
ComponentOne

Expander for WPF

Copyright © 1987-2012 GrapeCity, Inc. All rights reserved.

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)

Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of GrapeCity, Inc. All other trademarks used herein are the properties of their respective owners.

Warranty

ComponentOne warrants that the original CD (or diskettes) are free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective CD (or disk) to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for a defective CD (or disk) by sending it and a check for \$25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original CD (or disks) set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. We are not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

This manual was produced using [ComponentOne Doc-To-Help™](#).

Table of Contents

ComponentOne Expander for WPF.....	1
Help with ComponentOne Studio for WPF	1
Key Features.....	3
Expander for WPF Quick Start.....	3
Step 1 of 3: Creating an Application with a C1Expander Control	3
Step 2 of 3: Adding Content to the C1Expander Control.....	4
Step 3 of 3: Running the Project	5
Using C1Expander	5
C1Expander Elements.....	6
Expander Header.....	6
Expander Content Panel	7
Expanding and Collapsing C1Expander.....	7
Initial Expand State	8
Expand Direction	8
Expandability	9
Expander for WPF Layout and Appearance.....	9
ComponentOne ClearStyle Technology	10
How ClearStyle Works.....	10
C1Expander ClearStyle Properties.....	10
Expander for WPF Appearance Properties.....	11
Text Properties	11
Content Positioning Properties.....	11
Color Properties.....	12
Border Properties.....	12
Size Properties	12
Templates.....	13
C1Expander Themes.....	13
Expander for WPF Samples	17
Expander for WPF Task-Based Help	19

Adding Content to the Header Bar	19
Adding Text to the Header Bar	19
Adding a Control to the Header Bar	20
Adding Content to the Content Panel.....	21
Adding Text to the Content Panel.....	21
Adding a Control to the Content Panel.....	22
Adding Multiple Controls to the Content Panel	23
Changing the Expand Direction	24
Changing the Initial Expand State	25
Binding Data to the Header and Content Panel Using Templates	26
Preventing Expansion	28

ComponentOne Expander for WPF

Save precious screen real estate with **ComponentOne Expander™ for WPF**. **Expander for WPF** includes one control, **C1Expander**, which allows you create an expandable and collapsible information panel that can include text, images, and controls. Choose from four expand directions and take complete control of the control's style by customizing its appearance in Microsoft Expression Blend.



Getting Started

- [Using C1Expander](#) (page 5)
- [Quick Start](#) (page 3)
- [Task-Based Help](#) (page 19)

Help with ComponentOne Studio for WPF

Getting Started

For information on installing ComponentOne Studio for WPF, licensing, technical support, namespaces and creating a project with the control, please visit [Getting Started with Studio for WPF](#).

What's New

For a list of the latest features added to **ComponentOne Studio for WPF**, visit [What's New in Studio for WPF](#).

Key Features

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

- **Expand on Page Load**

You can choose whether or not the **C1Expander** control is expanded upon page load by using the **IsExpanded** property. By default, the **IsExpanded** property is set to **True** and the control is initially appears expanded. For more information, see the [Expandability](#) (page 9) topic.

- **Expand Direction**

The **C1Expander** control has the ability to expand in four different directions. The **ExpandDirection** property indicated in which direction the control expands and can be set to **Top**, **Right**, **Bottom**, or **Left**. For more information, see the [Expand Direction](#) (page 8) topic.

- **Custom Header**

The **C1Expander** control's header bar can be customized with both text and controls. For more information on the customizable header bar , see [Expander Header](#) (page 6).

- **Configure Items in an Organized Pattern**

Expander is designed to maximize space. Configure the size and position of **C1Expander** to hide items until needed.

Expander for WPF Quick Start

The following quick start guide is intended to get you up and running with **Expander for WPF**. In this quick start, you'll start in Visual Studio to create a new project, add a **C1Expander** control to your application, and then add content to the **C1Expander** control's content panel.

Step 1 of 3: Creating an Application with a C1Expander Control

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

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 **WPF Application**. Enter a **Name** for your project and click **OK**.
3. Navigate to the Toolbox and double-click the **C1Expander** icon to add the control to the project. The XAML markup resembles the following:
4. Click the **C1Expander** control to select it.
5. In the Properties window, set the following properties:
 - Set the **Height** property to "75".
 - Set the **Width** property to "140".
 - Set the **Header** property to "Expander Quick Start".
 - Set the **Background** property to **Aqua**.
 - Set the **ExpandDirection** property to **Up**.

In this step, you customized the appearance and behavior of the C1Expander control. In the next step, you will add content to the control.

Step 2 of 3: Adding Content to the C1Expander Control

In the last step, you customized the appearance and behavior of the C1Expander control. In this step, you will add controls to the C1Expander control's content panel.

Complete the following steps:

1. Place your cursor between the `<c1ext:C1Expander>` and `</c1ext:C1Expander>` tags and press ENTER.
2. Navigate to the Toolbox and double-click the **StackPanel** icon to add a **StackPanel** control to the C1Expander control. The XAML markup will resemble the following:

```
<c1ext:C1Expander Height="200" Width="250" Name="C1Expander1"
  Background="Aqua" Header="Expander Quick Start" ExpandDirection="Up">
  <StackPanel></StackPanel>
</c1ext:C1Expander>
```

You are adding a **StackPanel** control to the C1Expander control's content panel because you'll be adding more than one control to the content panel in this quick start. The C1Expander control, as a content control, can only accept one child item at a time; however, you can get around this limitation by adding a panel-based control, which can accept multiple child items, to the C1Expander control.

3. Add `HorizontalAlignment="Center"` to the `<StackPanel>` tag so that all content added to the panel will be centered. The XAML markup will resemble the following:

```
<StackPanel HorizontalAlignment="Center">
```

4. Place your cursor between the `<StackPanel>` and `</StackPanel>` tags and press ENTER.

5. Navigate to the Toolbox and double-click the **TextBlock** icon to add the control to the **StackPanel** control. Repeat this twice so that a total of three **TextBlock** controls are added as the **StackPanel**'s content. The XAML markup will resemble the following:

```
<StackPanel HorizontalAlignment="Center" >
  <TextBlock></TextBlock>
  <TextBlock></TextBlock>
  <TextBlock></TextBlock>
</StackPanel>
```

6. Add `Text="Control 1"` to the first `<TextBlock>` tag, `Text="Control 2"` to the second `<TextBlock>` tag, and `Text="Control 3"` to the third `<TextBlock>` tag so that the XAML markup resembles the following:

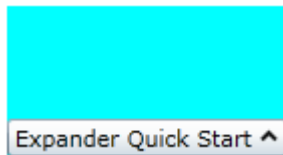
```
<TextBlock Text="Control 1"></TextBlock>
<TextBlock Text="Control 2"></TextBlock>
<TextBlock Text="Control 3"></TextBlock>
```

In this step, you added content to the C1Expander control. In the next step, you will run the project and observe the run-time features of the C1Expander control.

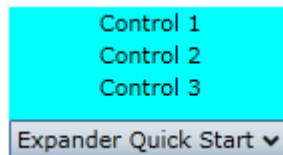
Step 3 of 3: Running the Project

In the first two steps of this quick start, you added created a WPF project containing a C1Expander control and then added content to the C1Expander control. In this step, you will run the project and observe the run-time appearance and behaviors of the C1Expander control.

1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time. Observe that the expander content isn't visible.



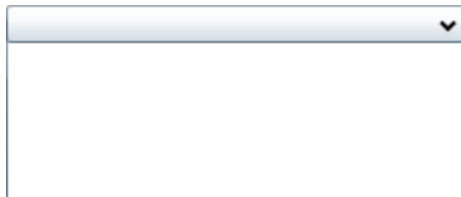
2. Click the C1Expander control's header bar to expand the content. Observe that the three **TextBox** controls that you added to the content panel appear



Congratulations! You have successfully completed the **Expander for WPF** quick start. In this quick start, you've created and customized an **Expander for WPF** application, added content to the control content panel, and observed several of the control's run-time features.

Using C1Expander

The C1Expander control is a **HeaderedContentControl** control that provides an expandable and collapsible pane for storing text, images, and controls. When you add the C1Expander control to a project, it exists as a header with a blank content panel. By default, the control's interface looks similar to the following image:



After the C1Expander control is added to your project, you can add a header and content to the control. You can also modify behaviors, such as the expandability and the direction, of the control. The following topics provide an overview of the C1Expander control's elements and features.

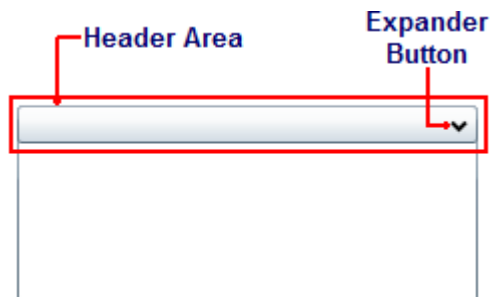
C1Expander Elements

This section provides a visual and descriptive overview of the elements that comprise the C1Expander control. The control is composed of two elements – the header bar and the content panel – which combine to make the complete C1Expander control.

Expander Header

By default, the header bar element of the C1Expander control appears at the top of the control and the expander button appears on the right side of the header. When the C1Expander control is first placed on the page, the header bar contains no text.

The following image labels the header bar of the **C1Expander** control.



To add text to the header element, simply set the **Header** property to a string. Once the text is added, you can style it using several font properties (see [Text Properties](#) (page 11)). You can also add WPF controls to the header.

The placement of the header element and expander button will change depending on the expand direction of the control. For more information on expand directions, see the [Expand Direction](#) (page 8).

Attribute Syntax versus Property Element Syntax

When you want to add something simple to the C1Expander header, such as an unformatted string, you can simply use the common XML attributes in your XAML markup, such as in the following:

```
<c1ext:C1Expander Header="Hello World" />
```

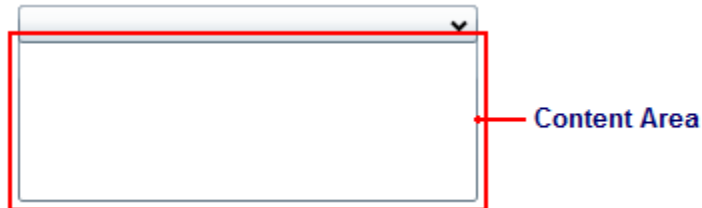
However, there may be times where you want to add more complex elements, such as grids or panels, to the content panel. In this case you would use property element syntax, such as in the following:

```
<c1ext:C1Expander ExpandDirection="Down" Width="150" Height="55"
Name="C1Expander1">
  <c1ext:C1Expander.Header>
    <Grid HorizontalAlignment="Stretch">
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="*" />
      </Grid.ColumnDefinitions>
      <TextBlock Text="C1Expander Header" />
    </Grid>
  </c1ext:C1Expander.Header>
</c1ext:C1Expander>
```

Expander Content Panel

The `C1Expander` 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 or in Visual Studio's Design view, elements in the content panel of the control can be added and moved on the control through a simple drag-and-drop operation.

The following image labels the content panel of the `C1Expander` control.



You can add text to the content panel by setting the `C1Expander` control's **Content** property or by adding a **TextBox** element to the content panel. Adding WPF elements to the content panel at run time is simple: You can either use simple drag-and-drop operations or XAML in Visual Studio or Blend. If you'd prefer to add a control at run time, you can use C# or Visual Basic code.

Content controls like `C1Expander` can only accept one child element at a time. However, you can get around this limitation by adding a panel-based control as the `C1Expander` control's child element. Panel-based controls, such as a **StackPanel** control, are able to hold multiple elements. The panel-based control meets the one control limitation of the `C1Expander` control, but its ability to hold multiple elements will allow you to show several controls in the content panel.

Attribute Syntax versus Property Element Syntax

When you want to add something simple to the `C1Expander` 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:C1Expander Content="Hello World"/>
```

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

```
<c1ext:C1Expander ExpandDirection="Down" Width="150" Height="55"
Name="C1Expander1">
    <c1ext:C1Expander.Content>
        <StackPanel>
            <TextBlock Text="Hello"/>
            <TextBlock Text="World"/>
        </StackPanel>
    </c1ext:C1Expander.Content>
</c1ext:C1Expander>
```

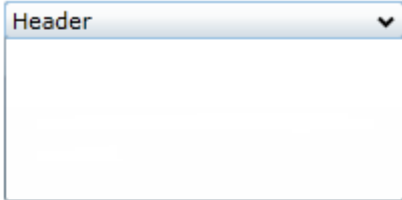
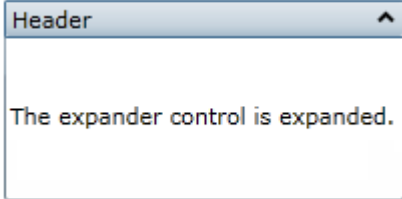
Expanding and Collapsing C1Expander

There are several options for customizing the way that the `C1Expander` control expands and collapses. The following sections describe how to set the initial expand state, set the direction to which the control expands, and how to prevent a `C1Expander` control from expanding.

Initial Expand State

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

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

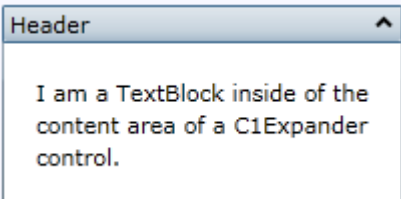
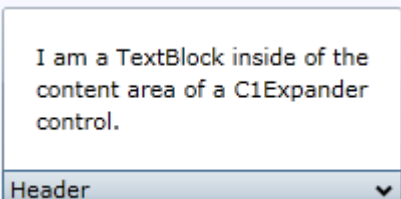
IsExpanded	Result
IsExpanded=False	
IsExpanded=True	

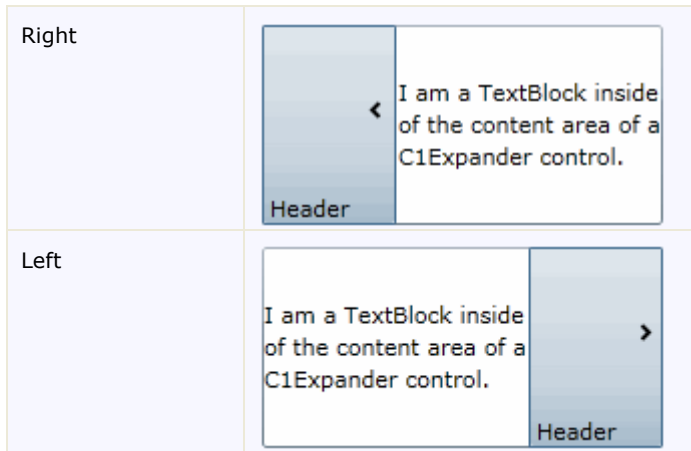
You can set the expand states in Design view, in XAML, or in code.

Expand Direction

The `C1Expander` control includes the option to specify the expand direction using the `ExpandDirection` property. In addition to setting how the direction the control expands, changing the `ExpandDirection` also changes the header's orientation to the content panel of the control. By default the `ExpandDirection` property is set to **Down** and the control expands from top to bottom.

The following table illustrates each `ExpandDirection` setting.

ExpandDirection	Result
Down	
Up	



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

Expandability

By default, the C1Expander control's `IsExpandable` property is set to **True**, meaning that the content can be expanded by clicking on the header bar. In some instances, such as when you'd like to prevent expansion until a specific event, you may not want to allow for expansion of the control; in this case, you would set the `IsExpandable` property to **False**.

Once the `IsExpandable` property is set to **False**, users can no longer interact with the control. If the user hovers over the control when the `IsExpandable` property is set to **False**, no mouse-over effect will appear; if the user clicks on the header when the `IsExpandable` property is set to **False**, the control will remain collapsed.

The table below illustrates the behavioral effect of each `IsExpandable` property setting.

<code>IsExpandable</code>	Result
True	
False	

Expander for WPF Layout and Appearance

The following topics detail how to customize the C1Expander control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the grid and take advantage of WPF's XAML-based styling. You can also use templates to format and layout the control and to customize the control's actions.

ComponentOne ClearStyle Technology

ComponentOne ClearStyle™ technology is a new, quick and easy approach to providing Silverlight and WPF control styling. ClearStyle allows you to create a custom style for a control without having to deal with the hassle of XAML templates and style resources.

Currently, to add a theme to all standard WPF controls, you must create a style resource template. In Microsoft Visual Studio, this process can be difficult; this is why Microsoft introduced Expression Blend to make the task a bit easier. Having to jump between two environments can be a bit challenging to developers who are not familiar with Blend or do not have the time to learn it. You could hire a designer, but that can complicate things when your designer and your developers are sharing XAML files.

That's where ClearStyle comes in. With ClearStyle the styling capabilities are brought to you in Visual Studio in the most intuitive manner possible. In most situations you just want to make simple styling changes to the controls in your application so this process should be simple. For example, if you just want to change the row color of your data grid this should be as simple as setting one property. You shouldn't have to create a full and complicated-looking template just to simply change a few colors.

How ClearStyle Works

Each key piece of the control's style is surfaced as a simple color property. This leads to a unique set of style properties for each control. For example, a **Gauge** has **PointerFill** and **PointerStroke** properties, whereas a **DataGrid** has **SelectedBrush** and **MouseOverBrush** for rows.

Let's say you have a control on your form that does not support ClearStyle. You can take the XAML resource created by ClearStyle and use it to help mold other controls on your form to match (such as grabbing exact colors). Or let's say you'd like to override part of a style set with ClearStyle (such as your own custom scrollbar). This is also possible because ClearStyle can be extended and you can override the style where desired.

ClearStyle is intended to be a solution to quick and easy style modification but you're still free to do it the old fashioned way with ComponentOne's controls to get the exact style needed. ClearStyle does not interfere with those less common situations where a full custom design is required.

C1Expander ClearStyle Properties

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

The following table outlines the brush properties of the **C1Expander** control:

Brush	Description
Background	Gets or sets the brush of the control's background.
ButtonForeground	Gets or sets the brush of the button's foreground.
ExpandedBackground	Gets or sets the brush of the header when the control is expanded.
FocusBrush	Gets or sets the brush of the control when the control is focused.
MouseOverBrush	Gets or sets the System.Windows.Media.Brush used to highlight the buttons when the mouse is hovered over them.
PressedBrush	Gets or sets the System.Windows.Media.Brush used to highlight the buttons when they are clicked on.

You can completely change the appearance of the **C1Expander** control by setting a few properties, such as the **HeaderBackground** property, which sets the background color of the expander's header. For example, if you set

the **HeaderBackground** property to "#FFE4005", the **C1Expander** control would appear similar to the following in its unexpanded state:



It's that simple with ComponentOne's ClearStyle technology. For more information on ClearStyle, see the [ComponentOne ClearStyle Technology](#) (page 10) topic.

Expander for WPF Appearance Properties

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

Text Properties

The following properties let you customize the appearance of text in the C1Expander control.

Property	Description
FontFamily	Gets or sets the font family of the control. This is a dependency property.
FontSize	Gets or sets the font size. This is a dependency property.
FontStretch	Gets or sets the degree to which a font is condensed or expanded on the screen. This is a dependency property.
FontStyle	Gets or sets the font style. This is a dependency property.
FontWeight	Gets or sets the weight or thickness of the specified font. This is a dependency property.
Header	Gets or sets the header of an expander control.
HeaderFontFamily	Gets or sets the font family of the header.
HeaderFontStretch	Gets or sets the font stretch of the header.
HeaderFontStyle	Gets or sets the font style of the header.
HeaderFontWeight	Gets or sets the font weight of the header.

Content Positioning Properties

The following properties let you customize the position of header and content panel content in the C1Expander control.

Property	Description
HeaderPadding	Gets or sets the padding of the header.

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

Color Properties

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

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

Border Properties

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

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

Size Properties

The following properties let you customize the size of the **C1Expander** control.

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

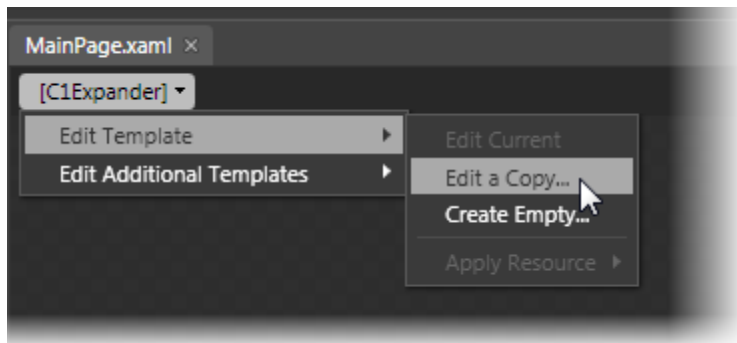
MinWidth	Gets or sets the minimum width constraint of the element. This is a dependency property.
Width	Gets or sets the width of the element. This is a dependency property.

Templates

One of the main advantages to using a WPF 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 WPF applications, you can provide your own UI for data managed by **ComponentOne Expander for WPF**. Extensible Application Markup Language (XAML; pronounced "Zammel"), an XML-based declarative language, offers a simple approach to designing your UI without having to write code.

Accessing Templates

You can access templates in Microsoft Expression Blend by selecting the C1Expander control and, in the menu, selecting **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or select **Create Empty** to create a new blank template.



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

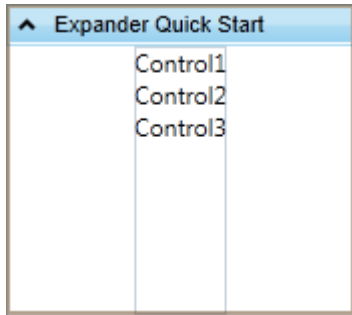
Note that you can use the [Template](#) property to customize the template.

Additional Templates

In addition templates, the C1Expander control includes a few additional templates. You can access these additional templates in Microsoft Expression Blend – in Blend select the C1Expander control and, in the menu, select **Edit Additional Templates**. Choose a template and select **Create Empty**.

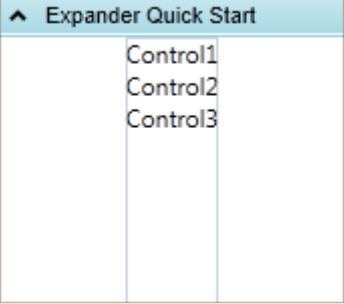
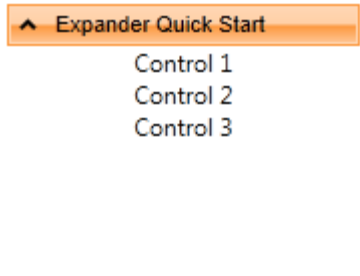
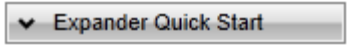
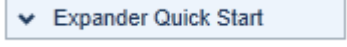
C1Expander Themes

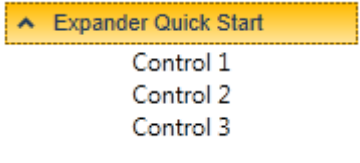
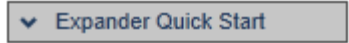
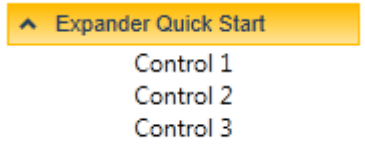
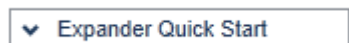
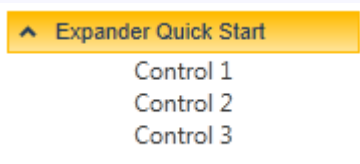
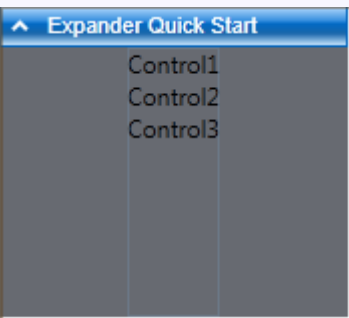
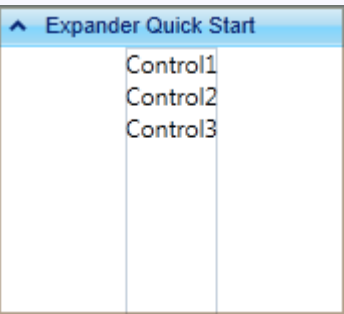
ComponentOne Expander for WPF incorporates several themes that allow you to customize the appearance of your grid. When you first add a C1Expander control to the page, it appears similar to the following image:



This is the control's default appearance. You can change this appearance by using one of the built-in themes or by creating your own custom theme. All of the built-in themes are based on WPF Toolkit themes. The built-in themes are described and pictured below; note that in the images below, a row has been selected to show selected styles:

Theme Name	Theme Preview
C1ThemeBureauBlack	
C1ThemeExpressionDark	
C1ThemeExpressionLight	

C1Blue	 <p>^ Expander Quick Start</p> <p>Control1 Control2 Control3</p>
C1ThemeOffice2007Blue	 <p>▼ Expander Quick Start</p> <p><i>Design Time</i></p>  <p>^ Expander Quick Start</p> <p>Control 1 Control 2 Control 3</p> <p><i>Run Time</i></p>
C1ThemeOffice2007Black	 <p>▼ Expander Quick Start</p> <p><i>Design Time</i></p>  <p>^ Expander Quick Start</p> <p>Control 1 Control 2 Control 3</p> <p><i>Run Time</i></p>
C1ThemeOffice2007Silver	 <p>▼ Expander Quick Start</p> <p><i>Design Time</i></p>  <p>^ Expander Quick Start</p> <p>Control 1 Control 2 Control 3</p> <p><i>Run Time</i></p>
C1ThemeOffice2010Blue	 <p>▼ Expander Quick Start</p> <p><i>Design Time</i></p>

	 <p>Control 1 Control 2 Control 3</p> <p><i>Run Time</i></p>
C1ThemeOffice2010Black	 <p><i>Design Time</i></p>  <p>Control 1 Control 2 Control 3</p> <p><i>Run Time</i></p>
C1ThemeOffice2010Silver	 <p><i>Design Time</i></p>  <p>Control 1 Control 2 Control 3</p> <p><i>Run Time</i></p>
C1ThemeShinyBlue	 <p>Control1 Control2 Control3</p>
C1ThemeWhistlerBlue	 <p>Control1 Control2 Control3</p>

To set an element's theme, use the **ApplyTheme** method. First add a reference to the theme assembly to your project, and then set the theme in code, like this:

- Visual Basic

```
Private Sub Window_Loaded(sender As System.Object, e As
System.Windows.RoutedEventArgs) Handles MyBase.Loaded
    Dim theme As New C1ThemeExpressionDark

    ' Using ApplyTheme
    C1Theme.ApplyTheme(LayoutRoot, theme)
```

- C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    C1ThemeExpressionDark theme = new C1ThemeExpressionDark();

    //Using ApplyTheme
    C1Theme.ApplyTheme(LayoutRoot, theme);
}
```

To apply a theme to the entire application, use the **System.Windows.ResourceDictionary.MergedDictionaries** property. First add a reference to the theme assembly to your project, and then set the theme in code, like this:

- Visual Basic

```
Private Sub Window_Loaded(sender As System.Object, e As
System.Windows.RoutedEventArgs) Handles MyBase.Loaded
    Dim theme As New C1ThemeExpressionDark

    ' Using Merged Dictionaries
    Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThem
eResources(theme))
End Sub
```

- C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
    C1ThemeExpressionDark theme = new C1ThemeExpressionDark();

    //Using Merged Dictionaries
    Application.Current.Resources.MergedDictionaries.Add(C1Theme.GetCurrentThem
eResources(theme));
}
```

Note that this method works only when you apply a theme for the first time. If you want to switch to another **ComponentOne** theme, first remove the previous theme from **Application.Current.Resources.MergedDictionaries**.

Expander for WPF Samples

Please be advised that these **ComponentOne** software tools are accompanied by various sample projects and/or demos which may make use of other development tools included with the **ComponentOne Studios**.

Samples can be accessed from the **ComponentOne Control Explorer**. To view samples, on your desktop, click the **Start** button and then click **ComponentOne | Studio for WPF | Samples | WPF ControlExplorer**.

The following pages within the **ControlExplorer** detail the **C1Expander** control:

Sample	Description
Expander	Illustrates the functionality of the C1Expander control.

Expander for WPF Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and know how to use the C1Expander control in general. If you are unfamiliar with the **ComponentOne Expander for WPF** product, please see the **Expander for WPF** Quick Start first.

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

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

Adding Content to the Header Bar

You can easily add both simple text and WPF controls to the C1Expander control's header bar. The topics in this section will provide step-by-step instructions about adding text content and controls to the header bar.

For more information on the header element, you can also visit the [Expander Header](#) (page 6) topic.

Adding Text to the Header Bar

By default, the C1Expander control's header bar is empty. You can add text to the control's header bar by setting the **Header** property to a string in the **Properties** window, in XAML, or in code.

At Design Time

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

1. Click the C1Expander control once to select it.
2. In the **Properties** window, 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:C1Expander>` tag so that it appears similar to the following:

```
<c1ext:C1Expander Name="C1Expander" Header="Hello World" Width="150" Height="55">
```

In Code

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

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

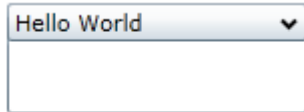
```
C1Expander1.Header = "Hello World"
```
 - C#

```
c1Expander1.Header = "Hello World";
```
2. Run the program.



This Topic Illustrates the Following:

The header bar of the C1Expander control now reads "Hello World". The end result of this topic should resemble the following:



Adding a Control to the Header Bar

The C1Expander control's header bar is able to accept a WPF control. In this topic, you will add a **Button** control to the header bar in XAML and in code.

In XAML

To add a **Button** control to the header bar in XAML, place the following XAML markup between the `<c1ext:C1Expander>` and `</c1ext:C1Expander>` tags:

```
<c1ext:C1Expander.Header>
    <Button Content="Button" Height="Auto" Width="50"/>
</c1ext:C1Expander.Header>
```

In Code

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

1. Enter Code view and add the following code beneath the **InitializeComponent()** method:

- Visual Basic

```
'Create the Button control
Dim NewButton As New Button()
NewButton.Content = "Button"
'Set the Button Control's Width and Height properties
NewButton.Width = 50
NewButton.Height = Double.NaN
'Add the Button to the header
C1Expander1.Header = (NewButton)
```

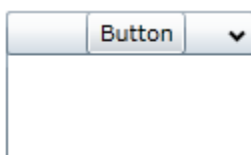
- C#

```
//Create the Button control
Button NewButton = new Button();
NewButton.Content = "Button";
//Set the Button Control's Width and Height properties
NewButton.Width = 50;
NewButton.Height = Double.NaN;
//Add the Button to the header
c1Expander1.Header = (NewButton);
```

2. Run the program.

✔ This Topic Illustrates the Following:

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



Adding Content to the Content Panel

You can easily add both simple text and WPF controls to the C1Expander control's content panel. The topics in this section will provide step-by-step instructions about how to add text content and controls to the header.

For more information on the header bar, you can also visit the [Expander Content Panel](#) (page 7) topic.

Adding Text to the Content Panel

You can easily add a simple line of text to the content panel of the C1Expander control by setting the **Content** property to a string in the **Properties** window, 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](#) (page 22).

At Design Time

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

1. Click the C1Expander control once to select it.
2. In the **Properties** window, set the **Content** property to a string (for example, "Hello World").
3. Run the program and then expand the C1Expander control.

In XAML

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

1. Add `Content="Hello World"` to the `<c1ext:C1Expander>` tag so that it appears similar to the following:

```
<c1ext:C1Expander Name="C1Expander1" Content="Hello World" Width="150" Height="55">
```

2. Run the program and then expand the C1Expander control.

In Code

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

1. Enter Code view and add the following code beneath the **InitializeComponent()** method:

- Visual Basic

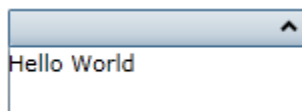
```
C1Expander1.Content = "Hello World"
```
- C#

```
c1Expander1.Content = "Hello World";
```

2. Run the program and then expand the C1Expander control.

✔ This Topic Illustrates the Following:

When the C1Expander control is expanded, it reads "Hello World". The end result of this topic should resemble the following:



Adding a Control to the Content Panel

The C1Expander control will accept one child control in its content panel. In this topic, you will add learn how to add a WPF button control in the **Properties** window, in XAML, and in code.

At Design Time

To add a control to the content panel, complete the following steps:

1. Click the C1Expander control once to select it.
2. In the Toolbox, double-click the **Button** icon to add a **Button** control the content panel of the C1Expander control.
3. Run the program and then expand the C1Expander control.

In XAML

To add a button control to the C1Expander control's content panel in XAML, complete the following steps:

1. Place the following markup between the `<c1ext:C1Expander>` and `</c1ext:C1Expander>` tags:

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

2. Run the program and then expand the C1Expander control.

In Code

To add a button control to the C1Expander control's content panel in code, complete the following:

1. 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
C1Expander1.Content = (NewButton)
```

- C#

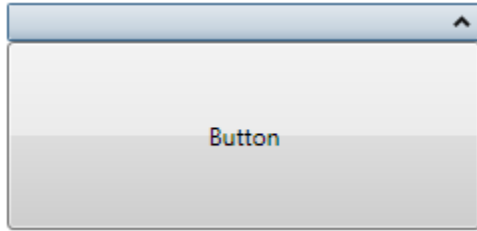
```
//Create the Button control
Button NewButton = new Button();
NewButton.Content = "Button";
//Set the Button Control's Width and Height properties
NewButton.Width = double.NaN;
NewButton.Height = double.NaN;
//Add the Button to the content panel
c1Expander1.Content = (NewButton);
```

2. Run the program and then expand the C1Expander control.



This Topic Illustrates the Following:

When the C1Expander control is expanded, the button control appears in its content panel and resembles the following image:



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 **C1Expander** control. When you add multiple controls to the panel-based control, each one will appear within the **C1Expander** control's content panel.

At Design Time

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

1. In the Design pane, click the **C1Expander** control once to select it.
2. In the Toolbox, double-click the **StackPanel** icon to add a **StackPanel** control to the content panel of the **C1Expander** control.
3. In the Source pane, click within the `<StackPanel>` tag to bring focus to the **StackPanel** control.
4. In the Toolbox, double-click the **TextBlock** icon to add a **TextBlock** control to the **StackPanel**. Repeat this step twice to add two more **TextBlock** controls to the **StackPanel**.
5. Click within the first `<TextBlock>` tag to reveal its properties in the **Properties** window and then set its **Text** property to "1st TextBlock".
6. Click within the second `<TextBlock>` tag to reveal its properties in the **Properties** window and then set its **Text** property to "2nd TextBlock".
7. Click within the third `<TextBlock>` tag to reveal its properties in the **Properties** window and then set its **Text** property to "3rd TextBlock".
8. Expand the **C1Expander** 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:C1Expander>` and `</c1ext:C1Expander>` tags:

```
<c1ext:C1Expander.Content>
  <StackPanel>
    <TextBlock Text="1st TextBlock"/>
    <TextBlock Text="2nd TextBlock"/>
    <TextBlock Text="3rd TextBlock"/>
  </StackPanel>
</c1ext:C1Expander.Content>
```

2. Run the program.
3. Expand the **C1Expander** 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 C1Expander
Dim StackPanel1 As New StackPanel()
C1Expander1.Content = (StackPanel1)
'Create three TextBlock control and set their Text properties
Dim TextBlock1 As New TextBlock()
Dim TextBlock2 As New TextBlock()
Dim TextBlock3 As New TextBlock()
TextBlock1.Text = "1st TextBlock"
TextBlock2.Text = "2nd TextBlock"
TextBlock3.Text = "3rd TextBlock"
'Add TextBlock controls to StackPanel
StackPanel1.Children.Add(TextBlock1)
StackPanel1.Children.Add(TextBlock2)
StackPanel1.Children.Add(TextBlock3)
```

- C#

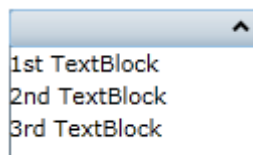
```
//Create a stack panel and add it to C1Expander
StackPanel StackPanel1 = new StackPanel();
c1Expander1.Content = (StackPanel1);
//Create three TextBlock control and set their Text properties
TextBlock TextBlock1 = new TextBlock();
TextBlock TextBlock2 = new TextBlock();
TextBlock TextBlock3 = new TextBlock();
TextBlock1.Text = "1st TextBlock";
TextBlock2.Text = "2nd TextBlock";
TextBlock3.Text = "3rd TextBlock";
//Add TextBlock controls to StackPanel
StackPanel1.Children.Add(TextBlock1);
StackPanel1.Children.Add(TextBlock2);
StackPanel1.Children.Add(TextBlock3);
```

2. Run the program.

3. Expand the C1Expander control and observe that each of the three **TextBlock** controls appear in the content panel.

✔ **This Topic Illustrates the Following:**

When the C1Expander control is expanded, three **TextBlock** controls will appear in the content panel as follows:



Changing the Expand Direction

By default, the C1Expander control expands from top-to-bottom because its **ExpandDirection** property is set to **Down**. You can easily change the expand direction by setting the **ExpandDirection** property to **Up**, **Right**, or **Left** in the **Properties** window, in XAML, or in code.

At Design Time

To set the `ExpandDirection` property, complete the following steps:

1. Click the `C1Expander` control once to select it.
2. In the **Properties** window, click the `ExpandDirection` drop-down arrow and select one of the options from the list. For this example, select **Right**.

In XAML

To set the `ExpandDirection` property to `Right` in XAML, add `ExpandDirection="Right"` to the `<c1ext:C1Expander>` tag so that it appears similar to the following:

```
<c1ext:C1Expander ExpandDirection="Right" Name="C1Expander1"
Width="150" Height="55">
```

In Code

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

1. Enter Code view and add the following code beneath the **InitializeComponent()** method:

- Visual Basic

```
C1Expander1.ExpandDirection = C1.WPF.Extended.ExpandDirection.Right
```

- C#

```
c1Expander1.ExpandDirection = C1.WPF.Extended.ExpandDirection.Right;
```

2. Run the program.



This Topic Illustrates the Following:

By following the instructions in this topic, you have learned how to set the `ExpandDirection` property. In this topic, you set the `ExpandDirection` property to **Right**, which will make the `C1Expander` control resemble the following:



Changing the Initial Expand State

By default, the `C1Expander` control's `IsExpanded` property is set to **False**, meaning that the control will appear in its collapsed state upon page load. If you'd prefer that the `C1Expander` control be expanded upon page load, you can set the `IsExpanded` property to **True** in the **Properties** window, in XAML, or in code.

At Design Time

To set the `IsExpanded` property to **True**, complete the following steps:

1. Click the `C1Expander` control once to select it.
2. In the **Properties** window, locate the `IsExpanded` check box and then select it.

In XAML

To set the `IsExpanded` property to **True** in XAML, add `IsExpanded="True"` to the `<c1ext:C1Expander>` tag so that it appears similar to the following:

```
<c1ext:C1Expander IsExpanded="True" Name="C1Expander1" Width="150"
Height="55">
```

In Code

To set the `IsExpanded` property to **True** in code, complete the following steps:

1. Enter Code view and add the following code beneath the **InitializeComponent()** method:

- Visual Basic

```
C1Expander1.IsExpanded = True
```

- C#

```
c1Expander1.IsExpanded = true;
```

2. Run the program.

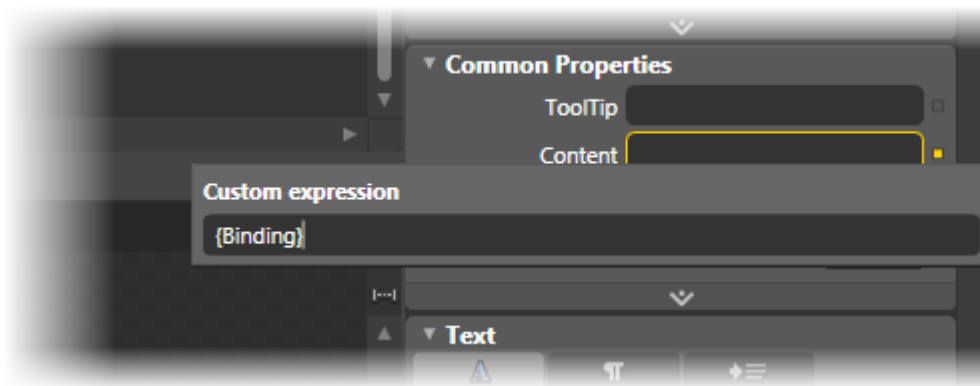
Binding Data to the Header and Content Panel Using Templates

In this topic, you will learn how to bind data to the `C1Expander` control's heading and content panel using the **ContentTemplate** template and **HeaderTemplate** template. This topic assumes that you are working in Microsoft Expression Blend.

Step 1: Add the `C1Expander` Control to the Project and Prepare it for Data Binding

Complete the following steps:

1. Add a `C1Expander` control to your WPF project.
2. Select the `C1Expander` control to expose its properties in the **Properties** tab and complete the following:
3. Set the **Name** property to "NameAgeHolder1".
4. Next to the **Content** property, click the **Advanced options** button and select **Custom Expression**. Set the **Custom expression** field to "{Binding}". This sets up the **Content** property to pass the `DataContext` directly to its template, which you will create in a later step.



5. Next to the **Header** property, click the **Advanced options** button and select **Custom Expression**. Set the **Custom expression** field to "{Binding}". This sets up the **Header** to pass the `DataContext` directly to its template, which you will create in a later step.

Step 2: Create Templates, Add a Control to Each Template, and Bind Each Control to a Data Source Property

Complete the following steps:

1. Create the first template, the **HeaderTemplate**, by completing the following steps:
 - a. Click the **NameAgeHolder1** breadcrumb and select **Edit Additional Templates | Edit HeaderTemplate | Create Empty**.

The **Create DataTemplate Resource** dialog box opens.

- b. In the **Name (Key)** field, enter "NameTemplate".
 - c. Click **OK** to close the **Create DataTemplate Resource** dialog box to create the new template.
 - d. Click the **Assets** tab and then, in the search field, enter "Label" to find the **Label** control.
 - e. Double-click the **Label** icon to add the **Label** control to your template.
 - f. Select the **Label** control to expose its properties in the **Properties** tab.
 - g. Next to the **Content** property, click the **Advanced options** button and select **Custom Expression**. Set the **Custom expression** field to "{Binding Name}". This sets the **Label** control's **Content** property to the value of the **Name** property, which is a property you'll create in code later.
2. Create the second template, the **ContentTemplate**, by completing the following steps:
- a. Click the **NameAgeHolder1** breadcrumb and select **Edit Additional Templates | Edit Generated Content (ContentTemplate) | Create Empty**.
The **Create DataTemplate Resource** dialog box opens.
 - b. In the **Name (Key)** field, enter "AgeTemplate".
 - c. Click **OK** to close the **Create DataTemplate Resource** dialog box to create the new template.
 - d. Click the **Assets** tab and then, in the search field, enter "Label" to find the **Label** control.
 - e. Double click the **Label** icon to add the **Label** control to your template.
 - f. Select the **Label** control to expose its properties in the **Properties** tab.
 - g. Next to the **Content** property, click the **Advanced options** button and select **Custom Expression**. Set the **Custom expression** field to "{Binding Age}". This sets the **Label** control's **Content** property to the value of the **Age** property, which is a property you'll create in code later.

Step 3: Create the Data Source

Complete the following steps:

1. Open the **MainPage.xaml** code page (this will be either **MainPage.xaml.cs** or **MainPage.xaml.vb** depending on which language you've chosen for your project).
2. Add the following class to your project, placing it beneath the namespace declaration:
 - Visual Basic

```
Public Class NameAndAge
    Public Sub New(name As String, age As Integer)
        Name = name
        Age = age
    End Sub
    Public Property Name() As String
        Get
        End Get
        Set
        End Set
    End Property
    Public Property Age() As Integer
        Get
        End Get
        Set
        End Set
    End Property
End Class
```

- C#

```
public class NameAndAge
{
    public NameAndAge(string name, int age)
    {
        Name = name;
        Age = age;
    }

    public string Name { get; set; }
    public int Age { get; set; }
}
```

This class creates a class with two properties: a string property named **Name** and a numeric property named **Age**.

3. Add the following code beneath the **InitializeComponent()** method to set the **Name** property and the **Age** property:

- Visual Basic

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

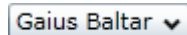
- C#

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

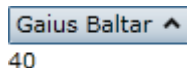
Step 4: Run the Project and Observe the Results

Complete the following steps:

1. Press F5 to run the project and observe that the value of the **Name** property appears on the control's header bar.



2. Click the header bar to expand the control and observe that the value of the **Age** property appears in the content panel:



Preventing Expansion

You can prevent a C1Expander control from being expanded by setting the **IsExpandable** property to **False** in the **Properties** window, in XAML, or in code.

At Design Time

To set the **IsExpandable** property to **False**, complete the following steps:

1. Click the C1Expander control once to select it.
2. In the **Properties** window, locate the **IsExpandable** check box and then deselect it.

In XAML

To set the **IsExpandable** property to **False** in XAML, add **IsExpandable="False"** to the `<cl:ext:C1Expander>` tag so that it appears similar to the following:


```
<clext:C1Expander IsExpandable="False" Width="150" Height="55">
```

In Code

To set the `IsExpanded` property to **True** in code, complete the following steps:

1. Enter Code view and add the following code beneath the **InitializeComponent()** method:

- Visual Basic

```
C1Expander1.IsExpandable = False
```

- C#

```
c1Expander1.IsExpandable = false;
```

2. Run the program.