ComponentOne

DropDown for WPF

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 DropDown for WPF Overview

Implement single item selection that's simple to setup and powerful to use with **ComponentOne DropDown™ for WPF**. **DropDown for WPF** provides a simple drop-down box control (like a **ComboBox** or a **DateTimePicker**). The **DropDown for WPF** controls have 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.

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

Getting Started

Get started with the following topics:

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

- **Three Drop-down Controls**

  DropDown for WPF includes three drop-down controls to give you flexibility in your applications. 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.

- **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 WPF Quick Start

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

You will create a simple project using a C1DropDown control. You'll create a WPF application, add a C1DropDown control to the application, customize and add content to the control, and observe some of the run-time interaction possible with **DropDown for WPF**.

Step 1 of 3: Creating the C1DropDown Application

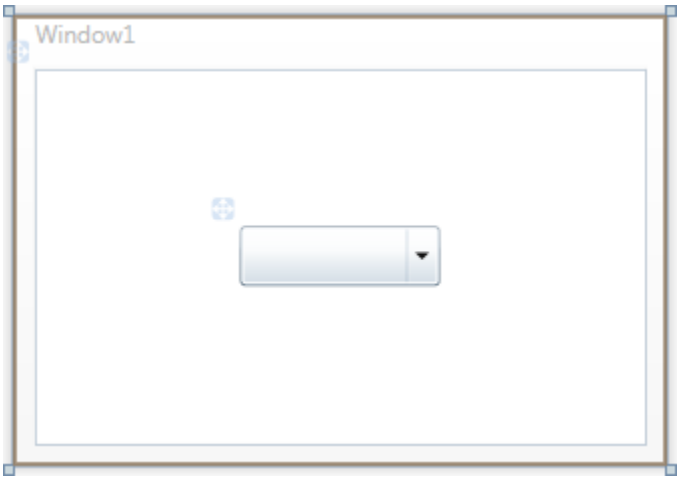
In this step you'll create a WPF application using **DropDown for WPF**. When you add a C1DropDown control to your application, you'll have a complete, functional drop-down box-like interface that you can add images, controls, and other elements to. To set up your project and add a C1DropDown control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

1. Create a new WPF project in Visual Studio.
2. In the Solution Explorer, right-click the **References** item and choose **Add Reference**. Select the **C1.WPF** and **WPFToolkit** assemblies and click **OK** to add references to your project.
3. Navigate to the Visual Studio Toolbox, and double-click the **C1DropDown** icon to add the control to the window.
4. Resize the Window and reposition the C1DropDown control in the Window.
5. Navigate to the Properties window and set the C1DropDown control's **Height** to **30** and **Width** to **100**.

   The XAML will appear similar to the following:

   ```
   <c1:C1DropDown Name="C1DropDown1" Height="30" Width="100" />
   ```

   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 WPF 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. Switch to XAML view. In the next steps you'll add XAML markup to your application to add content to the drop-down box.
2. Add markup to the C1DropDown control so that it appears similar to the following:

```
<c1:C1DropDown Name="C1DropDown1" Height="30" Width="100">
    <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.

3. Add the following markup just after the `</c1:C1DropDown.Header>` tag in the XAML markup you just added:

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

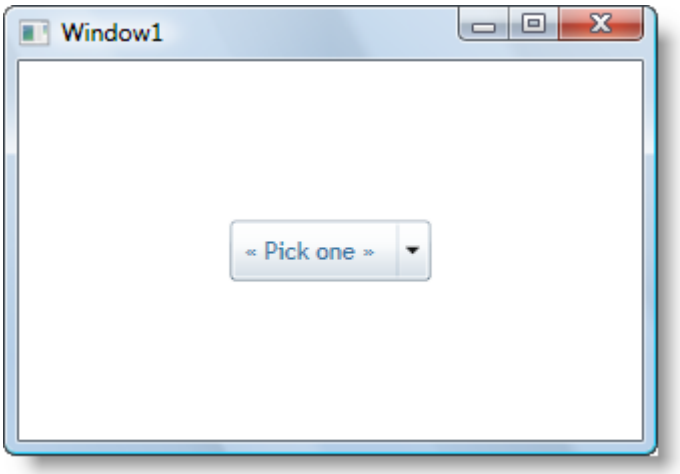
In this step you added content to the C1DropDown control. In the next step you'll further customize the control and run the application to observe run-time interactions.

Step 3 of 3: Running the C1DropDown Application

Now that you've created a WPF 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 WPF**'s run-time behavior, and then further customize the control, complete the following steps:

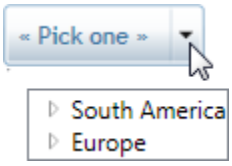
1. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time.

   The application will appear similar to the following:

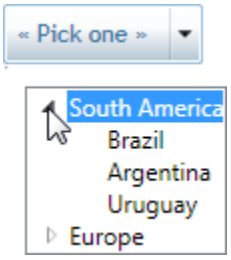


   The C1DropDown control appears as a simple drop-down box. Notice the text that you specified appears in the header of the control.

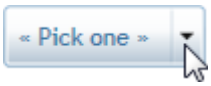
2. Click the C1DropDown control's drop-down arrow. Notice that the **TreeView** control is now visible:



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 WPF** quick start and created a simple WPF application, added and customized a **DropDown for WPF** control, and viewed some of the run-time capabilities of the control.

Working with DropDown for WPF

ComponentOne DropDown for WPF 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 functional input control that can be customized and include added content.

Basic Properties

ComponentOne DropDown for WPF 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 WPF Appearance Properties (page 11) 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 **ToggleButton** is shown.

Basic Events

ComponentOne DropDown for WPF 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.

DropDown 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. The image below identifies the header and content area:

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 Height="36" HorizontalAlignment="Left" Margin="10,12,0,0"
Name="C1DropDown1" VerticalAlignment="Top" Width="101">
    <c1:C1DropDown.Header>
        <TextBlock Height="21" Name="TextBlock1" Text="« Pick one »"
Width="120" />
    </c1:C1DropDown.Header>
    <c1:C1DropDown.Content>
        <TreeView HorizontalAlignment="Left" Name="TreeView1" Width="120">
            <TreeViewItem Header="South America">
                <TreeViewItem Header="Brazil" />
                <TreeViewItem Header="Argentina" />
                <TreeViewItem Header="Uruguay" />
            </TreeViewItem>
            <TreeViewItem Header="Europe">
                <TreeViewItem Header="Italy" />
                <TreeViewItem Header="France" />
                <TreeViewItem Header="England" />
                <TreeViewItem Header="Germany" />
            </TreeViewItem>
        </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 Direction (page 9) 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 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 (page 20).

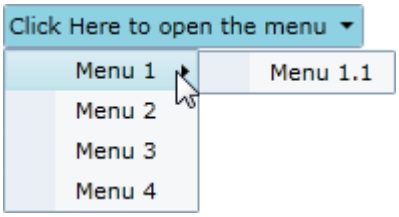
Additional Controls

In addition to the full-featured C1DropDown control, **DropDown for WPF** 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>
```

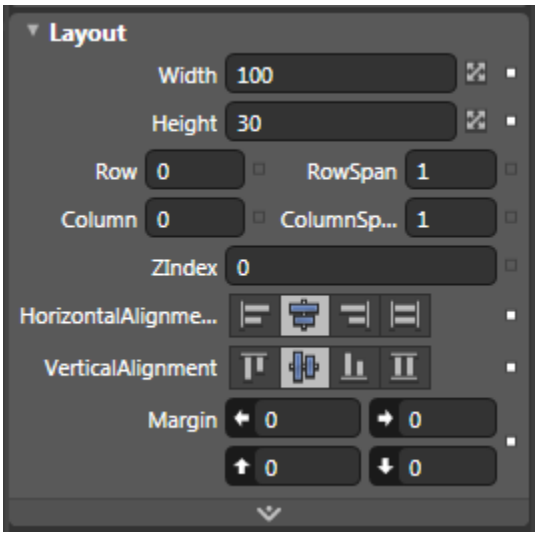
Note that the `<c1:C1DropDown.Header>` defines the header element and the content element is contained within the `<c1:C1SplitButton></c1:C1SplitButton>` tags.

DropDown for WPF 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 grid and take advantage of WPF's XAML-based styling. You can also use templates to format and lay out the control and to customize the control's actions.

Layout in a Panel

You can easily lay out the C1DropDown and other controls in your WPF application, using the attached layout properties. For example, you can lay out your control in a **Grid** panel with its **Row**, **ColumnSpan**, and **RowSpan** properties and in a **Canvas** panel with its **Left** and **Top** properties. For example, the C1DropDown control includes the following **Layout** properties when located within a **Grid** panel:



You can change the sizing, alignment, and location of the C1DropDown control within the **Grid** panel.

DropDown for WPF Appearance Properties

ComponentOne DropDown for WPF 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.

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.

ClearStyle Properties

The following table lists all of the ClearStyle-supported properties in the C1DropDown control as well as a description of the property:

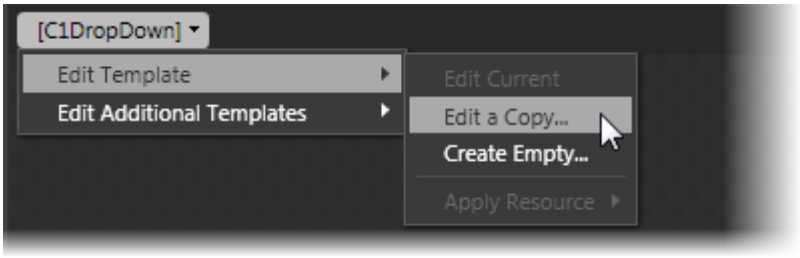
Property	Description
Background	Gets or sets a brush that describes the background of a control. The default **Background** color is White.
FocusBrush	A brush used to define the appearance of the control, when the control is in focus.
MouseOverBrush	A brush used to define the appearance of the control, when the control is in moused over.
PressedBrush	A brush used to define the appearance of the control, when the control is selected.

DropDown 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 DropDown 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 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.

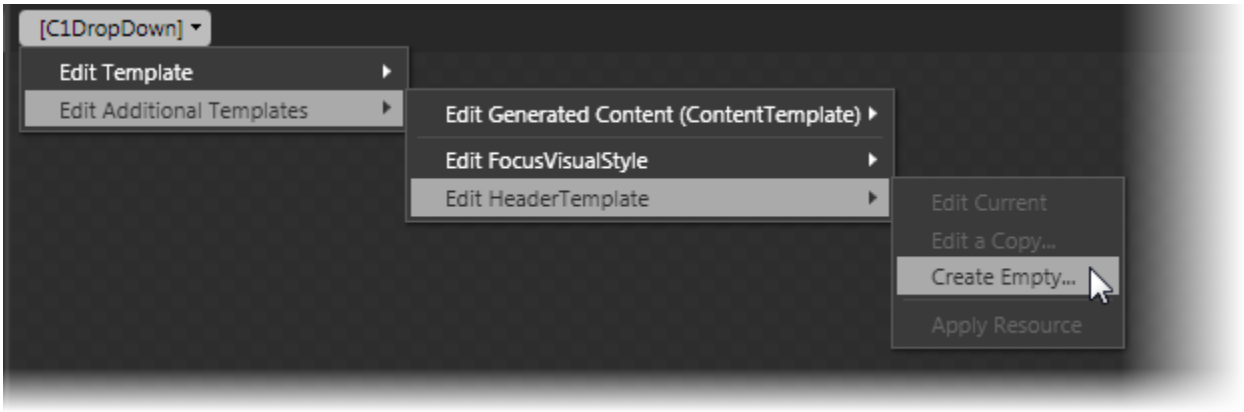


Once you've created a new template, the template will appear in the **Objects and Timeline** window. Note that you can use the Template property to customize the template.

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

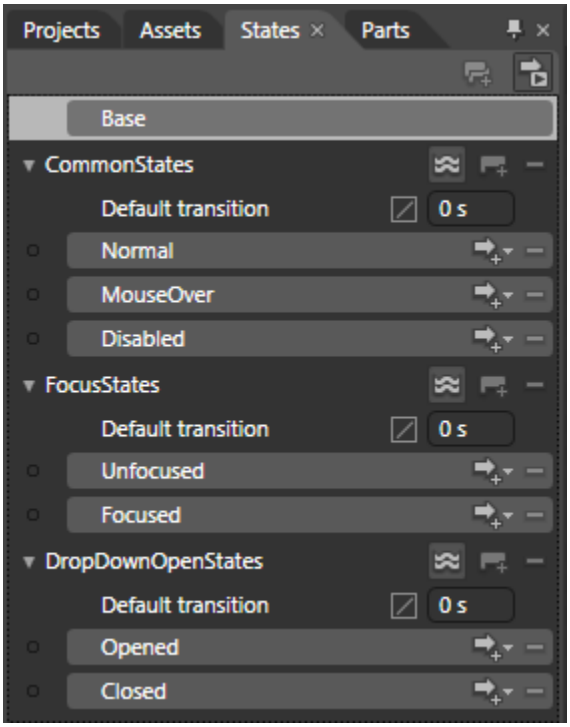
ComponentOne DropDown for WPF's C1DropDown control provides several style properties that you can use to change the appearance of the control. Some of the included styles are described in the table below:

Style	Description
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.
FontStyle	Gets or sets the font style. 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 WPF Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos, which may make use of other ComponentOne development tools included with the ComponentOne Studios. Samples can be accessed from the **ComponentOne Studio for WPF ControlExplorer**. To view samples, on your desktop, click the **Start** button and then click **All Programs | ComponentOne | Studio for WPF | Samples | WPF ControlExplorer**.

C# Samples

The following C# sample is included:

Sample	Description
ControlExplorer	The **DropDown** page in the **ControlExplorer** sample demonstrates how to add content to and customize the C1DropDown control.

DropDown for WPF 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 WPF** product, please see the DropDown for WPF Quick Start (page 3) first.

Each topic in this section provides a solution for specific tasks using the **ComponentOne DropDown for WPF** product. Most task-based help topics also assume that you have created a new WPF project and added a C1DropDown control named "c1dropdown1" to the project – for information about creating the control, see Creating a DropDown (page 17).

Creating a DropDown

You can easily create a C1DropDown control at design time, 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 it to have content at run time. For an example, see Adding Content to C1DropDown (page 19).

At Design Time in Blend

To create a C1DropDown control in Visual Studio, complete the following steps:

1. Click once on the design area of the window to select it.
2. Double-click the **C1DropDown** icon in the Toolbox to add the control to the panel. The C1DropDown control will now exist in your application.
3. If you choose, you can customize the control by selecting it and setting properties in the Properties window.

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.WPF.dll** assembly, and click **OK**.
2. Add a XAML namespace to your project by adding `xmlns:c1="http://schemas.componentone.com/wpf/Basic"` to the initial `<Window>` tag. It will appear similar to the following:

```
<Window x:Class="C1WPFPropertyGridCS102809.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="288" Width="418"
xmlns:c1="http://schemas.componentone.com/wpf/Basic">
```

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>
    <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.WPF.dll** assembly, and click **OK**.
2. In XAML view, give the initial grid in the window a name, by updating the tag so it appears similar to the following:

```
<Grid x:Name="LayoutRoot">
```

3. Right-click within the **Window1.xaml** window and select **View Code** to switch to Code view
4. Add the following import statements to the top of the page:
   - Visual Basic

```
Imports C1.WPF
```

   - C#

```
using C1.WPF;
```

5. 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 Window1()
{
    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 window.

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 for it to be interacted with at run time. For an example, see Adding Content to C1DropDown (page 19).

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, you 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 box 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 Window1()
{
    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 Direction (page 9).

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.WPF
   ```
   - C#
   ```
   using C1.WPF;
   ```
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,
   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 moves the cursor over the control at run time.

What You've Accomplished

In this topic you added code setting the IsDropDownOpen property so that the drop-down box opens on mouse over at run time. 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.