height="Auto">
```

   The **UserControl** will now resize to accommodate any content placed within it.
5. In the XAML window of the project, add the follow XAML markup just under the `<Grid>` tag:

   - XAML to Add

```
<Grid.Resources>
    <ResourceDictionary>
        <Style x:Key="letterStyle" TargetType="Border">
            <Setter Property="Height" Value="60" />
            <Setter Property="Width" Value="60" />
            <Setter Property="Margin" Value="2" />
            <Setter Property="CornerRadius" Value="3" />
            <Setter Property="BorderThickness" Value="4" />
            <Setter Property="BorderBrush" >
                <Setter.Value>
                    <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                        <GradientStop Color="#FFBDBDBD"/>
                        <GradientStop Color="#FFDADADA" Offset="0.5"/>
                        <GradientStop Color="#FFBDBDBD" Offset="1"/>
                    </LinearGradientBrush>
                </Setter.Value>
            </Setter>
        </Style>
        <DataTemplate x:Key="letterTemplate" >
            <Border Background="Blue" Style="{StaticResource
letterStyle}" >
                <TextBlock Text="{Binding}"
HorizontalAlignment="Center" VerticalAlignment="Center" FontSize="32"
TextAlignment="Center" TextWrapping="Wrap" >
                    <TextBlock.Foreground>
                        <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                            <GradientStop Color="BlueViolet"/>
                            <GradientStop Color="Thistle"
Offset="0.4"/>
                            <GradientStop Color="AntiqueWhite"
Offset="0.6"/>
                            <GradientStop Color="White" Offset="1"/>
                        </LinearGradientBrush>
                    </TextBlock.Foreground>
                </TextBlock>
            </Border>
        </DataTemplate>
    </ResourceDictionary>
```

```
</Grid.Resources>
```

This will add a style to format the content you will add to the C1HyperPanel panel.

6. In the XAML window of the project, place the cursor just after the `</Grid.Resources>` tag 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.
7. Navigate to the Toolbox and double-click the **C1HyperPanel** icon to add the panel to the project.
8. Give your control a name by adding `x:Name="c1hp1"` to the `<c1:C1HyperPanel>` tag so that it appears similar to the following:
```
<c1:C1HyperPanel x:Name="c1hp1"></c1:C1HyperPanel>
```
By giving it a unique identifier, you'll be able to access the control in code.
9. Resize your control by adding `Width="40" Height="Auto" Width="Auto"` to the `<c1:C1HyperPanel>` tag so that it appears similar to the following:
```
<c1:C1HyperPanel x:Name="c1hp1" Height="Auto"
Width="Auto"></c1:C1HyperPanel>
```
Your control will now fill the page and your complete markup will appear like the following:

- XAML Markup

```
<UserControl xmlns:c1="clr-
namespace:C1.Silverlight;assembly=C1.Silverlight"
x:Class="C1HyperPanel.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="Auto" Height="Auto">
    <Grid x:Name="LayoutRoot" Background="White">
        <Grid.Resources>
            <ResourceDictionary>
                <Style x:Key="letterStyle" TargetType="Border">
                    <Setter Property="Height" Value="60" />
                    <Setter Property="Width" Value="60" />
                    <Setter Property="Margin" Value="2" />
                    <Setter Property="CornerRadius" Value="3" />
                    <Setter Property="BorderThickness" Value="4" />
                    <Setter Property="BorderBrush" >
                        <Setter.Value>
                            <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                                <GradientStop Color="#FFBDBDBD"/>
                                <GradientStop Color="#FFDADADA"
Offset="0.5"/>
                                <GradientStop Color="#FFBDBDBD"
Offset="1"/>
                            </LinearGradientBrush>
                        </Setter.Value>
                    </Setter>
                </Style>
                <DataTemplate x:Key="letterTemplate" >
                    <Border Background="Blue" Style="{StaticResource
letterStyle}" >
                        <TextBlock Text="{Binding}"
HorizontalAlignment="Center" VerticalAlignment="Center" FontSize="32"
TextAlignment="Center" TextWrapping="Wrap" >
                    <TextBlock.Foreground>
```

```
                            <LinearGradientBrush EndPoint="0.5,1"
StartPoint="0.5,0">
                                  <GradientStop Color="BlueViolet"/>
                                  <GradientStop Color="Thistle"
Offset="0.4"/>
                                  <GradientStop Color="AntiqueWhite"
Offset="0.6"/>
                                  <GradientStop Color="White" Offset="1"/>
                            </LinearGradientBrush>
                        </TextBlock.Foreground>
                             </TextBlock>
                         </Border>
                    </DataTemplate>
                </ResourceDictionary>
            </Grid.Resources>
            <c1:C1HyperPanel x:Name="c1hp1" Height="Auto"
Width="Auto"></c1:C1HyperPanel>
        </Grid>
</UserControl>
```

You've successfully created a Silverlight application and added a C1HyperPanel panel to the application. In the next step you'll add controls to and customize C1HyperPanel.

Step 2 of 3: Adding Content to the Panel

In the previous step you created a new Silverlight project and added a C1HyperPanel panel to the application. In this step you'll continue by adding controls to the panel.

Complete the following steps:

1. In XAML view, place the cursor just between the `<c1:C1HyperPanel>` and `</c1:C1HyperPanel>` tags; now when you add items they will be added inside the panel.
2. Navigate to the Toolbox and double-click the **ContentControl** icon to add the control to C1HyperPanel. That's all you have to do to add an item to a C1HyperPanel panel – you can items as you would normally to other panels include the **Grid** and **Canvas**.

   Note that you can also simply add the XAML markup in the next step.
3. Switch to XAML view and update the **ContentControl**'s markup so that it appears similar to the following:

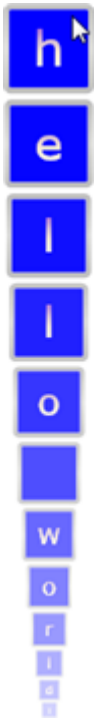
   ```
   <ContentControl Content="h" ContentTemplate="{StaticResource
   letterTemplate}"/>
   ```
4. Enter the following XAML under the **ContentControl**'s markup and within the C1HyperPanel to add additional controls to the panel:

   - XAML to Add

     ```
     <ContentControl Content="e" ContentTemplate="{StaticResource
     letterTemplate}"/>
     <ContentControl Content="l" ContentTemplate="{StaticResource
     letterTemplate}"/>
     <ContentControl Content="l" ContentTemplate="{StaticResource
     letterTemplate}"/>
     <ContentControl Content="o" ContentTemplate="{StaticResource
     letterTemplate}"/>
     <ContentControl Content=" " ContentTemplate="{StaticResource
     letterTemplate}"/>
     <ContentControl Content="w" ContentTemplate="{StaticResource
     letterTemplate}"/>
     ```

```
<ContentControl Content="o" ContentTemplate="{StaticResource
letterTemplate}"/>
<ContentControl Content="r" ContentTemplate="{StaticResource
letterTemplate}"/>
<ContentControl Content="l" ContentTemplate="{StaticResource
letterTemplate}"/>
<ContentControl Content="d" ContentTemplate="{StaticResource
letterTemplate}"/>
<ContentControl Content="!" ContentTemplate="{StaticResource
letterTemplate}"/>
```

5. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time. It will appear similar to the following:



Notice that if you move your mouse over the items in the panel they appear to move.

You've successfully created a Silverlight application and added C1HyperPanel and controls to the application. In the next step you'll customize C1HyperPanel.

Step 3 of 3: Customizing the Application

In the previous steps you created a new Silverlight project and added a C1HyperPanel panel and several **ContentControl**s to the application. In this step you'll continue by setting properties to customize those controls.

Complete the following steps:

1. In XAML view, set the C1HyperPanel control's orientation by adding `Orientation="Horizontal"` to the `<c1:C1HyperPanel>` tag so that it appears similar to the following:

```
<c1:C1HyperPanel x:Name="c1hp1" Height="Auto" Width="Auto"
Orientation="Horizontal">
```

The Orientation property determines if items in the panel are displayed horizontally or vertically. By default Orientation is set to **Vertical** and the panel displays content vertically; setting Orientation property to **Horizontal** will display content horizontally.

2. Set the C1HyperPanel control's distribution by adding `Distribution="0.2"` to the `<c1:C1HyperPanel>` tag so that it appears similar to the following:

```
<c1:C1HyperPanel x:Name="c1hp1" Height="Auto" Width="Auto"
Orientation="Horizontal" Distribution="0.2">
```

The Distribution property consists of a number between .1 and 1.0 and controls how elements are zoomed near the center of the panel. The smaller the value, the more visible the zoom effect. By default, the property is set to "0.5". Setting Distribution to ".02" will cause elements in the center to appear even more zoomed in then elements at the edge of the panel.

3. Set the C1HyperPanel control's scale by adding `MinElementScale="0.5"` to the `<c1:C1HyperPanel>` tag so that it appears similar to the following:

```
<c1:C1HyperPanel x:Name="c1hp1" Height="Auto" Width="Auto"
Orientation="Horizontal" Distribution="0.2" MinElementScale="0.5">
```

The MinElementScale property consists of a number between 0 and 1.0 and determines how small elements near the edge of the panel will appear when compared to elements near the center. By default, the property is set to "0". Setting MinElementScale will ensure that elements near the edge of the panel will appear half their original size at the smallest.

4. Set the C1HyperPanel control's center by adding `Center="0.1"` to the `<c1:C1HyperPanel>` tag so that it appears similar to the following:

```
<c1:C1HyperPanel x:Name="c1hp1" Height="Auto" Width="Auto"
Orientation="Horizontal" Distribution="0.2" MinElementScale="0.5"
Center="0.1">
```

The Center property consists of a number between 0 and 1.0 and sets where the center, or the most zoomed in element, of the panel is when the application is initially run. By default, the property is set to "0.5" and the center is in the middle. Setting Center will move the initially zoomed element to the left side of the control. This value is updated automatically as the mouse moves over the panel at run time.

5. Add the following markup just after the `<C1HyperPanel>` tag and before the content elements:

```
<c1:C1HyperPanel.Background>
    <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
        <GradientStop Color="#FF000000" Offset="0"/>
        <GradientStop Color="#FFFF00DF" Offset="1"/>
    </LinearGradientBrush>
</c1:C1HyperPanel.Background>
```

This markup will add a gradient background to the C1HyperPanel.

6. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time.

The application will initially appear similar to the following:

7. Move your mouse in the panel and notice that the Center of the content changes:

Congratulations! You've completed the **HyperPanel for Silverlight** quick start and created a **ComponentOne HyperPanel for Silverlight** application, customized the appearance and behavior of the controls, and viewed some of the run-time capabilities of your application.

Working with HyperPanel for Silverlight

ComponentOne HyperPanel for Silverlight includes the C1HyperPanel panel, a simple **StackPanel** which provides an automatic zoom effect for items near the mouse. When you add the C1HyperPanel panel to a XAML window, it exists as an empty panel where you can add controls and content as you would to any other panel, the **Grid** or **Canvas**.

Basic Properties

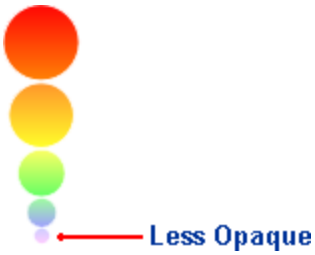
ComponentOne HyperPanel for Silverlight includes several properties that allow you to set the functionality of the panel. Some of the more important properties that vary from the standard **StackPanel** properties are listed below.

The following properties let you customize the C1HyperPanel panel:

Property	Description
ApplyOpacity	Gets or sets a Boolean value that determines if opacity is applied to elements away from the center of the panel.
Distribution	Gets or sets a value between 0.1 and 1.0 that controls how much zooming should be applied to elements near the center.
HorizontalContentAlignment	Gets or sets the horizontal alignment of the panel's content.
MinElementScale	Gets or sets a value between zero and one that determines the minimum scale to be applied to elements when they are away from the center.
Orientation	Gets or sets a value that indicates the dimension by which child elements are stacked.
VerticalContentAlignment	Gets or sets the vertical alignment of the panel's content.

Edge Opacity

The ApplyOpacity property lets you control how items further from the Center of the panel appear. If ApplyOpacity is **True** (default) then items further away from the center will be less opaque:



If ApplyOpacity is **False** then items further away from the center will appear at the same opacity level as the item in the center:

Horizontal and Vertical Orientation

The Orientation property determines how content is laid out within the C1HyperPanel panel. By default, Orientation is set to **Vertical** and content appears stacked from top to bottom vertically in the panel:

When Orientation is set to **Horizontal** content will appear stacked from left to right horizontally in the panel:

Alignment

The **HorizontalAlignment** and **VerticalAlignment** properties control how the C1HyperPanel panel is aligned in the containing panel or window. By default, both properties are set to **Stretch** and the panel is stretched to fill the entire space available.

HorizontalAlignment options include **Left**, **Center**, **Right**, and **Stretch**. If, for example, **HorizontalAlignment** was set to **Right**, the panel would appear at the right of the available area like in the following image:

VerticalAlignment options include **Top**, **Center**, **Bottom**, and **Stretch**. If, for example, **VerticalAlignment** was set to **Top**, the panel would appear at the top of the available area like in the following image:

Content Alignment

The HorizontalContentAlignment and VerticalContentAlignment properties control how content in the C1HyperPanel panel is aligned within the panel. By default, both properties are set to **Center** and content is centered within the panel.

HorizontalContentAlignment options include **Left**, **Center**, **Right**, and **Stretch**. VerticalContentAlignment options include **Top**, **Center**, **Bottom**, and **Stretch**. If, for example, HorizontalContentAlignment was set to **Left** and VerticalContentAlignment was set to **Bottom**, the panel would appear in the bottom-left corner of the panel like in the following image:

Distribution

The Distribution property gets or sets a value between 0.1 and 1.0 that controls how much zooming should be applied to elements near the center of the panel. The smaller the value, the more distant items away from the center of the panel will appear. By default Distribution is set to "0.5".

If Distribution is set to "1", all elements will appear at the same zoom level, for example in the following image:

If Distribution is set to a smaller number, such as "0.2", elements away from the center will appear further zoomed out:

Scale

The MinElementScale property gets or sets a value that determines the minimum scale to be applied to elements when they are away from the center. If MinElementScale is set to "0.1", items furthest away from the center will appear at smallest 10% of their original size.

By default, MinElementScale is set to "0" and items furthest away from the center of the panel appear completely zoomed out. The larger the number, the more zoomed in elements further from the center appear. For example, when MinElementScale is set to "1" as in the image below, all items appear to be the same distance away and elements appear at 100% of their original size. When the mouse is moved over elements in the panel they are not zoomed in or out at all and appear static:

HyperPanel 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 C1HyperPanel panel in general. If you are unfamiliar with the **ComponentOne HyperPanel for Silverlight** product, please see the HyperPanel for Silverlight Quick Start first.

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

Each task-based help topic also assumes that you have created a new Silverlight project and added a C1HyperPanel panel to your project.

Adding a Control to the Panel

By default the C1HyperPanel panel appears empty and contains no content. Adding controls and other content to the panel is as easy as adding content to any other panel, **Canvas**, or **Grid**. In the following steps you'll add a button to your C1HyperPanel.

At Design Time in Blend

To add a button to C1HyperPanel design time in Microsoft Expression Blend, complete the following:

1. Double-click the C1HyperPanel panel in the **Objects and Timelines** pane to select it.
2. If the **Button** option is not displayed in the Toolbox, right-click the **Common Controls** button and select the **Button** option. The **Button** option will be displayed.
3. Double-click the **Button** item in the Toolbox to add it to the **C1HyperPanel** control.

In XAML

To add a button to C1HyperPanel add the `<Button>` tag after the `<c1:C1HyperPanel>` tag so that it appears similar to the following:

```
<c1:C1HyperPanel Name="C1HyperPanel1">
    <Button Height="50" Name="button1" Width="50"></Button>
</c1:C1HyperPanel>
```

In Code

To add a button to C1HyperPanel, switch to Code view and add the following **UserControl_Loaded** event handler and code so it appears like the following:

- Visual Basic

```
Private Private Sub UserControl_Loaded(ByVal sender As System.Object,
ByVal e As System.Windows.RoutedEventArgs)
    Dim button1 = New Button
    button1.Height = 50
```

```
    button1.Width = 50
    Me.C1HyperPanel1.Children.Add(button1)
End Sub
```

- C#

```
private void UserControl_Loaded(object sender, RoutedEventArgs e)
{
    Button button1 = new Button();
    button1.Height = 50;
    button1.Width = 50;
    this.c1HyperPanel1.Children.Add(button1);
}
```

Run your project and observe:

The C1HyperPanel panel will appear with a button:

Changing the Background Color

By default the C1HyperPanel panel appears transparent and all elements below the panel will show through. If you choose, you can set the color of the panel using the **Background** property. For example, the following steps will detail how to change the color of the background to black.

At Design Time in Blend

To change **C1HyperPanel**'s background color to black in Microsoft Expression Blend, complete the following:

1. Click once on the **C1HyperPanel** to select it.
2. Navigate to the Properties tab and click the **Background** item.
3. Click the Solid Color brush tab, and choose **Black** in the color picker, or enter " #FF000000" in the **Hex value** text box.

In XAML

To change **C1HyperPanel**'s background color to black in XAML add `Background="Black"` to the `<c1:C1HyperPanel>` tag so that it appears similar to the following:

```
<c1:C1HyperPanel Name="C1HyperPanel1" Background="Black">
```

In Code

For example, to change the background color add the following code to your project:

- Visual Basic

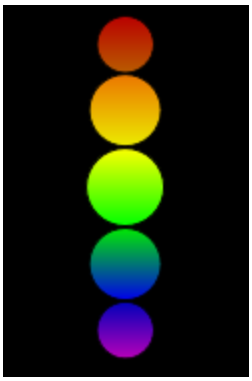
```
Me.C1HyperPanel1.Background = New
System.Windows.Media.SolidColorBrush(Colors.Black)
```

- C#

```
this.c1HyperPanel1.Background = new System.Windows.Media.
SolidColorBrush(Colors.Black);
```

Run your project and observe:

The C1HyperPanel panel will appear with a black background:



Changing the Scale

The MinElementScale property determines how small elements farthest away from the Center of the C1HyperPanel will appear. For example, using MinElementScale you can choose to set all elements to the same size, which would remove the zoom effect.

At Design Time in Blend

To set all C1HyperPanel elements to the same size at design time in Microsoft Expression Blend, complete the following:

1. Click once on the **C1HyperPanel** to select it.
2. Navigate to the Properties tab and locate the MinElementScale property.
3. Click in the text box next to the MinElementScale property and enter "1".

In XAML

To set all C1HyperPanel elements to the same size in XAML add `MinElementScale="1"` to the `<c1:C1HyperPanel>` tag so that it appears similar to the following:

```
<c1:C1HyperPanel Name="C1HyperPanel1" MinElementScale="1">
```

In Code

To set all C1HyperPanel elements to the same size in code, add the following code to your project:

- Visual Basic
```
Me.C1HyperPanel1.MinElementScale = "1"
```
- C#
```
this.c1HyperPanel1.MinElementScale = "1";
```

Run your project and observe:

All items in the C1HyperPanel panel will appear the same size and when you mouse over items there will be no zoom effect:

Setting the Orientation

By default the Orientation property is set to **Vertical** and content appears stacked from top to bottom vertically in the panel. If you choose, you can change the Orientation so that content appears stacked horizontally instead.

At Design Time in Blend

To set the Orientation so that content appears stacked horizontally at design time in Microsoft Expression Blend, complete the following:

1. Click once on the **C1HyperPanel** to select it.
2. Navigate to the Properties tab and locate the Orientation property.
3. Click the drop-down arrow next to the Orientation property and choose **Horizontal**.

In XAML

To set the Orientation so that content appears stacked horizontally in XAML add `Orientation="Horizontal"` to the `<c1:C1HyperPanel>` tag so that it appears similar to the following:

```
<c1:C1HyperPanel Name="C1HyperPanel1" Orientation="Horizontal">
```

In Code

To set the Orientation so that content appears stacked horizontally in code, add the following code to your project:

- Visual Basic

```
Me.C1HyperPanel1.Orientation = Orientation.Horizontal
```

- C#

```
this.c1HyperPanel1.Orientation = Orientation.Horizontal;
```

Run your project and observe:

When Orientation is set to **Horizontal** content will appear stacked from left to right horizontally in the panel:

Setting Element Distribution

The Distribution property controls how much zooming should be applied to elements near the center of the panel. The smaller the value, the smaller items away from the center of the panel will appear. In the following steps you'll set the Distribution so that items away from the center appear more zoomed out.

At Design Time in Blend

To set the Distribution so that items away from the center appear more zoomed out at design time in Expression Blend, complete the following:

1. Click once on the **C1HyperPanel** to select it.
2. Navigate to the Properties window and locate the Distribution property.
3. Click in the text box next to the Distribution property and enter "0.2".

In XAML

To set the Distribution so that items away from the center appear more zoomed out in XAML add `Distribution="0.2"` to the `<c1:C1HyperPanel>` tag so that it appears similar to the following:

```
<c1:C1HyperPanel Name="C1HyperPanel1" Distribution="0.2">
```

In Code

To set the Distribution so that items away from the center appear more zoomed out in code, add the following code to your project:

- Visual Basic

```
Me.C1HyperPanel1.Distribution = "0.2"
```

- C#

```
this.c1HyperPanel1.Distribution = "0.2";
```

Run your project and observe:

With the Distribution property set to a smaller number, elements away from the center appear further zoomed out:

Layout Panels

Control the flow and positioning of the content on your Silverlight app with **ComponentOne Layout Panels™ for Silverlight**. Wrap content vertically or horizontally using C1WrapPanel. Dock content along the edges of the panel with C1DockPanel. Display content in a grid using C1UniformGrid.

Layout Panels for Silverlight Features

Wrap Panel

- **Wrap Panel**

  Create flow type layouts that wrap content vertically or horizontally using the C1WrapPanel control.

- **Horizontal/Vertical Flow Layout**

  You can set the dimensions of the panel to determine the orientation of the layout; flow the child elements horizontally or vertically.

Dock Panel

- **Dock Panel**

  Dock content along the edges of the panel with the C1DockPanel control.

- **Dock Elements**

  Dock Panel for Silverlight enables you to dock elements to the top, bottom, left and right.

- **Order the Elements in the Dock Panel**

  Children elements are positioned in the dock panel in the order that they are declared in XAML.

Uniform Grid

- **Uniform Grid Layout**

  Neatly display child elements in columns and rows with **Uniform Grid for Silverlight**.

- **Show/Hide Columns and Rows**

  Using **Uniform Grid for Silverlight**, you can show or hide an entire column or row.

- **Span Columns and Rows**

  By setting **C1UniformGrid's ColumnSpan** property you can span columns. Or, span rows by setting the **RowSpan** property. This resembles the built-in Microsoft Grid.

Layout Panels for Silverlight Quick Starts

A quick start has been created for each control in **Layout Panels for Silverlight**. In each quick start, you'll begin by creating a Silverlight application and then you will add styles for the controls. For **WrapPanel**, you will wrap several HyperlinkButtons. For **DockPanel**, you will create the panel with several elements inside, demonstrating the four docking options for C1DockPanel. For UniformGrid, you will create a grid with three columns and two empty cells in the first row.

Choose one of the following quick starts to get started:

- WrapPanel for Silverlight Quick Start
- DockPanel for Silverlight Quick Start

- UniformGrid for Silverlight Quick Start

WrapPanel for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **WrapPanel for Silverlight**. In this quick start, you'll create a new project in Visual Studio, add styled HyperlinkButtons that can be wrapped, and change the orientation for the buttons.

Step 1 of 3: Creating a Silverlight Application

In this step you'll create a Silverlight application in Visual Studio 2008 using **ComponentOne Layout Panels for Silverlight**.

To set up your project, 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 (in this example, C# is used), and in the templates list select **Silverlight Application**.
3. Enter **C1WrapPanel** as the name of your project and click **OK**. The N**ew Silverlight Application** dialog box will appear.
4. Uncheck the **Host the Silverlight application in a new Web site** box, if necessary, and click **OK**. The **MainPage.xaml** file should open.

In the next step, you'll add, style, and wrap several HyperlinkButtons.

Step 2 of 3: Adding a C1WrapPanel

We're going to use simple HyperlinkButtons to show how content can be wrapped vertically or horizontally. This is the typical scenario to create a TagCloud view; very commonly used in Web applications.

1. First, remove the Grid tags from your project.
2. In the Visual Studio Toolbox, drag a C1WrapPanel control to the page.
3. Place your cursor in between the <c1:C1WrapPanel> tags and add the HyperlinkButtons using the following XAML:

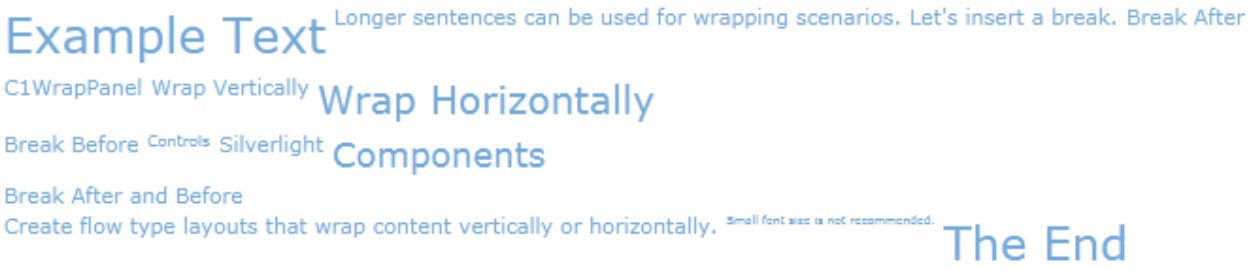
```
            <HyperlinkButton Content="Example Text" FontSize="25" />
            <HyperlinkButton Content="Longer sentences can be used for
wrapping scenarios." />
            <HyperlinkButton Content="Let's insert a break." />
            <HyperlinkButton c1:C1WrapPanel.BreakLine="After"
Content="Break After" />
            <HyperlinkButton Content="C1WrapPanel" />
            <HyperlinkButton Content="Wrap Vertically" />
            <HyperlinkButton Content="Wrap Horizontally" FontSize="20"  />
            <HyperlinkButton c1:C1WrapPanel.BreakLine="Before"
Content="Break Before" />
            <HyperlinkButton Content="Controls" FontSize="8" />
            <HyperlinkButton Content="Silverlight" />
            <HyperlinkButton Content="Components" FontSize="18" />
            <HyperlinkButton c1:C1WrapPanel.BreakLine="AfterAndBefore"
Content="Break After and Before" />
            <HyperlinkButton Content="Create flow type layouts that wrap
content vertically or horizontally." />
            <HyperlinkButton Content="Small font size is not recommended."
FontSize="6" />
            <HyperlinkButton Content="The End" FontSize="24" />
```

In the next step, you'll run the application.

Step 3 of 3: Running the Application

Now you're ready to run the application. Complete the following steps:

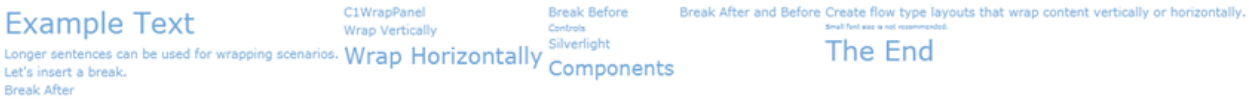
1. From the **Debug** menu, select **Start Debugging**. Your application will look similar to the following:



2. Click the **Stop Debugging** button to close the application.
3. Go back to **MainPage.xaml**. In the <c1:C1WrapPanel> tag, set the **Orientation** property to **Vertical**; the XAML will look like the following:

```
<c1:C1WrapPanel Orientation="Vertical">
```

4. Click **Start Debugging** again in the **Debug** menu. Your application will now look like this:



Notice how the buttons are stacked vertically.

Congratulations! You have successfully completed the **WrapPanel for Silverlight** quick start.

DockPanel for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **DockPanel for Silverlight**. In this quick start, you'll create a new project in Visual Studio, and add elements docked on the top, bottom, left, right, or even fill the C1DockPanel.

Step 1 of 3: Creating a Silverlight Application

In this step you'll create a Silverlight application in Visual Studio 2008 using **ComponentOne DockPanel for Silverlight**.

To set up your project, 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 (in this example, C# is used), and in the templates list select **Silverlight Application**.
3. Enter a **C1DockPanel** as the name of your project and click **OK**. The N**ew Silverlight Application** dialog box will appear.
4. Uncheck the **Host the Silverlight application in a new Web site** box, if necessary, and click **OK**. The **MainPage.xaml** file should open.

In the next step, you'll add and style **C1DockPanels**.

Step 2 of 3: Adding a C1DockPanel

In this step we'll add and style several **C1DockPanels**.

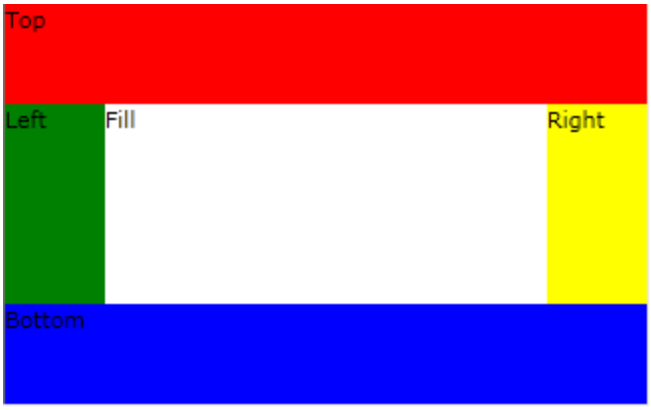
1. First, remove the Grid tags from your project.
2. In the Visual Studio Toolbox, drag a C1DockPanel control to the page.
3. Place your cursor in between the <c1:C1DockPanel> tags and enter the following XAML to add **C1DockPanels** on the left, right, top, and bottom and to fill the center of the screen:

```
<Border c1:C1DockPanel.Dock="Top" Height="50" Background="Red">
   <TextBlock Text="Top" />
</Border>
<Border c1:C1DockPanel.Dock="Bottom" Height="50" Background="Blue">
   <TextBlock Text="Bottom" />
</Border>
<Border c1:C1DockPanel.Dock="Right" Width="50" Background="Yellow">
   <TextBlock Text="Right" />
</Border>
<Border c1:C1DockPanel.Dock="Left" Background="Green" Width="50" >
   <TextBlock Text="Left" />
</Border>
<Border Background="White" >
   <TextBlock Text="Fill" />
</Border>
```

In the next step, you'll run the application.

Step 3 of 3: Running the Application

Now you're ready to run the application. From the **Debug** menu, select **Start Debugging**. Your application will look similar to the following, with four **C1DockPanels** on the top, right, left, and bottom and the center block filled:



Congratulations! You have successfully completed the **DockPanel for Silverlight** quick start.

UniformGrid for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **UniformGrid for Silverlight**. In this quick start, you'll create a new project in Visual Studio, add a C1UniformGrid to your application, set the Columns, FirstColumn, and **Width** properties, and then run the application.

Step 1 of 4: Creating a Silverlight Application

In this step you'll create a Silverlight application in Visual Studio 2008 using **ComponentOne UniformGrid for Silverlight**.

To set up your project, 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 (in this example, C# is used), and in the templates list select **Silverlight Application**.
3. Enter a **C1UniformGrid** as the name of your project and click **OK**. The N**ew Silverlight Application** dialog box will appear.
4. Uncheck the **Host the Silverlight application in a new Web site** box, if necessary, and click **OK**. The **MainPage.xaml** file should open.

In the next step, you'll add, style, and wrap several HyperlinkButtons.

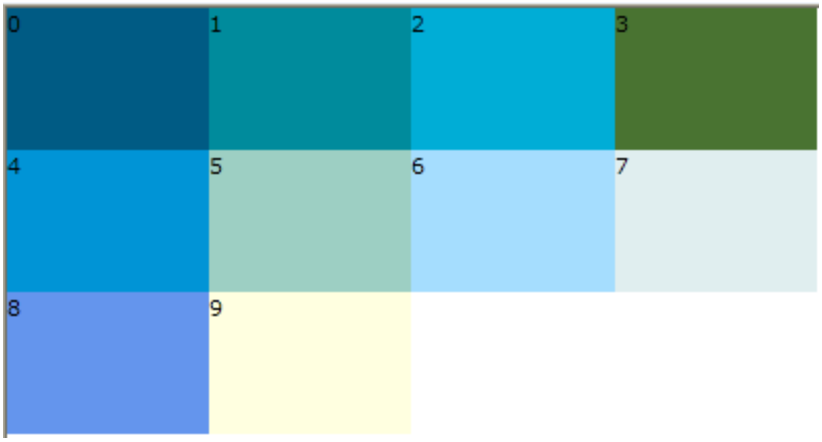
Step 2 of 4: Adding a C1UniformGrid

Next we are going to add a C1UniformGrid.

1. First, remove the Grid tags from your project.
2. In the Visual Studio Toolbox, drag a C1UniformGrid control to the page.
3. Place your cursor in between the <c1:C1UniformGrid> tags and enter the following XAML to add the grid's child elements, or numbered cells in this case:

```
<Border Background="#FF005B84" >
    <TextBlock Text="0" />
</Border>
<Border Background="#FF008B9C" >
    <TextBlock Text="1" />
</Border>
<Border Background="#FF00ADD6" >
    <TextBlock Text="2" />
</Border>
<Border Background="#FF497331" >
    <TextBlock Text="3" />
</Border>
<Border Background="#FF0094D6" >
    <TextBlock Text="4" />
</Border>
<Border Background="#FF9DCFC3" >
    <TextBlock Text="5" />
</Border>
<Border Background="#FFA5DDFE" >
    <TextBlock Text="6" />
</Border>
<Border Background="#FFE0EEEF" >
    <TextBlock Text="7" />
</Border>
<Border Background="CornflowerBlue" >
    <TextBlock Text="8" />
</Border>
<Border Background="LightYellow" >
    <TextBlock Text="9" />
</Border>
```

If you run the application at this point, it looks like the following image:



Step 3 of 4: Setting C1UniformGrid Properties

In this step, we'll set the set the Columns, FirstColumn, and **Width** properties. The Columns will set the number of columns in the grid, the **Width** property will set the width, in pixels, and the FirstColumn property will determine how many empty cells will appear in the first row.

1. Place your cursor within the opening <c1:C1UniformGrid> grid tag.
2. Set the Columns, FirstColumn, and **Width** properties using the following XAML:

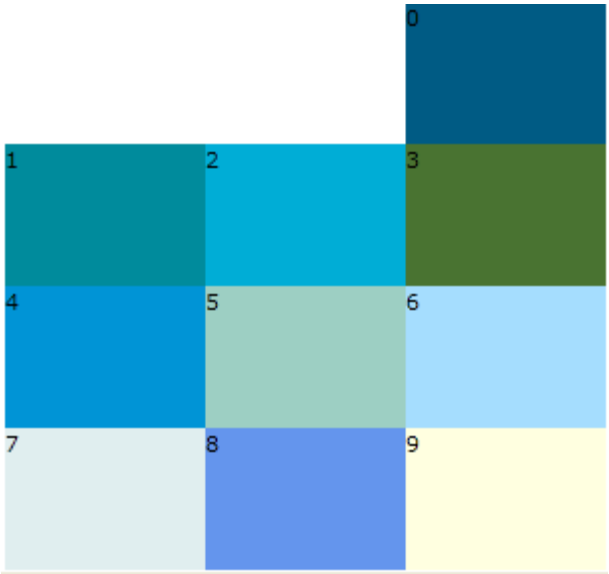
```
<c1:C1UniformGrid Columns="3" FirstColumn="2" Width="300">
```

This will give the grid three columns and a width of 300 pixels. There will be two empty cells in the first row of the grid, as specified with the FirstColumn property.

In the next step, you will run the application to see the results.

Step 4 of 4: Running the Application

Now you're ready to run the application. From the **Debug** menu, select **Start Debugging**. Your application will look similar to the following:



Notice the two empty cells in the first row.

Congratulations! You have successfully completed the **UniformGrid for Silverlight** quick start.

Layout Panels 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 Layout Panels in general. If you are unfamiliar with the **ComponentOne Layout Panels for Silverlight** product, please see the Layout Panels for Silverlight Quick Starts first.

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

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

Wrapping Items with C1WrapPanel

You can wrap items using the **C1WrapPanel.BreakLine Attached** property. In this example, HyperlinkButtons are used. Complete the following steps:

1. In your Silverlight project, drag a C1WrapPanel control from the Toolbox and place it before the closing </Grid> tag in the .xaml page.
2. Place your cursor in between the <c1:C1WrapPanel> tags and press ENTER.
3. Add the following XAML to wrap HyperlinkButtons:

```
            <Border  Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Orange">
                <HyperlinkButton Foreground="White"  Content="Example
Text" FontSize="25" />
            </Border>
            <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Green" c1:C1WrapPanel.BreakLine="After">
                <HyperlinkButton Foreground="White" Content="Break After"
/>
            </Border>
            <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Blue">
                <HyperlinkButton Foreground="White" Content="C1WrapPanel"
FontSize="16"/>
            </Border>
            <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Red">
                <HyperlinkButton Foreground="White" Content="Wrap
Vertically" />
            </Border>
            <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Purple">
                <HyperlinkButton Foreground="White" Content="Wrap
Horizontally" FontSize="20"  />
            </Border>
```

   Notice the **C1WrapPanel.BreakLine** property is set to **After** for the second HyperlinkButton. This will add a break after the button.

4. Run your project. The C1WrapPanel will resemble the following image:

Notice there is a break after the second HyperlinkButton.

Wrapping Items Vertically with C1WrapPanel

By default, items are wrapped horizontally. However, in some cases you may need them to wrap vertically. You can set the Orientation property to specify vertical wrapping. In this example, HyperlinkButtons are used. Complete the following steps:

1. In your Silverlight project, drag a C1WrapPanel control from the Toolbox and place it before the closing </Grid> tag in the .xaml page.
2. In the <c1:C1WrapPanel> tag, set the **Orientation** property to **Vertical**; the XAML will look like the following:
```
<c1:C1WrapPanel Orientation="Vertical">
```
3. Place your cursor in between the <c1:C1WrapPanel> tags and press ENTER.
4. Add the following XAML to wrap HyperlinkButtons:
```
            <Border  Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Orange">
                <HyperlinkButton Foreground="White"  Content="Example
Text" FontSize="25" />
            </Border>
            <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Green" c1:C1WrapPanel.BreakLine="After">
                <HyperlinkButton Foreground="White" Content="Break After"
/>
            </Border>
            <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Blue">
                <HyperlinkButton Foreground="White" Content="C1WrapPanel"
FontSize="16"/>
            </Border>
            <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Red">
                <HyperlinkButton Foreground="White" Content="Wrap
Vertically" />
            </Border>
            <Border Margin="2" BorderBrush="Black" BorderThickness="2"
Background="Purple">
                <HyperlinkButton Foreground="White" Content="Silverlight"
FontSize="20"  />
            </Border>
```
5. Run your project. The C1WrapPanel will resemble the following image:


Example Text
Break After
C1WrapPanel
Wrap Vertically
Silverlight

ListBox

Get two high performance controls for displaying lists of bound data with **ComponentOne ListBox™ for Silverlight**. Display lists with tile layouts or with optical zoom using the **C1ListBox** and **C1TileListBox** controls. These controls support UI virtualization so they are blazing-fast while able to display thousands of items with little-to-no loss of performance.

Key Features

ListBox for Silverlight's key features include the following:

- **Horizontal or Vertical Orientation**

  The **ComponentOne ListBox** controls support both horizontal and vertical orientation, allowing for more layout scenarios.

- **Display Items as Tiles**

  The **C1TileListBox** can arrange its items in both rows and columns creating tile displays. Specify the size and template of each item and select the desired orientation.

- **Optical Zoom**

  The **C1ListBox** control supports optical zoom functionality so users can manipulate the size of the items intuitively through pinch gestures. The zooming transformation is smooth and fluid so the performance of your application is not sacrificed.

- **UI Virtualization**

  The **ComponentOne ListBox** controls support UI virtualization so they are blazing-fast while able to display thousands of items with virtually no loss of performance. You can determine how many items are rendered in each layout pass by setting the **ViewportGap** and **ViewportPreviewGap** properties. These properties can be adjusted depending on the scenario.

- **Preview State**

  In order to have the highest performance imaginable, the **ListBox** controls can render items outside the viewport in a preview state. Like the standard **ItemTemplate**, the **Preview** template defines the appearance of items when they are in a preview state, such as zoomed out or during fast scroll. The controls will then switch to the full item template when the items have stopped scrolling or zooming.

C1ListBox Quick Start

The following quick start guide is intended to get you up and running with the **C1ListBox** control. In this quick start you'll start in Visual Studio and create a new project, add **C1ListBox** to your application, and customize the appearance and behavior of the control.

Step 1 of 3: Creating an Application with a C1ListBox Control

In this step, you'll create a Silverlight application in Visual Studio using **ListBox 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 accept default settings, close the **New Silverlight Application** dialog box, and create your project. The **MainPage.xaml** file should open.
4. Add the following `<StackPanel>` markup between the `<Grid>` and `</Grid>` tags to add a **StackPanel** containing a **TextBlock** and **ProgressBar**:

```
<StackPanel x:Name="loading" VerticalAlignment="Center">
    <TextBlock Text="Retrieving data from Flickr..."
TextAlignment="Center" />
    <ProgressBar IsIndeterminate="True" Width="200" Height="4" />
</StackPanel>
```

   The **TextBlock** and **ProgressBar** will indicate that the **C1ListBox** is loading.
5. Navigate to the Toolbox and double-click the C1ListBox icon to add the control to the grid. This will add the reference and XAML namespace automatically.
6. Edit the `<c1:C1ListBox>` tag to customize the control:

```
<c1:C1ListBox x:Name="listBox" ItemsSource="{Binding}"
Background="Transparent" Visibility="Collapsed" ItemWidth="800"
ItemHeight="600" Zoom="Fill" ViewportGap="0"
RefreshWhileScrolling="False"></c1:C1ListBox>
```

   This gives the control a name and customizes the binding, background, visibility, size, and refreshing ability of the control.
7. Add the following markup between the `<c1:C1ListBox>` and `</c1:C1ListBox>` tags:

```
<c1:C1ListBox.PreviewItemTemplate>
                <DataTemplate>
                    <Grid Background="Gray">
                        <Image Source="{Binding Thumbnail}"
Stretch="UniformToFill" />
                    </Grid>
                </DataTemplate>
            </c1:C1ListBox.PreviewItemTemplate>
            <c1:C1ListBox.ItemTemplate>
                <DataTemplate>
                    <Grid>
                        <Image Source="{Binding Content}"
Stretch="UniformToFill" />
                        <TextBlock Text="{Binding Title}"
Foreground="White" Margin="4 0 0 4" VerticalAlignment="Bottom" />
                    </Grid>
                </DataTemplate>
</c1:C1ListBox.ItemTemplate>
```

8. This markup adds data templates for the **C1ListBox** control's content. Note that you'll complete binding the control in code.

What You've Accomplished

You've successfully created a Silverlight application containing a **C1ListBox** control. In the next step, Step 2 of 3: Adding Data to the ListBox, you will add the data for **C1ListBox**.

Step 2 of 3: Adding Data to the ListBox

In the last step, you added the **C1ListBox** control to the application. In this step, you will add code to display images from a photo stream.

Complete the following steps to add data to the control programmatically:

1. Select the **Page**, navigate to the Properties window, click the lightning bolt **Events** button to view events, and scroll down and double-click the area next to the **Loaded** event.

   This will open the Code Editor and add the **Page_Loaded** event.

2. Add the following imports statements to the top of the page:

   - Visual Basic

```
Imports C1.Silverlight
Imports System
Imports System.Collections.Generic
Imports System.IO
Imports System.Linq
Imports System.Xml.Linq
Imports System.Net
Imports System.Collections.ObjectModel
```

   - C#

```
using C1.Silverlight;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Xml.Linq;
using System.Collections.ObjectMode;
```

3. Add the following code inside the initial event handler:

   - Visual Basic

```
DataContext = Enumerable.Range(0, 100)
AddHandler Loaded, AddressOf ListBoxSample_Loaded
```

   - C#

```
DataContext = Enumerable.Range(0, 100);
Loaded += new System.Windows.RoutedEventHandler(ListBoxSample_Loaded);
```

4. Add the following code below within the **MainPage** class:

   - Visual Basic

```
Private Sub ListBoxSample_Loaded(sender As Object, e As
RoutedEventArgs)
        LoadPhotos()
    End Sub

    Private Sub LoadPhotos()
        Dim flickrUrl =
"http://api.flickr.com/services/feeds/photos_public.gne"
        Dim AtomNS = "http://www.w3.org/2005/Atom"
        loading.Visibility = Visibility.Visible
        retry.Visibility = Visibility.Collapsed

        Dim photos = New List(Of Photo)()
        Dim client = New WebClient()
        AddHandler client.OpenReadCompleted, Function(s, e)
                                                 Try
                                                     '#Region "** parse
flickr data"
                                                     Dim doc =
XDocument.Load(e.Result)
```

```
                                                    For Each entry In
doc.Descendants(XName.[Get]("entry", AtomNS))
                                                        Dim title =
entry.Element(XName.[Get]("title", AtomNS)).Value
                                                        Dim enclosure
= entry.Elements(XName.[Get]("link", AtomNS)).Where(Function(elem)
elem.Attribute("rel").Value = "enclosure").FirstOrDefault()
                                                        Dim contentUri
= enclosure.Attribute("href").Value
                                                        photos.Add(New
Photo() With { _
                                                          .Title =
title, _
                                                          .Content =
contentUri, _
                                                          .Thumbnail =
contentUri.Replace("_b", "_m") _
                                                        })
                                                    Next
                                                    '#End Region


listBox.ItemsSource = photos
                                                    loading.Visibility
= Visibility.Collapsed
                                                    listBox.Visibility
= Visibility.Visible
                                                Catch

MessageBox.Show("There was an error when attempting to download data
from Flickr.")
                                                    loading.Visibility
= Visibility.Collapsed
                                                    retry.Visibility =
Visibility.Visible
                                                End Try

                                            End Function
        client.OpenReadAsync(New Uri(flickrUrl))
    End Sub

    Private Sub Retry_Click(sender As Object, e As RoutedEventArgs)
        LoadPhotos()
    End Sub

#Region "** public properties"

    Public Property Orientation() As Orientation
        Get
            Return listBox.Orientation
        End Get
        Set(value As Orientation)
            listBox.Orientation = value
        End Set
    End Property

    Public Property ItemWidth() As Double
```

```
        Get
            Return listBox.ItemWidth
        End Get
        Set(value As Double)
            listBox.ItemWidth = value
        End Set
    End Property

    Public Property ItemHeight() As Double
        Get
            Return listBox.ItemHeight
        End Get
        Set(value As Double)
            listBox.ItemHeight = value
        End Set
    End Property

    Public Property ZoomMode() As ZoomMode
        Get
            Return listBox.ZoomMode
        End Get
        Set(value As ZoomMode)
            listBox.ZoomMode = value
        End Set
    End Property
#End Region
```

- C#

```
void ListBoxSample_Loaded(object sender, RoutedEventArgs e)
        {
            LoadPhotos();
        }

        private void LoadPhotos()
        {
            var flickrUrl =
"http://api.flickr.com/services/feeds/photos_public.gne";
            var AtomNS = "http://www.w3.org/2005/Atom";
            loading.Visibility = Visibility.Visible;
            retry.Visibility = Visibility.Collapsed;

            var photos = new List<Photo>();
            var client = new WebClient();
            client.OpenReadCompleted += (s, e) =>
            {
                try
                {
                    #region ** parse flickr data
                    var doc = XDocument.Load(e.Result);
                    foreach (var entry in
doc.Descendants(XName.Get("entry", AtomNS)))
                    {
                        var title = entry.Element(XName.Get("title",
AtomNS)).Value;
                        var enclosure =
entry.Elements(XName.Get("link", AtomNS)).Where(elem =>
elem.Attribute("rel").Value == "enclosure").FirstOrDefault();
```

```
                        var contentUri =
enclosure.Attribute("href").Value;
                        photos.Add(new Photo() { Title = title, Content
= contentUri, Thumbnail = contentUri.Replace("_b", "_m") });
                    }
                    #endregion

                    listBox.ItemsSource = photos;
                    loading.Visibility = Visibility.Collapsed;
                    listBox.Visibility = Visibility.Visible;
                }
                catch
                {
                    MessageBox.Show("There was an error when attempting
to download data from Flickr.");
                    loading.Visibility = Visibility.Collapsed;
                    retry.Visibility = Visibility.Visible;
                }
            };
            client.OpenReadAsync(new Uri(flickrUrl));
        }

        private void Retry_Click(object sender, RoutedEventArgs e)
        {
            LoadPhotos();
        }

        #region ** public properties

        public Orientation Orientation
        {
            get
            {
                return listBox.Orientation;
            }
            set
            {
                listBox.Orientation = value;
            }
        }

        public double ItemWidth
        {
            get
            {
                return listBox.ItemWidth;
            }
            set
            {
                listBox.ItemWidth = value;
            }
        }

        public double ItemHeight
        {
            get
            {
```

```
                return listBox.ItemHeight;
            }
            set
            {
                listBox.ItemHeight = value;
            }
        }

        public ZoomMode ZoomMode
        {
            get
            {
                return listBox.ZoomMode;
            }
            set
            {
                listBox.ZoomMode = value;
            }
        }
        #endregion
```

5. The code above pulls images from Flickr's public photo stream and binds the **C1ListBox** to the list of images.
6. Add the following code just below the **MainPage** class:

- Visual Basic

```
Public Class Photo
    Public Property Title() As String
        Get
            Return m_Title
        End Get
        Set(value As String)
            m_Title = Value
        End Set
    End Property
    Private m_Title As String
    Public Property Thumbnail() As String
        Get
            Return m_Thumbnail
        End Get
        Set(value As String)
            m_Thumbnail = Value
        End Set
    End Property
    Private m_Thumbnail As String
    Public Property Content() As String
        Get
            Return m_Content
        End Get
        Set(value As String)
            m_Content = Value
        End Set
    End Property
    Private m_Content As String
End Class
```

- C#

```
public class Photo
    {
        public string Title { get; set; }
        public string Thumbnail { get; set; }
        public string Content { get; set; }
    }
```

What You've Accomplished

You have successfully added data to **C1TileListBox**. In the next step, Step 3 of 3: Running the ListBox Application, you'll examine the **ListBox for Silverlight** features.

Step 3 of 3: Running the ListBox 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 **ListBox for Silverlight**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time.

   The application will appear, displaying an image.
2. Use the scroll bar on the right of the control, to scroll through the image stream.
3. If you have touch capabilities, try pinching to zoom an image.

What You've Accomplished

Congratulations! You've completed the **ListBox for Silverlight** quick start and created an application using the **C1ListBox** control and viewed some of the run-time capabilities of your application.

C1TileListBox Quick Start

The following quick start guide is intended to get you up and running with the **C1TileListBox** control. In this quick start you'll start in Visual Studio and create a new project, add **C1TileListBox** to your application, and customize the appearance and behavior of the control.

Step 1 of 3: Creating an Application with a C1TileListBox Control

In this step, you'll create a Silverlight application in Visual Studio using **TileListBox 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 accept default settings, close the **New Silverlight Application** dialog box, and create your project. The **MainPage.xaml** file should open.
4. Place the cursor between the `<Grid>` and `</Grid>`, navigate to the Toolbox, and double-click the C1TileListBox icon to add the control to the grid. This will add the reference and XAML namespace automatically.
5. Edit the `<c1:C1TileListBox>` tag to customize the control:

   ```
   <c1:C1TileListBox x:Name="tileListBox" ItemsSource="{Binding}"
   ItemWidth="130" ItemHeight="130"></c1:C1TileListBox>
   ```

   This gives the control a name, sets the **ItemsSource** property (you'll customize this in code in a later step), and sets the size of the control.

6. Add markup between the `<c1:C1TileListBox>` and `</c1:C1TileListBox>` tags so it looks like the following:

```
<c1:C1TileListBox x:Name="tileListBox" ItemsSource="{Binding}"
ItemWidth="130" ItemHeight="130">
    <c1:C1TileListBox.PreviewItemTemplate>
        <DataTemplate>
            <Grid Background="Gray" />
                </DataTemplate>
                </c1:C1TileListBox.PreviewItemTemplate>
                <c1:C1TileListBox.ItemTemplate>
                <DataTemplate>
                <Grid Background="LightBlue">
                    <Image Source="{Binding Thumbnail}"
Stretch="UniformToFill" />
                    <TextBlock Text="{Binding Title}" Margin="4 0 0 4"
VerticalAlignment="Bottom" />
                </Grid>
        </DataTemplate>
    </c1:C1TileListBox.ItemTemplate>
</c1:C1TileListBox>
```

This markup adds data templates for the **C1TileListBox** control's content. Note that you'll complete binding the control in code.

What You've Accomplished

You've successfully created a Silverlight application containing a **C1TileListBox** control. In the next step, Step 2 of 3: Adding Data to the TileListBox, you will add data for **C1TileListBox**.

Step 2 of 3: Adding Data to the TileListBox

In the last step, you added the **C1TileListBox** control to the application. In this step, you will add code to bind the control to data.

Complete the following steps to add data to the control programmatically:

1. Right-click the page and select **View Code** to open the Code Editor.
2. Add the following imports statements to the top of the page:
   - Visual Basic

```
Imports System
Imports System.Collections.Generic
Imports System.IO
Imports System.Linq
Imports System.Xml.Linq
Imports System.Net
Imports C1.Silverlight
```

   - C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using C1.Silverlight;
```

3. Add the following code inside the initial event handler within the **MainPage** class:
   - Visual Basic

```
DataContext = Enumerable.Range(0, 100).[Select](Function(i) New Item()
With {.Title = i.ToString()})
```

- C#

```
DataContext = Enumerable.Range(0, 100).Select(i => new Item { Title =
i.ToString() });
```

4. Add the following code below the initial event handler but within the **MainPage** class:

- Visual Basic

```
#Region "** public properties"

    Public Property Orientation() As Orientation
        Get
            Return tileListBox.Orientation
        End Get
        Set(value As Orientation)
            tileListBox.Orientation = value
        End Set
    End Property

    Public Property ItemWidth() As Double
        Get
            Return tileListBox.ItemWidth
        End Get
        Set(value As Double)
            tileListBox.ItemWidth = value
        End Set
    End Property

    Public Property ItemHeight() As Double
        Get
            Return tileListBox.ItemHeight
        End Get
        Set(value As Double)
            tileListBox.ItemHeight = value
        End Set
    End Property

    Public Property ZoomMode() As ZoomMode
        Get
            Return tileListBox.ZoomMode
        End Get
        Set(value As ZoomMode)
            tileListBox.ZoomMode = value
        End Set
    End Property
#End Region
```

- C#

```
#region ** public properties

        public Orientation Orientation
        {
            get
            {
                return tileListBox.Orientation;
            }
            set
            {
                tileListBox.Orientation = value;
```

```
            }
        }

        public double ItemWidth
        {
            get
            {
                return tileListBox.ItemWidth;
            }
            set
            {
                tileListBox.ItemWidth = value;
            }
        }

        public double ItemHeight
        {
            get
            {
                return tileListBox.ItemHeight;
            }
            set
            {
                tileListBox.ItemHeight = value;
            }
        }

        public ZoomMode ZoomMode
        {
            get
            {
                return tileListBox.ZoomMode;
            }
            set
            {
                tileListBox.ZoomMode = value;
            }
        }
        #endregion
```

The code above binds the **C1TileListBox** to a list of numbers.

5. Add the following code just below the **MainPage** class:

- Visual Basic

```
Public Class Item
    Public Property Title() As String
        Get
            Return m_Title
        End Get
        Set(value As String)
            m_Title = Value
        End Set
    End Property
    Private m_Title As String
    Public Property Thumbnail() As String
        Get
            Return m_Thumbnail
```

```
        End Get
        Set(value As String)
            m_Thumbnail = Value
        End Set
    End Property
    Private m_Thumbnail As String
End Class
```

- C#

```
public class Item
    {
        public string Title { get; set; }
        public string Thumbnail { get; set; }
    }
```

What You've Accomplished

You have successfully added data to **C1TileTileListBox**. In the next step, Step 3 of 3: Running the TileListBox Application, you'll examine the **TileListBox for Silverlight** features.

Step 3 of 3: Running the TileListBox 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 **TileListBox for Silverlight**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time.

   The application will appear, displaying a tiled list of numbers.
2. Use the scroll bar on the right of the control, to scroll through the numbered squares.
3. If you have touch capabilities, try pinching to zoom an image.

What You've Accomplished

Congratulations! You've completed the **TileListBox for Silverlight** quick start and created an application using the **C1TileListBox** control and viewed some of the run-time capabilities of your application.

Top Tips

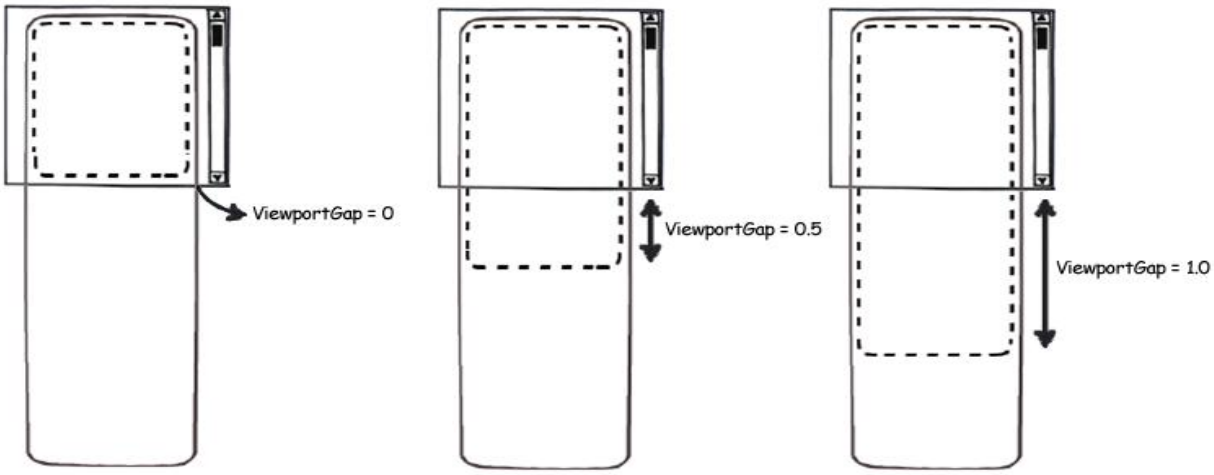
These tips will help you maximize your performance while using any of the **ComponentOne ListBox** controls.

- **Use PreviewTemplate** – In order to avoid making the layout heavier, the **PreviewTemplate** can be used to render a lighter template during preview states, such as while zooming and scrolling fast. For example you could display a thumbnail image in the **PreviewTemplate** and display the larger image in the full **ItemTemplate**.

```
<c1:C1ListBox x:Name="listBox"
                  ItemsSource="{Binding}"
                  RefreshWhileScrolling="False">
    <c1:C1ListBox.PreviewItemTemplate>
        <DataTemplate>
            <Grid Background="Gray">
                <Image Source="{Binding Thumbnail}"
Stretch="UniformToFill"/>
            </Grid>
        </DataTemplate>
    </c1:C1ListBox.PreviewItemTemplate>
    <c1:C1ListBox.ItemTemplate>
        <DataTemplate>
```

```
            <Grid>
                <Image Source="{Binding Content}"
Stretch="UniformToFill"/>
            </Grid>
        </DataTemplate>
    </c1:C1ListBox.ItemTemplate>
</c1:C1ListBox>
```

- **Adjust the ViewportGap and ViewportPreviewGap Properties** – These coefficient values determine what size of items outside the viewport to render in advance. The larger the value the more quickly off-screen items will appear to render, but the slower the control will take on each layout pass. For example, if set to 0.5 the view port will be enlarged to take up a half screen more at both sides of the original view port.



- **Set RefreshWhileScrolling to False** – Determines whether the view port is refreshed while scrolling. If set to false items will appear blank or say "Loading" while the user scrolls real fast until they stop and allow items to render.

Working with ListBox for Silverlight

ComponentOne ListBox for Silverlight includes the C1ListBox and C1TileListBox controls. **C1ListBox** is similar to the standard Silverlight **ListBox** control but it includes additional functionality, such as zooming. **C1TileListBox** allows you to create tiled lists of items. Display lists with tile layouts or with optical zoom using the **C1ListBox** and **C1TileListBox** controls.

Basic Properties

ComponentOne ListBox for Silverlight includes several properties that allow you to set the functionality of the control. Some of the more important properties are listed below.

The following properties let you customize the C1ListBox control:

Property	Description
ActualMaxZoom	Gets the actual maximum zoom.
ActualMinZoom	Gets the actual minimum zoom.
ActualZoom	Gets the actual zoom.
IsScrolling	Gets a value indicating whether the list is scrolling.

IsZooming	Gets a value indicating whether this list is zooming.
ItemHeight	Gets or sets the height of each item.
Items	Gets the collection used to generate the content of the control. (Inherited from ItemsControl.)
ItemsPanel	Gets or sets the template that defines the panel that controls the layout of items. (Inherited from ItemsControl.)
ItemsSource	Gets or sets a collection used to generate the content of the ItemsControl. (Inherited from ItemsControl.)
ItemStyle	Style applied to all the items of this item control. (Inherited from C1ItemsControl.)
ItemTemplate	Template applied to all the items of the list. (Inherited from C1ItemsControl.)
ItemTemplateSelector	Template selector used to specify different templates applied to items of the same type. (Inherited from C1ItemsControl.)
ItemWidth	Gets or sets the width of each item.
MaxZoom	Gets or sets the maximum zoom available.
MinZoom	Gets or sets the minimum zoom available.
Orientation	Gets or sets the orientation in which the list is displayed.
Panel	Gets the panel associated with this items control.
PreviewItemTemplate	Gets or sets the template used for previewing an item.
RefreshWhileScrolling	Gets or sets a value indicating whether the viewport must be refreshed while the scroll is running.
ScrollViewer	Gets the scroll viewer template part belonging to this items control.
ViewportGap	Gets or sets a coeficient which will determine in each layout pass the size of the viewport. If zero is specified the size of the viewport will be equal to the scrollviewer viewport. If 0.5 is specified the size of the viewport will be enlarged to take up half screen more at both sides of the original viewport.
ViewportPreviewGap	Gets or sets a coeficient which will determine in each layout pass the size of the viewport to render items in preview mode.
Zoom	Gets or set the zoom applied to this list.
ZoomMode	Gets or sets whether the zoom is enabled or disabled.

The following properties let you customize the C1ListBoxItem:

Description

PreviewContent	Gets or sets the content of the preview state.
PreviewContentTemplate	Gets or sets the DataTemplate used when in preview state.
State	Gets or sets the state of the item, which can be Preview or Full.

Optical Zoom

The **ListBox for Silverlight** controls support optical zoom functionality so users can manipulate the size of the items intuitively through pinch gestures. The zooming transformation is smooth and fluid so the performance of your application is not sacrificed.

You can customize the zoom using the **ZoomMode** and **Zoom** properties. The **ZoomMode** property gets or sets whether the zoom is enabled or disabled. The **Zoom** property the **Zoom** value applied to the control. The **ZoomChanged** event is triggered when the zoom value of the control is changed.

UI Virtualization

The **ComponentOne ListBox** controls support UI virtualization so they are blazing-fast while able to display thousands of items with virtually no loss of performance. You can determine how many items are rendered in each layout pass by setting the **ViewportGap** and **ViewportPreviewGap** properties. These properties can be adjusted depending on the scenario.

The **ViewportGap** property gets or sets a coefficient which will determine in each layout pass the size of the viewport. If zero is specified the size of the viewport will be equal to the scrollviewer viewport. If 0.5 is specified the size of the viewport will be enlarged to take up half screen more at both sides of the original viewport.

The **ViewportPreviewGap** property gets or sets a coefficient which will determine in each layout pass the size of the viewport to render items in preview mode.

Orientation

The **ComponentOne ListBox** controls support both horizontal and vertical orientation, allowing for more layout scenarios. To set the orientation of the control, set the **Orientation** property to **Horizontal** or **Vertical**.

Preview State

In order to have the highest performance imaginable, the **ListBox** controls can render items outside the viewport in a preview state. Like the standard **ItemTemplate**, the **Preview** template defines the appearance of items when they are in a preview state, such as zoomed out or during fast scroll. The controls will then switch to the full item template when the items have stopped scrolling or zooming.

MaskedTextBox

Validate input in your Silverlight applications! **ComponentOne MaskedTextBox™ for Silverlight** provides a text box with a mask that automatically validates entered input, similar to the standard Microsoft WinForms **MaskedTextBox** control.

Getting Started

Get started with the following topics:

- Key Features
- Quick Start
- Task-Based Help

MaskedTextBox for Silverlight Features

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

- **Validate Data and Enhance Your UI**

  The ComponentOne masked text box control (C1MaskedTextBox) provides a text box with a mask that automatically validates the input. The edit mask enhances the UI by preventing end-users from entering invalid characters into the control.

- **Provide Clues to Prompt Users for Information**

  The masked text box control includes a Watermark property, which lets end-users know what type of information is expected.

- **Easily Change Colors with ClearStyle**

  C1MaskedTextBox supports ComponentOne ClearStyle™ technology which allows you to easily change control brushes without having to override templates. By just setting a few brush properties in Visual Studio you can quickly style the entire control.

- **Silverlight Toolkit Themes Support**

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

MaskedTextBox for Silverlight Quick Start

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

You will create a simple form using several C1MaskedTextBox controls that will demonstrate the difference between the **Text** and **Value** properties. The controls will include various masks and different appearance and behavior settings so that you can explore the possibilities of using **MaskedTextBox for Silverlight**.

Step 1 of 4: Setting up the Application

In this step you'll begin in Visual Studio to create a Silverlight application using **MaskedTextBox for Silverlight**. When you add a C1MaskedTextBox control to your application, you'll have a complete, functional input editor. You can further customize the control to your application.

To set up your project and add C1MaskedTextBox controls to your application, complete the following steps:

1. In Visual Studio, select **File | New | Project**. The **New Project** dialog box will open.
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, place the cursor between the `<Grid>` and `</Grid>` tags and click once.
5. Navigate to the Toolbox and double-click the **StackPanel** icon to add the panel to **MainPage.xaml**. The XAML markup will now look similar to the following:

```
<UserControl x:Class="C1MaskedTextBox.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:DesignHeight="300" d:DesignWidth="400"
xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sd
k">
    <Grid x:Name="LayoutRoot" Background="White">
        <StackPanel></StackPanel>
    </Grid>
</UserControl>
```

6. Add `x:Name="sp1" Width="Auto" Height="Auto" Orientation="Vertical" HorizontalAlignment="Center" VerticalAlignment="Center"` to the `<StackPanel>` tag so that it appears similar to the following:

```
<StackPanel x:Name="sp1" Width="Auto" Height="Auto"
Orientation="Vertical" HorizontalAlignment="Center"
VerticalAlignment="Center"></StackPanel>
```

   Elements in the panel will now appear centered and vertically positioned.

You've successfully created a Silverlight application. In the next step you'll add and customize **Label** and **C1MakedTextBox** controls and complete setting up the application.

Step 2 of 4: Customizing the Application

In the previous step you created a new Silverlight project and added a **StackPanel** to the application. In this step you'll continue by adding and customizing **Label** and **C1MakedTextBox** controls.

Complete the following steps:

1. In the XAML window of the project, place the cursor between the `<StackPanel x:Name="sp1">` and `</StackPanel>` tags.
2. Navigate to the Toolbox and double-click the **Label** icon to add the control to the **StackPanel**.
3. Add `x:Name="lbl1" Content="Employee Information" Margin="2"` to the `<sdk:Label>` tag so that it appears similar to the following:

```
<sdk:Label x:Name="lbl1" Content="Employee Information" Margin="2" />
```

4. Place the cursor just after the `<sdk:Label>` tag, navigate to the Toolbox, and double-click the **C1MaskedTextBox** icon to add the control to the **StackPanel**.

   Note that the C1.Silverlight namespace and `<c1:C1MaskedTextBox></c1:C1MaskedTextBox>` tags have been added to the project.

5. If the **C1MaskedTextBox** tag contains any content, delete it. The XAML markup will now look similar to the following:

```
<UserControl x:Class="C1MaskedTextBox.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:DesignHeight="300" d:DesignWidth="400"
xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sd
k" xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml">
    <Grid x:Name="LayoutRoot" Background="White">
        <StackPanel x:Name="sp1" Width="Auto" Height="Auto"
Orientation="Vertical" HorizontalAlignment="Center"
VerticalAlignment="Center">
            <sdk:Label x:Name="lbl1" Content="Employee Information"
Margin="2" />
            <c1:C1MaskedTextBox />
        </StackPanel>
    </Grid>
</UserControl>
```

6. Give your control a name by adding `x:Name="c1mtb1"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Name="c1mtb1"></c1:C1MaskedTextBox>
```

   By giving it a unique identifier, you'll be able to access the control in code.

7. Resize your control and set the alignment and margin by adding `Height="23" VerticalAlignment="Top" Margin="2"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Name="c1mtb1" Height="23" VerticalAlignment="Top"
Margin="2" />
```

   Your control should now be resized on the page.

8. Set a watermark on the control by adding `Watermark="Employee ID"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Name="c1mtb1" Height="23" VerticalAlignment="Top"
Margin="2" Watermark="Employee ID" />
```

   The WaterMark property will set the text that appears n the control to prompt users at run time.

9. Set a numeric mask on the control by adding `Mask="000-00-0000"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Name="c1mtb1" Height="23" VerticalAlignment="Top"
Margin="2" Watermark="Employee ID" Mask="000-00-0000" />
```

   The Mask property will set the mask – users will now only be able to enter numbers into the control at run time.

10. Add an event handler by adding `TextChanged="c1mtb1_TextChanged"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Name="c1mtb1" Height="23" VerticalAlignment="Top"
Margin="2" Watermark="Employee ID" Mask="000-00-0000"
TextChanged="c1mtb1_TextChanged" />
```

    You will add the event handler code later in the tutorial.

11. Place the cursor just after the `<c1:C1MaskedTextBox>` tag and add the following XAML to add three additional **C1MaskedTextBox** and four additional **Label** controls:

```
<sdk:Label x:Name="lbl2" FontSize="9" Margin="2"/>
```

```
<c1:C1MaskedTextBox Name="c1mtb2" Height="23" VerticalAlignment="Top"
Margin="2" Watermark="Name" TextChanged="c1mtb2_TextChanged"/>
<sdk:Label x:Name="lbl3" FontSize="9" Margin="2"/>
<c1:C1MaskedTextBox Name="c1mtb3" Height="23" VerticalAlignment="Top"
Margin="2" Watermark="Hire Date" Mask="00/00/0000"
TextChanged="c1mtb3_TextChanged"/>
<sdk:Label x:Name="lbl4" FontSize="9" Margin="2"/>
<c1:C1MaskedTextBox Name="c1mtb4" Height="23" VerticalAlignment="Top"
Margin="2" Watermark="Phone Number" Mask="(999) 000-0000"
TextChanged="c1mtb4_TextChanged"/>
<sdk:Label x:Name="lbl5" FontSize="9" Margin="2"/>
```

Your application should now appear similar to the following in the Design view:

You've successfully set up your application's user interface. In the next step you'll add code to your application.

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 add additional functionality.

Complete the following steps:

1. Select **View | Code** to switch to Code view.
2. In Code view, add the following import statement to the top of the page:
   - Visual Basic
     ```
     Imports C1.Silverlight
     ```
   - C#
     ```
     using C1.Silverlight;
     ```
3. Add the following **C1MaskedTextBox_TextChanged** event handlers to the project:
   - Visual Basic
     ```
     Private Sub c1mtb1_TextChanged(ByVal sender As System.Object, ByVal e
     As System.Windows.Controls.TextChangedEventArgs) Handles
     c1mtb1.TextChanged
         Me.lbl2.Content = "Mask: " & Me.c1mtb1.Mask & "  Value: " &
     Me.c1mtb1.Value & "  Text: " & Me.c1mtb1.Text
     End Sub
     ```

```
Private Sub c1mtb2_TextChanged(ByVal sender As System.Object, ByVal e
As System.Windows.Controls.TextChangedEventArgs) Handles
c1mtb2.TextChanged
    Me.lbl3.Content = "Mask: " & Me.c1mtb2.Mask & "  Value: " &
Me.c1mtb2.Value & "  Text: " & Me.c1mtb2.Text
End Sub
Private Sub c1mtb3_TextChanged(ByVal sender As System.Object, ByVal e
As System.Windows.Controls.TextChangedEventArgs) Handles
c1mtb3.TextChanged
    Me.lbl4.Content = "Mask: " & Me.c1mtb3.Mask & "  Value: " &
Me.c1mtb3.Value & "  Text: " & Me.c1mtb3.Text
End Sub
Private Sub c1mtb4_TextChanged(ByVal sender As System.Object, ByVal e
As System.Windows.Controls.TextChangedEventArgs) Handles
c1mtb4.TextChanged
    Me.lbl5.Content = "Mask: " & Me.c1mtb4.Mask & "  Value: " &
Me.c1mtb4.Value & "  Text: " & Me.c1mtb4.Text
End Sub
```

- C#

```
private void c1mtb1_TextChanged(object sender, TextChangedEventArgs e)
{
    this.lbl2.Content = "Mask: " + this.c1mtb1.Mask + " Value: " +
this.c1mtb1.Value + " Text: " + this.c1mtb1.Text;
}
private void c1mtb2_TextChanged(object sender, TextChangedEventArgs e)
{
    this.lbl3.Content = "Mask: " + this.c1mtb2.Mask + " Value: " +
this.c1mtb2.Value + " Text: " + this.c1mtb2.Text;
}
private void c1mtb3_TextChanged(object sender, TextChangedEventArgs e)
{
    this.lbl4.Content = "Mask: " + this.c1mtb3.Mask + " Value: " +
this.c1mtb3.Value + " Text: " + this.c1mtb3.Text;
}
private void c1mtb4_TextChanged(object sender, TextChangedEventArgs e)
{
    this.lbl5.Content = "Mask: " + this.c1mtb4.Mask + " Value: " +
this.c1mtb4.Value + " Text: " + this.c1mtb4.Text;
}
```

4. Add code to the page's constructor so that it appears like the following:

- Visual Basic

```
Public Sub New()
    InitializeComponent()
    Me.c1mtb1_TextChanged(Nothing, Nothing)
    Me.c1mtb2_TextChanged(Nothing, Nothing)
    Me.c1mtb3_TextChanged(Nothing, Nothing)
    Me.c1mtb4_TextChanged(Nothing, Nothing)
End Sub
```

- C#

```
public Page()
{
    InitializeComponent();
    this.c1mtb1_TextChanged(null, null);
    this.c1mtb2_TextChanged(null, null);
```

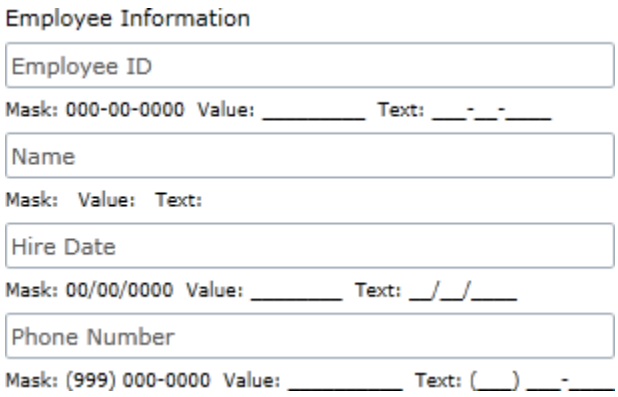
```
    this.c1mtb3_TextChanged(null, null);
    this.c1mtb4_TextChanged(null, null);
}
```

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

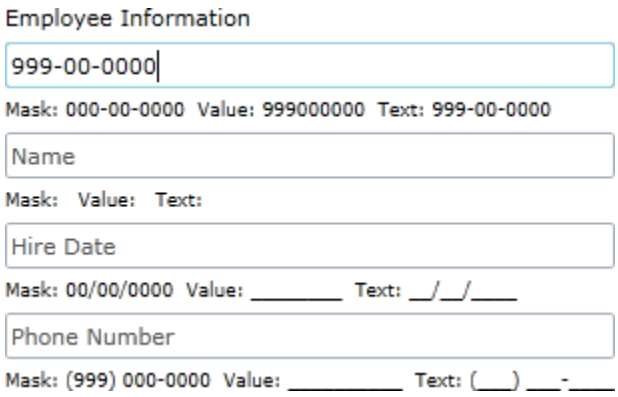
Now that you've created a Silverlight application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **MaskedTextBox for Silverlight**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time. It will appear similar to the following:



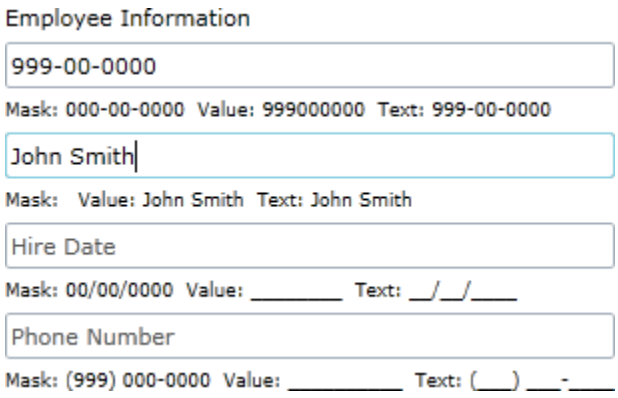
   Notice the watermark that appears in each **C1MaskedTextBox** control.

2. Enter a number in the first **C1MaskedTextBox** control:



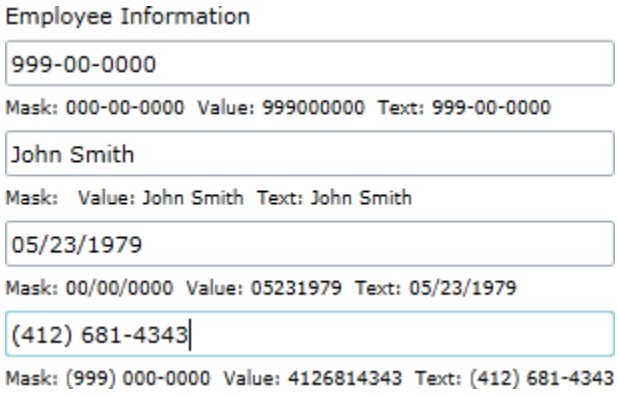
   The label below the control displays the mask, current value, and current text.

3. Enter a string in the second **C1MaskedTextBox** control:



Notice that there was no mask added to this control – if you chose, you could type numbers or other characters in the control.

4. Try entering a string in the third **C1MaskedTextBox** control. Notice that you cannot – that is because the Mask property was set to only accept numbers. Enter a numeric value instead – notice that this does work.
5. Enter numbers in each of the remaining controls. The application will appear similar to the following image:



Notice that the Value property displayed under each C1MaskedTextBox control does not include literal characters, while the **Text** property does.

Congratulations! You've completed the **MaskedTextBox for Silverlight** quick start and created a **MaskedTextBox for Silverlight** application, customized the appearance and behavior of the controls, and viewed some of the run-time capabilities of your application.

Working With MaskedTextBox

ComponentOne MaskedTextBox for Silverlight includes the C1MaskedTextBox control, a simple control which provides a text box with a mask that automatically validates entered input. When you add the C1MaskedTextBox control to a XAML window, it exists as a completely functional text box which you can further customize with a mask. By default, the control's interface looks similar to the following image:

The C1MaskedTextBox control appears like a text box and includes a basic text input area which can be customized.

Basic Properties

ComponentOne MaskedTextBox 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 MaskedTextBox for Silverlight Appearance Properties for more information about properties that control appearance.

The following properties let you customize the C1MaskedTextBox control:

Property	Description
Mask	Gets or sets the input mask to use at run time. See Mask Formatting for more information.
PromptChar	Gets or sets the character used to show spaces where user is supposed to type.
Text	Gets or sets the text content of this element.
TextMaskFormat	Gets or sets a value that determines whether literals and prompt characters are included in the Value property.
Value	Gets the formatted content of the control as specified by the TextMaskFormat property.
Watermark	Gets or sets the content of the watermark.

The **Text** property of the C1MaskedTextBox exposes the control's full content. The Value property exposes only the values typed by the user, excluding template characters specified in the Mask. For example, if the Mask property is set to "99-99" and the control contains the string "55-55", the **Text** property would return "55-55" and the Value property would return "5555".

Mask Formatting

You can provide input validation and format how the content displayed in the C1MaskedTextBox control will appear by setting the Mask property. **ComponentOne MaskedTextBox for Silverlight** supports the standard number formatting strings defined by Microsoft and the Mask property uses the same syntax as the standard **MaskedTextBox** control in WinForms. This makes it easier to re-use masks across applications and platforms.

By default, the Mask property is not set and no input mask is applied. When a mask is applied, the Mask string should consist of one or more of the masking elements. Other elements that may be displayed in the control are literals and prompts which may also be used if allowed by the TextMaskFormat property.

The following table lists some example masks:

Mask	Behavior
00/00/0000	A date (day, numeric month, year) in international date format. The "/" character is a logical date separator, and will appear to the user as the date separator appropriate to the application's current culture.
00->L<LL-0000	A date (day, month abbreviation, and year) in United States format in which the three-letter month abbreviation is displayed with an initial uppercase letter followed by two lowercase letters.
(999)-000-0000	United States phone number, area code optional. If users do not want to enter the optional characters, they can either enter spaces or place the mouse pointer directly at the position in the mask represented by the first 0.
$999,999.00	A currency value in the range of 0 to 999999. The currency, thousandth, and decimal characters

	will be replaced at run time with their culture-specific equivalents.

You can set the TextMaskFormat property to one of the following elements to define what is included in the mask:

Option	Description
IncludePrompt	Return text input by the user as well as any instances of the prompt character.
IncludeLiterals	Return text input by the user as well as any literal characters defined in the mask.
IncludePromptAndLiterals	Return text input by the user as well as any literal characters defined in the mask and any instances of the prompt character.
ExcludePromptAndLiterals	Return only text input by the user.

The following topics detail mask, literal, and prompt elements that can be used or displayed.

Mask Elements

ComponentOne MaskedTextBox for Silverlight supports the standard number formatting strings defined by Microsoft. The Mask string should consist of one or more of the masking elements as detailed in the following table:

Element	Description
0	Digit, required. This element will accept any single digit between 0 and 9.
9	Digit or space, optional.
#	Digit or space, optional. If this position is blank in the mask, it will be rendered as a space in the **Text** property. Plus (+) and minus (-) signs are allowed.
L	Letter, required. Restricts input to the ASCII letters a-z and A-Z. This mask element is equivalent to [a-zA-Z] in regular expressions.
?	Letter, optional. Restricts input to the ASCII letters a-z and A-Z. This mask element is equivalent to [a-zA-Z]? in regular expressions.
&	Character, required.
C	Character, optional. Any non-control character.
A	Alphanumeric, optional.
a	Alphanumeric, optional.
.	Decimal placeholder. The actual display character used will be the decimal symbol appropriate to the format provider.
,	Thousands placeholder. The actual display character used will be the thousands placeholder appropriate to the format provider.
:	Time separator. The actual display character used will be the time symbol appropriate to the format provider.
/	Date separator. The actual display character used will be the date symbol appropriate to the format provider.

$	Currency symbol. The actual character displayed will be the currency symbol appropriate to the format provider.
<	Shift down. Converts all characters that follow to lowercase.
>	Shift up. Converts all characters that follow to uppercase.
|	Disable a previous shift up or shift down.
\	Escape. Escapes a mask character, turning it into a literal. "\\" is the escape sequence for a backslash.
All other characters	Literals. All non-mask elements will appear as themselves within C1MaskedTextBox. Literals always occupy a static position in the mask at run time, and cannot be moved or deleted by the user.

The decimal (.), thousandths (,), time (:), date (/), and currency ($) symbols default to displaying those symbols as defined by the application's culture.

Literals

In addition to the mask elements defined in the Mask Formatting topic, other characters can be included in the mask. These characters are *literals*. Literals are non-mask elements that will appear as themselves within C1MaskedTextBox. Literals always occupy a static position in the mask at run time, and cannot be moved or deleted by the user.

For example, if the Mask property has been set to "(999)-000-0000" to define a phone number, the mask characters include the "9" and "0" elements. The remaining characters, the dashes and parentheses, are literals. These characters will appear as they in the C1MaskedTextBox control.

Note that the TextMaskFormat property must be set to **IncludeLiterals** or **IncludePromptAndLiterals** for literals to be used. If you do not want literals to be used, set TextMaskFormat to **IncludePrompt** or **ExcludePromptAndLiterals**.

Prompts

You can choose to include prompt characters in the **C1MaskedTextBox** control. The prompt character defined that text that will appear in the control to prompt the user to enter text. The prompt character indicates to the user that text can be entered, and can be used to detail the type of text allowed. By default the underline "_" character is used.

Note that the TextMaskFormat property must be set to **IncludePrompt** or **IncludePromptAndLiterals** for prompt characters to be used. If you do not want prompt characters to be used, set TextMaskFormat to **IncludeLiterals** or **ExcludePromptAndLiterals**.

Watermark

Using the Watermark property you can provide contextual clues of what value users should enter in a C1MaskedTextBox control. The watermark is displayed in the control while not text has been entered. To add a watermark, add the text `Watermark="Watermark Text"` to the `<c1:C1MaskedTextBox>` tag in the XAML markup for any **C1MaskedTextBox** control.

So, for example, enter `Watermark="Enter Text"` to the `<c1:C1MaskedTextBox>` tag so that appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" Width="120" Name="C1MaskedTextBox1"
Watermark="Enter Text" />
```

The control will appear similar to the following at run time:

If you click within the control and enter text, you will notice that the watermark disappears.

MaskedTextBox for Silverlight Layout and Appearance

The following topics detail how to customize the C1MaskedTextBox 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.

MaskedTextBox for Silverlight Appearance Properties

ComponentOne MaskedTextBox 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.

Content Properties

The following properties let you customize the appearance of content in the **C1MaskedTextBox** control:

Property	Description
Mask	Gets or sets the input mask to use at run time. See Mask Formatting for more information.
PromptChar	Gets or sets the character used to show spaces where user is supposed to type.
Watermark	Gets or sets the content of the watermark.

Text Properties

The following properties let you customize the appearance of text in the **C1MaskedTextBox** 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.
TextAlignment	Gets or sets how the text should be aligned in the **C1MaskedTextBox**.

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 **C1MaskedTextBox** control:

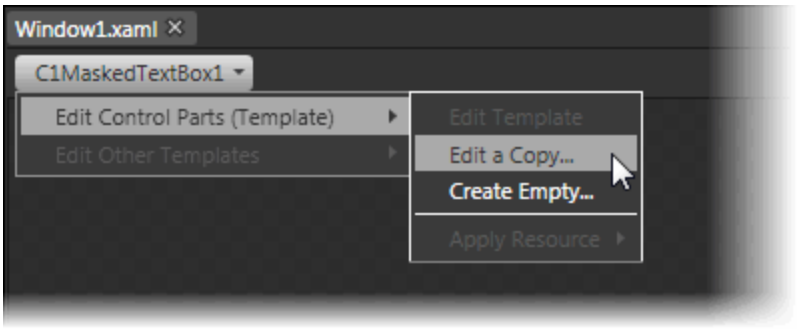
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
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 MaskedTextBox 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 C1MaskedTextBox 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.

MaskedTextBox 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 C1MaskedTextBox control in general. If you are unfamiliar with the **ComponentOne MaskedTextBox for Silverlight** product, please see the MaskedTextBox for Silverlight Quick Start first.

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

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

Setting the Value

The Value property determines the currently visible text. By default the C1MaskedTextBox control starts with its Value not set 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 Value property in Blend, complete the following steps:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties tab and enter a number, for example "123", in the text box next to the Value property.

   This will set the Value property to the number you chose.

In XAML

To set the Value property add `Value="123"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" Width="120" Name="C1MaskedTextBox1"
Value="123"></c1:C1MaskedTextBox>
```

In Code

To set the Value property, add the following code to your project:

- Visual Basic

```
C1MaskedTextBox1.Value = "123"
```

- C#

```
c1MaskedTextBox1.Value = "123";
```

Run your project and observe:

Initially **123** (or the number you chose) will appear in the control:

Adding a Mask for Currency

You can easily add a mask for currency values using the Mask property. By default the C1MaskedTextBox control starts with its Mask not set but you can customize this at design time in Microsoft Expression Blend, in XAML, and in code. For more details about mask characters, see Mask Elements.

At Design Time in Blend

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

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties tab and enter "$999,999.00" in the text box next to the Mask property.

   This will set the Mask property to the number you chose.

In XAML

To set the Mask property add `Mask="$999,999.00"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" Width="120" Name="C1MaskedTextBox1"
Mask="$999,999.00"></c1:C1MaskedTextBox>
```

In Code

To set the Value property add the following code to your project:

- Visual Basic

```
C1MaskedTextBox1.Mask = "$999,999.00"
```

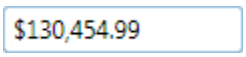
- C#

```
c1MaskedTextBox1.Mask = "$999,999.00";
```

Run your project and observe:

The mask will appear in the control:

Enter a number; notice that the mask is filled:



Changing the Prompt Character

The PromptChar property sets the characters that are used to prompt users in the C1MaskedTextBox control. By default the PromptChar property is set to an underline character ("_") but you can customize this at design time in Microsoft Expression Blend, in XAML, and in code. For more details about the PromptChar property, see Prompts.

At Design Time in Blend

To set the PromptChar property at in Blend, complete the following steps:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties tab and enter "0000" in the text box next to the Mask property to set a mask.
3. In the properties window, enter "#" (the pound character) in the text box next to the PromptChar property

In XAML

To set the PromptChar property add `Mask="0000" PromptChar="#"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" Name="C1MaskedTextBox1" Width="120"
Mask="0000" PromptChar="#"></c1:C1MaskedTextBox>
```

In Code

To set the PromptChar property add the following code to your project:

- Visual Basic

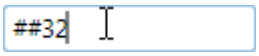
```
Dim x As Char = "#"c
C1MaskedTextBox1.Mask = "0000"
C1MaskedTextBox1.PromptChar = x
```

- C#

```
char x = '#';
this.c1MaskedTextBox1.Mask = "0000";
this.c1MaskedTextBox1.PromptChar = x;
```

Run your project and observe:

The pound character will appear as the prompt in the control. In the following image, the number 32 was entered in the control:



Changing Font Type and Size

You can change the appearance of the text in the grid by using the text properties in Microsoft Expression Blend's Properties tab, through XAML, or through code.

At Design Time n Blend

To change the font of the grid to Arial 10pt in Blend, complete the following:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties tab, and set **FontFamily** property to "Arial" (or a font of your choice).
3. In the Properties window, set the **FontSize** property to **10**.

   This will set the control's font size and style.

In XAML

For example, to change the font of the control to Arial 10pt in XAML add `FontFamily="Arial" FontSize="10"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" Name="C1MaskedTextBox1" Width="120"
FontSize="10" FontFamily="Arial"></c1:C1MaskedTextBox>
```

In Code

For example, to change the font of the grid to Arial 10pt add the following code to your project:

- Visual Basic

```
C1MaskedTextBox1.FontSize = 10
C1MaskedTextBox1.FontFamily = New System.Windows.Media.FontFamily("Arial")
```

- C#

```
c1MaskedTextBox1.FontSize = 10;
c1MaskedTextBox1.FontFamily = new
System.Windows.Media.FontFamily("Arial");
```

Run your project and observe:

The control's content will appear in Arial 10pt font:

Locking the Control from Editing

By default the C1MaskedTextBox control's Value property is editable by users at run time. If you want to lock the control from being edited, you can set the **IsReadOnly** property to **True**.

At Design Time in Blend

To lock the C1MaskedTextBox control from run-time editing, complete the following steps:

1. Click the C1MaskedTextBox control once to select it.
2. Navigate to the Properties tab and check the **IsReadOnly** check box.

   This will set the **IsReadOnly** property to **False**.

In XAML

To lock the C1MaskedTextBox control from run-time editing in XAML add `IsReadOnly="True"` to the `<c1:C1MaskedTextBox>` tag so that it appears similar to the following:

```
<c1:C1MaskedTextBox Height="23" Name="C1MaskedTextBox1" Width="120"
IsReadOnly="True"></c1:C1MaskedTextBox>
```

In Code

To lock the C1MaskedTextBox control from run-time editing add the following code to your project:

- Visual Basic

```
C1MaskedTextBox1.IsReadOnly = True
```

- C#

```
c1MaskedTextBox1.IsReadOnly = true;
```

Run your project and observe:

The control is locked from editing. Try to click the cursor within the control – notice that the text insertion point (the blinking vertical line) will not appear in the control.

Menu and ContextMenu

Add a complete menu system to your Silverlight app with **ComponentOne Menu™ for Silverlight** and **ComponentOne ContextMenu™ for Silverlight**. Use the C1Menu control to create menu and C1ContextMenu control to attach pop-up menu to your interface.

Menu for Silverlight includes the following controls:

- **C1ContextMenu**

  The C1ContextMenu provides a pop-up menu that provides frequently used commands that are associated with the selected object.

- **C1Menu**

  The C1Menu is a control that allows hierarchical organization of elements associated with event handlers.

Getting Started

- Working with Menus
- Quick Start
- Task-Based Help

Menu and ContextMenu for Silverlight Key Features

ComponentOne Menu for Silverlight and **ComponentOne ContextMenu for Silverlight** allow you to create customized, rich applications. Make the most of **Menu for Silverlight** by taking advantage of the following key features:

- **Browser Boundaries Detection**

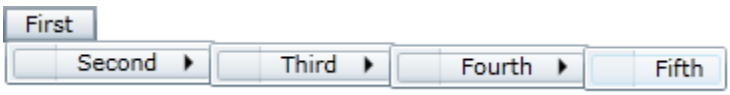
  Menus are positioned automatically and always stay within the page bounds. See Boundary Detection for more information.

- **Keyboard Navigation Support**

  The C1Menu control supports keyboard navigation, making your Silverlight applications more accessible.

- **Deep Menus**

  Nest menus to any depth. See Nested Submenus for more information.

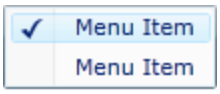


- **Icons for Menu Items**

  Menu allows you to use an icon for each menu item along with the menu label.

- **Checkable Menu Items**

  Declare menu items with checked/unchecked states. See Checkable Menu Items for more information.



- **Context Menus**

  The C1ContextMenu control enables you to provide pop-up menus that associate frequently used commands with selected objects. The C1ContextMenuService class exposes context menus as extender properties that can be attached to any **FrameworkElement** objects on the page, much like the **ToolTip** property provided by the **ToolTipService** class. See C1ContextMenu Elements for more information.

- **Silverlight Toolkit Themes Support**

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

Menu and ContextMenu for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **Menu for Silverlight** and **ContextMenu for Silverlight**. In this quick start, you'll start in Expression Blend to create a new project with the C1Menu control. You will also add menu items, submenus, and a context menu to the control.

Step 1 of 5: Creating an Application with a C1Menu Control

In this step, you'll begin in Expression Blend to create a Silverlight application using the C1Menu control.

Complete the following steps:

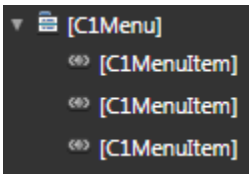
1. In Expression Blend, select **File | New Project**.
2. In the **New Project** dialog box, select the Silverlight project type in the left pane and, in the right-pane, select **Silverlight Application + Website**.
3. Enter a **Name** and **Location** for your project, select a **Language** in the drop-down box, and click **OK**. Blend creates a new application, which opens with the **MainPage.xaml** file displayed in Design view.
4. Add the C1Menu control to your project by completing the following steps:
   a. On the menu, select **Window | Assets** to open the **Assets** tab.
   b. Under the **Assets** tab, enter "C1Menu" into the search bar.
   c. The C1Menu control's icon appears.
   d. Double-click the C1Menu icon to add the control to your project.
5. Under the **Objects and Timeline** tab, select **[C1Menu]** and then, under the **Properties** panel, set the following properties:
   - Locate the **Width** property and click its glyph to set the width of the control to **Auto**.
   - Locate the **Height** property and click its glyph to set the height of the control to **Auto**.

You have completed the first step of the **Menu and ContextMenu for Silverlight** quick start. In this step, you created a project and added a C1Menu control to it. In the next step, you will add top-level menu items to the control.

Step 2 of 5: Adding Menu Items to the Control

In the last step, you created a Silverlight project with a C1Menu control attached to it. In this step, you will add three menu items to the C1Menu control.

1. Add a C1MenuItem icon to the **Tools** panel by completing the following steps:
   a. In the **Tools** panel, click the **Assets** button (the double-chevron button).
   b. In the search bar, enter "C1MenuItem" to bring up the **C1MenuItem** icon.
   c. Double-click the **C1MenuItem** icon to add it to the Tools panel.
2. Add a menu item to the menu by completing the following steps:
   a. Under the **Objects and Timeline** tab, select **[C1Menu]**.
   b. On the **Tools** panel, double-click the **C1MenuItem** icon (this is located beneath the Assets button) to add the **C1MenuItem** to the **C1Menu** control.
3. Repeat step 2 twice to add a total of three menu items to the C1Menu control. The hierarchy under the **Objects and Timeline** tab should appear as follows:



   If the hierarchy doesn't appear as shown in the above image, you can use drag-and-drop operations to rearrange the C1MenuItems.
4. Under the **Objects and Timeline** tab, select the first **[C1MenuItem]** and then, under the **Properties** panel, set the following properties:
   - Locate the **Width** property and click its glyph to set the width of the control to **Auto**.
   - Locate the **Height** property and click its glyph to set the height of the control to **Auto**.
   - Locate the **Header** property and set it to "File".
5. Under the **Objects and Timeline** tab, select the second **[C1MenuItem]** and then, under the **Properties** panel, set the following properties:
   - Locate the **Width** property and click its glyph to set the width of the control to **Auto**.
   - Locate the **Height** property and click its glyph to set the height of the control to **Auto**.
   - Locate the **Header** property and set it to "Edit".
6. Under the **Objects and Timeline** tab, select the third **[C1MenuItem]** and then, under the **Properties** panel, set the following properties:
   - Locate the **Width** property and click its glyph to set the width of the control to **Auto**.
   - Locate the **Height** property and click its glyph to set the height of the control to **Auto**.
   - Locate the **Header** property and set it to "Added Items".

In this step, you added top-level menu items to the C1Menu control in Expression Blend. In the next step, you will use XAML to add submenus to two of those items.

Step 3 of 5: Adding Submenus to the Menu Items

In the last step, you added the top-level menu items in Expression Blend. In this step, you will add submenus to two of the menu items using XAML.

1. Switch to XAML view.
2. Modify the first two <c1:C1MenuItem> tags so that they have both opening and closing tags. They should resemble the following:

```
<c1:C1MenuItem Header="File"></c1:C1MenuItem>
<c1:C1MenuItem Header="Edit"></c1:C1MenuItem>
```

   Adding opening and closing tags to these items will allow you to nest other C1MenuItems between these tags in the next step. This step is necessary to create submenus in XAML.
3. To add a submenu to the "File" menu item, add the following markup between the <c1:C1MenuItem Header="File"> and </c1:C1MenuItem> tags:

```
<c1:C1MenuItem Height="Auto" Width="Auto" Header="New">
      <c1:C1MenuItem
      Height="Auto"
      Width="Auto"
      Header="Document"/>
      <c1:C1MenuItem Height="Auto" Width="Auto" Header="Project"/>
</c1:C1MenuItem>
<c1:C1MenuItem Height="Auto" Width="Auto" Header="Open">
      <c1:C1MenuItem
           Height="Auto"
           Width="Auto"
           Header="Document"/>
      <c1:C1MenuItem
           Height="Auto"
           Width="Auto"
           Header="Project"/>
      <c1:C1Separator/>
      <c1:C1MenuItem Header="Recent Document 1"
           Height="Auto"
           Width="Auto"
           GroupName="CheckedDocuments"
           IsCheckable="True"
           IsChecked="True">
</c1:C1MenuItem>
      <c1:C1MenuItem
           Header="Recent Document 2"
           Height="Auto"
```

```
            Width="Auto"
            GroupName="CheckedDocuments"
            IsCheckable="True">
        </c1:C1MenuItem>
    </c1:C1MenuItem>
    <c1:C1Separator/>
    <c1:C1MenuItem
        Height="Auto"
        Width="Auto"
        Header="Close"/>
    <c1:C1MenuItem
        Height="Auto"
        Width="Auto"
        Header="Close Solution"/>
    <c1:C1Separator/>
    <c1:C1MenuItem
        Height="Auto"
        Width="Auto"
        Header="Exit"/>
```

4. To add a submenu to the "Edit" menu item, following markup between the `<c1:C1MenuItem Header="Edit">` and `</c1:C1MenuItem>` tags:

```
    <c1:C1MenuItem
        Height="Auto"
        Width="Auto"
        Header="Undo"/>
    <c1:C1MenuItem
        Height="Auto"
        Width="Auto"
        Header="Redo"/>
```

In this step, you added submenus to two of the C1Menu control's menu items. In the next step, you will add a C1ContextMenu control to the C1Menu control.

Step 4 of 5: Adding a C1ContextMenu to the C1Menu Control

In the last step, you added submenus to two of the C1Menu control's menu items. In this step, you will add a C1ContextMenu control to the C1Menu control. This context menu will have one item that, when clicked, will add submenu items to the C1Menu control's top-level "Added Items" top-level menu item that you created in Step 2 of 5: Adding Menu Items to the Control.

Complete the following steps:

1. In XAML view, place the following XAML markup right before the `</c1:C1Menu>` tag:

```
<c1:C1ContextMenuService.ContextMenu>
     <c1:C1ContextMenu Width="Auto" Height="Auto">
          <c1:C1MenuItem
               Height="Auto"
               Width="Auto"
               Header="Add Item"
               Click="C1MenuItem_AddOnClick"/>
     </c1:C1ContextMenu>
</c1:C1ContextMenuService.ContextMenu>
```

   The above markup adds a C1ContextMenu control to the C1Menu control using the **C1ContextMenuService** helper class. Note that the C1ContextMenu control contains one C1MenuItem that is attached to a **Click** event named "C1MenuItem_AddOnClick".

2. Add `x:Name="AddedItems"` to the `<c1:C1MenuItem Header="Added Items"/>` tag. This gives the item a unique identifier so that you can call it in code.
3. Open the **MainPage.xaml.cs** page and add the following **Click** event handler to the project:
   - Visual Basic

```
Private Sub C1MenuItem_AddOnClick(ByVal sender As Object, ByVal
e As C1.Silverlight.SourcedEventArgs)
      Dim C1MenuItemAdd As New C1.Silverlight.C1MenuItem()
      C1MenuItemAdd.Header = "Added Item"
      AddedItems.Items.Add(C1MenuItemAdd)
 End Sub
```

   - C#

```
private void C1MenuItem_AddOnClick(object sender,
C1.Silverlight.SourcedEventArgs e)
 {
      C1.Silverlight.C1MenuItem C1MenuItemAdd = new
C1.Silverlight.C1MenuItem();
      C1MenuItemAdd.Header = "Added Item";
      AddedItems.Items.Add(C1MenuItemAdd);
 }
```

   When the **Click** event is raised, this code will create a new C1MenuItem and add it to the "Added Items" menu item.

4. Click the **Projects** tab and then double-click **Default.htm** to open the HTML document.

Note: If you are using Silverlight 4, you can skip step 5.

5. In **Default.htm**, add `<param name="windowless" value="true" />` between the `<Object>` and `</Object>` tags. This makes the application windowless so that users can right-click the C1Menu control to bring up the C1ContextMenu control.

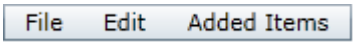
In this step, you added a C1ContextMenu control to the C1Menu control. In the next step, you will run the project and see the result of the **Menu for Silverlight** quick start.

Step 5 of 5: Running the Project

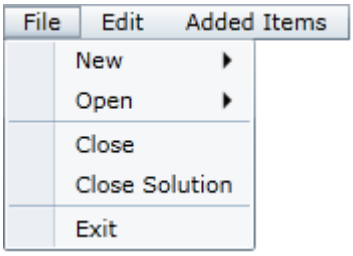
Now that you have created a Silverlight project with a C1Menu control and a C1ContextMenu control, the only thing left to do is run the project and observe the results of your work.

Complete the following steps:

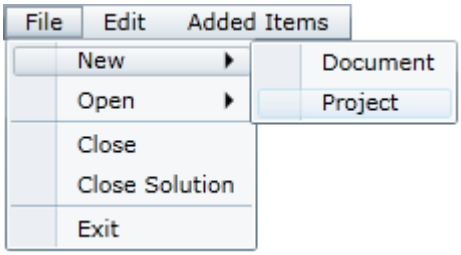
1. From the toolbar, select **Project | Run Project** to run the application. The application appears as follows:



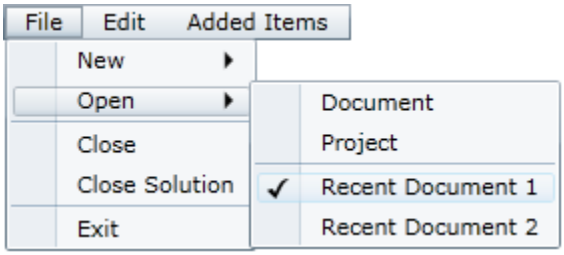
2. Click **File** and observe that a submenu appears.



3. Hover your cursor over **New** and observe that another submenu appears.



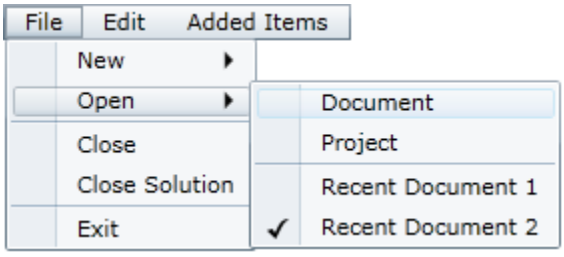
4. Hover your cursor over **Open** and observe that another submenu appears. Note that **Recent Document 1** is checked.



5. Click **Recent Document 2**.

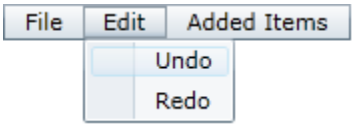
   The **Open** submenu closes.

6. Select **File | Open** and observe that **Recent Document 2** is checked instead of **Recent Document 1**.

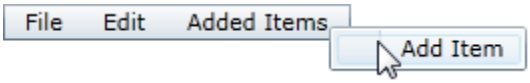


   The two checkable items, **Recent Document 1** and **Recent Document 2**, are grouped together to form a list of mutually exclusive checkable items. You can read more about this by visiting the Checkable Menu Items topic.

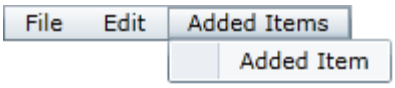
7. Click **Edit** and observe that the **Edit** submenu appears.



8. Click **Added Items** and observe that it has no submenu.
9. Right-click the menu to bring up the context menu.
10. On the context menu, click **Add Item**.



11. Click **Added Items** and observe that it has a submenu consisting of one new item, which is named **Added Item**.



Congratulations! You have completed the **Menu for Silverlight** quick start. Now that you have learned some of the basics of the control, we recommend that you visit the Working with C1Menu and C1ContextMenu section for an overview of the controls and their features.

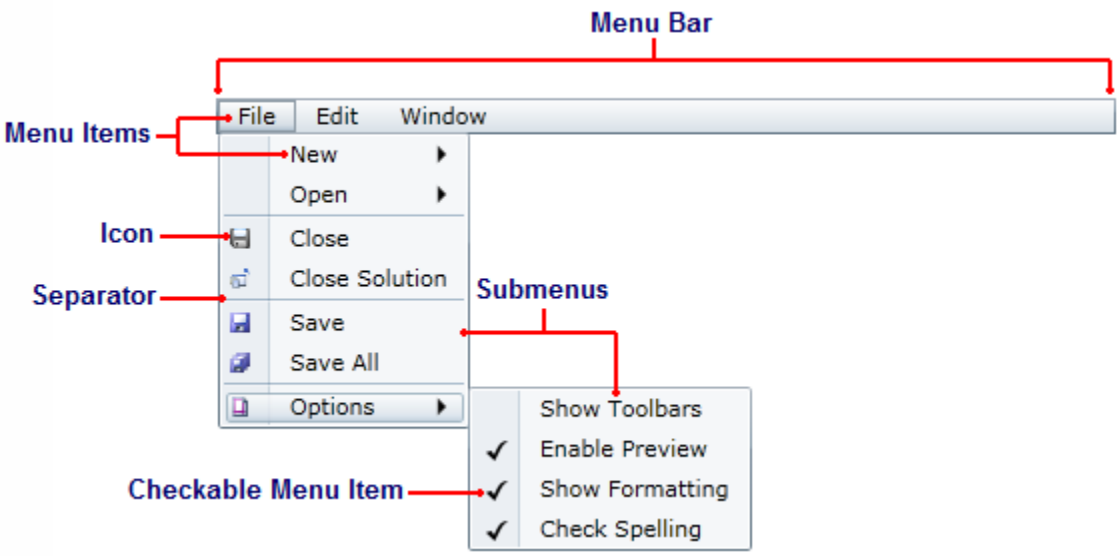
Working with C1Menu and C1ContextMenu

The following topics outline the basics of working with the C1Menu and C1ContextMenu controls. In this section, you will find outlines of the controls' elements and descriptions of some of the controls' most popular features.

C1Menu Elements

The C1Menu is a control that allows hierarchical organization of elements associated with event handlers. The menus can be nested to any depth that you desire, and you can add as many items to the menu as you need to add.

The following image diagrams the elements of the C1Menu control.



The elements of the C1Menu control can be described as follows:

- **Menu Bar**

  The main menu is a horizontal, top-level menu. It is comprised of first-level C1MenuItems and can hold any feature available to C1MenuItems.

- **Submenus**

  Submenus are menus that can only be accessed from other menus. A submenu is created when you nest a C1MenuItem within another C1MenuItem.

- **Menu Items**

  Menu items are represented by the C1MenuItem class. Menu items can have labels and icons, and they can also be specified as checkable menu items. Each menu item is associated with a click event handler.

- **Checkable Menu Item**

Checkable menu items are objects of the C1MenuItem class that can be selected or cleared by users. You can make checkable menu items standalone, or you can group them with other checkable menu items to create a list of mutually exclusive check boxes. Checkable menu items can't be used in the menu bar.

- **Icon**

  Icons are created by adding an **Image** control to the Icon property. Icons can't be used in the menu bar.

- **Separator**

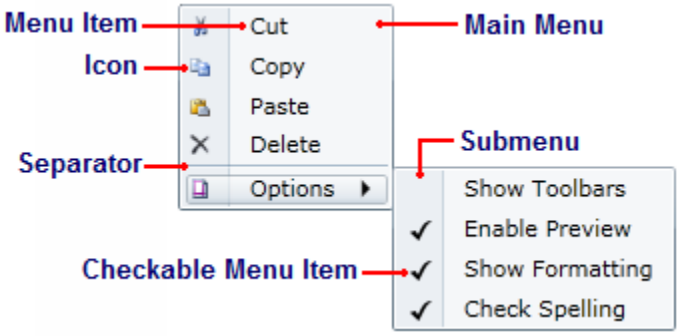
  Separators are created through the C1Separator class. Icons can't be used in the menu bar.

C1ContextMenu Elements

The C1ContextMenu provides a pop-up menu that provides frequently used commands that are associated with the selected object. The C1ContextMenuService class exposes context menus as extender properties that can be attached to any **FrameworkElement** objects on the page, much like the **ToolTip** property provided by the **ToolTipService** class. Once the menu is attached to an object, the users can right-click that object to open the menu.

Note: The C1ContextMenu will only work in a windowless Silverlight application prior to Silverlight 4. For information about creating a windowless application, see Creating a Context Menu.

The following image diagrams the elements of the C1ContextMenu control.



The elements of the C1Menu control can be described as follows:

- **Main Menu**

  The main menu is a horizontal bar comprised of first-level C1MenuItems. Menu items can be standalone items, or they can contain lists of submenu items.

- **Submenus**

  Submenus are menus that can only be accessed from other menus. A submenu is created when you nest a C1MenuItem within another C1MenuItem.

- **Menu Items**

  Menu items are represented by the C1MenuItem class. Menu items can have labels and icons, and they can also be specified as checkable menu items. Each menu item is associated with a click event handler.

- **Checkable Menu Item**

  Checkable menu items are objects of the C1MenuItem class that can be selected or cleared by users. You can make checkable menu items standalone, or you can group them with other checkable menu items to create a list of mutually exclusive check boxes.

- **Icon**

  Icons are created by adding an **Image** control to the Icon property.

- **Separator**

  Separators are created through the C1Separator class. They can be used to create visual groups of menu items.

C1Menu and C1ContextMenu Features

The C1Menu control and the C1ContextMenu control share a common object model, and both controls take C1MenuItem items. This means that the C1Menu and C1ContextMenu controls possess the same features. This section outlines some popular features shared by the menu controls.

Automatic Closing

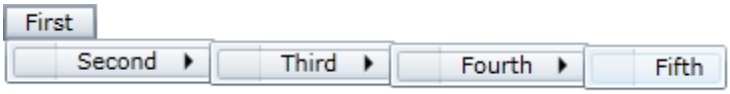
By default, the C1ContextMenu control, its submenus, and the submenus of a C1Menu control will remain open even when a user clicks outside of the menu. The only way users can close the menu is if they click the C1Menu control or the C1ContextMenu control. However, you can change this behavior by enabling the automatic closing feature, which will allow users to close menus by clicking outside of the menu control's boundaries. To turn on automatic closing, set the AutoClose property to **True**.

Nested Submenus

The C1Menu control and the C1ContextMenu control can hold submenus. These submenus are created when C1MenuItems are nested within the tags of other C1MenuItems. For example, placing the following XAML markup

```
<c1:C1MenuItem Header="First">
     <c1:C1MenuItem Header="Second">
          <c1:C1MenuItem Header="Third">
               <c1:C1MenuItem Header="Fourth">
                    <c1:C1MenuItem Header="Fifth"/>
</c1:C1MenuItem>
```

between the opening and closing tags of a C1Menu would create the following:



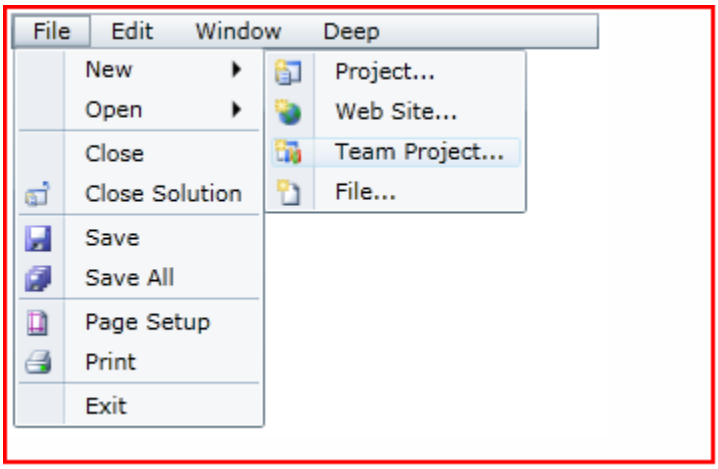
You can have as many nested menus as you want, although it's best not to have more than two or three submenus in a hierarchy for usability purposes.

For task-based help on creating a nested submenu for a C1Menu or a C1ContextMenu control, see Creating a Submenu.

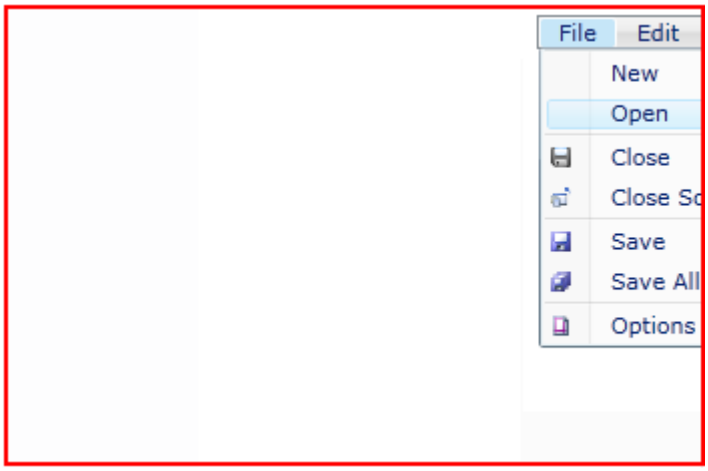
Boundary Detection

Boundary detection ensures that the menus a user opens will always stay within the page bounds. This is helpful if you have several nested submenus within your menu, as these menus can expand until they're beyond the scope of a project page.

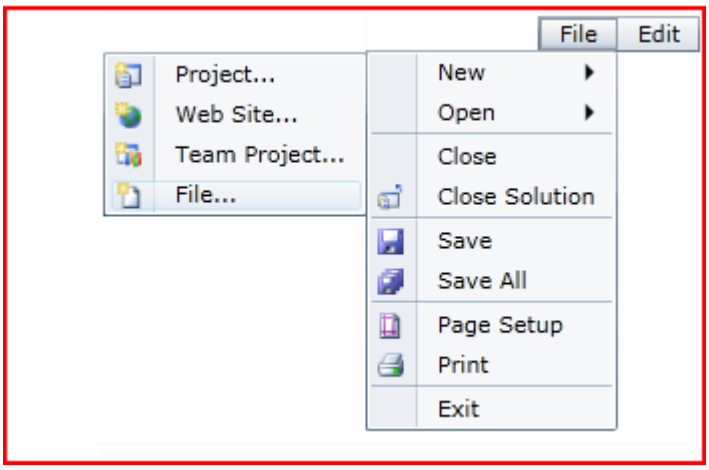
The image below illustrates a menu contained with a boundary large enough to allow for the expansion of two submenus.



The image below illustrates what that same menu would look like if the boundaries were shifted several centimeters to the left with the boundary detection feature disabled. Observe that the menu is cut off despite there being an extensive blank space to the left of the control.



The image below illustrates that same menu and same boundary, only this time boundary detection is enabled. Observe that the submenus shift to the left to avoid being cut off by the tight boundary on the right side of the page.



By default, boundary detection is disabled, but you can enable it by setting the DetectBoundaries property to **True**. For task-based help about enabling boundary detection, see the Enabling Boundary Detection topic.

Checkable Menu Items

You can make any C1MenuItem a checkable menu item by setting its IsCheckable property to **True**. The value of the IsChecked property determines whether or not the menu item is checked or unchecked. By default, the IsChecked property of an item is **False**. When the item is clicked, the value of the IsChecked property sets to **True**.

You can create a group of mutually exclusive checkable items by setting the GroupName property of each item you wish to add to the group. For example, the XAML below will create a group of three mutually exclusive checkable items.

```
<c1:C1MenuItem Header="MenuItem1" IsCheckable="True"
GroupName="MutuallyExclusiveGroup"/>
<c1:C1MenuItem Header="MenuItem2" IsCheckable="True"
GroupName="MutuallyExclusiveGroup" />
<c1:C1MenuItem Header="MenuItem3" IsCheckable="True"
GroupName="MutuallyExclusiveGroup" />
```

Working with C1ScrollViewer

C1ScrollViewer is used by **C1Menu** and is a replacement for the standard **ScrollViewer** control. **C1ScrollViewer** supports scrolling in both horizontal and vertical directions. **C1ScrollViewer** includes some advantages over using the Standard **ScrollViewer** control:

- **ClearStyle support**

  ComponentOne ClearStyle makes it easy to customize the appearance of the control simply by changing a few properties.

- **Unique and Unified Style**

  The **C1ScrollViewer** control has a unique style, different from the standard control. The **C1ScrollViewer** also control uses the same style in both **Studio for Silverlight** and **Studio for WPF** versions. The Standard **ScrollViewer** uses different styles for the two platforms.

- **Scrolling on MouseOver**

  C1ScrollViewer allows users the ability to scroll on mouseover at run time. You can customize this feature using the **ScrollMode** property.

The **C1ScrollViewer** element encapsulates a content element and up to two **ScrollBar** controls. The extent includes all the content of the **C1ScrollViewer**. The visible area of the content is the viewport.

The HorizontalScrollBarVisibility and VerticalScrollBarVisibility properties control the conditions under which the vertical and horizontal ScrollBar controls appear.

Menu and ContextMenu for Silverlight Layout and Appearance

The following topics detail how to customize the C1Menu and C1ContextMenu controls' 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.

C1Menu and ContextMenu ClearStyle Properties

Menu and ContextMenu 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 **C1Menu** control:

Brush	Description
Background	Gets or sets the brush of the control's background.
HighlightedBackground	Gets or sets the System.Windows.Media.Brush used to highlight the menu items when they are hovered over.
OpenedBackground	Gets or sets the System.Windows.Media.Brush used to highlight the menu items when they are opened.

The following table outlines the brush properties of **C1ContextMenu** control:

ClearStyle Property	Description
Background	Gets or sets the brush of the control's background.
HighlightedBackground	Gets or sets the System.Windows.Media.Brush used to highlight the menu items when they are hovered over.
OpenedBackground	Gets or sets the System.Windows.Media.Brush used to highlight the menu items when they are opened.

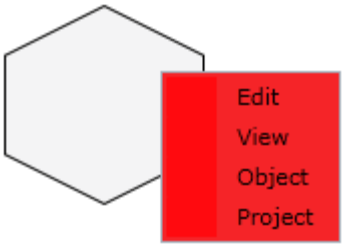
The following table outlines the brushes of the **C1MenuItem** control:

ClearStyle Property	Description
Background	Gets or sets the brush of the control's background.
HighlightedBackground	Gets or sets the System.Windows.Media.Brush used to highlight the menu item when it is hovered over.
OpenedBackground	Gets or sets the System.Windows.Media.Brush used to highlight the menu item when it is opened.
HeaderForeground	Gets or sets the foreground brush of the header.

HeaderBackground	Gets or sets the background brush of the header.

You can completely change the appearance of the **C1Menu** or **C1ContextMenu** controls by setting a few properties, such as the **C1Menu**'s **Background** property, which sets the background color of the menu. For example, if you set the **Background** property to "#FFE40005", the **C1Menu** control would appear similar to the following:

And if you set the **C1ContextMenu**'s **Background** property to "#FFE40005", it would appear similar to the following:



It's that simple with 'ComponentOne's ClearStyle technology.

C1Menu and C1ContextMenu Appearance Properties

ComponentOne Menu and ContextMenu 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 C1Menu control, the C1ContextMenu control, and the C1MenuItem items.

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 of header and content area content in the C1Menu control, the C1ContextMenu control, and the C1MenuItem items.

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.

Color Properties

The following properties let you customize the colors used in the C1Menu control, the C1ContextMenu control, and the C1MenuItem items.

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 borders of the C1Menu control, the C1ContextMenu control, and the C1MenuItem items.

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 C1Menu control, the C1ContextMenu control, and the C1MenuItem items.

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.

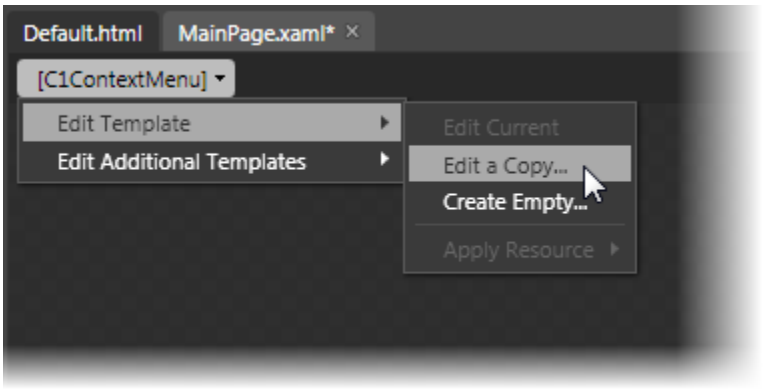
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

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 Menu 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 C1Menu or C1ContextMenu 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.The image below illustrates this using the C1ContextMenu control.



If you want to edit a C1MenuItem template, simply select the C1MenuItem 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 Menu Templates

In addition to the default templates, the C1Menu control and the C1ContextMenu control include a few additional templates. These additional templates can also be accessed in Microsoft Expression Blend – in Blend select the C1Menu or C1ContextMenu control and, in the menu, select **Edit Additional Templates**. Choose a template, and select **Create Empty**,

Additional Menu Item Templates

In addition to the default templates, the C1Menu control and the C1MenuItem control include a few additional templates. These additional templates can also be accessed in Microsoft Expression Blend – in Blend select the

C1Menu or C1MenuItem control and, in the menu, select **Edit Additional Templates**. Choose a template, and select **Create Empty.**

Item Templates

ComponentOne Menu for Silverlight's menu and context controls are **ItemsControl**s that serve as a container for other elements. As such, both controls include templates to customize items places within the menu. These templates include an **ItemTemplate** and an **ItemsPanel** 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 C1Menu or C1ContextMenu 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**.

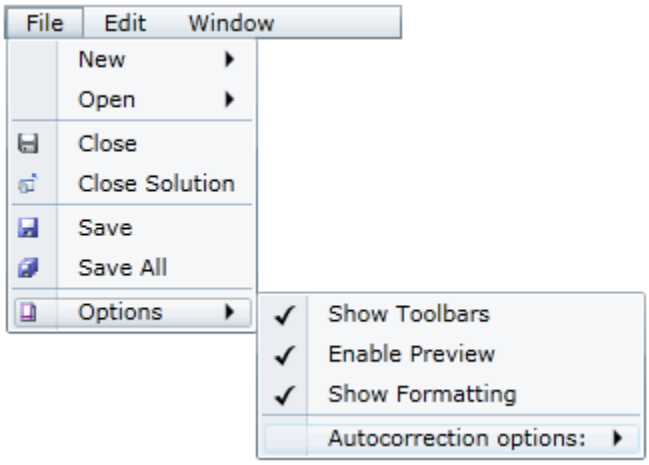
A dialog box will appear allowing you to name the template and determine where to define the template.

Menu 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.

> **Note**: This topic only illustrates themes on the C1Menu control. However, the themes for the C1ContextMenu control are identical to the theming of the C1Menu control's submenus.

When you add the C1Menu control to your project, it appears with the default blue theme:



You can theme the C1Menu control with one of our six included Silverlight themes: BureauBlack, Cosmopolitan, ExpressionDark, ExpressionLight, RainierOrange, ShinyBlue, and WhistlerBlue. The table below shows a sample of each theme.

You can add any of these themes to the menu controls by declaring the theme around the control in markup. For task-based help about adding a theme to the menu controls, see Working with Themes.

Menu and ContextMenu 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 C1Menu control in general. If you are unfamiliar with the **ComponentOne Menu for Silverlight** and

ComponentOne ContextMenu for Silverlight products, please see the **Menu and ContextMenu for Silverlight** Quick Start first.

Each topic in this section provides a solution for specific tasks using the **C1Menu** and **C1ContextMenu** controls.

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

Creating Menus

The following topics detail how to create different types of menus.

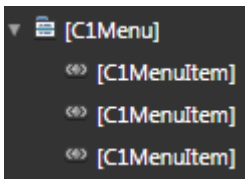
Creating a Top-Level Menu

In this topic, you will learn how to create a top-level menu for the C1Menu and C1ContextMenu controls.

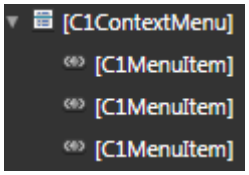
In Blend

Complete the following steps:

1. Add a C1Menu control or a C1ContextMenu (see Creating a Context Menu) control to your project.
2. Select the C1Menu or the C1ContextMenu control and set the following properties:
   - Locate the **Width** property and click its glyph to set the width of the control to **Auto**.
   - Locate the **Height** property and click its glyph to set the height of the control to **Auto**.
3. Add a C1MenuItem icon to the **Tools** panel by completing the following steps:
   a. In the **Tools** panel, click the **Assets** button (the double-chevron button).
   b. In the search bar, enter "C1MenuItem" to bring up the **C1MenuItem** icon.
   c. Double-click the **C1MenuItem** icon to add it to the Tools panel.
4. Add a menu item to the menu by completing the following steps:
   a. Under the **Objects and Timeline** tab, select **[C1Menu]** or **[C1ContextMenu]**.
   b. On the **Tools** panel, double-click the **C1MenuItem** icon (this is located beneath the double-chevron button) to add the **C1MenuItem** to the control.
5. Repeat step 2 twice to add a total of three menu items to the C1Menu control. The hierarchy under the **Objects and Timeline** tab should appear as follows:
   - For a C1Menu



   - For a C1ContextMenu



6. Select the first **[C1MenuItem]** and set the following properties:
   - Locate the **Width** property and click its glyph to set the width of the control to **Auto**.
   - Locate the **Height** property and click its glyph to set the height of the control to **Auto**.
   - Locate the **Header** property and set it to "MenuItem1".
7. Select the second **[C1MenuItem]** and set the following properties:
   - Locate the **Width** property and click its glyph to set the width of the control to **Auto**.
   - Locate the **Height** property and click its glyph to set the height of the control to **Auto**.
   - Locate the **Header** property and set it to "MenuItem2".
8. Select the third **[C1MenuItem]** and set the following properties:
   - Locate the **Width** property and click its glyph to set the width of the control to **Auto**.
   - Locate the **Height** property and click its glyph to set the height of the control to **Auto**.
   - Locate the **Header** property and set it to "MenuItem3".

> **Note**: If you are working with the C1ContextMenu control, you will have to attach it to another FrameWork element using the C1ContextMenuService class. For information about attaching C1ContextMenu to an element, see Creating a Context Menu.

9. Run the program and complete one of the following:
   - If you are using a C1Menu control, observe that a menu bar with three items appears.

     OR
   - If you are using a C1ContextMenu control, right-click the element the context menu is attached to. Observe that a menu with three items appears.

In XAML

Complete the following steps:

1. Place the following XAML between the `<c1:C1Menu>` and `</c1:C1Menu>` tags or the `<c1:C1ContextMenu>`and `</c1:C1ContextMenu>` tags.

```
<c1:C1MenuItem Header="MenuItem1" Height="Auto" Width="Auto"/>
<c1:C1MenuItem Header="MenuItem2" Height="Auto" Width="Auto"/>
<c1:C1MenuItem Header="MenuItem3" Height="Auto" Width="Auto"/>
```

2. Add `Height="Auto"` and `Width="Auto"` to the `<c1:C1Menu>` or `<c1:C1ContextMenu>` tag.
3. Run the program and complete one of the following:
   - If you are using a C1Menu control, observe that a menu bar with three items appears.

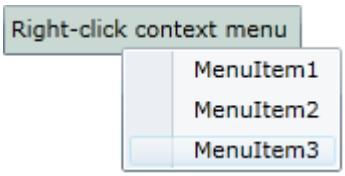
     OR
   - If you are using a C1ContextMenu control, right-click the element the context menu is attached to. Observe that a menu with three items appears.

This Topic Illustrates the Following:

This topic illustrates one of the following results:

- If you used a C1Menu to complete this task, the result will resemble the following:

- If you used a C1ContextMenu to complete this task, the result will resemble the following:



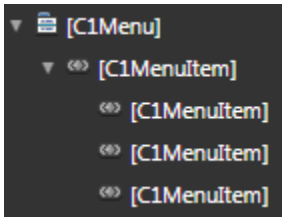
Creating a Submenu

In this topic, you will create a submenu that's attached to one of a menu's items. This topic assumes that you have created a top-level menu (see Creating a Top-Level Menu) with at least one menu item.

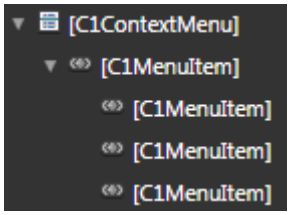
Complete the following steps:

1. Add a C1Menu control or a C1ContextMenu control to your project.
2. Select the C1Menu or the C1ContextMenu control and set the following properties:
   - Locate the **Width** property and click its glyph to set the width of the control to **Auto**.
   - Locate the **Height** property and click its glyph to set the height of the control to **Auto**.
3. Under the **Assets** tab, enter "C1MenuItem" in the search bar.

   The C1MenuItem icon appears.
4. Double-click the C1MenuItem icon to add a menu item the menu control. Repeat this step three times to add a total of four menu items to your project.
5. Switch to the **Objects and Timeline** tab.
6. Use drag-and-drop operations to create the following hierarchy:
   - For a C1Menu



   - For a C1ContextMenu



7. Select the first **[C1MenuItem]** and set the following properties:
   - Locate the **Width** property and click its glyph to set the width of the control to **Auto**.
   - Locate the **Height** property and click its glyph to set the height of the control to **Auto**.
   - Locate the **Header** property and set it to "Click here".
8. Set the properties of each submenu item as follows:
   - Locate the **Width** property and click its glyph to set the width of the control to **Auto**.
   - Locate the **Height** property and click its glyph to set the height of the control to **Auto**.
   - Locate the **Header** property and set it to "Submenu Item".

> **Note**: If you are working with the C1ContextMenu control, you will have to attach it to another **FrameWork** element using the C1ContextMenuService class. For information about attaching C1ContextMenu to an element, see Creating a Context Menu.

9. Run the program and complete one of the following:
   - If you are using a C1Menu control, click **Click here**. Observe that a submenu with three items appears.

     OR
   - If you are using a C1ContextMenu control, right-click the element the context menu is attached to. Click **Click Here** and observe that a submenu with three items appears.

In XAML

Complete the following steps:

1. Place the following XAML between the `<c1:C1Menu>` and `</c1:C1Menu>` tags or the `<c1:C1ContextMenu>`and `</c1:C1ContextMenu>` tags.

```
      <c1:C1MenuItem Header="Click here" Height="Auto" Width="Auto">
           <c1:C1MenuItem Header="Submenu Item" Height="Auto"
Width="Auto"/>
           <c1:C1MenuItem Header="Submenu Item" Height="Auto"
Width="Auto"/>
           <c1:C1MenuItem Header="Submenu Item" Height="Auto"
Width="Auto"/>
      </c1:C1MenuItem>
```

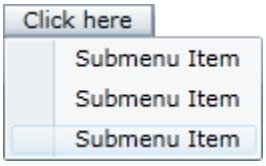
2. Add `Height="Auto"` and `Width="Auto"` to the `<c1:C1Menu>` or `<c1:C1ContextMenu>` tag.
3. Run the program and complete one of the following:
   - If you are using a C1Menu control, click **Click here**. Observe that a submenu with three items appears.

     OR
   - If you are using a C1ContextMenu control, right-click the element you attached it to in order to open the context menu. Click **Click Here** and observe that a submenu with three items appears.

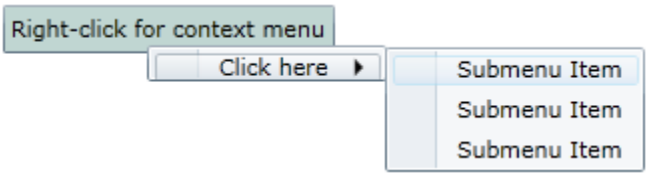
This Topic Illustrates the Following:

This topic illustrates one of the following results:

- If you used a C1Menu to complete this task, the result will resemble the following:



- If you used a C1ContextMenu to complete this task, the result will resemble the following:



Creating a Context Menu

In this topic, you will use the C1ContextMenuService class to attach a context menu to a **TextBox** control.

Complete the following steps:

1. Add a reference to the **C1.Silverlight** assembly to your Silverlight project.
2. Open Source view (XAML view if you're using Blend).
3. Add the following markup to the <UserControl> tag to import the namespace of the assembly:

```
xmlns:c1="clr-namespace:C1.Silverlight;assembly=C1.Silverlight"
```

4. Replace the <Grid> tag with the following markup to add a **TextBox** control to the project:

```
<Grid x:Name="LayoutRoot" Background="White">
    <TextBox Text="Right-click for context menu"  Width="170" Height="25"
Background="#FFC4D8D4">
    </TextBox>
</Grid>
```

5. Add C1ContextMenuService to the **TextBox** by placing the following markup between the <TextBox> and </TextBox> tags:

```
<c1:C1ContextMenuService.ContextMenu>
</c1:C1ContextMenuService.ContextMenu>
```

The C1ContextMenuService class exposes context menus as extender properties that can be attached to any **FrameworkElement** objects on the page.

6. Place the following XAML between the `<c1:C1ContextMenuService.ContextMenu>` and `</c1:C1ContextMenuService.ContextMenu>` tags to create a context menu and a few context menu items:

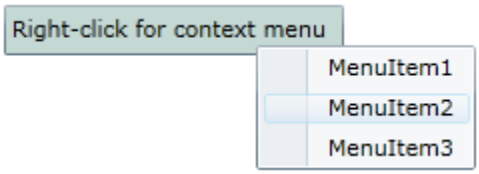
```
<c1:C1ContextMenu Width="Auto" Height="Auto">
     <c1:C1MenuItem Header="MenuItem1"></c1:C1MenuItem>
     <c1:C1MenuItem Header="MenuItem2"></c1:C1MenuItem>
     <c1:C1MenuItem Header="MenuItem3"></c1:C1MenuItem>
</c1:C1ContextMenu>
```

Note: If you are using Silverlight 4, you can skip step 7.

7. In order to get the C1ContextMenu control to work properly, you will have to change your project to a windowless application. To accomplish this, open the **[ProjectName]TestPage.aspx** page (**Default.htm** if you're using Blend) and add the following markup between the `<Object>` and `</Object>` tags:

```
<param name="windowless" value="true" />
```

8. Run the project and right-click the **TextBox** control. Observe that a context menu with three items appears.



Working with Themes

The following topics illustrate how to add Silverlight themes to the C1ContextMenu control and the C1Menu control.

Adding a Theme to the C1ContextMenu Control

Because the C1ContextMenu control is always attached to another control using a service, adding a theme to the C1ContextMenu control is going to be slightly different than adding a theme to another Silverlight control. You will have to wrap the theme around a grid or a panel and then use the implicit style manager to ensure that the theme will be passed down to the C1ContextMenu control.

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. Navigate to the **Tools** panel and double-click the **Grid** icon to add a grid to your project.
5. Click the **Objects and Timeline** tab.
6. Select **[Grid]** and, using a drag-and-drop operation, place it underneath **[C1ThemeRanierOrange]**.

7. With **[Grid]** still selected, return to the **Assets** tab.
8. Add a control, such as a **TextBox** control or a **Border** control, to the grid and then add a C1ContextMenu control to that control (see Creating a Context Menu).
9. Switch to XAML view.
10. Add the following namespace to the `<UserControl>` tag:

```
xmlns:c1Theming="clr-
namespace:C1.Silverlight.Theming;assembly=C1.Silverlight.Theming"
```

11. Add `c1Theming:ImplicitStyleManager.ApplyMode="OneTime"` to the `<c1:C1ContextMenu>` tag that your markup resembles the following:

```
<c1:C1ContextMenu c1Theming:ImplicitStyleManager.ApplyMode="Auto">
```

12. Run your 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 and press **Enter**.
5. In the **Tools** panel, double-click the **Grid** icon. The grid tags appear between the theme tags, as follows:

```
<my:C1ThemeRainierOrange>
    <Grid></Grid>
</my:C1ThemeRainierOrange>
```

6. Add a C1ContextMenu between the `<Grid>` and `</Grid>` tags. For task-based help about adding a context menu to a Visual Studio project, see Creating a Context Menu.
7. Add following namespace to the `<UserControl>` tag:

```
xmlns:c1Theming="clr-
namespace:C1.Silverlight.Theming;assembly=C1.Silverlight.Theming"
```

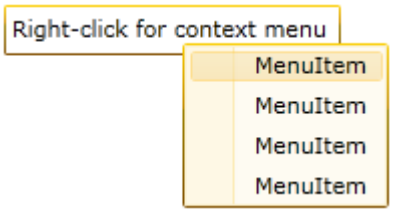
8. Set the by **ApplyMode** property by adding `c1Theming:ImplicitStyleManager.ApplyMode="Auto"` to the `<c1ext:C1Accordion>` tag so that your markup resembles the following:

```
<c1:C1ContextMenu c1Theming:ImplicitStyleManager.ApplyMode="Auto">
```

9. Run your project.

This Topic Illustrates the Following:

The following image depicts a C1ContextMenu control with the C1ThemeRainierOrange theme.



Adding a Theme to the C1Menu Control

The C1Menu control comes equipped with a light blue default theme, but you can also apply six themes (see Menu Theming) to the control. In this topic, you will change the C1Menu 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 "C1Menu" to search for the C1Menu control.
5. Double-click the C1Menu icon to add the C1Menu control to your project.
6. Under the **Objects and Timeline** tab, select **[C1Menu]** 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>
```

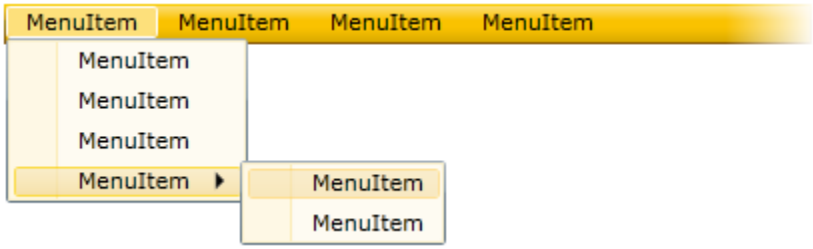
4. Place your cursor between the `<my:C1ThemeRainierOrange>` and `</my:C1ThemeRainierOrange>` tags.
5. In the **Tools** panel, double-click the C1Menu icon to add the control to the project. Its tags will appear as children of the `<my:C1ThemeRainierOrange>` tags, causing the markup to resemble the following:

```
<my:C1ThemeRainierOrange>
            <c1:C1Menu></c1:C1Menu>
</my:C1ThemeRainierOrange>
```

6. Run your project.

This Topic Illustrates the Following:

The following image depicts a C1Menu control with the C1ThemeRainierOrange theme.



Working with Checkable Menu Items

In the following topics, you will learn how to create standalone and mutually exclusive checkable menu items.

Creating a Checkable Menu Item

In this topic, you will create a checkable menu item that can be selected or cleared by a user. In order to complete this topic, you must have a C1ContextMenu control that holds at least one item or a C1Menu control with at least one submenu.

In Blend

Complete the following steps:

1. In the **Objects and Timeline** tab, select the **[C1MenuItem]** that you wish to make checkable.
2. Under the **Properties** panel, select the IsCheckable check box to set the IsCheckable property to **True**. This makes the item checkable at run time.

   Optional: If you want the item to be checked at run time, you can also select the IsChecked check box, but doing so isn't necessary to make an item checkable.
3. Run the project.

In XAML

Complete the following steps:

1. Locate the `<c1:C1MenuItem>` tag for the menu item you wish to make checkable and then add `IsCheckable="True"` to the tag so that the XAML resembles the following:

   ```
   <c1:C1MenuItem Header="C1MenuItem" IsCheckable="True"/>
   ```

   Optional: If you want the item to be checked at run time, you can also add `IsChecked="False"` to the tag.
2. Run the project.

In Code

Complete the following steps:

1. In Source view (XAML view if you're using Blend), locate the `<c1:C1MenuItem>` tag for the item you wish to make checkable and add `x:Name="CheckableMenuItem"` to it. This will give the item a unique identifier that you can use in code.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     CheckableMenuItem.IsCheckable = True
     ```

   - C#

     ```
     CheckableMenuItem.IsCheckable = true;
     ```

   Optional: If you want the item to be checked at run time, you can add the following code to the project:

   - Visual Basic

     ```
     CheckableMenuItem.IsChecked = True
     ```

   - C#

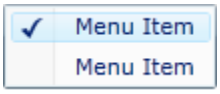
     ```
     CheckableMenuItem.IsChecked = true;
     ```

3. Run the program

This Topic Illustrates the Following:

Once the program is run, open the context menu or submenu. If you completed the optional step that created a checked item, a check mark will appear in the column to the left of the menu label; to clear the check mark, click the menu item. If you didn't complete the optional step, the check mark won't appear; to add a check mark, click the menu item.

The graphic below illustrates a checkable menu item.



Creating Mutually Exclusive Checkable Menu Items

In this topic, you learn how to create a list of checkable menu items that are grouped together so that only one item can be checked at a time.

In Blend

Complete the following steps:

1. Under the **Objects and Timeline** tab, select the first **[C1MenuItem]** you wish to make checkable and then set the following properties under the **Properties** panel:
   - Set the IsCheckable property to **True**.
   - Set the GroupName property to a string, such as "CheckableGroup".
2. Repeat step 1 for any other menu items you wish to add to the group of mutually exclusive checkable items.
3. Run the program and click the first item in the group. Observe that a check is added to the menu item. Now click the second item in the group and observe that the check is removed from the first item and then added to the second item.

In XAML

Complete the following steps:

1. Add `IsCheckable="True"` and `GroupName="CheckableGroup"` to the `<c1:C1MenuItem>` tag of each menu item you wish to add to the group of mutually exclusive checkable items.
2. Run the program and click the first item in the group. Observe that a check is added to the menu item. Now click the second item in the group and observe that the check is removed from the first item and then added to the second item.

In Code

Complete the following steps:

1. Set the **x:Name** property of each C1MenuItem you wish to add to the group of mutually exclusive checkable items.
2. Open the **MainPage.xaml.cs** page.
3. Set the IsCheckable and GroupName property of each menu item, replacing "ItemName" with the value of the menu item's **x:Name** property.
   - Visual Basic

     ```
     ItemName.IsCheckable = True
     ```

   - C#

     ```
     ItemName.IsCheckable = true;
     ```

4. Run the program and click the first item in the group. Observe that a check is added to the menu item. Now click the second item in the group and observe that the check is removed from the first item and then added to the second item.

Enabling Boundary Detection

Boundary detection ensures that the menus a user opens will always stay within page bounds. This is helpful if you have several nested submenus within your menu, as these menus can expand until they're beyond the scope of a project page. Boundary detection is disabled by default, but you can enable it by setting the DetectBoundaries property to **True**.

In Blend

Complete the following steps:

1. Select the C1ContextMenu control or the C1Menu control.

   The control's properties appear underneath the **Properties** panel.
2. Select the DetectBoundaries check box.
3. Run the program.

In XAML

Complete the following steps:

1. Add `DetectBoundaries="True"` to the `<c1:C1Menu>` or the `<c1:C1ContextMenu>` tag.
2. Run the program.

In Code

Complete the following steps:

1. In Source view (XAML view, if you're using Blend), locate the `<c1:C1MenuItem>` tag for the item you wish to make checkable and add `x:Name="C1Menu1"` to it. This will give the item a unique identifier that you can use in code.

2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     C1Menu1.DetectBoundaries = True
     ```

   - C#

     ```
     C1Menu1.DetectBoundaries = true;
     ```

     Run the program.

This Topic Illustrates the Following:

When you run the project, expand all of your submenus and observe that they will shift position rather than being cut off at the edge of the Web page. For more information on boundary detection, see Boundary Detection.

Enabling Automatic Menu Closing

Automatic menu closing allows users to close menus by clicking outside of the menu area. To enable automatic menu closing, set the AutoClose property to **True**.

In Blend

Complete the following steps:

1. Select the C1ContextMenu control or the C1Menu control.

   The control's properties appear underneath the **Properties** panel.
2. Select the AutoClose check box.
3. Run the program.

In XAML

Complete the following steps:

1. Add `AutoClose="True"` to the `<c1:C1Menu>` or the `<c1:C1ContextMenu>` tag.
2. Run the program.

In Code

Complete the following steps:

1. In Source view (XAML view, if you're using Blend), locate the `<c1:C1MenuItem>` tag for the item you wish to make checkable and add `x:Name="C1Menu1"` to it. This will give the item a unique identifier that you can use in code.
2. Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     C1Menu1.AutoClose = True
     ```

   - C#

     ```
     C1Menu1.AutoClose = true;
     ```

     Run the program.

This Topic Illustrates the Following:

After you've run the project, open the C1ContextMenu or one of the C1Menu control's submenus. With the submenu open, click outside of the submenu and observe that the submenu closes.

Adding a Separator Between Menu Items

In this topic, you will learn how to add separator bars between menu items.

In Blend

Complete the following steps:

1. Click the **Assets** tab and, in the search bar, enter "C1Separator".
2. Double-click the C1Separator icon to add the separator to your project.
3. Click the **Objects and Timeline** tab.
4. Select **[C1Separator]** and then use a drag-and-drop operation to place it in between two C1MenuItems.

In XAML

Complete one of the following:

To add a separator bar to a C1ContextMenu control, place `<c1:C1Separator />` between two `<c1:C1MenuItem>` tags. The result will resemble the following:

```
<c1:C1MenuItem Header="File" />
    <c1:C1Separator/>
<c1:C1MenuItem Header="Options" />
```

Adding an Icon to a Menu Item

In this step, you will learn how to add an icon to a C1MenuItem in XAML and in code. These steps will work in both Visual Studio and Blend.

In XAML

Complete the following steps:

1. Add an icon image to your Silverlight project. A 12x12 pixel image is best.
2. Add the following XAML markup between the `<c1:C1MenuItem>` and `</c1:C1MenuItem>` tags, replacing the value of the Source property with your image's name:

```
      <c1:C1MenuItem.Icon>
           <Image Source="YourImage.png" Height="12" Width="12"
Margin="5,0,0,0"/>
      </c1:C1MenuItem.Icon>
```

3. Run the project.

In Code

Complete the following steps:

1. Add an icon image to your Silverlight project. A 12x12 pixel image is best.
2. Add `x:Name="C1MenuItem1"` to the item you wish to add an icon to.
3. Import the following namespace:
   - Visual Basic

     `Imports System.Windows.Media.Imaging`
   - C#

     `using System.Windows.Media.Imaging;` Enter Code view and add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

```
'Create an image and assign it a source, margin, and width
Dim ItemIcon As New Image()
ItemIcon.Source = New BitmapImage(New Uri("02.png",
UriKind.RelativeOrAbsolute))
ItemIcon.Margin = New Thickness(5, 0, 0, 0)
ItemIcon.Height = 12
ItemIcon.Width = 12
'Set the C1MenuItem's icon property to the new image
C1MenuItem.Icon = ItemIcon
```

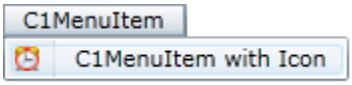
- C#

```
//Create an image and assign it a source, margin, and width
Image ItemIcon = new Image();
ItemIcon.Source = new BitmapImage(new Uri("02.png",
UriKind.RelativeOrAbsolute));
ItemIcon.Margin = new Thickness(5,0,0,0);
ItemIcon.Height = 12;
ItemIcon.Width = 12;
//Set the C1MenuItem's icon property to the new image
C1MenuItem1.Icon = ItemIcon;
```

5. Run the project.

This Topic Illustrates the Following:

The following image depicts a C1MenuItem with a 12x12 pixel icon.



NumericBox

Display and edit numeric values in your Silverlight applications! **ComponentOne NumericBox™ for Silverlight** provides a numeric text box control, C1NumericBox, which is similar to the standard Windows Forms **NumericUpDown** control and provides functionality for numeric input and editing right out of the box.

The C1NumericBox control contains a single numeric value that can be incremented or decremented by clicking the up or down buttons of the control. The user can also enter in a value, unless the IsReadOnly property is set to **True**.

Getting Started

Get started with the following topics:

- Key Features
- Quick Start
- Task-Based Help

NumericBox for Silverlight Features

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

- **Flexible Formatting**

  The Format property enables you to use the familiar .NET format strings to display data in any way you wish.

- **Numeric Range Support**

  Easily change the maximum and minimum values allowed for the editor.

- **Up/Down Buttons**

  The C1NumericBox control includes up/down buttons to increment or decrement the value.

NumericBox for Silverlight Quick Start

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

You will create an application that includes five C1NumericBox controls. The controls will function as a lock and when the correct code number has been entered in each, the controls will become locked and inactive and a button will appear directing users to a Web site.

Step 1 of 4: Adding NumericBox for Silverlight to your Project

In this step you'll begin in Visual Studio to create a Silverlight application using **NumericBox for Silverlight**. When you add a C1NumericBox control to your application, you'll have a complete, functional numeric editor. You can further customize the control to your application.

To set up your project and add a C1NumericBox control 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 close the **New Silverlight Application** dialog box and create your project.
4. In the XAML window of the project, place the cursor between the `<Grid>` and `</Grid>` tags and click once.
5. Navigate to the Toolbox and double-click the **StackPanel** icon to add the panel to **MainPage.xaml**. The XAML markup will now look similar to the following:

```
<UserControl xmlns:c1="clr-
namespace:C1.Silverlight;assembly=C1.Silverlight"
x:Class="C1NumericBox.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="White">
<StackPanel></StackPanel>
    </Grid>
```

6. Add `x:Name="sp1" Width="Auto" Height="Auto" Orientation="Vertical" HorizontalAlignment="Center" VerticalAlignment="Center"` to the `<StackPanel>` tag so that it appears similar to the following:

```
<StackPanel x:Name="sp1" Width="Auto" Height="Auto"
Orientation="Vertical" HorizontalAlignment="Center"
VerticalAlignment="Center"></StackPanel>
```

   Elements in the panel will now appear centered and vertically positioned.
7. In the XAML window of the project, place the cursor between the `<StackPanel>` and `</StackPanel>` tags.
8. Navigate to the Toolbox and double-click the **StackPanel** icon to add the panel to the existing **StackPanel**.
9. Add `x:Name="sp2" Width="Auto" Height="Auto" Orientation="Horizontal" HorizontalAlignment="Center" VerticalAlignment="Center"` to the `<StackPanel>` tag so that it appears similar to the following:

```
<StackPanel x:Name="sp2" Width="Auto" Height="Auto"
Orientation="Horizontal" HorizontalAlignment="Center"
VerticalAlignment="Center"></StackPanel>
```

   Elements in the panel will now appear centered and horizontally positioned.
10. In the XAML window of the project, place the cursor between the `<StackPanel x:Name="sp2">` and `</StackPanel>` tags.
11. Navigate to the Toolbox and double-click the **C1NumericBox** icon to add the control to the **StackPanel**. The XAML markup will now look similar to the following:

```
<UserControl xmlns:controls="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Tool
kit" xmlns:c1="clr-namespace:C1.Silverlight;assembly=C1.Silverlight"
x:Class="C1NumericBox.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="White">
<StackPanel x:Name="sp1" Width="Auto" Height="Auto"
Orientation="Vertical" HorizontalAlignment="Center"
VerticalAlignment="Center">
    <StackPanel x:Name="sp2" Width="Auto" Height="Auto"
Orientation="Horizontal" HorizontalAlignment="Center"
VerticalAlignment="Center">
            <c1:C1NumericBox></c1:C1NumericBox>
    </StackPanel>
```

```
</StackPanel>
</Grid>
</UserControl>
```

Note that the C1.Silverlight namespace and `<c1:C1NumericBox></c1:C1NumericBox>` tags have been added to the project.

12. Give your control a name by adding `x:Name="c1nb1"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:
```
<c1:C1NumericBox x:Name="c1nb1">
```
By giving it a unique identifier, you'll be able to access the control in code.

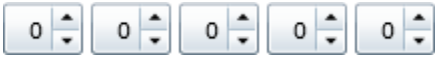
13. Resize your control and add a margin by adding `Width="40" Height="25" Margin="2"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:
```
<c1:C1NumericBox x:Name="c1nb1" Width="40" Height="25" Margin="2">
```
Your control should now appear smaller on the page.

14. Set your control's limits by adding `Minimum="0" Maximum="9"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:
```
<c1:C1NumericBox x:Name="c1nb1" Width="40" Height="25" Margin="2"
Minimum="0" Maximum="9">
```
The Minimum and Maximum properties will set the minimum and maximum values that are allowed in the control. Users will not be able to enter values outside of that range providing built-in data validation.

15. Add `ValueChanged="c1nb1_ValueChanged"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:
```
<c1:C1NumericBox x:Name="c1nb1" Width="40" Height="25" Margin="2"
Minimum="0" Maximum="9" ValueChanged="c1nb1_ValueChanged">
```
You will add code for the **c1nb1_ValueChanged** event handler in a later step.

16. Add the following XAML just below the existing `<c1:C1NumericBox x:Name="c1nb1"></c1:C1NumericBox>` tags so that it appears similar to the following:
```
<c1:C1NumericBox x:Name="c1nb2" Width="40" Height="25" Margin="2"
Minimum="0" Maximum="9"
ValueChanged="c1nb2_ValueChanged"></c1:C1NumericBox>
<c1:C1NumericBox x:Name="c1nb3" Width="40" Height="25" Margin="2"
Minimum="0" Maximum="9"
ValueChanged="c1nb3_ValueChanged"></c1:C1NumericBox>
<c1:C1NumericBox x:Name="c1nb4" Width="40" Height="25" Margin="2"
Minimum="0" Maximum="9"
ValueChanged="c1nb4_ValueChanged"></c1:C1NumericBox>
<c1:C1NumericBox x:Name="c1nb5" Width="40" Height="25" Margin="2"
Minimum="0" Maximum="9"
ValueChanged="c1nb5_ValueChanged"></c1:C1NumericBox>
```
This will add four additional C1NumericBox controls, so that you have a total of five controls on the page.

Your application should now look similar to the following:



You've successfully created a Silverlight application, added **C1NumericBox** controls to the application, and customized those controls. In the next step you'll complete setting up the application.

Step 2 of 4: Customizing the Application

In the previous step you created a new Silverlight project and added five **C1NumericBox** controls to the application. In this step you'll continue by adding additional controls to customize the application.

Complete the following steps:

1. In the XAML window of the project, place the cursor just after the `<StackPanel x:Name="sp1">` tag.
2. Navigate to the Visual Studio Toolbox and double-click the standard **Label** control to it to your project.
3. Name the Label and add content to it by adding `x:Name="lbl1" Content="Enter Combination"` to the `<controls:Label>` tag so that it appears similar to the following:

```
<controls:Label x:Name="lbl1" Content="Enter Combination"
Margin="2"></controls:Label>
```

4. Place the cursor between the first `</StackPanel>` tag and the second `</StackPanel>` tag and add the following markup to create a second label:

```
<controls:Label x:Name="lbl2" Content="Invalid Combination"
Foreground="Red" Margin="2"></controls:Label>
```

5. Place the cursor between the `</controls:Label>` tag and the second `</StackPanel>` tag and add the following markup to create a hidden button:

```
<Button x:Name="btn1" Content="Enter" Height="25"
Visibility="Collapsed" Click="btn1_Click" Margin="2"></Button>
```

You will add the **btn1_Click** event handler later in code. Your application will now look similar to the following:

Your markup will now appear similar to the following:

```
<UserControl xmlns:controls="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Tool
kit" xmlns:c1="clr-namespace:C1.Silverlight;assembly=C1.Silverlight"
x:Class="C1NumericBox.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <Grid x:Name="LayoutRoot" Background="White">
<StackPanel x:Name="sp1" Width="Auto" Height="Auto"
Orientation="Vertical" HorizontalAlignment="Center"
VerticalAlignment="Center">
<controls:Label x:Name="lbl1" Content="Enter Combination"
Margin="2"></controls:Label>
    <StackPanel x:Name="sp2" Width="Auto" Height="Auto"
Orientation="Horizontal" HorizontalAlignment="Center"
VerticalAlignment="Center">
            <c1:C1NumericBox x:Name="c1nb1" Width="40" Height="25"
Minimum="0" Maximum="9" Margin="2"
ValueChanged="c1nb1_ValueChanged"></c1:C1NumericBox>
            <c1:C1NumericBox x:Name="c1nb2" Width="40" Height="25"
Minimum="0" Maximum="9" Margin="2"
ValueChanged="c1nb2_ValueChanged"></c1:C1NumericBox>
```

```
            <c1:C1NumericBox x:Name="c1nb3" Width="40" Height="25"
Minimum="0" Maximum="9" Margin="2"
ValueChanged="c1nb3_ValueChanged"></c1:C1NumericBox>
            <c1:C1NumericBox x:Name="c1nb4" Width="40" Height="25"
Minimum="0" Maximum="9" Margin="2"
ValueChanged="c1nb4_ValueChanged"></c1:C1NumericBox>
            <c1:C1NumericBox x:Name="c1nb5" Width="40" Height="25"
Minimum="0" Maximum="9" Margin="2"
ValueChanged="c1nb5_ValueChanged"></c1:C1NumericBox>
        </StackPanel>
<controls:Label x:Name="lbl2" Content="Invalid Combination"
Foreground="Red" Margin="2"></controls:Label>
    <Button x:Name="btn1" Content="Enter" Visibility="Collapsed"
Height="25" Margin="2" Click="Btn1_Click"></Button>
</StackPanel>
    </Grid>
</UserControl>
```

You've successfully set up your application's user interface. In the next step you'll add code to your application.

Step 3 of 4: Adding Code to the Application

In the previous steps you set up the application's user interface and added **C1NumberBox**, **Label**, and **Button** controls to your application. In this step you'll add code to your application to finalize it.

Complete the following steps:

1. Select **View | Code** to switch to Code view.
2. Add the following imports statements to the top of the page:
   - Visual Basic
     ```
     Imports C1.Silverlight
     Imports System.Windows.Media
     Imports System.Windows.Browser
     ```
   - C#
     ```
     using C1.Silverlight;
     using System.Windows.Media;
     using System.Windows.Browser;
     ```
3. Initialize the following global variables just inside the **Page** class:
   - Visual Basic
     ```
     Dim nb1 As Integer = 5
     Dim nb2 As Integer = 2
     Dim nb3 As Integer = 3
     Dim nb4 As Integer = 7
     Dim nb5 As Integer = 9
     ```
   - C#
     ```
     int nb1 = 5;
     int nb2 = 2;
     int nb3 = 3;
     int nb4 = 7;
     int nb5 = 9;
     ```

   These numbers will be used as the correct 'code' in the application. When the user enters the correct combination of numbers at run time the button will appear.
4. Add code to the **Button1_Click** event handler so that it appears like the following:

- Visual Basic

```
Private Sub btn1_Click(ByVal sender As System.Object, ByVal e As
System.Windows.RoutedEventArgs) Handles btn1.Click
    HtmlPage.PopupWindow(New Uri("http://www.componentone.com"), "new",
Nothing)
End Sub
```

- C#

```
private void btn1_Click(object sender, RoutedEventArgs e)
{
    HtmlPage.PopupWindow(New Uri("http://www.componentone.com"), "new",
null);
}
```

When the button is pressed at run time it will open the ComponentOne Web site.

5. Next add the following custom **NBValidation** event to your code:

- Visual Basic

```
Private Sub NBValidation()
    If Me.c1nb1.Value = nb1 And Me.c1nb2.Value = nb2 And Me.c1nb3.Value
= nb3 And Me.c1nb4.Value = nb4 And Me.c1nb5.Value = nb5 Then
        Me.lbl2.Foreground = New SolidColorBrush(Colors.Green)
        Me.lbl2.Content = "Combination Valid"
        Me.c1nb1.IsReadOnly = True
        Me.c1nb2.IsReadOnly = True
        Me.c1nb3.IsReadOnly = True
        Me.c1nb4.IsReadOnly = True
        Me.c1nb5.IsReadOnly = True
        Me.btn1.Visibility = Windows.Visibility.Visible
    End If
End Sub
```

- C#

```
private void NBValidation()
{
    if (this.c1nb1.Value == nb1 & this.c1nb2.Value == nb2 &
this.c1nb3.Value == nb3 & this.c1nb4.Value == nb4 & this.c1nb5.Value ==
nb5)
    {
        this.lbl2.Foreground = new SolidColorBrush(Colors.Green);
        this.lbl2.Content = "Combination Valid";
        this.c1nb1.IsReadOnly = true;
        this.c1nb2.IsReadOnly = true;
        this.c1nb3.IsReadOnly = true;
        this.c1nb4.IsReadOnly = true;
        this.c1nb5.IsReadOnly = true;
        this.btn1.Visibility = Windows.Visibility.Visible;
    }
}
```

When the user enters the correct numbers (as indicated in step 3 above) the C1NumericBox controls will be set to read only and will no longer be editable, the text of the label below the controls will change to indicate the correct code has been entered, and a button will appear allowing users to enter the ComponentOne Web site.

6. Add **C1NumericBox_ValueChanged** event handlers to initialize **NBValidation**. The code will look like the following:

- Visual Basic

```
Private Sub c1nb1_ValueChanged(ByVal sender As System.Object, ByVal e
As C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
c1nb1.ValueChanged
    NBValidation()
End Sub
Private Sub c1nb2_ValueChanged(ByVal sender As System.Object, ByVal e
As C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
c1nb2.ValueChanged
    NBValidation()
End Sub
Private Sub c1nb3_ValueChanged(ByVal sender As System.Object, ByVal e
As C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
c1nb3.ValueChanged
    NBValidation()
End Sub
Private Sub c1nb4_ValueChanged(ByVal sender As System.Object, ByVal e
As C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
c1nb4.ValueChanged
    NBValidation()
End Sub
Private Sub c1nb5_ValueChanged(ByVal sender As System.Object, ByVal e
As C1.Silverlight.PropertyChangedEventArgs(Of System.Double)) Handles
c1nb5.ValueChanged
    NBValidation()
End Sub
```

- C#

```
private void c1nb1_ValueChanged(object sender,
PropertyChangedEventArgs<double> e)
{
    NBValidation();
}
private void c1nb2_ValueChanged(object sender,
PropertyChangedEventArgs<double> e)
{
    NBValidation();
}
private void c1nb3_ValueChanged(object sender,
PropertyChangedEventArgs<double> e)
{
    NBValidation();
}
private void c1nb4_ValueChanged(object sender,
PropertyChangedEventArgs<double> e)
{
    NBValidation();
}
private void c1nb5_ValueChanged(object sender,
PropertyChangedEventArgs<double> e)
{
    NBValidation();
}
```

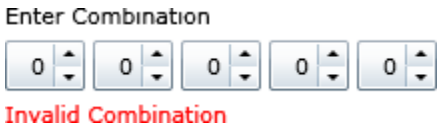
In this step you completed adding code to your application. In the next step you'll run the application and observe run-time interactions.

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 **NumericBox for Silverlight**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time.

   The application will appear similar to the following:



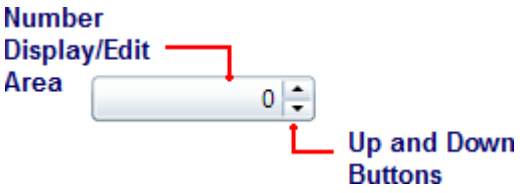
2. Click the **Up** arrow in the first (left-most) **C1NumericBox** control until **5** is displayed. Note that the number increased by 1 each time you click – this is because the Increment property is set to **1** by default.
3. Click inside the second **C1NumericBox**, highlight the "0" value, and type "2" to replace it.
4. Try clicking the **Down** button in the third **C1NumericBox** control and notice that the number does not change. This is because the Minimum property was set to **0** and so the control will not accept values less than zero. Click the **Up** button until **3** is displayed.
5. In the fourth **C1NumericBox** control, place the cursor in front of the **0** and click. Enter "5" so that "50" is displayed.
6. Click inside the last **C1NumericBox** control. Notice that the **50** inside the fourth **C1NumericBox** was reset to **9**. That's because the Maximum property was set to **9** so the control will not accept values greater than nine.
7. Enter **9** in the last **C1NumericBox** control.
8. Click the **Down** button of the fourth **C1NumericBox** control twice so **7** is displayed. Note that the text of the second Label changed and the button is now visible:

9. Try typing inside a **C1NumericBox** control or clicking its **Up** or **Down** buttons, notice that you cannot. That is because the IsReadOnly property was set to **True** when the correct number sequence was entered and the controls are now locked from editing.
10. Click the now-visible **Enter** button to navigate to the ComponentOne Web site.

Congratulations! You've completed the **NumericBox for Silverlight** quick start and created a **NumericBox for Silverlight** application, customized the appearance and behavior of the controls, and viewed some of the run-time capabilities of your application.

About C1NumericBox

ComponentOne NumericBox for Silverlight includes the C1NumericBox control, a simple control which provides numeric input and editing. When you add the C1NumericBox control to a XAML window, it exists as a completely functional numeric editor. By default, the control's interface looks similar to the following image:



It consists of the following elements:

- **Up and Down Buttons**

  The **Up** and **Down** buttons allow users to change the value displayed in the control. Each time a button is clicked the Value changes by the amount indicated by the Increment property (by default 1). By default the **Up** and **Down** buttons are visible; to hide the buttons set the ShowButtons property to **False**.

- **Number Display/Edit Area**

  The current Value is displayed in the number display/editing area. Users can type in the box to change the Value property. By default users can edit this number; to lock the control from editing set IsReadOnly to **True**.

Basic Properties

ComponentOne NumericBox 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 NumericBox for Silverlight Appearance Properties for more information about properties that control appearance.

The following properties let you customize the C1NumericBox control:

Property	Description
Value	Gets or sets the numeric value in the C1NumericBox.
Minimum	Gets or sets the minimum value allowed for the C1NumericBox control.
Maximum	Gets or sets the maximum value allowed for the C1NumericBox.
Increment	Gets or sets the increment applied when the user presses the up/down arrow keys or the **Up** or **Down** buttons.
Format	Gets or sets the format of the C1NumericBox control.

Number Formatting

You can change how the number displayed in the C1NumericBox control will appear by setting the Format property. **ComponentOne NumericBox for Silverlight** supports the standard number formatting strings defined by Microsoft. For more information, see MSDN.

The Format string consists of a letter or a letter and number combination defining the format. By default, the Format property is set to "F0". The letter indicates the format type, here "F" for fixed-point, and the number indicates the number of decimal places, here none.

The following formats are available:

Format Specifier	Name	Description
C or c	Currency	The number is converted to a string that represents a currency amount. The conversion is controlled by the currency format information of the current NumberFormatInfo object. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default currency precision given by the current NumberFormatInfo object is used.
D or d	Decimal	This format is supported only for integral types. The number is converted to a string of decimal digits (0-9), prefixed by a minus sign if the number is negative. The precision specifier indicates the minimum number of digits desired in the resulting string. If required, the number is padded with zeros to its left to produce the number of digits given by the precision specifier. The following example formats an Int32 value with the Decimal format specifier.
E or e	Scientific (exponential)	The number is converted to a string of the form "-d.ddd…E+ddd" or "-d.ddd…e+ddd", where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative. One digit always precedes the decimal point. The precision specifier indicates the desired number of digits after the decimal point. If the precision specifier is omitted, a default of six digits after the decimal point is used. The case of the format specifier indicates whether to prefix the exponent with an 'E' or an 'e'. The exponent always consists of a plus or minus sign and a minimum of three digits. The exponent is padded with zeros to meet this minimum, if required.
F or f	Fixed-point	The number is converted to a string of the form "-ddd.ddd…" where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision is given by the NumberDecimalDigits property of the current NumberFormatInfo object.
G or g	General	The number is converted to the most compact of either fixed-point or scientific notation, depending on the type of the number and whether a precision specifier is present. If the precision specifier is omitted or zero, the type of the number determines the default precision, as indicated by the following list. • Byte or SByte: 3 • Int16 or UInt16: 5

		• Int32 or UInt32: 10 • Int64: 19 • UInt64: 20 • Single: 7 • Double: 15 • Decimal: 29 Fixed-point notation is used if the exponent that would result from expressing the number in scientific notation is greater than -5 and less than the precision specifier; otherwise, scientific notation is used. The result contains a decimal point if required and trailing zeroes are omitted. If the precision specifier is present and the number of significant digits in the result exceeds the specified precision, then the excess trailing digits are removed by rounding. The exception to the preceding rule is if the number is a Decimal and the precision specifier is omitted. In that case, fixed-point notation is always used and trailing zeroes are preserved. If scientific notation is used, the exponent in the result is prefixed with 'E' if the format specifier is 'G', or 'e' if the format specifier is 'g'. The exponent contains a minimum of two digits. This differs from the format for scientific notation produced by the 'E' or 'e' format specifier, which includes a minimum of three digits in the exponent.
N or n	Number	The number is converted to a string of the form "-d,ddd,ddd.ddd…", where '-' indicates a negative number symbol if required, 'd' indicates a digit (0-9), ',' indicates a thousand separator between number groups, and '.' indicates a decimal point symbol. The actual negative number pattern, number group size, thousand separator, and decimal separator are specified by the NumberNegativePattern, NumberGroupSizes, NumberGroupSeparator, and NumberDecimalSeparator properties, respectively, of the current NumberFormatInfo object. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision is given by the NumberDecimalDigits property of the current NumberFormatInfo object.
P or p	Percent	The number is converted to a string that represents a percent as defined by the NumberFormatInfo.PercentNegativePattern property if the number is negative, or the NumberFormatInfo.PercentPositivePattern property if the number is positive. The converted number is multiplied by 100 in order to be presented as a percentage. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision given by the current NumberFormatInfo object is used.
R or r	Round-trip	This format is supported only for the Single and Double types. The round-trip specifier guarantees that a numeric value converted to a string will be parsed back into the same

		numeric value. When a numeric value is formatted using this specifier, it is first tested using the general format, with 15 spaces of precision for a Double and 7 spaces of precision for a Single. If the value is successfully parsed back to the same numeric value, it is formatted using the general format specifier. However, if the value is not successfully parsed back to the same numeric value, then the value is formatted using 17 digits of precision for a Double and 9 digits of precision for a Single. Although a precision specifier can be present, it is ignored. Round trips are given precedence over precision when using this specifier.
X or x	Hexadecimal	This format is supported only for integral types. The number is converted to a string of hexadecimal digits. The case of the format specifier indicates whether to use uppercase or lowercase characters for the hexadecimal digits greater than 9. For example, use 'X' to produce "ABCDEF", and 'x' to produce "abcdef". The precision specifier indicates the minimum number of digits desired in the resulting string. If required, the number is padded with zeros to its left to produce the number of digits given by the precision specifier.
Any other single character	(Unknown specifier)	(An unknown specifier throws a FormatException at runtime.)

Input Validation

You can use the Minimum and Maximum properties to set a numeric range that users are limited to at run time. If the Minimum and Maximum properties are set, users will not be able to pick a number larger than the Minimum or smaller than the Maximum.

When setting the Minimum and Maximum properties, the Minimum should be smaller than the Maximum. Also be sure to set the Value property to a number within the Minimum and Maximum range.

You can also choose a mode for range validation using the RangeValidationMode property. This property controls when the entered number is validated. You can set RangeValidationMode to one of the following options:

Option	Description
Always	This mode does not allow users to enter out of range values.
AlwaysTruncate	This mode does not allow users to enter out of range values. The value will be truncated if the limits are exceeded.
OnLostFocus	This mode truncates the value when the control loses focus.

NumericBox for Silverlight Layout and Appearance

The following topics detail how to customize the C1NumericBox 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 grid and to customize grid actions.

NumericBox for Silverlight Appearance Properties

ComponentOne NumericBox 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.

Content Properties

The following properties let you customize the appearance of content in the **C1NumericBox** control:

Property	Description
Format	Gets or sets the value for the Format of the C1NumericBox.
Watermark	Gets or sets the watermark content displayed when the control is empty.

Text Properties

The following properties let you customize the appearance of text in the **C1NumericBox** 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.
TextAlignment	Gets or sets how the text should be aligned in the **C1NumericBox**.

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.
SelectionBackground	Gets or sets the brush that fills the background of the selected text.
SelectionForeground	Gets or sets the brush used for the selected text in the C1NumericBox.

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.

Style Properties

The following properties let you set styles:

Property	Description
FocusVisualStyle	Gets or sets a property that enables customization of appearance, effects, or other style characteristics that will apply to this element when it captures keyboard focus. This is a dependency property.
Style	Gets or sets the style used by this element when it is rendered. This is a dependency property.

Size Properties

The following properties let you customize the size of the **C1NumericBox** control:

Property	Description
ActualHeight	Gets the rendered height of this element. This is a dependency property.
ActualWidth	Gets the rendered width of this element. This is a dependency property.
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.

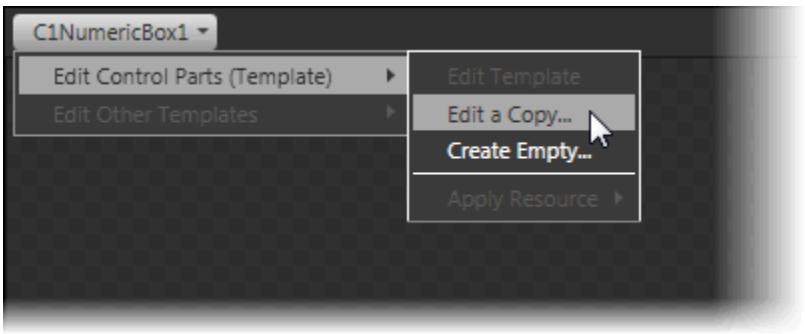
Templates

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 NumericBox 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 C1NumericBox 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.

NumericBox 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 C1NumericBox control in general. If you are unfamiliar with the **ComponentOne NumericBox for Silverlight** product, please see the NumericBox for Silverlight Quick Start first.

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

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

Setting the Start Value

The Value property determines the currently selected number. By default the C1NumericBox control starts with its Value set to **0** but you can customize this number at design time in Microsoft Expression Blend, in XAML, and in code.

At Design Time in Blend

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

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties tab, and enter a number, for example "123", in the text box next to the Value property.

   This will set the Value property to the number you chose.

In XAML

For example, to set the Value property add `Value="123"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Width="40" Height="25"
Value="123"></c1:C1NumericBox>
```

In Code

For example, to set the Value property add the following code to your project:

- Visual Basic
```
C1NumericBox1.Value = 123
```
- C#
```
c1NumericBox1.Value = 123;
```

Run your project and observe:

Initially **123** (or the number you chose) will appear in the control:

Setting the Increment Value

The Increment property determines by how much the Value property changes when the **Up** or **Down** button is clicked at run time. By default the C1NumericBox control starts with its Increment set to **1** but you can customize this number at design time in Microsoft Expression Blend, in XAML, and in code.

At Design Time in Blend

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

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties tab, and enter a number, for example "20", in the text box next to the Increment property.

   This will set the Increment property to the number you chose.

In XAML

For example, to set the Increment property to **20** add `Increment="20"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Width="40" Height="25"
Increment="20"></c1:C1NumericBox>
```

In Code

For example, to set the Increment property to **20** add the following code to your project:

- Visual Basic
```
C1NumericBox1.Increment = 20
```
- C#
```
c1NumericBox1.Increment = 20;
```

Run your project and observe:

Click the **Up** and then the **Down** button a few times or press the **Up** and **Down** arrow keys on the keyboard. Notice that the Value changes in steps of 20. You can still edit the value directly by clicking in the text box and entering a number that falls between that step.

Setting the Minimum and Maximum Values

You can use the Minimum and Maximum properties to set a numeric range that users are limited to at run time. If the Minimum and Maximum properties are set, users will not be able to pick a number larger than the Minimum or smaller than the Maximum.

Note: When setting the Minimum and Maximum properties, the Minimum should be smaller than the Maximum. Also be sure to set the Value property to a number within the Minimum and Maximum range. In the following example, the default value **0** falls within the range chosen.

At Design Time in Blend

To set the Minimum and Maximum in Blend, complete the following steps:

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties tab, and enter a number, for example **500**, next to the Maximum property.
3. In the Properties tab, enter a number, for example **-500**, next to the Minimum property.

   This will set Minimum and Maximum values.

In XAML

To set the Minimum and Maximum in XAML add `Maximum="500" Minimum="-500"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Width="40" Height="25"
Maximum="500" Minimum="-500"></c1:C1NumericBox>
```

In Code

To set the Minimum and Maximum add the following code to your project:

- Visual Basic

```
C1NumericBox1.Minimum = -500
C1NumericBox1.Maximum = 500
```

- C#

```
c1NumericBox1.Minimum = -500;
c1NumericBox1.Maximum = 500;
```

Run your project and observe:

Users will be limited to the selected range at run time.

Changing Font Type and Size

You can change the appearance of the text in the grid by using the text properties in the C1NumericBox Properties tab in Blend, through XAML, or through code.

At Design Time in Blend

To change the font of the grid to Arial 10pt in Blend, complete the following:

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties tab, click the **FontFamily** drop-down arrow, and set the property to "Arial" (or a font of your choice).
3. In the Properties tab, click the **FontSize** drop-down arrow and set the property to **10**.

   This will set the control's font size and style.

In XAML

For example, to change the font of the control to Arial 10pt in XAML add `FontFamily="Arial" FontSize="10"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Width="40" Height="25"
FontFamily="Arial" FontSize="10"></c1:C1NumericBox>
```

In Code

For example, to change the font of the grid to Arial 10pt add the following code to your project:

- Visual Basic
```
C1NumericBox1.FontSize = 10
C1NumericBox1.FontFamily = New System.Windows.Media.FontFamily("Arial")
```
- C#
```
c1NumericBox1.FontSize = 10;
c1NumericBox1.FontFamily = new System.Windows.Media.FontFamily("Arial");
```

Run your project and observe:

The control's content will appear in Arial 10pt font:

Hiding the Up and Down Buttons

By default buttons are visible in the C1NumericBox control to allow users to increment and decrement the value in the box by one step. You can choose to hide the **Up** and **Down** buttons in the C1NumericBox control at run time. To hide the **Up** and **Down** buttons you can set the ShowButtons property to **False**.

At Design Time in Blend

To hide the **Up** and **Down** buttons in Blend, complete the following steps:

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties tab, and uncheck the ShowButtons check box.

   This will set the ShowButtons property to **False**.

In XAML

For example, to hide the **Up** and **Down** buttons in XAML add `ShowButtons="False"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Width="40" Height="25"
ShowButtons="False"></c1:C1NumericBox>
```

In Code

For example, to hide the **Up** and **Down** buttons add the following code to your project:

- Visual Basic
```
C1NumericBox1.ShowButtons = False
```
- C#
```
c1NumericBox1.ShowButtons = false;
```

Run your project and observe:

The **Up** and **Down** buttons will not be visible:

Locking the Control from Editing

By default the C1NumericBox control's Value property is editable by users at run time. If you want to lock the control from being edited, you can set the IsReadOnly property to **True**.

At Design Time in Blend

To lock the C1NumericBox control from run-time editing in Blend, complete the following steps:

1. Click the C1NumericBox control once to select it.
2. Navigate to the Properties tab, and check the IsReadOnly check box.

   This will set the IsReadOnly property to **False**.

In XAML

For example, to hide the **Up** and **Down** buttons in XAML add `IsReadOnly="True"` to the `<c1:C1NumericBox>` tag so that it appears similar to the following:

```
<c1:C1NumericBox x:Name="C1NumericBox1" Width="40" Height="25"
IsReadOnly="True"></c1:C1NumericBox>
```

In Code

For example, to hide the **Up** and **Down** buttons add the following code to your project:

- Visual Basic

```
C1NumericBox1.IsReadOnly = True
```

- C#

```
c1NumericBox1.IsReadOnly = true;
```

Run your project and observe:

The control is locked from editing; notice that the **Up** and **Down** buttons are grayed out and inactive:

RangeSlider

Add smooth numeric data selection to your Silverlight applications. **ComponentOne RangeSlider™ for Silverlight** extends the basic slider control and provides two thumb elements instead of one, allowing users to select ranges instead of single values.

Getting Started

Get started with the following topics:

- Key Features
- Quick Start
- Task-Based Help

RangeSlider for Silverlight Key Features

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

- **Allow Smooth Range Selection and Enhance Your UI**

  ComponentOne RangeSlider for Silverlight extends the basic slider control and provides two thumb elements instead of one, allowing users to select ranges instead of single values.

- **Switch Orientation**

  Set the **C1RangeSlider** control to use a horizontal or vertical orientation to customize it to your application.

RangeSlider for Silverlight Quick Start

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

You will create a simple project using a C1RangeSlider and a standard **Rectangle** control. The C1RangeSlider control will control a gradient that is applied to the **Rectangle**, so that at run time moving the slider thumbs will change the gradient – letting you explore the possibilities of using **RangeSlider for Silverlight**.

Step 1 of 4: Setting up the Application

In this step you'll begin in Visual Studio to create a Silverlight application using **RangeSlider for Silverlight**. When you add a C1RangeSlider control to your application, you'll have a complete, functional input editor in the form of a slider. You can then further customize the control to your application.

To set up your project and add a C1RangeSlider control to your application, 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 `Width="400" Height="300"` to `Width="Auto" Height="Auto"` in the `<UserControl>` tag so that it appears similar to the following:

```
<UserControl x:Class="C1RangeSlider.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="Auto"
Height="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 **Rectangle** icon to add the standard control to the **Grid**.
7. 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>
```

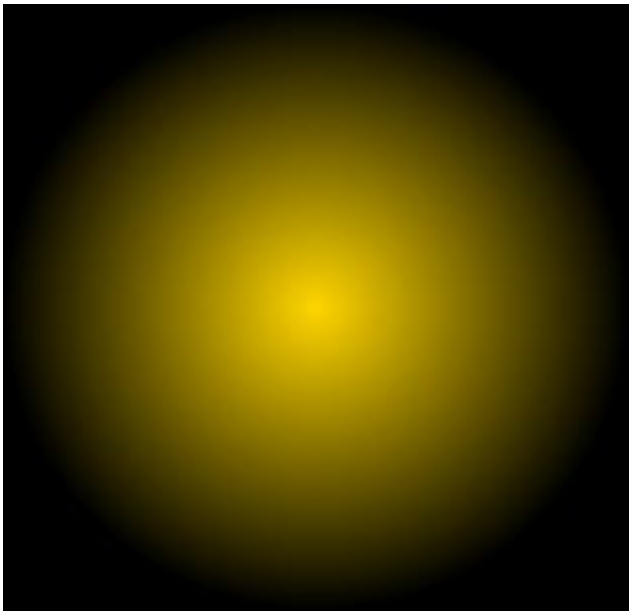
The rectangle will now fill the **Grid**.

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

```
<Rectangle x:Name="rc1" Width="Auto" Height="Auto">
    <Rectangle.Fill>
        <RadialGradientBrush x:Name="colors">
            <GradientStop x:Name="goldcol" Color="Gold" Offset="0" />
            <GradientStop x:Name="blackcol" Color="Black" Offset="1" />
        </RadialGradientBrush>
    </Rectangle.Fill>
</Rectangle>
```

This will add a black and gold radial gradient fill to the rectangle.

9. Run your application now and observe that it looks similar to the following:



You've successfully created a Silverlight application and customized the **Rectangle** control. In the next step you'll add and customize the C1RangeSlider control.

Step 2 of 4: Adding a C1RangeSlider Control

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 a C1RangeSlider control that will control the gradient fill in the **Rectangle**.

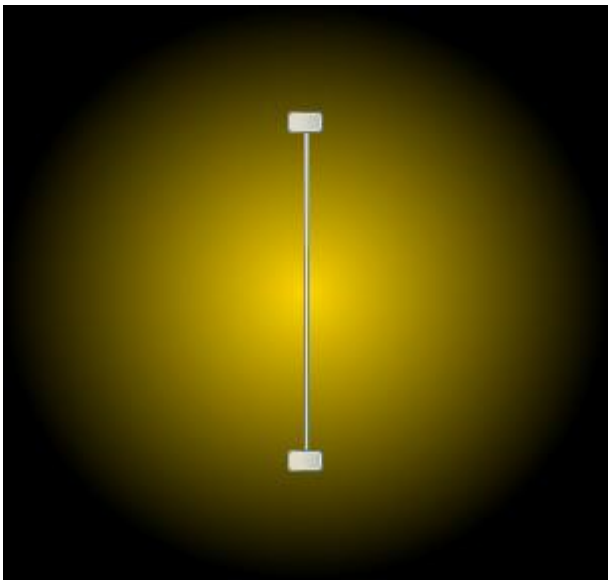
Complete the following steps:

1. In the XAML window of the project, place the cursor between the `</Rectangle>` and `</Grid>` tags and click once.
2. Navigate to the Toolbox and double-click the C1RangeSlider icon to add the control to the application on top of the **Rectangle**.
3. Give your control a name by adding `x:Name="c1rs1"` to the `<c1:C1RangeSlider>` tag so that it appears similar to the following:
   ```
   <c1:C1RangeSlider x:Name="c1rs1">
   ```
   By giving it a unique identifier, you'll be able to access the control in code.
4. Add a margin by adding `Margin="50"` to the `<c1:C1RangeSlider>` tag so that it appears similar to the following:
   ```
   <c1:C1RangeSlider x:Name="c1rs1" Margin="50">
   ```
   This will set each edge the same distance away from the grid's border.
5. Set the Orientation property to **Vertical** by adding `Orientation="Vertical"` to the `<c1:C1RangeSlider>` tag so that it appears similar to the following:
   ```
   <c1:C1RangeSlider x:Name="c1rs1" Margin="50" Orientation="Vertical">
   ```
   By default Orientation is **Horizontal** and the control appears across the page.
6. Set the UpperValue property to 1 by adding `UpperValue="1"` to the `<c1:C1RangeSlider>` tag so that it appears similar to the following:
   ```
   <c1:C1RangeSlider x:Name="c1rs1" Margin="50" Orientation="Vertical"
   UpperValue="1">
   ```
   The upper thumb will now begin at **1**.
7. Set the Maximum property to 1 by adding `Maximum="1"` to the `<c1:C1RangeSlider>` tag so that it appears similar to the following:
   ```
   <c1:C1RangeSlider x:Name="c1rs1" Margin="50" Orientation="Vertical"
   UpperValue="1" Maximum="1">
   ```
   Users will now not be able to select a value greater than 1.
8. Set the ValueChange property to 0.1 by adding `ValueChange="0.1"` to the `<c1:C1RangeSlider>` tag so that it appears similar to the following:
   ```
   <c1:C1RangeSlider x:Name="c1rs1" Margin="50" Orientation="Vertical"
   UpperValue="1" Maximum="1" ValueChange="0.1">
   ```
   When you click on the slider track at run time, the slider thumb will now move by 0.1 units.
9. Set the **Opacity** property to "0.8" by adding `Opacity="0.8"` to the `<c1:C1RangeSlider>` tag so that it appears similar to the following:
   ```
   <c1:C1RangeSlider x:Name="c1rs1" Margin="50" Orientation="Vertical"
   UpperValue="1" Maximum="1" ValueChange="0.1" Opacity="0.8">
   ```
   By default this property is set to **1** and the control appears completely opaque. Changing this to a lower number will make the control appear slightly transparent.
10. Indicate event handlers by adding `LowerValueChanged="c1rs1_LowerValueChanged"` `UpperValueChanged="c1rs1_UpperValueChanged"` to the `<c1:C1RangeSlider>` tag so that it appears similar to the following:

```
<c1:C1RangeSlider x:Name="c1rs1" Margin="50" Orientation="Vertical"
UpperValue="1" Maximum="1" ValueChange="0.1" Opacity="0.8"
LowerValueChanged="c1rs1_LowerValueChanged"
UpperValueChanged="c1rs1_UpperValueChanged">
```

You'll add code for these event handlers in a later step.

11. Run your application now and observe that it looks similar to the following:



You've successfully set up your application's user interface, but right now the slider will do nothing if you move it. In the next step you'll add code to your application to add functionality.

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. Select **View | Code** to switch to Code view.
2. In Code view, add the following import statement to the top of the page:
   - Visual Basic
     ```
     Imports C1.Silverlight
     ```
   - C#
     ```
     using C1.Silverlight;
     ```
3. Add the following code just after the page's constructor to update the gradient values:
   - Visual Basic
     ```
     Private Sub UpdateGradient()
         If c1rs1 IsNot Nothing Then
             Me.goldcol.Offset = Me.c1rs1.LowerValue
             Me.blackcol.Offset = Me.c1rs1.UpperValue
         End If
     End Sub
     ```

- C#

```
UpdateGradient()
{
    if (c1rs1 != null)
    {
        this.goldcol.Offset = this.c1rs1.LowerValue;
        this.blackcol.Offset = this.c1rs1.UpperValue;
    }
}
```

4. Return to Design view.
5. Click once on the C1RangeSlider control to select it and then navigate to the Properties window.
6. Click the lightning bolt **Events** icon at the top of the Properties window to view events.
7. Double-click the **LowerValueChanged** event to switch to Code view and create the **C1RangeSlider1_LowerValueChanged** event handler. Return to Design view and repeat this step with the **UpperValueChanged** event to create the **C1RangeSlider1_UpperValueChanged** event handler.
8. Add the **c1rs1_LowerValueChanged** event handler so that it appears like the following:

   - Visual Basic

```
Private Sub c1rs1_LowerValueChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles c1rs1.LowerValueChanged
    UpdateGradient()
End Sub
```

   - C#

```
private void c1rs1_LowerValueChanged(object sender, EventArgs e)
{
    UpdateGradient();
}
```

9. Add the **c1rs1_UpperValueChanged** event handler so that it appears like the following:

   - Visual Basic

```
Private Sub c1rs1_UpperValueChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles c1rs1.UpperValueChanged
    UpdateGradient()
End Sub
```

   - C#

```
c1rs1_UpperValueChanged(object sender, EventArgs e)
{
    UpdateGradient();
}
```

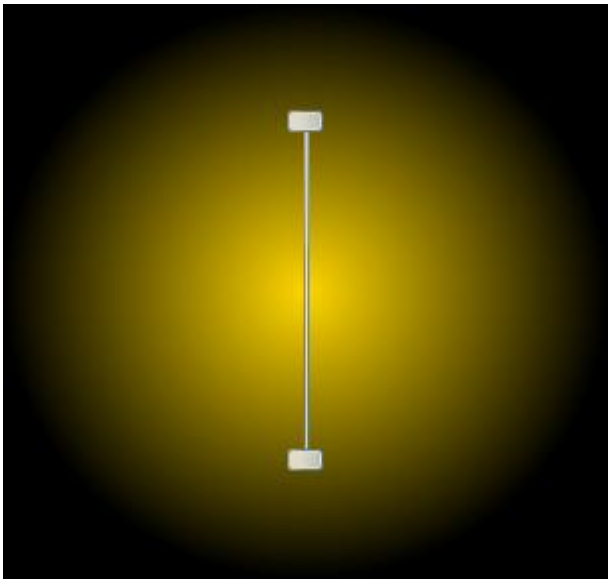
In this step you completed adding code to your application. In the next step you'll run the application and observe run-time interactions.

Step 4 of 4: Running the Application

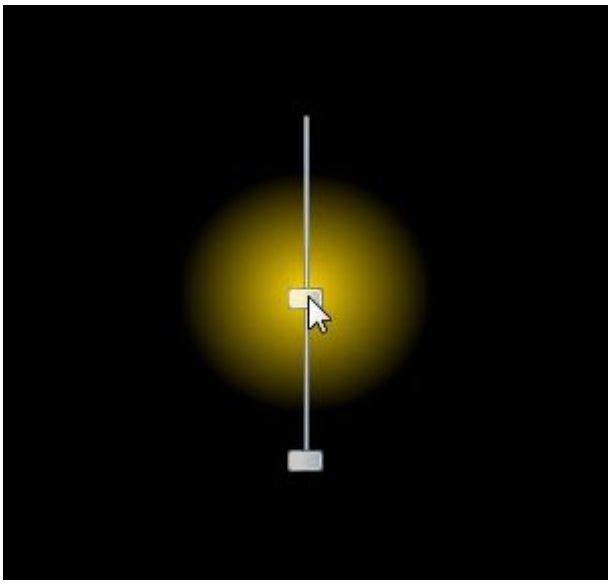
Now that you've created a Silverlight application and customized the application's appearance and behavior, the only thing left to do is run your application. To run your application and observe **RangeSlider for Silverlight**'s run-time behavior, complete the following steps:

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

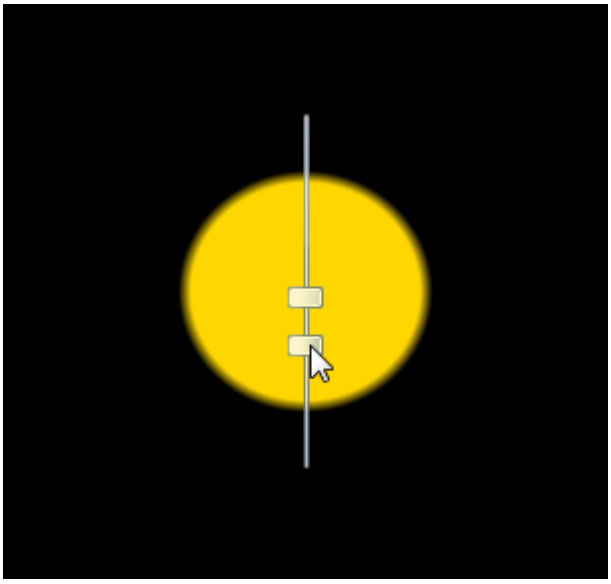
   The application will appear similar to the following:



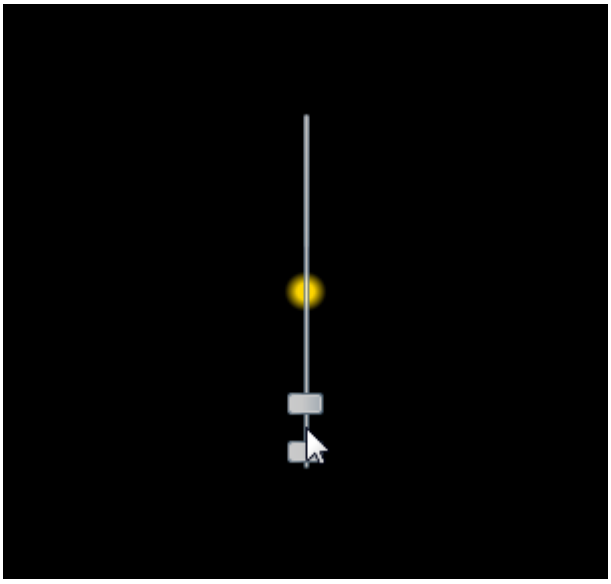
2. Move the top slider thumb down. Notice that the gradient's appearance changes:



3. Move the bottom thumb up, notice that the gradient effect appears less diffused:



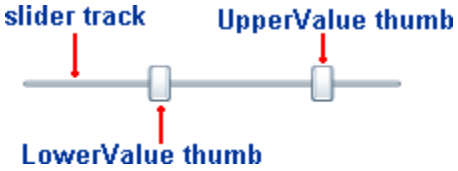
4. Click once between the slider thumbs and drag the cursor down the slider track – notice that both thumbs move together:



Congratulations! You've completed the **RangeSlider for Silverlight** quick start and created a **RangeSlider for Silverlight** application, customized the appearance and behavior of the controls, and viewed some of the run-time capabilities of your application.

Using RangeSlider for Silverlight

ComponentOne RangeSlider for Silverlight includes the C1RangeSlider control, a simple input control that moves beyond the typical slider and includes two thumbs for selecting a range of values. When you add the C1RangeSlider control to a XAML window, it exists as a completely functional slider control which you can further customize. The control's interface looks similar to the following image:



Basic Properties

ComponentOne RangeSlider 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 RangeSlider for Silverlight Appearance Properties for more information about properties that control appearance.

The following properties let you customize the C1RangeSlider control:

Property	Description
Delay	Gets or sets the time, in milliseconds, the **RepeatButtons** (at the left of the LowerValue thumb and at the right of the UpperValue thumb) wait when they are pressed before they start repeating the click action.
Interval	Gets or sets the time, in milliseconds, between repetitions of the click action, as soon as repeating starts (for the **RepeatButtons** at the left of the LowerValue thumb and at the right of the UpperValue thumb).
LowerValue	Gets or sets the current lower magnitude of the range control.
Maximum	Gets or sets the maximum possible value of the range element.
Minimum	Gets or sets the minimum possible value of the range element.
Orientation	The orientation of the C1RangeSlider (horizontal or vertical).
UpperValue	Gets or sets the current upper magnitude of the range control.
ValueChange	Gets or sets a value to be added to or subtracted from the UpperValue/LowerValue of a **RangeBase** control.

Minimum and Maximum

The Minimum and Maximum properties set the possible range of values allowable in the C1RangeSlider control. The thumb with the smaller number, the LowerValue thumb will not be able to bet set to a value lower than the Minimum and the UpperValue thumb will not be able to bet set to a value lower than the Maximum.

By default, the Minimum property is set to **0** and the Maximum property is set to **100**.

Thumb Values and Range

The C1RangeSlider control includes two thumbs for selecting a range of values. The UpperValue and the LowerValue thumbs move along the slider track. By default, the UpperValue property is set to **100** and the LowerValue property is set to **0**.

The value range is determined by the difference between the UpperValue and the LowerValue:

The ValueChange property determines by what value the UpperValue and the LowerValue thumbs move along the slider track when the track is clicked, note that if the tack is clicked between the UpperValue and LowerValue thumbs (in the range) the thumbs will not move.

The UpperValue property cannot be less than the Minimum property and the LowerValue cannot be less than the Maximum property.

Orientation

C1RangeSlider includes the ability to orient the control either horizontally or vertically using the Orientation property. By default the control initially appears with a horizontal orientation when added to the application. You can easily change the orientation from the Properties window, in XAML, and in code using the Orientation property. For more information, see Changing the Orientation.

C1RangeSlider includes the following orientations:

Orientation	Preview
Horizontal (default)	
Vertical	

RangeSlider for Silverlight Layout and Appearance

The following topics detail how to customize the C1RangeSlider 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.

RangeSlider for Silverlight Appearance Properties

ComponentOne RangeSlider 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 **C1RangeSlider** 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.

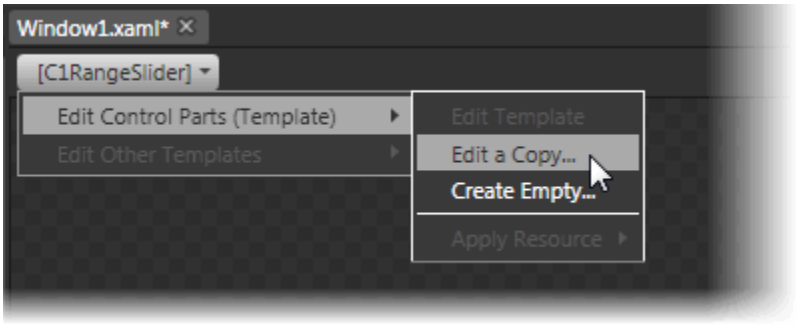
Templates

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 RangeSlider 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 C1RangeSlider 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.

RangeSlider 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 C1RangeSlider control in general. If you are unfamiliar with the **ComponentOne RangeSlider for Silverlight** product, please see the RangeSlider for Silverlight Quick Start first.

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

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

Setting the Thumb Values

The UpperValue and LowerValue properties get or set the value of the two C1RangeSlider thumbs. By default the C1RangeSlider control starts with the UpperValue property set to "100" and LowerValue property set to "0" set 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 UpperValue and LowerValue properties in Blend, complete the following steps:

1. Click the C1RangeSlider control once to select it.
2. Navigate to the Properties window, and enter a number, for example "10", in the text box next to the LowerValue property.
3. In the Properties tab, enter a number, for example "90", in the text box next to the UpperValue property.

   This will set the UpperValue and LowerValue properties to the values you chose.

In XAML

For example, to set the UpperValue and LowerValue properties add `UpperValue="90" LowerValue="10"` to the `<c1:C1RangeSlider>` tag so that it appears similar to the following:

```
<c1:C1RangeSlider Name="C1RangeSlider1" Width="26" Height="18"
UpperValue="90" LowerValue="10" />
```

In Code

For example, to set the UpperValue and LowerValue properties add the following code to your project:

- Visual Basic
```
Me.C1RangeSlider1.LowerValue = 10
Me.C1RangeSlider1.UpperValue = 90
```
- C#
```
this.c1RangeSlider1.LowerValue = 10;
this.c1RangeSlider1.UpperValue = 90;
```

Setting the Value Change

The UpperValue and LowerValue thumbs move along the track when the C1RangeSlider track is clicked. The ValueChange property determines by how much the thumbs move. By default, the ValueChange property is set to "10" and a slider thumb will move by 10 units when the track next to it is clicked. You can customize this value at design time in Microsoft Expression Blend, in XAML, and in code.

At Design Time in Blend

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

1. Click the C1RangeSlider control once to select it.
2. Navigate to the Properties tab, and enter a number, for example "5", in the text box next to the ValueChange property.

   This will set the ValueChange property to the value you chose.

In XAML

For example, to set the ValueChange property add `ValueChange="5"` to the `<c1:C1RangeSlider>` tag so that it appears similar to the following:
```
<c1:C1RangeSlider Name="C1RangeSlider1" Height="18" Width="26"
ValueChange="5" />
```

In Code

For example, to set the ValueChange property add the following code to your project:

- Visual Basic
```
Me.C1RangeSlider1.ValueChange = 5
```
- C#
```
this.c1RangeSlider1.ValueChange = 5;
```

Run the application and observe:

When you click the track of the C1RangeSlider control, the closest thumb will now move by 5 units.

Changing the Background Color

The **Background** property gets or sets the value of the C1RangeSlider control's background color. By default the C1RangeSlider 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 C1RangeSlider 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:C1RangeSlider>` tag so that it appears similar to the following:

```
<c1:C1RangeSlider Name="C1RangeSlider1" Height="18" Width="26"
Background="Red" />
```

In Code

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

- Visual Basic

```
Me.C1RangeSlider1.Background = System.Windows.Media.Brushes.Red
```

- C#

```
this.c1RangeSlider1.Background = System.Windows.Media.Brushes.Red;
```

Run the application and observe:

The background of the C1RangeSlider control will appear red:

Changing the Orientation

By default the Orientation property is set to **Horizontal** and the slider appears horizontally across the page. If you choose, you can change the Orientation so that content control appears vertically placed instead.

At Design Time in Blend

To set the Orientation property to **Vertical** in Microsoft Expression Blend, complete the following steps:

1. Click the C1RangeSlider control once to select it.
2. Navigate to the Properties tab, and locate the Orientation property.
3. Click the drop-down arrow next to the Orientation property and choose **Vertical**.

   This will change the Orientation property so that the control appears vertically.

In XAML

To set the Orientation property to **Vertical** add `Orientation="Vertical"` to the `<c1:C1RangeSlider>` tag so that it appears similar to the following:

```
<c1:C1RangeSlider Name="C1RangeSlider1" Width="26" Orientation="Vertical"
/>
```

In Code

For example, to set the Orientation property to **Vertical**, add the following code to your project:

- Visual Basic

```
Me.C1RangeSlider1.Orientation = Orientation.Vertical
```

- C#

```
this.c1RangeSlider1.Orientation = Orientation.Vertical;
```

Run the application and observe:

The background of the C1RangeSlider control will appear vertical:

TabControl

Arrange content in an efficient, organized manner using **ComponentOne TabControl for Silverlight** The **C1TabControl** control allows you to add tabs and corresponding content pages, thus enabling you to dispense substantial amounts of information while reducing screen space usage.

Getting Started

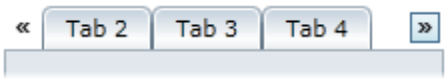
- Working with the C1TabControl Control
- Quick Start
- Task-Based Help

TabControl for Silverlight Key Features

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

- **Scroll Elements**

  The C1TabControl control will automatically insert scroll buttons when the amount of tabs exceeds the specified width or height of the control.



- **Tabstrip Placement**

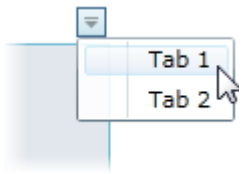
  The C1TabControl control's can be placed at the top, bottom, left, or right of the control.

- **Tab Closing Options**

  Control whether the user can close tabs and where to show the close button. Display the close button inside each tab item or in a global location outside the tabstrip, just like Visual Studio does in its Documents tab.

- **Show Tabs in a Menu**

  The C1TabControl control has an optional tab menu that can be activated by setting the **TabStripMenuVisibility** property to **Visible**. The tab menu enables users to open tab pages using a drop-down menu.



- **Customize the Header's Shape**

You can modify the shape of the tab headers using the TabItemShape property in the TabControl; select from **Rounded**, **Rectangle**, and **Sloped**. This is ideal for non-designer users; you don't need to change the template of the control to change the shape of the tab headers.

- **New Tab Item**

  C1TabControl includes built-in support for the new tab item with a uniform look and feel with the rest of the tabs, just like Microsoft Internet Explorer 8.

- **Align Items in the Header**

  Define if the tab items are overlapped with the right-most in the back or the left-most in the back. The selected item is always on top.

- **Change the Background**

  Keeping the appearance of the tab, you can change its background. While this seems like a very simple feature, C1TabControl is the only tab control in the market that allows you to change the background without having to customize the full template.

- **Overlap Headers**

  Overlap between tab items headers can be customized to show jagged tabs, like the Documents tab in Microsoft Visual Studio.

TabControl for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **TabControl for Silverlight**. In this quick start, you'll start in Blend to create a new project with the C1TabControl control. You will also customize the C1TabControl control, add tabs pages filled with content, and then observe some of the run-time features of the control.

Step 1 of 4: Creating an Application with a C1TabControl Control

In this step, you'll begin in Expression Blend to create a Silverlight application using **TabControl for Silverlight**.

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 the C1TabControl control to your project by completing the following steps:
   d. On the menu, select **Window | Assets** to open the **Assets** tab.
   e. Under the **Assets** tab, enter "C1TabControl" into the search bar.
   f. The C1TabControl control's icon appears.
   g. Double-click the C1TabControl icon to add the control to your project.
5. Add three tabs to the control by completing the following steps:
   a. Under the **Objects and Timeline** tab, select **[C1TabControl]**.
   b. Click the **Assets** tab.
   c. In the search box, enter "C1TabItem".
   d. The C1TabItem icon appears.

e. Double-click the C1TabItem icon to add a tab page to the control.

f. Repeat step 5-e twice to add two more tab pages to the control. The C1TabControl control should contain a total of three tab pages.

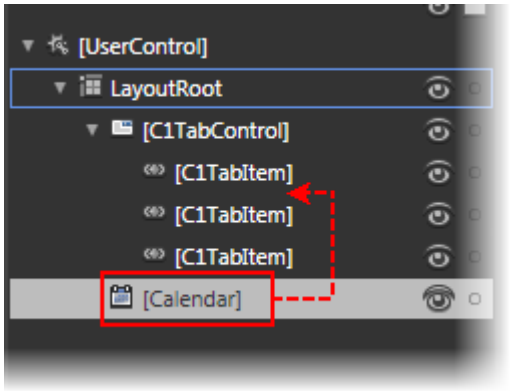
You have completed the first step of the **TabControl for Silverlight** quick start. In this step, you created a project and added a C1TabControl control with three tab pages to it. In the next step, you will add tabs and tab pages to the control.

Step 2 of 4: Adding Tab Pages to the C1TabControl Control

In the last step, you created a Silverlight project in Expression Blend and added a C1TabControl control with three tab pages to it. In this step, you'll customize each of the three tab pages.

Complete the following steps:

1. Modify the first tab page by completing the following steps:
   a. Under the **Objects and Timeline** tab, select the first **[C1TabItem].**
   b. Under the **Properties** panel, set the **Header** property to "Tab 1".
   c. Navigate to the **Assets** tab and, in the text box, enter "Calendar".
   d. Double-click the **Calendar** icon to add the **Calendar** control to the project.
   e. Under the **Objects and Timeline** tab, place the **Calendar** control under the first **[C1TabItem]** using a click-and-drag operation.



   f. Select the Calendar control and, under the **Properties** panel, set the following properties:
      - Click the **Height** glyph to set the **Height** property to **Auto**.
      - Click the **Width** glyph to set the **Width** property to **Auto**.
2. Modify the second tab page by completing the following steps:
   a. Under the **Objects and Timeline** tab, select the second **[C1TabItem].**
   b. Under the **Properties** panel, set the following properties:
      - Set the **Header** property to "Tab 2".
      - Set the **Content** property to "I am the Content property set to a string"
3. Modify the third tab page by completing the following steps:

a. Under the **Objects and Timeline** tab, select the third **[C1TabItem].**

b. Under the **Properties panel**, set the following properties:

- Set the **Header** property to "Tab 3".
- Set the **Content** property to "You can't close this tab. Try it."
- Set the CanUserClose property to **False** by clearing the CanUserClose check box. This will prevent the user from closing the tab.

You have completed step 2 of 4. In this step, you customized the three tab pages of the C1TabControl control. In the next step, you'll customize the appearance and behavior of the C1TabControl control.

Step 3 of 4: Customizing the C1TabControl Control

In the last step, you added three customized tab pages to the C1TabControl control. In this step, you'll customize the C1TabControl control by setting several of its properties.

Complete the following steps:

1. Under the **Objects and Timeline** tab, select **[C1TabControl]**.
2. Under the **Properties** panel, set the following properties:
   - Set the **Height** property to"200".
   - Set the **Width** property to "300".
   - Set the TabItemClose property to **InEachTab**. This will add close buttons to each tab except to the third tab, which you specified shouldn't be allowed to close in the last step of the quick start.
   - Set the TabItemShape property to **Sloped**. This will change the shape of the tab items so that they resemble tabs on an office folder.
   - Set the TabStripMenuVisibility property to **Visible**.
   - Set the TabStripPlacement property to **Bottom**. This will place the tabstrip at the bottom of the control.

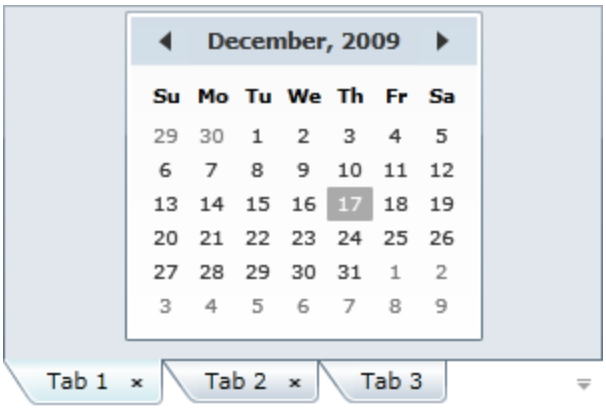
You have completed step 2 of 4 by customizing the features of the C1TabControl control. In the next step, you will run the program and observe what you've accomplished during this quick start.

Step 4 of 4: Running the Project

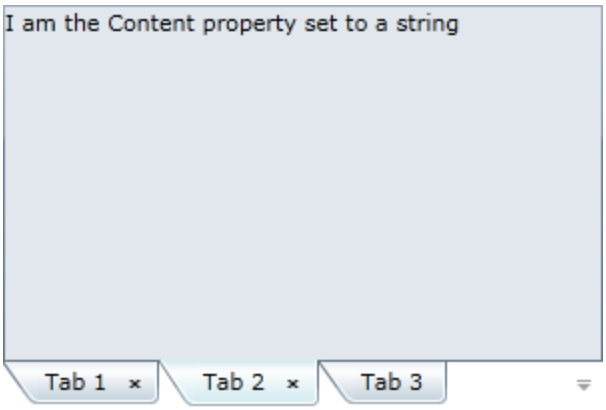
In the previous steps, you created a project with a C1TabControl control, added tab pages to the control, and modified the control's appearance and behaviors. In this step, you will run the program and observe all of the changes you made to the C1TabControl control.

Complete the following steps:

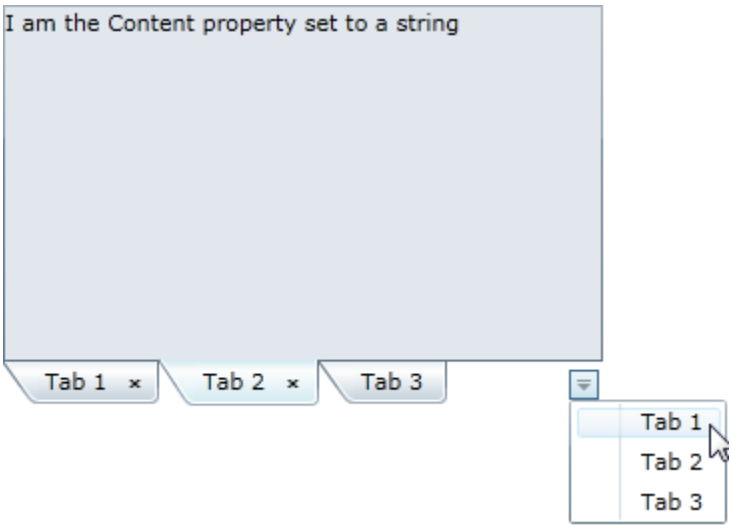
1. Select **Project | Run Project** to run the project. Observe that the C1TabControl control's tabstrip runs along the bottom of the control and features sloped tabs. Also note that it loads with the first tab page, which features a **Calendar** control as content, in view.



2. Click **Tab 2** and observe that the content is just the **Content** property set to a string.

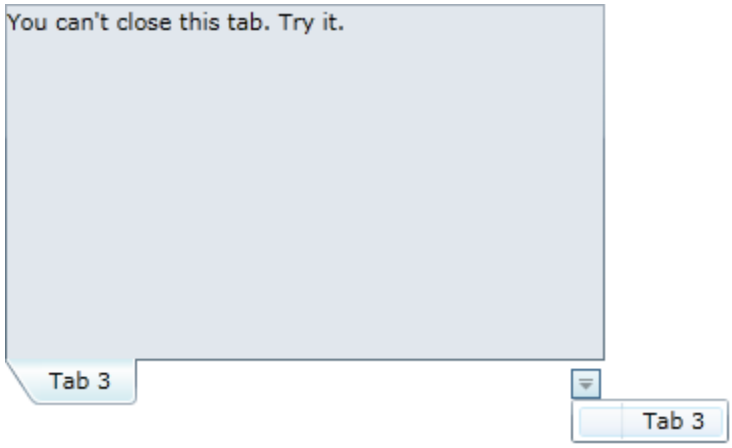


3. Click the drop-down button to open the tab menu and select **Tab 1**.



Tab 1 takes focus again.

4. Click **Tab 1**'s close button to close the tab.
5. Click **Tab 2**'s close button to close the tab. **Tab 3**'s content comes into focus; you can't close this tab because you set its CanUserClose property to **False** in a previous step.
6. Click the menu button and note that only the current tab page, **Tab 3**, is listed because the other two are closed.

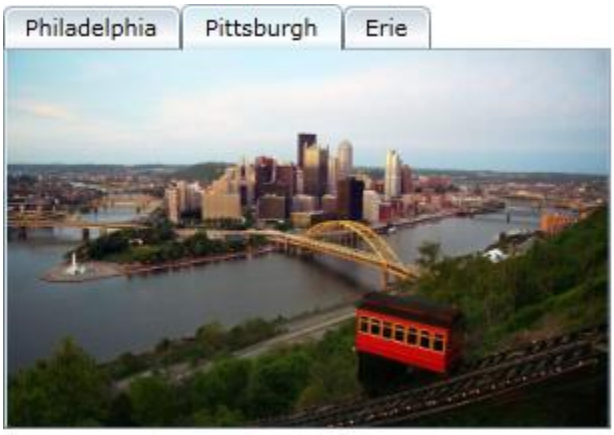


Congratulations! You have completed all four steps of the **TabControl for Silverlight** quick start. In this quick start, you created a project with a fully customized C1TabControl. From here, you can visit Working with the C1TabControl Control to learn more about the control's essentials or visit TabControl for Silverlight Task-Based Help to jump right into using the control.

Working with the C1TabControl Control

The C1TabControl control is a container that that can hold a series of tab pages. Each tab page can store text, images, and controls. Each tab and tab page is represented by the C1TabItem class.

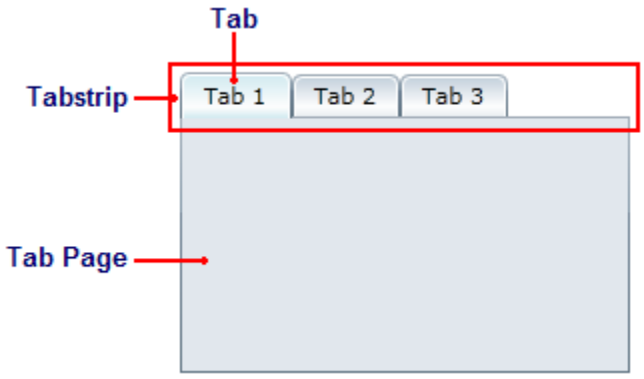
When you add the C1TabControl control to a project, it exists as nothing more than a container. But once the control is added to your project, you can easily add tabs to it in Design view, in XAML, or in code. The following image depicts a C1TabControl control with three tabs.



The following topics provide an overview of the C1TabControl control's elements and features.

C1TabControl Elements

This section provides a visual and descriptive overview of the elements that comprise the C1TabControl control. The control is comprised of three elements – the tab, the tabstrip and the tab page – that combine to make the complete C1TabControl control.



The topics below describe each element of the C1TabControl control.

Tabs

Each tab on the C1TabControl control is represented by the C1TabItem class. The header of the tab is exposed by the Header property of the C1TabItem. Each tab is associated with a tab page (see Tab Page).

Tabs can appear rounded, sloped, or rectangular. When a tab is added to a page, its default appearance is sloped – it looks similar to a tab on an office folder.

You can add a close button to each tab by setting the TabItemClose property to **InEachTab**. This will allow users to close any tab in the control; however, you can deny users the ability to close a particular tab by setting that tab's CanUserClose property to **False**.

Customizing the Header

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

```
<c1:C1TabItem Header="Hello World"/>
```

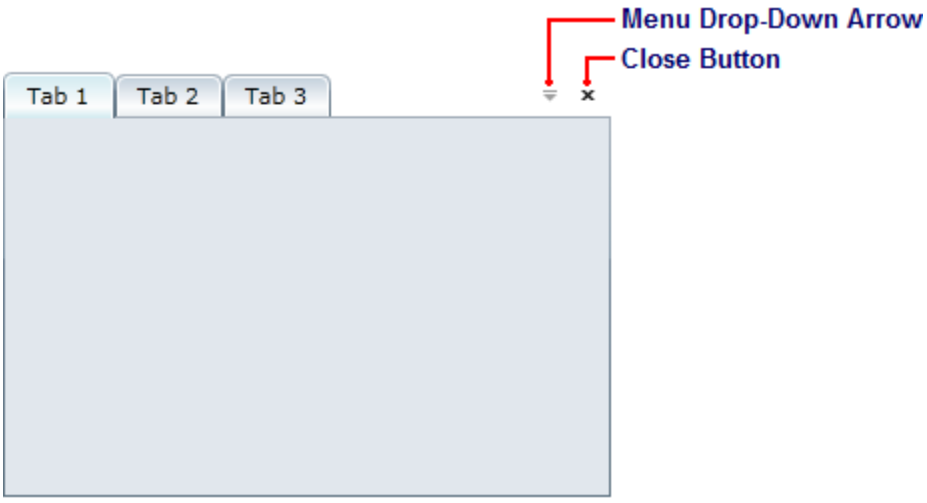
You can also customize the header using the HeaderTemplate.

Tabstrip

The C1TabControl control's tabstrip is created by adding one or more C1TabItem items to the control. Each tab in the strip is associated with a tab page (see Tabs and Tab Page).

By default, the tabstrip appears along the top of the C1TabControl control, but you can adjust the position of the strip by setting the TabStripPlacement property You can also adjust the overlap of the tabs by setting the TabStripOverlap and TabStripOverlapDirection properties.

The tabstrip can also contain two optional items: a close button and a menu drop-down arrow. The close button, when clicked, closes the selected tab; the menu drop-down arrow, when clicked, exposes a drop-down menu from which users can open a different tab.



To learn more about the close button, see Tab Closing. To learn more about the menu drop-down, see Optional Tab Menu.

Tab Page

When a tab (C1TabItem) is added to a C1TabControl control, it will have a corresponding tab page that will initially appear as an empty space. In the tab page, you can add grids, text, images, and arbitrary controls. When working in Blend or Visual Studio 2010's Design view, you can add elements to the tab page using a simple drag-and-drop operation.

You can add text to the tab page by setting the item's **Content** property or by adding a **TextBox** element to the tab page. Adding elements to the tab page at design 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.

A C1TabItem item can only accept one child element at a time. However, you can circumvent this issue 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 C1TabItem item, but its ability to hold multiple elements will allow you to show several controls in the tab page.

Attribute Syntax versus Property Element Syntax

When you want to add something simple to the tab page, 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:C1TabItem Content="Hello World"/>
```

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

```
<c1:C1TabItem>
    <c1:C1TabItem.Content>
        <StackPanel>
            <TextBlock Text="Hello"/>
            <TextBlock Text="World"/>
        </StackPanel>
    </c1:C1TabItem.Content>
</c1:C1TabItem>
```

You can also customize the content using the **ContentTemplate**.

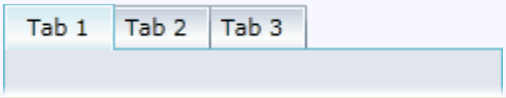
C1TabControl Features

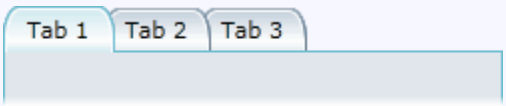
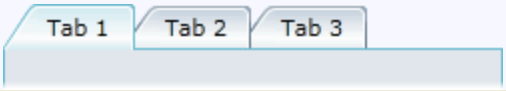
This section details several important features of the C1TabControl control.

Tab Shaping

Tabs can appear rounded, sloped, or rectangular. By default, the tab will appear rounded, but. you can change the shape of the tabs to any of the three settings by setting the **C1TabControl**.TabItemShape property to **Rectangle**, **Rounded**, or **Sloped**.

The following table illustrates each tab shape.

Tab Shape	Illustration
Rectangle	

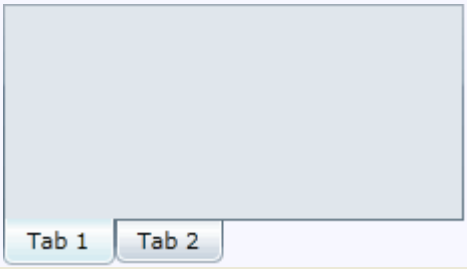
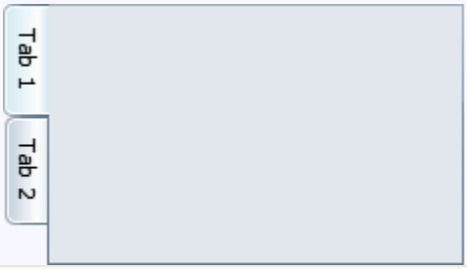
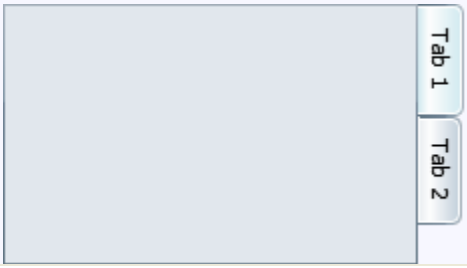
Rounded	
Sloped	

For task-based help, see Changing the Shape of Tabs.

Tabstrip Placement

The C1TabControl control's tabstrip, by default, will appear along the top of the control. However, you can set the **C1TabControl**.TabStripPlacement property to **Bottom**, **Left**, or **Right** to change the position of the tabstrip.

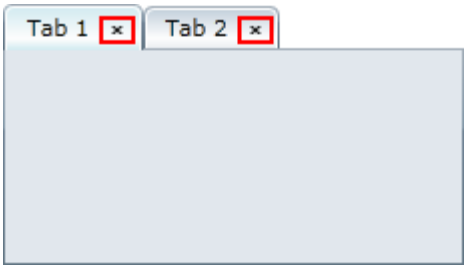
The following table illustrates each **C1TabControl**.TabStripPlacement setting.

ExpandDirection	Result
Top	
Bottom	
Left	
Right	

For task-based help, see Changing the Tabstrip Placement.

Tab Closing

You can add a close button to each tab by setting the TabItemClose property to **InEachTab**. This will allow users to close any tab in the control. On-tab close buttons appears as follows:



If you prefer, you can create a global close button that will appear on the tabstrip instead of directly on the tabs. To add the global close button, you set the TabItemClose property to **GlobalClose**. Clicking the global close button will close the currently selected tab. The global close button appears as follows:

If you want to prevent a user from closing a particular tab, just set that tab's CanUserClose property to **False**. For task-based help, see Allowing Users to Close Tabs.

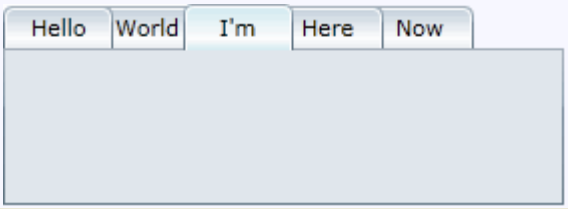
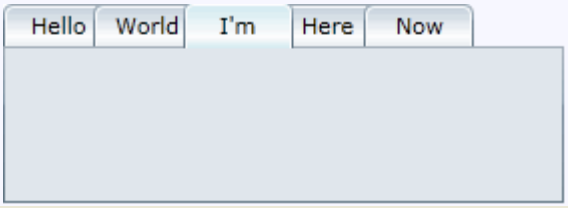
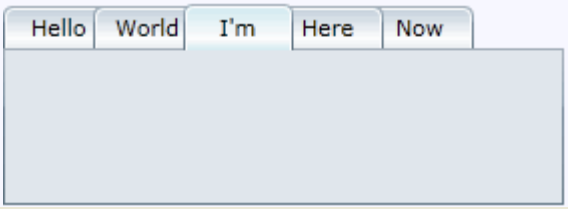
Optional Tab Menu

The C1TabControl control allows you to add a drop-down menu that allows users to select a tab and tab page from a menu. When enabled, this drop-down menu is accessible from the control's tabstrip. The drop-down menu appears as follows:

To activate the drop-down menu, simply set the **C1TabControl**.TabStripMenuVisibility property to **Visible**. For task-based help, see Adding a Menu to the Tabstrip.

Tab Overlapping

You can control the overlapping of tabs by setting the TabStripOverlap property and the TabStripOverlapDirection property. The TabStripOverlap property controls how many pixels of the tab overlap, and the TabStripOverlapDirection direction property allows you to select an enumeration value that sets the direction of the overlap. The TabStripOverlapDirection property has three enumeration values – **Right**, **Left**, and **RightLeftFromSelected** – which are described and illustrated in the table below. Please note that the TabStripOverlap property for each of the following examples has been set to a value of 10.

Enumeration Value	Description	Illustration
Right	The rightmost tabs are in the back while the selected tab is in the front.	
Left	The leftmost tabs are in the back while the selected tab is in the front.	
RightLeftFromSelected	Leftmost tabs are in the back, rightmost tabs are in the back, and the selected tab is in the front.	

TabControl for Silverlight Layout and Appearance

The following topics detail how to customize the C1TabControl 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.

C1TabControl ClearStyle Properties

TabControl 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 **C1TabControl** control:

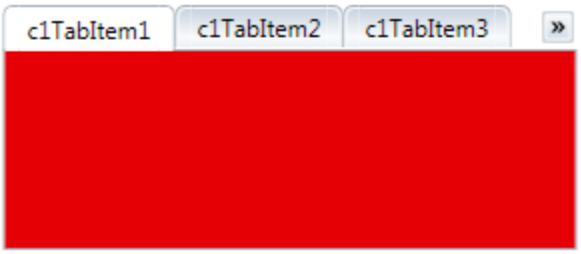
Brush	Description
Background	Gets or sets the brush for the background of each C1TabItem's content area.
TabStripBackground	Gets or sets the brush for the background of the C1TabStrip's background.

MouseOverBrush	Gets or sets the System.Windows.Media.Brush used to highlight the tabs when the mouse is hovered over them.
PressedBrush	Gets or sets the System.Windows.Media.Brush used to highlight the tabs when they are clicked on.

The following table outlines the brush properties of the **C1TabItem** control:

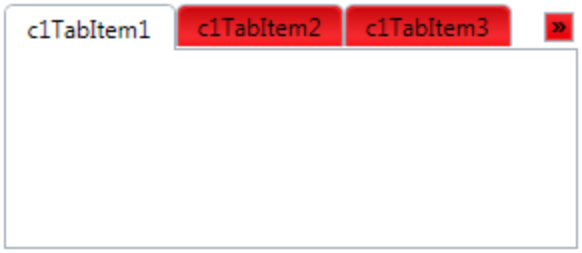
Brush	Description
Background	Gets or sets the backgrounds of all C1TabItems associated with the control.
MouseOverBrush	Gets or sets the System.Windows.Media.Brush used to highlight the tab when the mouse is hovered over them.
PressedBrush	Gets or sets the System.Windows.Media.Brush used to highlight the tab when they are clicked on.

You can completely change the appearance of the **C1TabControl** control by setting a few properties, such as the **Background** property, which sets the background color of the tab control's content area. For example, if you set the **Background** property to "#FFE40005", the **C1TabControl** control would appear similar to the following:



If you clicked through the tabs, you'd notice that the background of each tab's content area is red. If you'd like to change the color of one tab, simply set the **C1TabItem**'s Background property rather than the **C1TabControl**'s **Background** property.

You can also set the color of the tabstrip by setting the **C1TabControl**'s **TabStripBackground** property. In the following example, the **C1TabControl**'s **TabStripBackground** property is set to "#FFE40005":



It's that simple with ComponentOne's ClearStyle technology.

TabControl for Silverlight Appearance Properties

ComponentOne TabControl 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 C1TabControl control and its items.

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 tab. This is a dependency property.
Header	Gets or sets the header of a tab 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 C1TabControl control and its items.

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 C1TabControl control and its items.

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 C1TabControl control and its items.

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 C1TabControl control and its items.

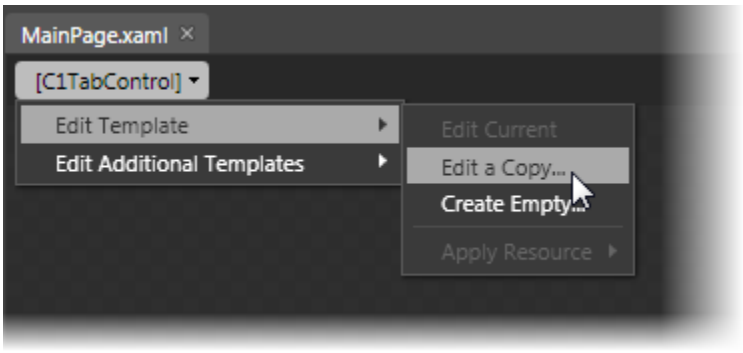
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.
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 TabControl 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 C1TabControl 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 C1TabItem template, simply select the C1TabItem 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 C1TabControl control and C1TabItem control include a few additional templates. These additional templates can also be accessed in Microsoft Expression Blend – in Blend select the C1TabControl or C1TabItem control and, in the menu, select **Edit Additional Templates**. Choose a template, and select **Create Empty.**

Item Templates

ComponentOne TabControl for Silverlight's tab 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 tab control. These templates include an **ItemTemplate**, an **ItemsPanel**, and an **ItemContainerStyle** template. You use the **ItemTemplate** to specify the visualization of the data objects, the **ItemsPanel** to define the panel that controls the layout of items, and the **ItemStyleContainer** to set the style of all container items.

Accessing Templates

You can access these templates in Microsoft Expression Blend by selecting the C1TabControl control and, in the menu, selecting **Edit Additional Templates**. Choose **Edit Generated Items (ItemTemplate)**, **Edit Layout of Items (ItemsPanel)**, or **Edit Generated Item Container (ItemStyleContainer)** 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.

TabControl 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 C1TabControl control to your project, it appears with the default blue theme:

You can also theme the C1TabControl control with one of our six included Silverlight themes: Cosmopolitan, BureauBlack, ExpressionDark, ExpressionLight, RainierOrange, ShinyBlue, and WhistlerBlue.

You can add any of these themes to a C1TabControl control by declaring the theme around the control in markup. For task-based help about using themes, see Using C1TabControl Themes.

TabControl 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 C1TabControl control in general. If you are unfamiliar with the **ComponentOne TabControl for Silverlight** product, please see the **TabControl for Silverlight** Quick Start first.

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

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

Adding a Tab to the C1TabControl Control

In this topic, you will add tabs to the C1TabControl control in Blend, in XAML, and in code.

At Design Time in Blend

Complete the following steps:

1. Click the C1TabControl control once to select it.
2. Under the **Assets** tab, type "C1TabItem" into the search bar.

   The C1TabItem icon appears in the list.
3. Double-click the C1TabItem icon.

   A C1TabItem appears within the C1TabControl control.

In XAML

To add a tab to the C1TabControl control, place the following markup between the `<c1:C1TabControl>` and `</c1:C1TabControl>` tags:

```
<c1:C1TabItem Content="C1TabItem">
</c1ext:C1TabItem>
```

In Code

Complete the following steps:

1. Add `x:Name="C1TabControl1"` to the `<c1:C1TabControl>` 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
     ```

- C#

```
using C1.Silverlight;
```

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

- Visual Basic

```
'Create the tab and add content
Dim C1TabItem1 As New C1TabItem()
C1TabItem1.Content = "C1TabItem1"
'Add the tab to the control
C1TabControl1.Items.Add(C1TabItem1)
```

- C#

```
//Create the tab and add content
C1TabItem C1TabItem1 = new C1TabItem();
C1TabItem1.Content = "C1TabItem1";
//Add the tab to the control
C1TabControl1.Items.Add(C1TabItem1);
```

4. Run the program.

Adding Content to a Tab Page

You can add content to the tab page using the **Content** property. This topic demonstrates how to add content – in this case, a standard **Button** control – to a tab page in Blend, in XAML, and in code. This topic assumes that you have added at least one C1TabItem to the C1TabControl control.

At Design Time in Blend

Complete the following steps:

1. Under the **Objects and Timeline** tab, select **[C1TabItem]**.
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 tab page'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 tab page'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 tab page's content area.
8. Run the program and click the tab.

In XAML

To add a **Button** control to the content area in XAML, place the following markup between the `<c1:C1TabItem>` and `</c1:C1TabItem>` tags:

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

In Code

Complete the following steps:

1. Add `x:Name="C1TabItem1"` to the `<c1:C1TabItem>` 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
C1TabItem1.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
C1TabItem1.Content = (NewButton);
```

3. Run the program.

This Topic Illustrates the Following:

The image below depicts a C1TabItem with a **Button** control as content.

Specifying a Tab Header

In this topic, you will add a header to a tab by setting the Header property Blend, in XAML, and in code. This topic assumes that you have added at least one C1TabItem to the C1TabControl control.

At Design Time in Blend

Complete the following steps:

1. Under the **Objects and Timeline** tab, select **[C1TabItem]**.
2. In the **Properties** panel, set the Header property to "Hello World".

In XAML

To add a header to a tab, add `Header="Hello World"` to the `<c1:C1TabItem>` tag so that the markup resembles the following:

```
<c1:C1TabItem Header="Hello World"></c1:C1TabControl>
```

In Code

Complete the following steps:

1. Add `x:Name="C1TabItem1"` to the `<c1:C1TabItem>` 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

     ```
     C1TabItem1.Header = Hello World
     ```

   - C#

     ```
     C1TabItem1.Header = Hello World;
     ```

3. Run the program.

This Topic Illustrates the Following:

The image below depicts a C1TabItem with a header.

Changing the Tabstrip Placement

The tabstrip of a C1TabControl control is placed on the top by default, but you can also place it to the left, right, or bottom of the control by setting the TabStripPlacement property. In this topic, you will set the TabStripPlacement property to **Right** in Blend, in XAML, and in code. For more information, see Tabstrip Placement. For more information on tabstrip placement settings, see Tabstrip Placement.

At Design Time in Blend

Complete the following steps:

1. Under the **Objects and Timeline** tab, select **[C1TabControl]**.
2. In the **Properties** panel, click the TabStripPlacement drop-down arrow and select **Right** from the list.

In XAML

To change the tabstrip placement, add `TabStripPlacement="Right"` to the `<c1:C1TabControl>` tab so that the markup resembles the following:

```
<c1:C1TabControl TabStripPlacement="Right"></c1:C1TabControl>
```

In Code

Complete the following steps:

1. Add `x:Name="C1TabControl1"` to the `<c1:C1TabControl>` 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

     ```
     C1TabControl1.TabStripPlacement = Right
     ```

   - C#

     ```
     C1TabControl1.TabStripPlacement = Right;
     ```

3. Run the program.

This Topic Illustrates the Following:

The image below depicts a C1TabControl control with its tabstrip placed on its right side.

Changing the Shape of Tabs

The tabs of a C1TabControl control can be rectangular, rounded, or sloped. By default, the tabs are rounded, but you can change the shape of the tabs by setting the C1TabControl control's TabItemShape property. In this topic, you will change the shape of the tabs to **Sloped** in Blend, in XAML, and in code. For more information on tabstrip shaping settings, see Tab Shaping.

At Design Time in Blend

Complete the following steps:

1. Under the **Objects and Timeline** tab, select **[C1TabControl]**.
2. In the **Properties** panel, click the TabItemShape drop-down arrow and select **Sloped** from the list.

In XAML

To change the shape of the tabs, add `TabItemShape="Sloped"` to the `<c1:C1TabControl>` tag so that the markup resembles the following:

```
<c1:C1TabControl TabItemShape="Sloped"></c1:C1TabControl>
```

In Code

Complete the following steps:

1. Add `x:Name="C1TabControl1"` to the `<c1:C1TabControl>` 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

     ```
     C1TabControl1.TabItemShape = Sloped
     ```

   - C#

     ```
     C1TabControl1.TabItemShape = Sloped;
     ```

3. Run the program.

This Topic Illustrates the Following:

The image below depicts a C1TabControl control with sloped tabs.

Allowing Users to Close Tabs

By default, users can't close the tabs on a C1TabControl control. You can alter this by setting the TabItemClose property to **InEachTab** or **GlobalClose** so that users can close tabs by clicking a button inside the tab or by selecting a tab and then clicking a universal close button (see Tab Closing for more information). In this topic, you will set the TabItemClose property to **GlobalClose** in Blend, in XAML, and in code.

At Design Time in Blend

Complete the following steps:

1. Under the **Objects and Timeline** tab, select **[C1TabControl]**.
2. In the **Properties** panel, click the TabItemClose drop-down arrow and select **GlobalClose** from the list.

In XAML

To allow users to close tabs using a universal button, add `TabItemClose="GlobalClose"` to `<c1:C1TabControl>` tab so that the markup resembles the following:

```
<c1:C1TabControl TabItemClose="GlobalClose"></c1:C1TabControl>
```

In Code

Complete the following steps:

1. Add `x:Name="C1TabControl1"` to the `<c1:C1TabControl>` 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

     ```
     C1TabControl1.TabItemClose = GlobalClose
     ```

   - C#

     ```
     C1TabControl1.TabItemClose = GlobalClose;
     ```

3. Run the program.

This Topic Illustrates the Following:

The image below depicts a C1TabControl control tabstrip with a global close button.

Preventing a User from Closing a Specific Tab

When tab closing enabled (see Tab Closing and Allowing Users to Close Tabs for more information), users can disable any tab on the strip by default. However, you can prevent users from closing a specific tab by setting that C1TabItem item's CanUserClose property to **False**.

At Design Time in Blend

Complete the following steps:

1. Under the **Objects and Timeline** tab, select the **[C1TabItem]** you want to prevent users from closing.
2. In the **Properties** panel, clear the CanUserClose check box.

In XAML

To prevent a user from closing a tab, add `CanUserClose="False"` to the `<c1:C1TabItem>` tag of the tab you want to prevent users from closing. The XAML will resemble the following:

```
<c1:C1TabItem CanUserClose="False"></c1:C1TabControl>
```

In Code

Complete the following steps:

1. Add `x:Name="C1TabItem1"` to the `<c1:C1TabItem>` 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

     ```
     C1TabItem1.CanUserClose = False
     ```

- C#

```
C1TabItem1.CanUserClose = false;
```

3. Run the program.

Adding a Menu to the Tabstrip

You can add a menu to the tabstrip so that users can open and close tabs from a context menu instead of by clicking on tabs. To add the tab menu, set the C1TabControl control's TabStripMenuVisibility property to **Visible**.

At Design Time in Blend

Complete the following steps:

1. Under the **Objects and Timeline** tab, select **[C1TabControl]**.
2. In the **Properties** panel, click the TabStripMenuVisibility drop-down arrow and select **Visible** from the list.

In XAML

To add a menu to the tabstrip, add `TabStripMenuVisibility="Visible"` to the `<c1:C1TabControl>` tag so that the markup resembles the following:

```
<c1:C1TabControl TabStripMenuVisibility="Visible"></c1:C1TabControl>
```

In Code

Complete the following steps:

1. Add `x:Name="C1TabControl1"` to the `<c1:C1TabControl>` 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

```
C1TabControl1.TabStripMenuVisibility = Visible
```

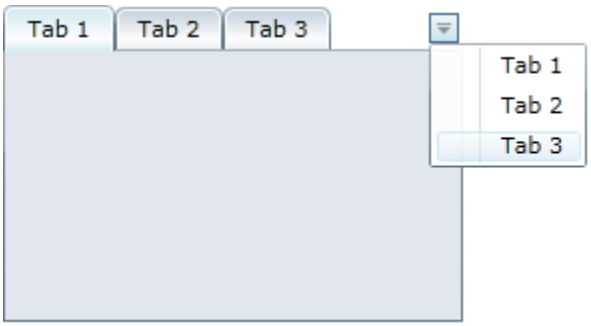
   - C#

```
C1TabControl1.TabStripMenuVisibility = Visible;
```

3. Run the program.

This Topic Illustrates the Following:

The image below depicts a C1TabControl control tabstrip with tab menu.



Overlapping Tabs on a Tabstrip

You can control the overlapping of tabs by setting the TabStripOverlap property and the TabStripOverlapDirection property (for more information, see Tab Overlapping). In this topic, you'll set the TabStripOverlap property and TabStripOverlapDirection properties in Blend, in XAML, and in Code. This topic assumes that you have added a C1TabControl with at least three tabs to your project (see Adding a Tab to the C1TabControl Control).

In Blend

Complete the following steps:

1. Select the C1TabControl control.
2. In the Properties panel, set the following properties:
   - Set the TabStripOverlap property to "12".
   - Set the TabStripOverlapDirection property to **Left**.

In XAML

Add `TabStripOverlap="12"` and `TabStripOverlapDirection="Left"` to the `<c1:C1TabControl>` tab so that the markup resembles the following:

```
<c1:C1TabControl HorizontalAlignment="Left" VerticalAlignment="Top"
Height="156" Width="226" TabStripOverlap="12"
TabStripOverlapDirection="Left">
```

In Code

Complete the following steps:

1. Add `x:Name="C1TabControl1"` to the `<c1:C1TabControl>` 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

     ```
     C1TabControl1.TabStripOverlap = 12

     C1TabControl1.TabStripOverlapDirection = C1.Silverlight.
     C1TabPanelOverlapDirection.Left
     ```

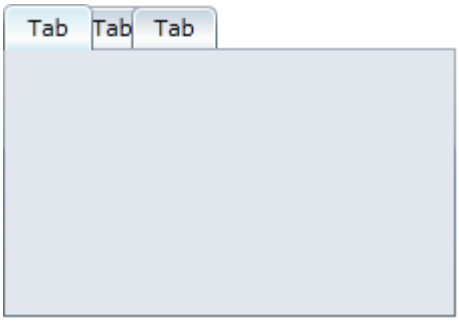
   - C#

     ```
     C1TabControl1.TabStripOverlap = 12;

     C1TabControl1.TabStripOverlapDirection = C1.Silverlight.
     C1TabPanelOverlapDirection.Left;
     ```

3. Run the program.

This Topic Illustrates the Following:

The following image depicts the result of this topic.



Using C1TabControl Themes

The C1TabControl control comes equipped with a light blue default theme, but you can also apply six themes (see TabControl Theming) to the control. In this topic, you will change the C1TabControl 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 "C1TabControl" to search for the C1TabControl control.
5. Double-click the C1TabControl icon to add the C1TabControl control to your project.
6. Under the **Objects and Timeline** tab, select **[C1TabControl]** 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>
```

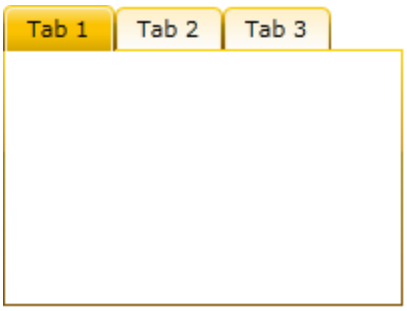
4. Place your cursor between the `<my:C1ThemeRainierOrange>` and `</my:C1ThemeRainierOrange>` tags.
5. In the **Tools** panel, double-click the C1TabControl icon to add the control to the project. Its tags will appear as children of the `<my:C1ThemeRainierOrange>` tags, causing the markup to resemble the following:

```
<my:C1ThemeRainierOrange>
            <c1:C1TabControl></c1:C1TabControl>
</my:C1ThemeRainierOrange>
```

6. Run your project.

This Topic Illustrates the Following:

The following image depicts a C1TabControl control with the C1ThemeRainierOrange theme.



TreeView

Get a hierarchical view of your data items with **ComponentOne TreeView for Silverlight**. It's similar to the TreeView controls available in WPF and Window Forms, but provides more features like keyboard-based search, drag and drop functionality, auto-search, hierarchical templates, and more.

Getting Started

- Quick Start
- Task-Based Help

TreeView for Silverlight Features

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

- **Drag-and-drop Nodes within the TreeView**

  TreeView supports drag-and-drop operations within the tree. Simply set the AllowDragDrop property to true and users will be able to reorder nodes within the tree by dragging them with the mouse.

- **Customizable Drag-drop Behavior**

  TreeView fires events during drag-drop operations so you can customize their behavior. For example, you can prevent some nodes from being dragged or some nodes from acting as drop targets.

- **Auto-search**

  With our TreeView control, you can easily jump to a letter in the node with auto-search. Just type a letter to go to a specific tree node.

- **Hierarchical Templates**

  You can use different templates for different node types without having to subclass the C1TreeViewItem class.

- **Customizable Nodes**

  Node headers are content elements, so they can host any type of element. Add images, checkboxes, or whatever your application requires.

- **Keyboard Navigation**

  Use the cursor keys to navigate the nodes, expanding and collapsing them as you go. Or use the auto-search feature to find specific nodes quickly and easily.

- **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.

- **Supports ClearStyle Technology**

  TreeView for Silverlight supports ComponentOne's ClearStyle technology, which allows you to easily change control colors without having to change control templates. By setting a few color properties, you can quickly style the **C1TreeView** control.

TreeView for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **TreeView for Silverlight**. In this quick start, you'll start in Visual Studio to create a new project, add a C1TreeView control to your application, and then add content to the C1TreeView control's content area.

Step 1 of 3: Creating an Application with a C1TreeView Control

In this step, you'll begin in Visual Studio to create a Silverlight application using **TreeView 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. 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 Toolbox and double-click the **C1TreeView** icon to add the treeview control to **MainPage.xaml**. The XAML markup will now look similar to the following:

```
<UserControl xmlns:c1="clr-
namespace:C1.Silverlight;assembly=C1.Silverlight"
x:Class="TreeViewQuickStart.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="640" d:DesignHeight="480">
  <Grid x:Name="LayoutRoot">
<c1:C1TreeView></c1:C1TreeView>
  </Grid>
</UserControl>
```

   Note that the C1.Silverlight namespace and `<c1:C1TreeView></c1:C1TreeView>` tags have been added to the project.

6. Give your grid a name by adding `x:Name="Tree"` to the `<c1:C1TreeView>` tag so that it appears similar to the following:

```
<c1:C1TreeView x:Name="Tree">
```

   By giving the control a unique identifier, you'll be able to access the **C1TreeView** control in code.

You've successfully created a Silverlight application containing a C1TreeView control. In the next step, you will customize the appearance and behavior of the C1TreeView control.

Step 2 of 3: Adding C1TreeView Items to C1TreeView

This lesson will show you how to add static **C1TreeView** items to the **C1TreeView** control in the XAML markup and in the code behind file.

To add static **C1TreeViewItems** to the **C1TreeView** control in the XAML:

1. Add the C1TreeViewItem to create the top-level node called "Book List". Within the `<c1:C1TreeViewItem>` tag add `Header="Book List`. This will create a top-level node that at run time. The XAML markup appears as follows:

```
<c1:C1TreeViewItem Header="Book List"></c1:C1TreeViewItem>
```

2. Add two child **C1TreeViewItems** below the `<c1:C1TreeViewItem>` tag to create two child nodes beneath the Book List node and add `Header="Language Books"`. In the second child node add `Header="Security Books"`. The XAML markup appears as follows:

```
<c1:C1TreeViewItem Header="Language Books"/>
<c1:C1TreeViewItem Header="Security Books"/>
```

3. Add another `<c1:C1TreeViewItem>` tag to create a new top level node that will hold two child nodes. The XAML markup appears as follows:

```
<c1:C1TreeViewItem Header="Classic Books">
<c1:C1TreeViewItem Header="Catch-22"/>
<c1:C1TreeViewItem Header="The Great Gatsby"/>
```

4. Add two closing `<c1:C1TreeViewItem>` tags to close the **Book List** node and **Classic Books** node. You should have all of the following XAML markup now included in your MainPage.xaml:

```
<UserControl xmlns:c1="clr-
namespace:C1.Silverlight;assembly=C1.Silverlight"
x:Class="TreeViewQuickStart.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="640" d:DesignHeight="480">
  <Grid x:Name="LayoutRoot">
<c1:C1TreeView x:Name="Tree">
        <c1:C1TreeViewItem Header="Book List">
              <c1:C1TreeViewItem Header="Language Books"/>
              <c1:C1TreeViewItem Header="Security Books"/>
              <c1:C1TreeViewItem Header="Classic Books">
                  <c1:C1TreeViewItem Header="Catch-22"/>
                  <c1:C1TreeViewItem Header="The Great Gatsby"/>
                 </c1:C1TreeViewItem>
                  </c1:C1TreeViewItem>
</c1:C1TreeView>
    </Grid>
</UserControl>
```

5. Run the project and notice that the Book node is not expanded. You can expand it by clicking on the arrow image.

To add static **C1TreeView** items to the **C1TreeView** control in the codebehind file:

- Visual Basic

```
Imports C1.Silverlight
Class MainPage
    Public Sub New()
```

```
        InitializeComponent()
        InitializeTreeView()
    End Sub
    Private Sub InitializeTreeView()
        ' Remove items that were added at design time
        Tree.Items.Clear()
        Dim booklist As New C1TreeViewItem()
        booklist.Header = "Book List"
        Tree.Items.Add(booklist)

        ' Adding child items
        Dim language As New C1TreeViewItem()
        language.Header = "Language Books"
        booklist.Items.Add(language)

        ' Adding child items
        Dim security As New C1TreeViewItem()
        security.Header = "Security Books"
        booklist.Items.Add(security)

        ' Adding child items
        Dim classic As New C1TreeViewItem()
        classic.Header = "Classic Books"
        booklist.Items.Add(classic)

        ' Adding child items
        Dim subclassic As New C1TreeViewItem()
        subclassic.Header = "Catch-22"
        classic.Items.Add(subclassic)
        Dim subclassic2 As New C1TreeViewItem()
        subclassic2.Header = "The Great Gatsby"
        classic.Items.Add(subclassic2)
    End Sub
End Class
```

- C#

```
using C1.Silverlight;
public MainPage()
{
    InitializeComponent();
    InitializeTreeView();
}
void InitializeTreeView()
{
    // Remove items that were added at design time
    Tree.Items.Clear();
    C1TreeViewItem booklist = new C1TreeViewItem();
    booklist.Header = "Book List";
    Tree.Items.Add(booklist);

   // Adding child items
    C1TreeViewItem language = new C1TreeViewItem();
    language.Header = "Language Books";
    booklist.Items.Add( language );

    // Adding child items
    C1TreeViewItem security = new C1TreeViewItem();
```

```
    security.Header = "Security Books";
    booklist.Items.Add(security);

    // Adding child items
    C1TreeViewItem classic = new C1TreeViewItem();
    classic.Header = "Classic Books";
    booklist.Items.Add(classic);

    // Adding child items
    C1TreeViewItem subclassic = new C1TreeViewItem();
    subclassic.Header = "Catch-22";
    classic.Items.Add(subclassic);
    C1TreeViewItem subclassic2 = new C1TreeViewItem();
    subclassic2.Header = "The Great Gatsby";
    classic.Items.Add(subclassic2);
}
```

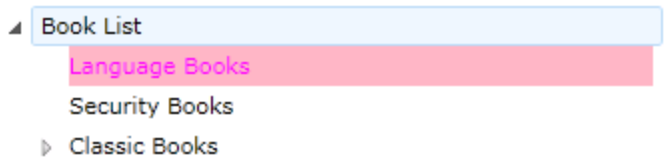
In this step, you added several **C1TreeView** items to the C1TreeView control. In the next step, you will customize the behavior and appearance of the **C1TreeView** control.

Step 3 of 3: Customizing TreeView's Appearance and Behavior

In the previous step you worked in Visual Studio to create **C1TreeViewItems** in XAML. In this step you'll customize the **C1TreeView** control's appearance and behavior in using XAML code.

To customize **TreeView for Silverlight**, complete the following steps:

1. Place your cursor within the `<c1:C1TreeView>` tag. Within the `<c1:C1TreeView>` tag add `SelectionMode="Extended"`. This will create a top-level node that you will be able to select multiple tree items by holding the shift and control keys. The XAML markup appears as follows:
```
<c1:C1TreeView x:Name="Tree" SelectionMode="Extended">
```
2. Place your cursor within the first `<c1:C1TreeViewItem>` tag. Within the `<c1:C1TreeViewItem>` add the tag `IsExpanded="True"`and `IsSelected="True"`. This will create a top-level node that appears selected and expanded at run time. The XAML markup appears as follows:
```
<c1:C1TreeViewItem Header="Book List" IsExpanded="True" IsSelected="True">
```
3. Locate the tag that reads `<c1:C1TreeViewItem Header="Language Books">`. Within the `<c1:C1TreeViewItem Header="Language Books">` add `Foreground="Fuchsia" Background="LightPink"`. This will add a light pink background to the "Classic Books" tree item and a fuchsia color to the text. The XAML markup will resemble the following:
```
  <c1:C1TreeViewItem Header="Language Books" Foreground="Fuchsia"
Background="LightPink"/>
```
4. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time. Observe the following behavioral and appearance changes for C1TreeView:



- The C1TreeView appears expanded.
- The first C1TreeViewItem appears selected.
- The second C1TreeViewItem has light pink background and a fuchsia color text.

- You can select multiple C1TreeViewItems holding down the control and shift key.

Congratulations! You have successfully completed the **TreeView for Silverlight** quick start. In this quick start, you've created and customized a **TreeView for Silverlight** application, added static C1TreeViewItems, and observed several of the control's run-time features.

C1TreeView Structure

The **C1TreeView** class is a **StackPanel** with two elements:

- A header that represents the actual node, with a button to collapse and expand the children.
- A body that is another **StackPanel** and contains other nodes.

You can add images to a node by grabbing its first child (the header), casting that to a **StackPanel**, and inserting an image element at whatever position you prefer. For example:

```
    StackPanel nodeHeader = TreeNode.Children[0] as StackPanel;
    nodeHeader.Children.Insert(0, myImage);
```

TreeView Creation

C1TreeViewItems can be added to the **C1TreeView** control as static items defined either in the XAML or in the code behind or can be defined on your page or user control by using any of the following methods:

- Static creation using XAML syntax or programmatically through the code behind file
- Dynamic creation using a constructor to create new instances of the C1TreeViewItem class.
- Data source creation through binding **C1TreeView** to a **SiteMapDataSource**, **XMLDataSource**, or an **AccessDataSource**.

Static TreeView Creation

Each node in the Tree is represented by a name/value pair, defined by the text and value properties of treenode, respectively. The text of a node is rendered, whereas the value of a node is not rendered and is typically used as additional data for handling postback events.

A static menu is the simplest way to create the treeview structure.

To display static C1TreeViewNodes using XAML syntax, first nest opening and closing <Nodes> tags between opening and closing tags of the C1TreeView control. Next, create the treeview structure by nesting <asp:C1TreeViewNode> elements between opening and closing <Nodes> tags. Each <asp:C1TreeViewNode> element represents a node in the control and maps to a C1TreeViewNode object.

Declarative syntax can be used to define the C1TreeViewNodes inline on your page.

For example:

```
<Grid x:Name="LayoutRoot">
<c1:C1TreeView x:Name="Tree">
         <c1:C1TreeViewItem Header="Book List" IsExpanded="True"
IsSelected="True">
                <c1:C1TreeViewItem Header="Language Books"/>
                <c1:C1TreeViewItem Header="Security Books"/>
                <c1:C1TreeViewItem Header="Classic Books">
                    <c1:C1TreeViewItem Header="Catch-22"/>
                    <c1:C1TreeViewItem Header="The Great Gatsby"/>
                  </c1:C1TreeViewItem>
                   </c1:C1TreeViewItem>
 </c1:C1TreeView>
    </Grid>
```

Dynamic TreeView Creation

Dynamic treeviews can be created on the server side or client side. When creating dynamic treeview on the server side, use a constructor to dynamically create a new instance of the C1TreeView class. For example:

- Visual Basic

```
Namespace TreeViewQuickStart
    Public Partial Class MainPage
        Inherits UserControl
        Public Sub New()
            InitializeComponent()

            InitializeTreeView()
        End Sub
        Private Sub InitializeTreeView()

            ' Remove items that were added at design time

            Tree.Items.Clear()

            Dim booklist As New C1TreeViewItem()
            booklist.Header = "Book List"
            Tree.Items.Add(booklist)

            ' Adding child items
            Dim language As New C1TreeViewItem()
            language.Header = "Language Books"
            booklist.Items.Add(language)

            ' Adding child items
            Dim security As New C1TreeViewItem()
            security.Header = "Security Books"
            booklist.Items.Add(security)

            ' Adding child items
            Dim classic As New C1TreeViewItem()
            classic.Header = "Classic Books"
            booklist.Items.Add(classic)
            'Add checkbox
            classic.Header = New CheckBox()

            ' Adding child items
            Dim subclassic As New C1TreeViewItem()
            subclassic.Header = "Catch-22"
            classic.Items.Add(subclassic)
            Dim subclassic2 As New C1TreeViewItem()
            subclassic2.Header = "The Great Gatsby"
            classic.Items.Add(subclassic2)
        End Sub
    End Class
End Namespace
```

- C#

```
namespace TreeViewQuickStart
{
    public partial class MainPage : UserControl
    {
        public MainPage()
```

```
        {
            InitializeComponent();
            InitializeTreeView();

        }
       void InitializeTreeView()
       {

            // Remove items that were added at design time

            Tree.Items.Clear();

            C1TreeViewItem booklist = new C1TreeViewItem();
            booklist.Header = "Book List";
           Tree.Items.Add(booklist);

           // Adding child items
          C1TreeViewItem language = new C1TreeViewItem();
          language.Header = "Language Books";
          booklist.Items.Add( language );

         // Adding child items
         C1TreeViewItem security = new C1TreeViewItem();
         security.Header = "Security Books";
         booklist.Items.Add(security);

        // Adding child items
        C1TreeViewItem classic = new C1TreeViewItem();
        classic.Header = "Classic Books";
        booklist.Items.Add(classic);
        //Add checkbox
        classic.Header = new CheckBox();

       // Adding child items
       C1TreeViewItem subclassic = new C1TreeViewItem();
       subclassic.Header = "Catch-22";
       classic.Items.Add(subclassic);
       C1TreeViewItem subclassic2 = new C1TreeViewItem();
       subclassic2.Header = "The Great Gatsby";
       classic.Items.Add(subclassic2);
}
    }
}
```

Data Source TreeView Creation

TreeView items can be created from a hierarchal datasource control such as an **XMLDataSource** or **SiteMapDataSource**. This allows you to update the treeview items without having to edit code.

When using multi-level data as ItemsSource for the C1TreeView, you need to specify a C1HierarchicalDataTemplate for the items.

The template will tell the C1TreeView where to find the next level of data, this is done through the "ItemsSource" property of the C1HierarchicalDataTemplate.

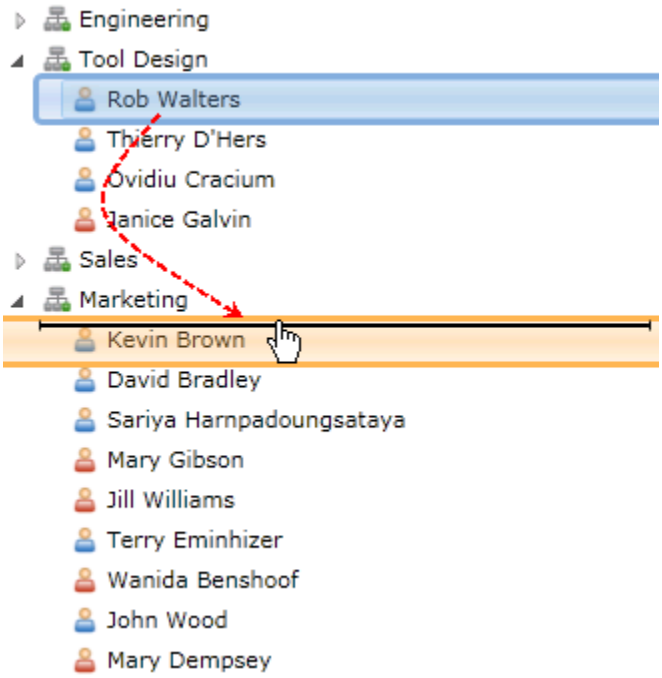
TreeView Behavior

The following topics describe the behavior for the **C1TreeView** control.

Drag-and-Drop Nodes

You can drag-and-drop **C1TreeViewNode**s on nodes, in between nodes, or from one tree to another tree when the AllowDragDrop property is set to **True**.

The following image shows a **C1TreeViewNode** being dragged from one **C1TreeView** to another **C1TreeView**. An arrow or vertical line can be used as a visual cue to show you where the **C1TreeViewItem** is going to be dropped when either the DragDropArrowMarker or DragDropLineMarker properties are applied.



Load on Demand

Instead of fully populating each node when the application starts, you can use a technique called "delayed loading", where nodes are populated on demand, when the user expands them. This allows the application to load faster and use resources more efficiently

To implement the delayed loading nodes, use the following code:

```
public Page()
{
    InitializeComponent();
    // No changes here.
    // ...
    // Initialize the C1TreeView
    InitializeTreeView();
}
void InitializeTreeView()
{
  // Remove items that were added at design time
  _tv.Items.Clear();
```

```
   // Scan every type in the assembly
  foreach (Type t in _tv.GetType().Assembly.GetTypes())
  {
    if (t.IsPublic && !t.IsSpecialName && !t.IsAbstract)
    {
      // Add node for this type
      C1TreeViewItem node = new C1TreeViewItem();
      node.Header = t.Name;
      node.FontWeight = FontWeights.Bold;
      _tv.Items.Add(node);
      // Add subnodes for properties, events, and methods
      node.Items.Add(CreateMemberNode("Properties", t,
MemberTypes.Property));
      node.Items.Add(CreateMemberNode("Events", t, MemberTypes.Event));
      node.Items.Add(CreateMemberNode("Methods", t,
MemberTypes.Method));
    }
  }
}
C1TreeViewItem CreateMemberNode(string header, MemberTypes memberTypes)
{
  // Create the node
  C1TreeViewItem node = new C1TreeViewItem();
  node.Header = header;
  node.Foreground = new SolidColorBrush(Colors.DarkGray);
   // Hook up event hander to populate the node before expanding it
  node.Expanding += node_Expanding;
   // Save information needed to populate the node
  node.Tag = memberTypes;
   // Add a dummy node so this node can be expanded
  node.Items.Add(new C1TreeViewItem());
  node.IsExpanded = false;
   // Finish
  return node;
}
//populate the node
void node_Expanding(object sender, RoutedEventArgs e)
{
  // Get the node that fired the event
  C1TreeViewItem node = sender as C1TreeViewItem;
  // Unhook event handler (we'll populate the node and be done with it)
  node.Expanding -= node_Expanding;
   // Remove dummy node
  node.Items.Clear();
   // Populate the node
  Type type = (Type)node.Parent.Tag;
  MemberTypes memberTypes = (MemberTypes)node.Tag;
   BindingFlags bf = BindingFlags.Public | BindingFlags.Instance;
  foreach (MemberInfo mi in type.GetMembers(bf))
  {
    if (mi.MemberType == memberTypes)
    {
      if (!mi.Name.StartsWith("get_") && !mi.Name.StartsWith("set_"))
      {
        C1TreeViewItem item = new C1TreeViewItem();
        item.Header = mi.Name;
        item.FontSize = 12;
```

```
            node.Items.Add(item);
        }
      }
    }
}
```

This implementation hooks up an event hander for the **Expanding** event, so we can populate the node when the user tries to expand it. We also save the information we will need to populate the node in the Tag property. Finally, we add a dummy child node so the user will be able to expand this node and trigger the Expanding event that will populate the node.

Note that instead of using the **Tag** property, we could also have derived a custom class from **C1TreeViewNode** and built all the delay-load logic into that class. This would be a more elegant approach, but unfortunately Silverlight doesn't support template inheritance. If you derive a class from a class that has a template (such as **Button** or **C1TreeViewNode**), the template is not inherited, and you have to provide a template yourself or your derived class will be just an empty control.

Node Selection

When you click on a node at run time it is automatically marked as selected. Clicking a node will raise the SelectionChanged event to provide custom functionality. To have the nodes marked as selected without clicking them you can enable the IsSelected property.

When the user selects a new item, the **C1TreeView** fires the SelectionChanged event. You can then retrieve the item that was selected using the SelectedItem property.

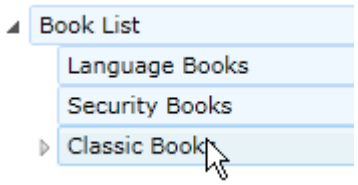
There're several ways to do this. One is to assign additional data to the **Tag** property of each **C1TreeViewItem** as you create them. Later, you can inspect the **Tag** property to retrieve the information. For example:

- Visual Basic
```
' Create a node and assign some data to its Tag property
Dim item As New C1TreeViewItem()
item.Header = "Beverages"
item.Tag = beveragesID
```
- C#
```
// Create a node and assign some data to its Tag property
C1TreeViewItem item = new C1TreeViewItem();
item.Header = "Beverages";
item.Tag = beveragesID;
```

Later, use the information in whatever way you see fit:

- Visual Basic
```
Dim item As C1TreeViewItem = _tv.SelectedItem
    ' Handle beverages node
If TypeOf item.Tag Is Integer AndAlso CInt(item.Tag) = beveragesID Then
End If
```
- C#
```
C1TreeViewItem item = _tv.SelectedItem;
if (item.Tag is int && (int)item.Tag == beveragesID)
{
    // Handle beverages node
}
```

If the SelectionMode property is set to Multiple then multiple nodes can be selected at one time by holding down the control key while mouse clicking multiple nodes. To unselect a node, click on it again. The nodes are marked as selected in the following C1TreeView:



Node Navigation

C1TreeView supports mouse and keyboard navigation.

Navigating C1TreeViewNodes using the mouse

The following table describes the actions and corresponding mouse commands when navigating through the **C1TreeViewNode**s:

Action	Mouse Command
Expand a node	Click on the plus sign at the left of the node's name.
Collapse a node	Click on the minus sign at the left of the node's name.
Select a node	Click on the node's name.

Navigating C1TreeViewNodes using the keyboard

The following table describes the actions and their associated keys to use when navigating through **C1TreeViewNode**s:

Action	Keyboard Command
Expand a node	+ KEY
Collapse a node	- KEY
Move up a node	UP ARROW KEY
Move down a down	DOWN ARROW KEY
Select multiple nodes	MOUSE + CTRL KEY

TreeView for Silverlight Layout and Appearance

The following topics detail how to customize the C1TreeView 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 treeview and take advantage of Silverlight's XAML-based styling.

You can customize the appearance of your treeview item by using the Header property. You can set the Header to be any object such as a stack panel containing an image and some text:

```
<C1TreeViewItem>
   <C1:C1TreeViewItem.Header>
      <StackPanel Orientation="Horizontal">
         <Image Source="myImage,jpg"/>
         <TextBlock Text="My Text"/>
      </StackPanel>
   </C1:C1TreeViewItem.Header>
</C1:C1TreeViewItem>
```

TreeView for Silverlight Appearance Properties

ComponentOne TreeView 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 C1TreeView 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.
C1HierarchicalPresenter.Header	Gets or sets the item that labels the control.

Content Positioning Properties

The following properties let you customize the position of header and content area content in the C1TreeView control.

Property	Description
Padding	Gets or sets the padding inside a control. This is a dependency property.
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..
HorizontalContentAlignment	Gets or sets the horizontal alignment of the control's content. 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.
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.

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 **C1TreeView** control.

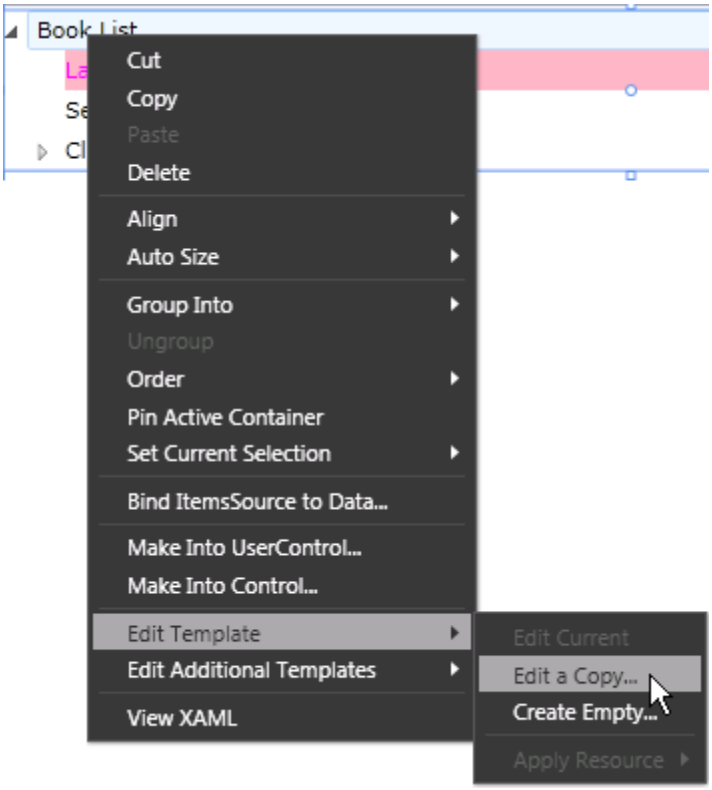
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.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

C1TreeView 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 TreeView 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 C1TreeView control and, in the menu, selecting **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or **Create Empty**, to create a new blank template.

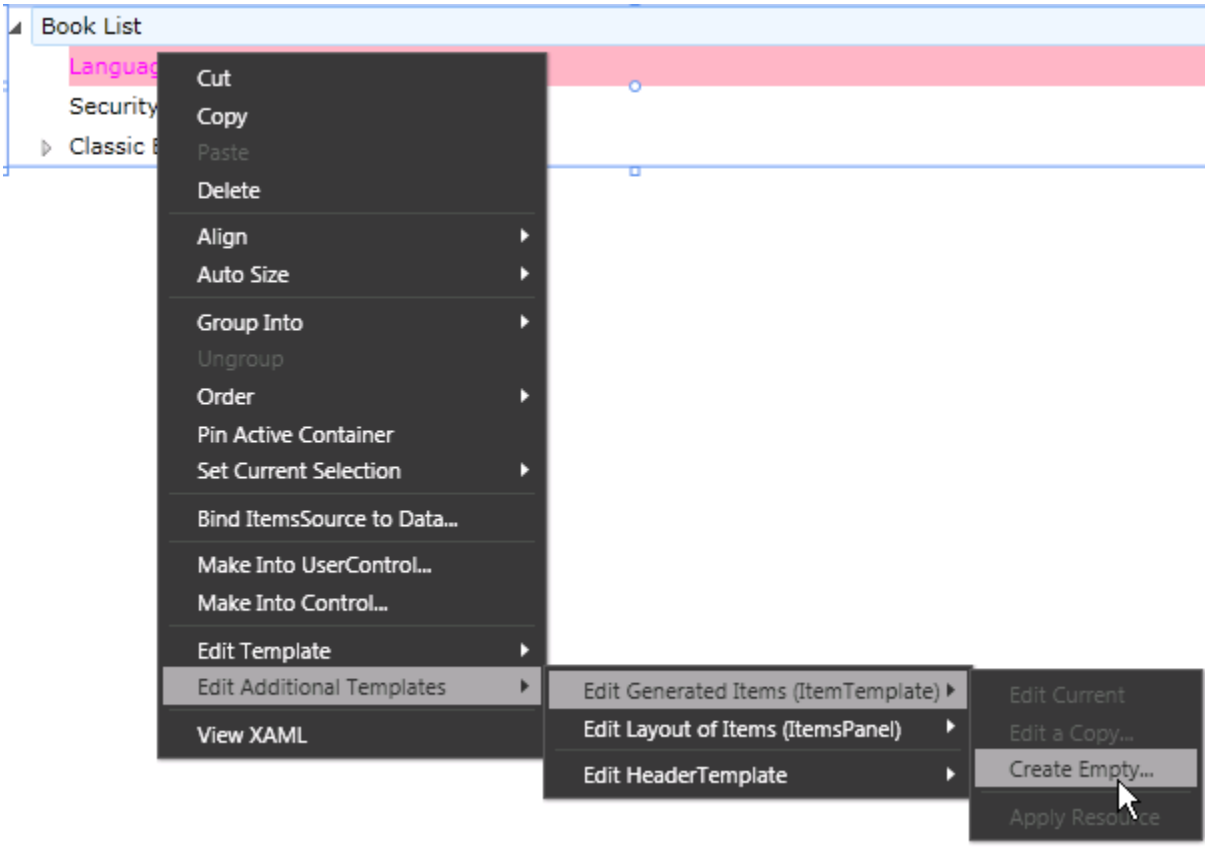


Once you've created a new template, the template will appear in the **Objects and Timeline** window. Note that you can use the Template property to customize the template.

> **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.

Additional Templates

In addition to the default template, the C1TreeView control includes a few additional templates. These additional templates can also be accessed in Microsoft Expression Blend – in Blend select the C1TreeView control and, in the menu, select **Edit Additional Templates**. Choose a template, and select **Create Empty**:



C1TreeView Styles

ComponentOne TreeView for Silverlight's TreeView 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.

TreeView Theming

Silverlight themes, in short, 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.

But the C1TreeView control can also be themed with one of our six included Silverlight themes: Cosmopolitan, BureauBlack, ExpressionDark, ExpressionLight, RanierOrange, ShinyBlue, and WhistlerBlue. You can add any of these themes to a C1TreeView control by declaring it around the control in markup. For more information about adding themes, see the Using Silverlight Themes.

TreeView ClearStyle Properties

TreeView for Silverlight supports ComponentOne's ClearStyle technology, which allows you to easily change control colors without having to change control templates. By setting a few color properties, you can quickly style the **C1TreeView** control. The supported properties for **C1TreeView** are listed in the following table:

Property	Description
Background	Gets or sets the background used to fill the **C1DockControl**.
MouseOverBrush	Gets or sets the brush used to highlight the control when the mouse is hovering over it.
SelectedBackground	Gets or sets the background color of the selected item.

You can completely change the appearance of the **C1TreeView** by setting these properties. For example, if you set the **C1TreeView.Background** property to **SkyBlue** so the XAML markup appears similar to the following:

```
<c1:C1TreeView Background="SkyBlue" x:Name="Tree">
    <c1:C1TreeViewItem Header="Book List">
        <c1:C1TreeViewItem Header="Language Books"/>
        <c1:C1TreeViewItem Header="Security Books"/>
        <c1:C1TreeViewItem Header="Classic Books">
            <c1:C1TreeViewItem Header="Catch-22"/>
            <c1:C1TreeViewItem Header="The Great Gatsby"/>
        </c1:C1TreeViewItem>
    </c1:C1TreeViewItem>
</c1:C1TreeView>
```

The **C1TreeView** will look similar to the following:

Experiment with the other properties to quickly change the look of the **C1TreeView** elements. For example, the following XAML sets the **Background**, **MouseOverBrush**, and **SelectedBackground** properties:

```
<c1:C1TreeView Background="SkyBlue" MouseOverBrush="Gold"
SelectedBackground="Green" x:Name="Tree">
    <c1:C1TreeViewItem Header="Book List">
        <c1:C1TreeViewItem Header="Language Books"/>
        <c1:C1TreeViewItem Header="Security Books"/>
        <c1:C1TreeViewItem Header="Classic Books">
            <c1:C1TreeViewItem Header="Catch-22"/>
            <c1:C1TreeViewItem Header="The Great Gatsby"/>
        </c1:C1TreeViewItem>
    </c1:C1TreeViewItem>
```

```
        </c1:C1TreeView>
```

The **C1TreeView** will look similar to the following when you run the project:

When you mouse over a header, it appears yellow. When you select an item, it appears green.

TreeView 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 C1TreeView control in general. If you are unfamiliar with the **ComponentOne TreeView for Silverlight** product, please see the **TreeView for Silverlight** Quick Start first.

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

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

Using Silverlight Themes

The C1TreeView control comes equipped with a light blue default theme, but you can also apply six themes (see TreeView Theming) to the control. In this topic, you will change the C1TreeView control's theme to **C1ThemeRanierOrange**.

Complete the following steps:

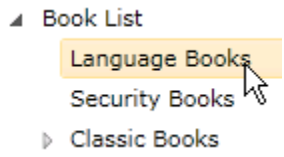
1. Open the **.xaml** page in Visual Studio.
2. Place your cursor between the `<Grid></Grid>` tags.
3. In the Toolbox, double-click the **C1ThemeRanierOrange** icon to declare the theme. Its tags will appear as follows:
```
<my:C1ThemeRainierOrange></my:C1ThemeRainierOrange>
```
4. Place your cursor between the `<my:C1ThemeRanierOrange>` and `</my:C1ThemeRanierOrange>` tags.
5. In the Toolbox, double-click the C1TreeView icon to add the control to the project. Its tags will appear as children of the `<my:C1ThemeRanierOrange>` tags, causing the markup to resemble the following:
```
<my:C1ThemeRainierOrange>
    <c1:C1TreeView></c1:C1TreeView>
</my:C1ThemeRainierOrange>
```
6. To create the **C1TreeViewItem**s, enter the following markup between the <c1:C1TreeView> and </c1:C1TreeView> tags:
```
    <c1:C1TreeViewItem Header="Book List">
        <c1:C1TreeViewItem Header="Language Books"/>
            <c1:C1TreeViewItem Header="Security Books"/>
            <c1:C1TreeViewItem Header="Classic Books">
```

```
            <c1:C1TreeViewItem Header="Catch-22"/>
            <c1:C1TreeViewItem Header="The Great Gatsby"/>
        </c1:C1TreeViewItem>
    </c1:C1TreeViewItem>
```

7. Now you have to set the **ApplyMode** property; before you do that, however, you will have to add the following statement to the <UserControl> tag to reference the C1.Silverlight.Theming assembly:
```
xmlns: c1Theming="clr-
namespace:C1.Silverlight.Theming;assembly=C1.Silverlight.Theming"
```
8. Set the **ApplyMode** property by adding c1Theming:ImplicitStyleManager.ApplyMode=”Auto” to the <c1ext:C1TreeView> tag so that your markup resembles the following:
```
<c1:C1TreeView c1Theming:ImplicitStyleManager.ApplyMode="Auto"
x:Name="Tree">
```

What You've Accomplished

Run your project and observe that the TreeView resembles the following:



Getting the Text or Value of the SelectedItem in a TreeView

The **Header** property will return the values contained in your **C1TreeViewItem**s, you can get the string value using the following code:

- Visual Basic
```
Dim item As C1TreeViewItem = _tree.SelectedItem
_textBlock.Text = item.Header.ToString()
```
- C#
```
C1TreeViewItem item = _tree.SelectedItem;
_textBlock.Text = item.Header.ToString();
```

What You've Accomplished

Run your application and observe that the **Header** property will return the values contained in your **C1TreeViewItem**s.

Adding Check Boxes to the TreeView

You can easily add check boxes to the **C1TreeView** control, check boxes can appear before text and allow users to select a tree view item. The following XAML markup adds check boxes to the **C1TreeView**:

- XAML
```
<c1:C1TreeView Name="C1TreeView1" Height="300" Width="200" >
    <c1:C1TreeViewItem IsExpanded="True" Margin="10">
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
                    <TextBlock Text="Desktop" />
                </CheckBox.Content>
            </CheckBox>
        </c1:C1TreeViewItem.Header>
```

```
    <c1:C1TreeViewItem>
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
                    <TextBlock Text="User" />
                </CheckBox.Content>
            </CheckBox>
        </c1:C1TreeViewItem.Header>
    </c1:C1TreeViewItem>
    <c1:C1TreeViewItem>
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
                    <TextBlock Text="Public" />
                </CheckBox.Content>
            </CheckBox>
        </c1:C1TreeViewItem.Header>
        <c1:C1TreeViewItem>
            <c1:C1TreeViewItem.Header>
                <CheckBox>
                    <CheckBox.Content>
                    <TextBlock Text="Favorites" />
                    </CheckBox.Content>
                </CheckBox>
            </c1:C1TreeViewItem.Header>
    </c1:C1TreeViewItem>
        <c1:C1TreeViewItem>
            <c1:C1TreeViewItem.Header>
                <CheckBox>
                    <CheckBox.Content>
                    <TextBlock Text="Public Downloads" />
                    </CheckBox.Content>
                </CheckBox>
            </c1:C1TreeViewItem.Header>
    </c1:C1TreeViewItem>
        <c1:C1TreeViewItem>
            <c1:C1TreeViewItem.Header>
                <CheckBox>
                    <CheckBox.Content>
                    <TextBlock Text="Public Music" />
                    </CheckBox.Content>
                </CheckBox>
            </c1:C1TreeViewItem.Header>
    </c1:C1TreeViewItem>
        <c1:C1TreeViewItem>
            <c1:C1TreeViewItem.Header>
                <CheckBox>
                    <CheckBox.Content>
                    <TextBlock Text="Public Pictures" />
                    </CheckBox.Content>
                </CheckBox>
            </c1:C1TreeViewItem.Header>
    </c1:C1TreeViewItem>
        <c1:C1TreeViewItem>
            <c1:C1TreeViewItem.Header>
                <CheckBox>
                    <CheckBox.Content>
```

```
                    <TextBlock Text="Public Videos" />
                    </CheckBox.Content>
                </CheckBox>
            </c1:C1TreeViewItem.Header>
    </c1:C1TreeViewItem>
            </c1:C1TreeViewItem>
            <c1:C1TreeViewItem IsExpanded="True">
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
                    <TextBlock Text="Computer" />
                </CheckBox.Content>
            </CheckBox>
        </c1:C1TreeViewItem.Header>
                <c1:C1TreeViewItem IsExpanded="True">
            <c1:C1TreeViewItem.Header>
                <CheckBox>
                    <CheckBox.Content>
                    <TextBlock Text="Local Disk (C:)" />
                    </CheckBox.Content>
                </CheckBox>
            </c1:C1TreeViewItem.Header>
            <c1:C1TreeViewItem>
                <c1:C1TreeViewItem.Header>
                    <CheckBox>
                        <CheckBox.Content>
                        <TextBlock Text="Program Files" />
                        </CheckBox.Content>
                    </CheckBox>
                </c1:C1TreeViewItem.Header>
        </c1:C1TreeViewItem>
            <c1:C1TreeViewItem>
                <c1:C1TreeViewItem.Header>
                    <CheckBox>
                        <CheckBox.Content>
                            <TextBlock Text="Users" />
                        </CheckBox.Content>
                    </CheckBox>
                </c1:C1TreeViewItem.Header>
        </c1:C1TreeViewItem>
            <c1:C1TreeViewItem>
                <c1:C1TreeViewItem.Header>
                    <CheckBox>
                        <CheckBox.Content>
                            <TextBlock Text="Windows" />
                        </CheckBox.Content>
                    </CheckBox>
                </c1:C1TreeViewItem.Header>
        </c1:C1TreeViewItem>
    </c1:C1TreeViewItem>
        <c1:C1TreeViewItem>
            <c1:C1TreeViewItem.Header>
                <CheckBox>
                    <CheckBox.Content>
                        <TextBlock Text="DVD Drive (D:)" />
                    </CheckBox.Content>
                </CheckBox>
```

```
            </c1:C1TreeViewItem.Header>
        </c1:C1TreeViewItem>
    </c1:C1TreeViewItem>
    <c1:C1TreeViewItem>
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
                    <TextBlock Text="Network" />
                </CheckBox.Content>
            </CheckBox>
        </c1:C1TreeViewItem.Header>
            </c1:C1TreeViewItem>
    <c1:C1TreeViewItem>
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
                    <TextBlock Text="Control Panel" />
                </CheckBox.Content>
            </CheckBox>
        </c1:C1TreeViewItem.Header>
            </c1:C1TreeViewItem>
    <c1:C1TreeViewItem>
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
                    <TextBlock Text="Recycle Bin" />
                </CheckBox.Content>
            </CheckBox>
        </c1:C1TreeViewItem.Header>
        </c1:C1TreeViewItem>
    </c1:C1TreeViewItem>
</c1:C1TreeView>
```

Enabling Drag-and-Drop

C1TreeView supports drag-and-drop behavior. For more information see the Drag-and-Drop Nodes topic. To enable drag-and-drop behavior, set the AllowDragDrop property to **True**:

- Visual Basic
```
C1TreeView.AllowDragDrop = True
```
- C#
```
C1TreeView.AllowDragDrop = true;
```

To enable visual drag-and-drop indicators you can set the DragDropArrowMarker and DragDropLineMarker properties.

Windows

Display modal and modeless child windows, wizards, dialog boxes, and message boxes in your Silverlight applications. Replace standard child windows with **ComponentOne Windows™ for Silverlight**. The **C1Window** control shows content in a floating window within the Silverlight plug-in.

Getting Started

Get started with the following topics:

- Key Features
- Quick Start
- Task-Based Help

Windows for Silverlight Key Features

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

- **Modal and Modeless Dialog Windows**

  Unlike the standard Silverlight child windows, **ComponentOne Windows for Silverlight** supports modeless dialog windows. Meaning, you can display multiple windows and still interact with the main page.

- **Define Window Objects in Separate XAML Files**

  C1Window objects are not children of any elements on the page. Because of this, they are defined in separate XAML files whose root element is a **C1Window** object.

- **Move and Resize Windows with the Mouse**

  You can easily determine whether or not the window can be moved and resized, fully customizing how the user interacts with the window in your application.

- **Determine the State of the Window**

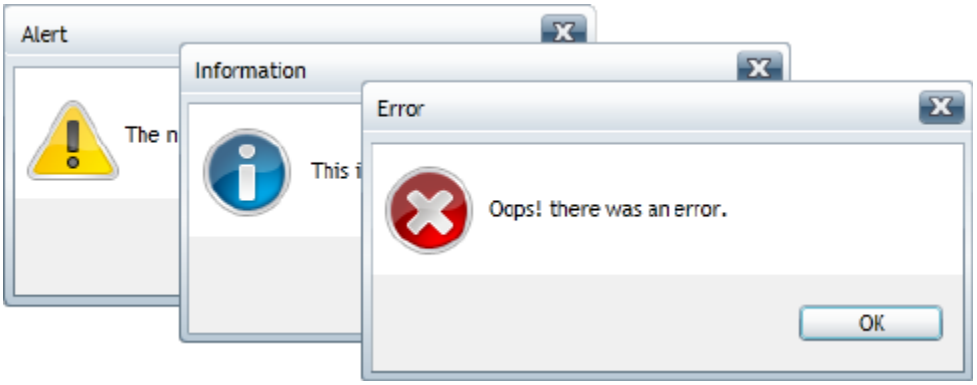
  Windows for Silverlight can be minimized and restored to its original size; the developer can set the current state of the window. The developer can also configure the window so it cannot be maximized.

- **Add Elements to the Window**

  You can edit the XAML file in Microsoft Expression Blend and add any elements you want. With **Windows for Silverlight**, creating and maintaining **C1Window** objects is very easy.

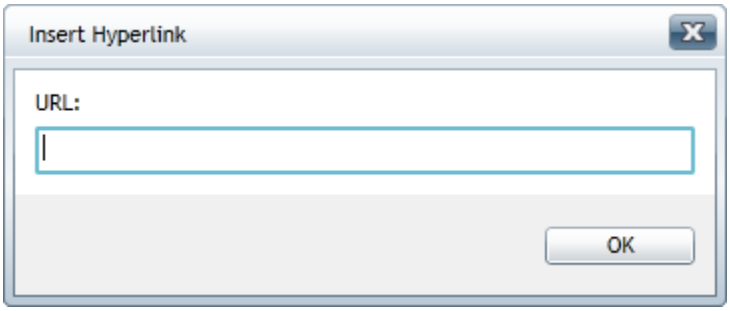
- **Message Box**

  Display standard message boxes with the **MessageBox** class. It's possible to customize the message, title, buttons, and the icon to be displayed. Plus, there are overloads available that allow you to omit most of these parameters. **C1MessageBox** includes built-in Windows Vista-style icons you can use by default.



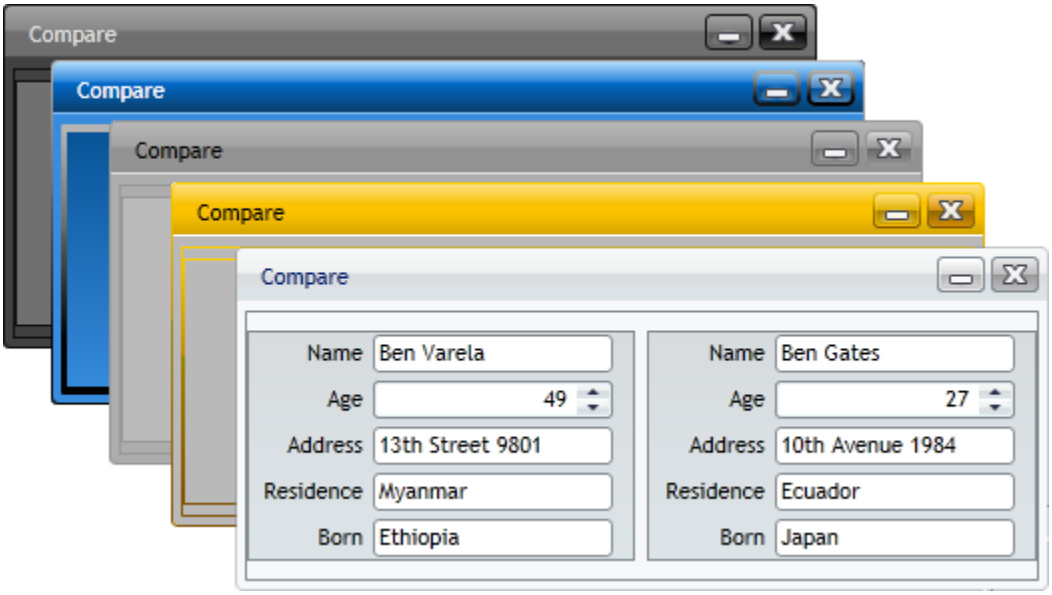
- **Prompt Dialog Box**

  Prompt the user with a message, an area for the end-user to type input, and display **OK** and **Cancel** buttons. The prompt dialog box is not limited to text: for rich formatting, supply a string of HTML.



- **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.



Windows for Silverlight Quick Start

The following quick start guide is intended to get you up and running with **Windows for Silverlight**. In this quick start you'll work in Visual Studio and create a new Silverlight project, add a C1Window control to your application, customize the appearance and behavior of the control, and observe some of the run-time interactions possible with the control.

Note that while this example uses Visual Studio, you should also similarly be able to complete this quick start in Microsoft Expression Blend.

Step 1 of 3: Creating a Silverlight Application

In this step you'll create a Silverlight application in Visual Studio using **Windows for Silverlight**. When you add a C1Window control to your application, you'll have a complete, functional dialog window. To set up your project and add C1Window control 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 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. Right-click the project in the Solution Explorer window and select **Add Reference**.
5. In the **Add Reference** dialog box, locate and select the **C1.Silverlight.dll** assembly and click **OK** to add a reference to your project.
6. In the XAML window of the project, place the cursor between the `<Grid>` and `</Grid>` tags and click once.
7. Add the following XAML markup between the `<Grid>` and `</Grid>` tags in the **MainPage.xaml** file:

```
<StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
      <TextBlock Text="Click a button to open a dialog window:"
Margin="5"/>
      <Button Click="OpenWindow" Margin="5" Width="200" Height="30"
Content="Open a modeless window." />
```

```
      <Button Click="ShowDialog" Margin="5" Width="200" Height="30"
Content="Open a modal window." />
</StackPanel>
```

This markup creates a container with two buttons. Users will be able to click on the first button to open a modeless window and on the second button to open a modal window.

What You've Accomplished

You've successfully created and set up a Silverlight application and added controls to the page. In the next step you'll add create a **C1Window** control and add code to your application to add functionality to the controls.

Step 2 of 3: Creating a C1Window Control

In the previous step you created a new Silverlight project and added button controls to the application. In this step you'll continue by adding a **C1Window** control in a new user control.

Complete the following steps:

1. Right-click the project in the Visual Studio Solution Explorer and select the **Add | New Item** option. The **Add New Item** dialog box will open.
2. In the **Add New Item** dialog box, select the **Silverlight User Control** option, name the new control "MyWindow.xaml" and click **Add** to add the new user control. If it does not open automatically, double-click the **MyWindow.xaml** file in the Solution Explorer to open it.
3. Replace the default XAML markup in the **MyWindow.xaml** file with the following markup, making sure to replace "MyProject" with the name of your project if different:

```
<UserControl x:Class="MyProject.MyWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Width="200"
Height="110">
    <Grid x:Name="LayoutRoot" Background="White">
        <TextBlock Text="Hello World!" />
    </Grid>
</UserControl>
```

   This **C1Window** control only contains a **Grid** and a **TextBlock**, but you can add any controls you choose to the dialog window.
4. In the Solution Explorer, right-click the **MainPage.xaml** file and select **View Code**.
5. In Code view, add the following import statement to the top of the page:
   - Visual Basic

```
Imports C1.Silverlight
```

   - C#

```
using C1.Silverlight;
```

6. Add the following event handlers to the file to initiate the buttons added earlier:
   - Visual Basic

```
Private Sub OpenWindow(ByVal sender As Object, ByVal e As
RoutedEventArgs)
    Dim window = New C1Window()
    window.Content = New MyWindow()
    window.CenterOnScreen()
    window.Show()
End Sub

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

```
    Dim window = New C1Window()
    window.Content = New MyWindow()
    window.CenterOnScreen()
    window.ShowModal()
End Sub
```

- *C#*

```
void OpenWindow(object sender, RoutedEventArgs e)
{
  var window = new C1Window();
  window.Content = new MyWindow();
  window.CenterOnScreen();
  window.Show();
}

void ShowDialog(object sender, RoutedEventArgs e)
{
  var window = new C1Window();
  window.Content = new MyWindow();
  window.CenterOnScreen();
  window.ShowModal();
}
```

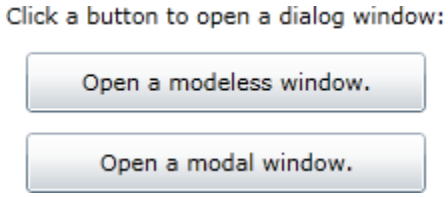
Now, when a button is clicked a dialog window will open.

You've successfully set up your application's user interface and interactions and added code to you application. Now all that's left is to run the application and observe some of the possible run-time interactions.

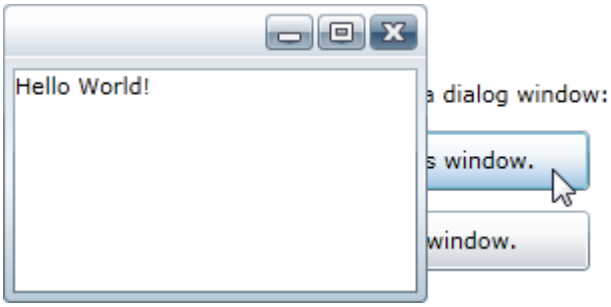
Step 3 of 3: Running the Application

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 **Windows for Silverlight**'s run-time behavior, complete the following steps:

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time. The application will appear similar to the following:

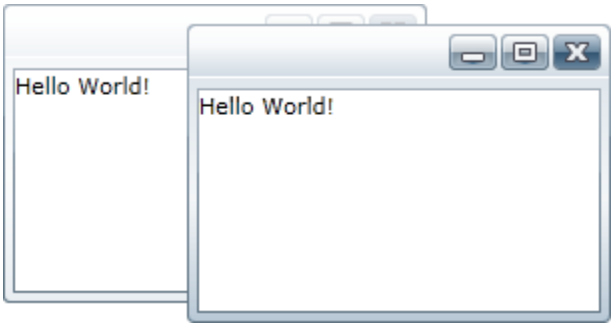


2. Click the **Open a modeless window.** button. A modeless dialog window will open:

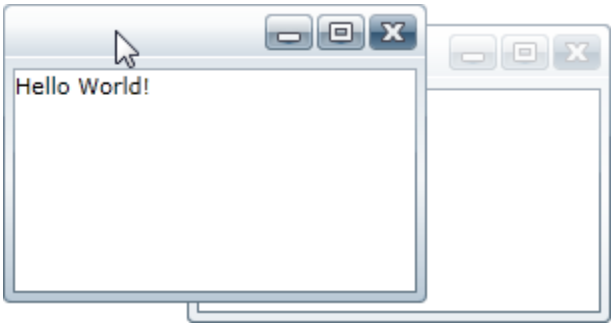


With modeless dialog windows, you can interact with other items on the page while the window is open.

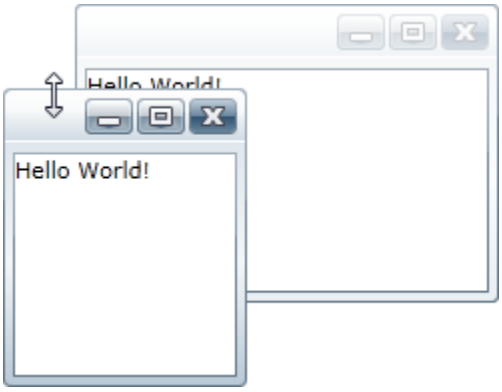
3. Click the **Open a modeless window.** button again. Another modeless dialog window will open.



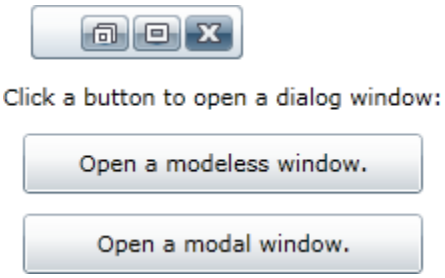
4. Click the first dialog box and notice that the focus shifts to it:



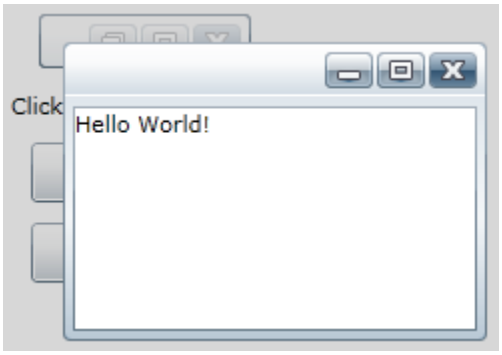
5. Click the dialog window's header and drag it to move it.
6. Click and drag a border to resize the dialog window:



7. Click the "X" button to close the dialog window.
8. Click the "_" button to minimize the second dialog window:



9. Click the **Open a modal window.** button. A modal dialog window will open:



   Notice that except for the new dialog window the entire page is grayed out. With modal dialog windows, users will not be able to interact with any other element while the dialog window is open.
10. Try to select the minimized dialog window and notice that you cannot.
11. Click on the "X" close button to close the modal dialog box. Notice that you can now interact with elements on the page again.

Congratulations! You've completed the **Windows for Silverlight** quick start and created a simple Silverlight application, added controls, including a **C1Window** control, and viewed some of the run-time capabilities of **Windows for Silverlight**.

Modal and Modeless Dialog Windows

Dialog boxes are commonly used in applications to retrieve input from the user. In some applications a dialog box is used to prompt the user for input and once the application retrieves the input the dialog box is automatically closed or destroyed.

On the other hand, some applications use dialog boxes to display information while the user works in other windows. For example, when you check spelling in Microsoft Word a dialog box remains open so you can go through and edit your text in the document while the spell checker looks for the next misspelled word. To support the different ways applications use dialog boxes, **C1Window** supports two different types of dialog windows: modal and modeless dialog windows.

Modal Dialog Windows

A modal dialog window is a child window that must be closed before the user can continue working on the current application. Typically modal dialog windows either take control of the entire system or application displaying them until they are closed. For example, you can use a modal dialog window to retrieve login information from a user before the user can continue working on an application. Modal windows are useful in presenting important information or requiring user interaction.

You can show the C1Window control as a modal dialog box using the ShowModal method.

Modeless Dialog Windows

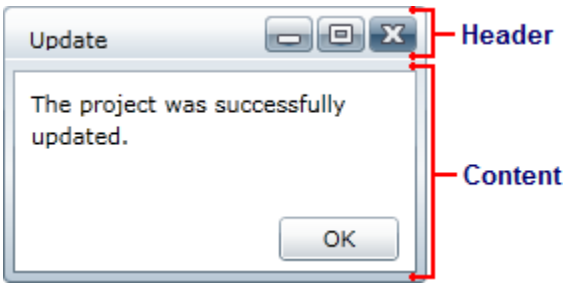
A modeless dialog window enables users to interact with other windows while the dialog window is present. Use this type of dialog window when the requested information is not necessary to continue. Modeless dialog windows do not keep the input focus so you can work on two applications at once.

A modeless dialog window is commonly used in menus and help systems where the user can use the dialog window and the application window concurrently. For example, a toolbar is a modeless dialog window because it can be detached from the application and the user can select items in the toolbar to apply features to the detached or separated application.

You can show the C1Window control as a modeless dialog box using the Show method.

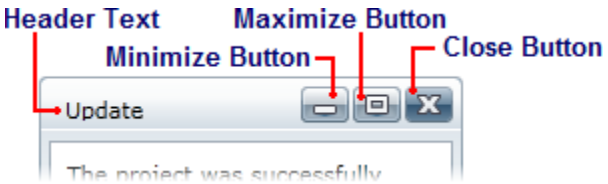
C1Window Elements

This section provides a visual and descriptive overview of the elements that comprise the C1Window control. Like a typical dialog box, the **C1Window** control includes two main elements: a header and a content area:



Header

The header area includes the typical caption bar elements including title text and **Minimize**, **Maximize**, and **Close** buttons:



You can set the text in the header with the **Header** property. You can set the visibility of the buttons using the ShowCloseButton, ShowMaximizeButton, and ShowMinimizeButton properties. By default all three properties are **True** and the buttons are visible. You can also set the **Header** property to a **UIElement**. For example, this would allow you to add an icon next to the text in the caption bar.

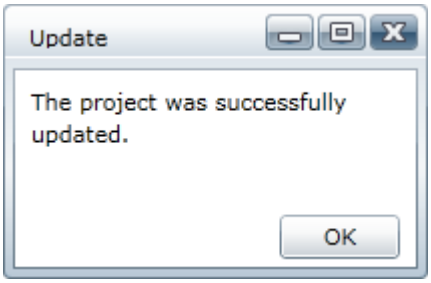
Content Area

The content area of the C1Window control can be set using the **Content** property. The content area can contain controls, a panel (such as a **Grid** or **StackPanel**) that contains controls, text, or other elements. You can, for example, set the **Content** property to a **UserControl** that contains many controls and elements.

Working with Windows for Silverlight

ComponentOne Windows for Silverlight includes the C1Window control, a simple dialog window control that shows content in a floating window within the Silverlight plug-in. When added to your application, the C1Window control appears similar to a traditional dialog box within a Silverlight application.

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



Basic Properties

ComponentOne Windows 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 Window Appearance Properties for more information about properties that control appearance.

The following properties let you customize the C1Window control:

Property	Description
Content	Gets or sets the content of a ContentControl. This is a dependency property.
ContentTemplate	Gets or sets the data template used to display the content of the ContentControl. This is a dependency property.
DialogResult	Gets or sets the dialog box result for the window.
Header	Gets or sets the header of this control.
HeaderTemplate	Gets or sets the data template used to display the header.

IsResizable	Gets or sets whether the window can be resized and maximized.
Language	Gets or sets localization/globalization language information that applies to an element.
ModalBackground	Gets or sets the brushed used on the background when showing a modal window.
ShowCloseButton	Gets or sets whether the close button of this window is shown.
ShowMaximizeButton	Gets or sets whether the maximize button of this window is shown.
ShowMinimizeButton	Gets or sets whether the minimize button of this window is shown.
WindowState	Gets or sets a value that indicates whether a window is restored, minimized, or maximized.

Basic Events

ComponentOne Windows 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 C1Window control:

Event	Description
Closed	Event fired when the window is closed by the user or the Close() method.
Closing	Event fired when the window is about to close, allows the handler to stop the window from being closed.
PositionChanged	Fires when the window position changes.
WindowStateChanged	Event raised when the WindowState property has changed

Basic Methods

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

The following methods let you customize the C1Window control:

Event	Description
BringToFront	Puts the window in front of all windows.
CenterOnScreen	Centers the window in its container.
Close	Closes the window.
Hide	Hides the window without closing it.
Position	Positions the window so that the specified point in the window matches the specified point of a FrameworkElement.
Show	Opens the window as a modeless dialog window.
ShowModal	Opens the window as a modal dialog window.

Window State

The C1Window dialog window can be floating, maximized, or minimized. You can customizing set the dialog window's state by setting the WindowState property to one of the following options:

Event	Description
Floating	The window is floating (neither maximized nor minimized).
Maximized	The window is maximized.
Minimized	The window is minimized.

Window Layout and Appearance

The following topics detail how to customize the C1Window 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.

Window Appearance Properties

ComponentOne Windows 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.
ModalBackground	Gets or sets the brushed used on the background when showing a modal window. For an example, see Setting the Modal Background Color.

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 **C1Window** 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.

Window Themes

C1Window includes Visual Styles allowing you to easily change the control's appearance. The control includes several built-in visual styles, including Vista and Office 2007 styles, to quickly style your application. You can easily customize the appearance of a single C1Window control with the themes and you can also change the theme across an application.

Window 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 Windows 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 C1Window 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.

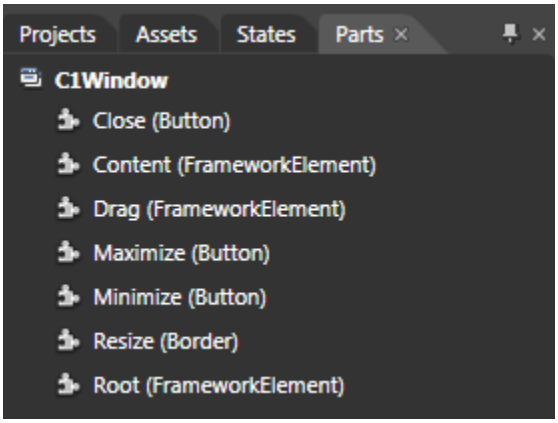
Window Styles

ComponentOne Windows for Silverlight's **C1Window** 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.
HeaderFontStyle	Gets or sets the font style of the header.
Style	Gets or sets the style used by this element when it is rendered. This is a dependency property.

Window Template Parts

In Microsoft Expression Blend, you can view and edit template parts by creating a new template (for example, click the **C1Window** 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.

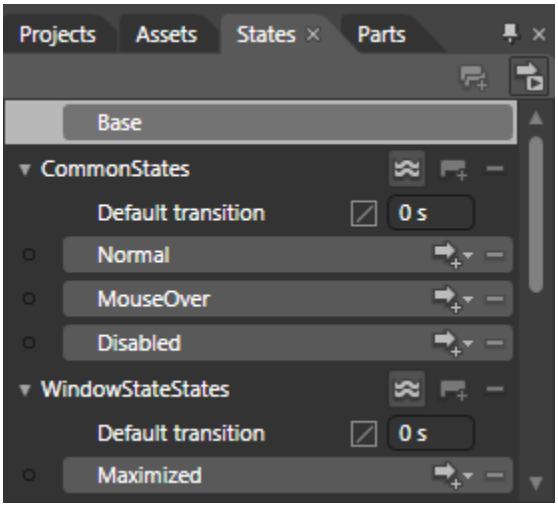
Template parts available in the **C1Window** control include:

Name	Type	Description
Close	Button	Represents a button control used to close the dialog window.
Content	FrameworkElement	This element contains the window's content.
Drag	FrameworkElement	This element listens for mouse events to drag the window.

Maximize	Button	Represents a button control used to maximize the dialog window.
Minimize	Button	Represents a button control used to minimize the dialog window.
Resize	Border	Draws a border, background, or both around another object. Here the border is used to resize the control.
Root	FrameworkElement	The root element of the template.

Window 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. 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. Window state states include **Maxmized** for when the window is maximized, **Minimized** for when the window is minimized, and **Floating** for when the window is neither maximized nor minimized.

Active states include **Active** when the window is active and in focus and **Inactive** for when the window is inactive and not in focus. Drag states include **Still** for when the window is not being dragged, and **Dragged** when the user is in the process of completing a drag-and-drop operation with the window.

Windows 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 C1Window control in general. If you are unfamiliar with the **ComponentOne Windows for Silverlight** product, please see the Windows for Silverlight Quick Start first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne Windows for Silverlight** product. Each task-based help topic assumes that you have created a new Silverlight project and most also assume that you have added a C1Window control to the project.

Setting the Title Text

You can easily customize **Windows for Silverlight** control's header area by adding a title. For more information the header area, see C1Window Elements. By default the window does not include text in the caption bar, but you can customize this at design time in Microsoft Expression Blend, in XAML, and in code by setting the **Header** property.

At Design Time in Blend

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

1. Click the C1Window control once to select it.
2. Navigate to the Properties tab and locate the **Header** item.
3. Click in the text box next to the **Header** item, and enter "Hello World!" or some other text.

   This will set the **Header** property and the text in the caption bar of the dialog window to the text you chose.

In XAML

For example, to set the **Header** property add `Header="Hello World!"` to the `<c1:C1Window>` tag so that it appears similar to the following:

```
<c1:C1Window Height="110" HorizontalAlignment="Right" Margin="0,54,71,0"
VerticalAlignment="Top" Width="220" Content="C1Window" Header="Hello
World!"/>
```

In Code

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

- Visual Basic

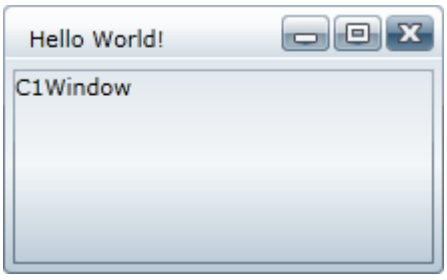
  ```
  Me.C1Window1.Header = "Hello World!"
  ```

- C#

  ```
  this.c1window1.Header = "Hello World!";
  ```

Run the application and observe:

The caption bar of the C1Window control will appear with "Hello World!" or the text you chose:



Hiding Header Buttons

You can easily customize user interaction by setting what buttons are visible on the **Windows for Silverlight** control's caption bar. For more information the header area, see C1Window Elements. By default the window displays **Minimize**, **Maximize**, and **Close** buttons, but you can customize this in Microsoft Expression Blend, in XAML, and in code.

At Design Time in Blend

To hide the **Maximize** and **Minimize** buttons in Blend, complete the following steps:

1. Click the C1Window control once to select it.

2. Navigate to the Properties window tab.
3. Locate the **ShowMaximizeButton** item and clear the check box next to the item.
4. Locate the **ShowMinimizeButton** item and clear the check box next to the item.

   This will hide the **Maximize** and **Minimize** buttons.

In XAML

For example, to remove the **Maximize** and **Minimize** add `ShowMaximizeButton="False" ShowMinimizeButton="False"` to the `<c1:C1Window>` tag so that it appears similar to the following:

```
<c1:C1Window Height="129" HorizontalAlignment="Right" Margin="0,54,71,0"
VerticalAlignment="Top" Width="220" Content="C1Window"
ShowMaximizeButton="False" ShowMinimizeButton="False"/>
```

In Code

For example, to hide the **Maximize** and **Minimize** buttons, add the following code to your project:

- Visual Basic

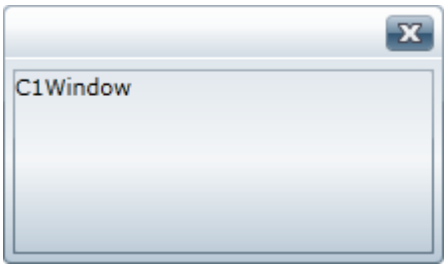
```
Me.C1Window1.ShowMaximizeButton = False
Me.C1Window1.ShowMinimizeButton = False
```

- C#

```
this.c1window1.ShowMaximizeButton = false;
this.c1window1.ShowMinimizeButton = false;
```

Run the application and observe:

The caption bar of the C1Window control will appear without the **Maximize** and **Minimize** buttons displayed:



Note that to hide the **Close** button, you can set the ShowCloseButton property.

Minimizing and Maximizing the Window

The C1Window control includes caption bar buttons that will minimize or maximize the control. However you can also customize the appearance of the control using the WindowState property. In this topic you'll add a C1Window control to the form and three buttons: one that will minimize the window, one that will maximize the window, and one that will restore a minimized or maximized window.

Complete the following steps:

1. Create a new Silverlight project in Microsoft Expression Blend.
2. Click the **Assets** button in the Toolbar (the double-chevron icon).
3. In the dialog box locate and select the **C1Window** icon.

   Note that if you cannot locate the **C1Window** icon in the dialog box, you may need to add a reference to the **C1.Silverlight** assembly first.

4. Select the page and double-click the **C1Window** icon in the Toolbar. The dialog window will be added to the page.
5. Select the **C1Window** control, navigate to the Properties window, and set the following properties:
   - Set the control's **Name** property to "window".
   - Set the control's **Height** to **150** and **Width** to **200**.
   - Set the control's **Margin** property to **5**.
   - Set the control's **Content** property to " Minimize or Maximize me!".
6. In the Toolbox, click the Panels icon (by default a Grid) a select the **StackPanel** item.
7. Double-click the **StackPanel** to add one to the application.
8. Select the **StackPanel** and in the Properties window set its **HorizontalAlignment** and **VerticalAlignment** properties to **Center**.
9. Set the **StackPanel**'s **Height** and **Width** properties to **Auto**.
10. Double-click the **Button** icon in the Toolbox three times to add three **Button** controls to the **StackPanel** below the **C1Window** control.
11. Select each of the buttons in turn and set the following properties in the Properties window:
    - Set the first button's **Name** and **Content** properties to "Minimize".
    - Set the second button's **Name** and **Content** properties to "Maximize".
    - Set the third button's **Name** and **Content** properties to "Restore".
12. Set the **Margin** property for each of the buttons to **5**.
13. Select the **Minimize** button, click the lightning bolt **Events** icon in the Properties window.
14. Double-click the space next to the **Click** event to create an event handler and switch to Code view. Return to Design view and repeat this step with the remaining buttons to create a **Button_Click** event handler for each one.
15. In Code view, add the following import statement to the top of the page:
    - Visual Basic
      ```
      Imports C1.Silverlight
      ```
    - C#
      ```
      using C1.Silverlight;
      ```
16. Add code to the **Button_Click** event handlers you created earlier. They will appear similar to the following:
    - Visual Basic
      ```
      Private Sub Minimize_Click(ByVal sender as Object, ByVal e as
      System.Windows.RoutedEventArgs)
          window1.WindowState = C1WindowState.Minimized
      End Sub

      Private Sub Maximize_Click(ByVal sender as Object, ByVal e as
      System.Windows.RoutedEventArgs)
          window1.WindowState = C1WindowState.Maximized
      End Sub

      Private Sub Restore_Click(ByVal sender as Object, ByVal e as
      System.Windows.RoutedEventArgs)
          window1.WindowState = C1WindowState.Floating
      ```

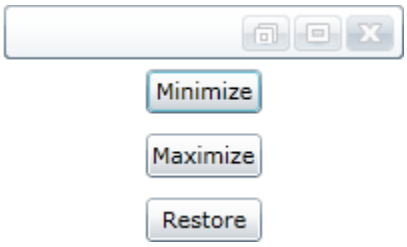
```
End Sub
```

- C#

```
private void Minimize_Click(object sender,
System.Windows.RoutedEventArgs e)
{
    this.window.WindowState = C1WindowState.Minimized;
}

private void Maximize_Click(object sender,
System.Windows.RoutedEventArgs e)
{
    this.window.WindowState = C1WindowState.Maximized;
}

private void Restore_Click(object sender,
System.Windows.RoutedEventArgs e)
{
    this.window.WindowState = C1WindowState.Floating;
}
```

Run the application and observe:

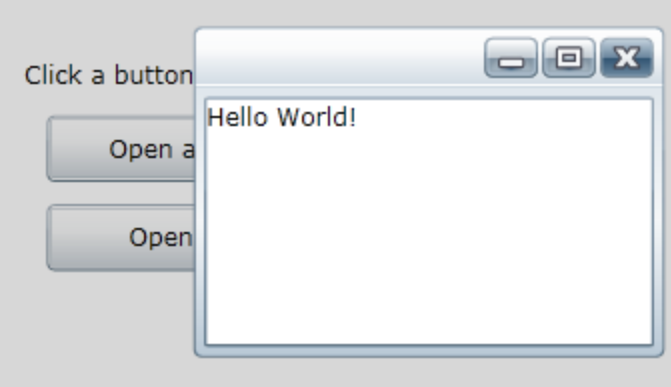
A dialog box appears with three buttons on the page. You can click the **Minimize** button to minimize the **C1Window** control:



You can maximize the **C1Window** dialog window control by clicking the **Maximize** button, and you can return the **C1Window** control to its original appearance by clicking the **Restore** button.

Setting the Modal Background Color

At run time when you open a modal dialog window, you will notice that the area behind the window is grayed out. This indicates that the dialog window must be closed before the user can interact with elements on the page. For example:



You can customize this background color by setting the ModalBackground property. For an example, complete the following steps:

1. Create an application that includes a button that opens a modal dialog window named "window". For example, complete the steps in the outlined in the Windows for Silverlight Quick Start.
2. In Code view, add the following code to the **Button_Click** event:

   - Visual Basic

     ```
     Dim bgcol As New SolidColorBrush()
     bgcol.Color = Color.FromArgb(150, 255, 0, 0)
     window.ModalBackground = bgcol
     ```

   - C#

     ```
     SolidColorBrush bgcol = new SolidColorBrush();
     bgcol.Color = Color.FromArgb(150, 255, 0, 0);
     window.ModalBackground = bgcol;
     ```

   This code will create a new red-colored brush and set the ModalBackground property to the brush. The code in the **Button_Click** event will now appear similar to the following:

   - Visual Basic

     ```
     Private Sub ShowDialog(ByVal sender As Object, ByVal e As
     RoutedEventArgs)
         Dim window = New C1Window()
         window.Content = New MyWindow()
         window.CenterOnScreen()
         Dim bgcol As New SolidColorBrush()
         bgcol.Color = Color.FromArgb(150, 255, 0, 0)
         window.ModalBackground = bgcol
         window.ShowModal()
     End Sub
     ```

   - C#

     ```
     void ShowDialog(object sender, RoutedEventArgs e)
     {
         var window = new C1Window();
         window.Content = new MyWindow();
         window.CenterOnScreen();
         SolidColorBrush bgcol = new SolidColorBrush();
         bgcol.Color = Color.FromArgb(150, 255, 0, 0);
         window.ModalBackground = bgcol;
         window.ShowModal();
     }
     ```

Run the application and observe:

Run the application and open the dialog window as a modal dialog box. You will notice that the background color behind the window appears red or the color you chose: