ComponentOne

ComboBox for WPF

By GrapeCity, Inc.

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

Internet: **info@ComponentOne.com**

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

Sales

E-mail: sales@componentone.com

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

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

Table of Contents

ComponentOne ComboBox for WPF Overview

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

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

Getting Started

- Working with the C1ComboBox Control (page 13)

- Quick Start (page 7)

- Task-Based Help (page 25)

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

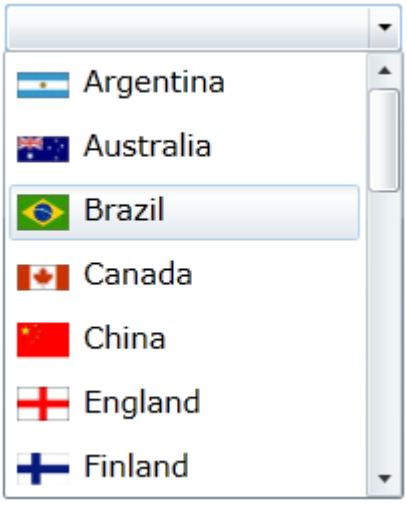
ComponentOne ComboBox for WPF allows you to create customized, rich applications. Make the most of **ComboBox for WPF** by taking advantage of the following key features:

- **Auto-Searchable Drop-Down List**

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

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

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



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

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

ComboBox for WPF Quick Start

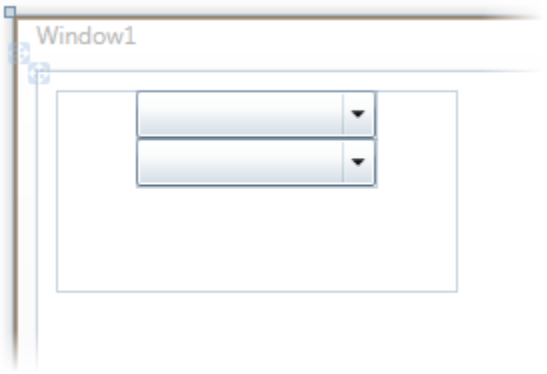
The following quick start guide is intended to get you up and running with **ComboBox for WPF**. In this quick start, you'll start in Visual Studio 2008 to create a new project with two C1ComboBox controls. The first control will be populated with a list of three items that, when clicked, will determine the list that appears in the second combo box.

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

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

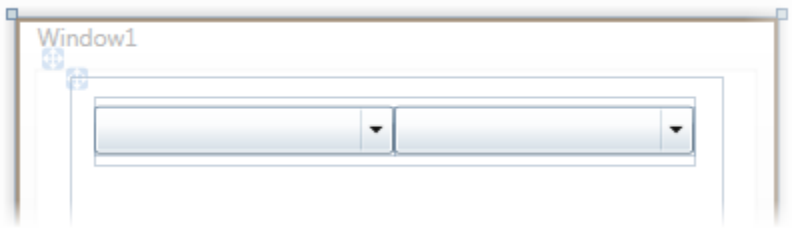
Complete the following steps:

1. In Visual Studio, select **File | New | Project**.
2. In the **New Project** dialog box, select a language in the left pane, and in the templates list select **WPF Application**.
3. Enter a **Name** for your project and click **OK**.
4. Add two C1ComboBox controls to the project by completing the following steps:
   a. In the **Toolbox**, double-click the **StackPanel** icon to add it to the project.
   b. Select the **StackPanel** control.
   c. Double-click the C1ComboBox icon to add the control to the **StackPanel**.
   d. Repeat steps 4b and 4c to add another C1ComboBox to the **StackPanel**. The project resembles the following:



5. Set the **StackPanel** control's properties as follows:
   - Set the **Width** property to "300".
   - Set the **Height** property to "35".
   - Set the **Orientation** property to **Horizontal**.
6. Set **c1ComboBox1**'s properties as follows:
   - Set the **Width** property to "150".
   - Set the **Height** property to "35".
   - Set the **Name** property to "Category"
7. Set **c1ComboBox2**'s properties as follows:
   - Set the **Width** property to "150".
   - Set the **Height** property to "35".
   - Set the **Name** property to "Shows".

   The project resembles the following:



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

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

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

Complete the following steps:

1. Select the first C1ComboBox, **Category**.
2. In the **Properties** window, click the **Items** ellipsis button to open the **Collection Editor: Items** dialog box.
3. Click **Add** three times to add three C1ComboBoxItems to the control. Three C1ComboBoxItems named **c1ComboBoxItem1**, **c1ComboBoxItem2**, and **c1ComboBoxItem3**, are added to the control.
4. Set **c1ComboBoxItem1**'s properties as follows:
   - Set the **Content** property to "Comedy".
   - Set the **Height** property to "25".
5. Set **c1ComboBoxItem2**'s properties as follows:
   - Set the **Content** property to "Drama".
   - Set the **Height** property to "25".
6. Set **C1ComboBoxItem3**'s properties as follows:
   - Set the **Content** property to "Science Fiction".
   - Set the **Height** property to "25".
7. Click **OK** to close the **Collection Editor: Items** dialog box.

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

Step 3 of 4: Adding Code to the Control

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

1. Select the first C1ComboBox control ("Category").
2. In the **Properties** window, click the **Events**  button.
3. Double-click the inside the **SelectedIndexChanged** text box to add the **C1ComboBox1_SelectedIndexChanged** event handler.

   The **MainPage.xaml.cs** page opens.

4. Import the following namespace into your project:
   - Visual Basic

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

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

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

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

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

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

- C#

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

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

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

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

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

Step 4 of 4: Running the Project

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

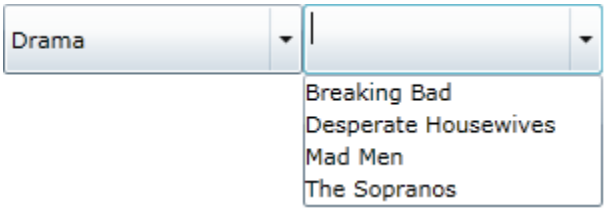
Complete the following steps:

1. Press F5 to run the project. The project loads with two blank combo boxes:

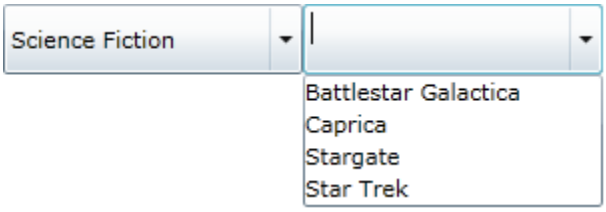
2. Click the second combo box's drop-down arrow and observe that the drop-down list is empty:

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

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



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



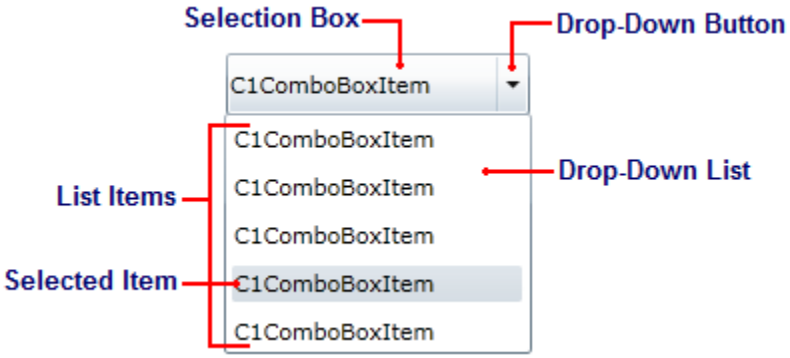
Congratulations! You have completed the **ComboBox for WPF** quick start.

Working with the C1ComboBox Control

This section provides an overview of C1ComboBox control basics. If you haven't used the control, we recommend starting with the ComboBox for WPF Quick Start (page 7) topic.

C1ComboBox Elements

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



See below for a description of each C1ComboBox element.

- **Selection Box**

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

- **Drop-Down Button**

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

- **Drop-Down List**

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

- **List Items**

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

- **Selected Item**

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

C1ComboBox Features

The following topics detail a few of the C1ComboBox control's features. For more information on utilizing these features, see the ComboBox for WPF Task-Based Help (page 25) section.

Drop-Down List Direction

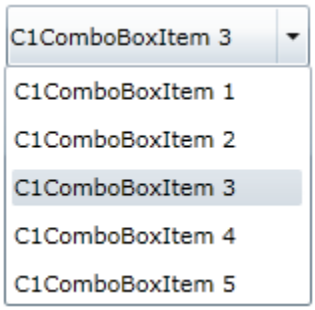
By default, when the user clicks the C1ComboBox control's drop-down arrow at run-time, the drop-down list will appear below the control; if that is not possible, it will appear above the control. You can, however, change the direction in which the drop-down list appears by setting the DropDownDirection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

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

For instructions about how to change the drop-down direction, see Changing the Drop-Down List Direction (page 28).

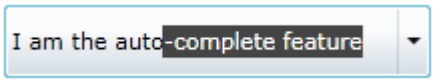
Item Selection

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



AutoComplete

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



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

The AutoComplete feature can be disabled by setting the AutoComplete property to **False**. To learn how to disable the feature at design time, in XAML, and in code, see Disabling AutoComplete (page 29).

Drop-Down List Sizing

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

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

For task-based help on drop-down list sizing, see Setting the Maximum Height and Maximum Width of the Drop-Down List (page 30).

ComboBox for WPF Layout and Appearance

The following topics detail how to customize the C1ComboBox control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the grid and take advantage of 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.

C1ComboBox and C1ComboBoxItem ClearStyle Properties

ComboBox 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 **C1ComboBox** control:

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

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

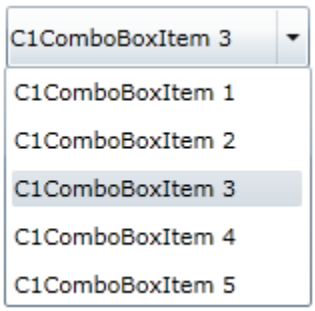
Brush	Description
Background	Gets or sets the brush of the control's background.

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

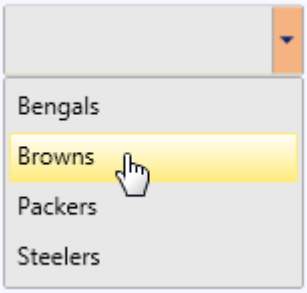
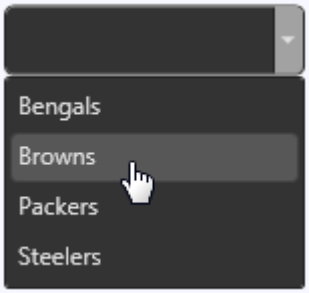
It's that simple with ComponentOne's ClearStyle technology. For more information on ClearStyle, see the ComponentOne ClearStyle Technology (page 15) topic.

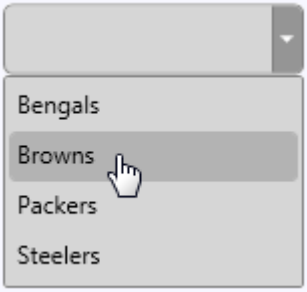
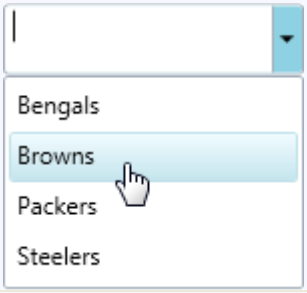
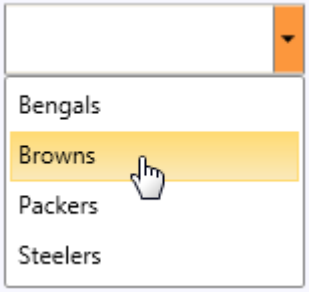
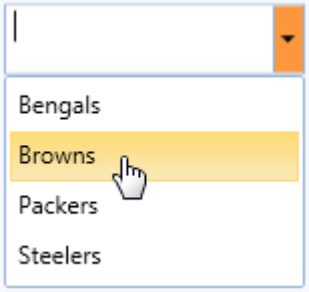
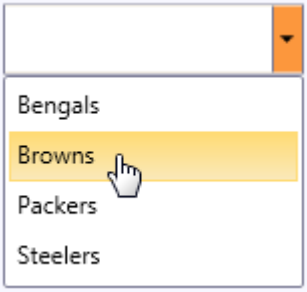
C1ComboBoxThemes

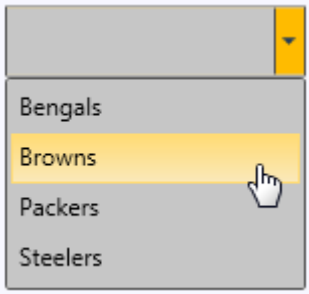
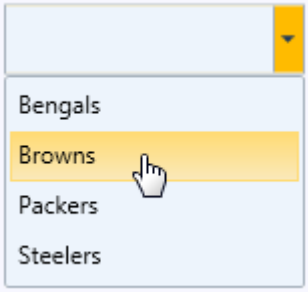
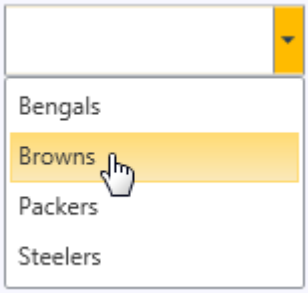
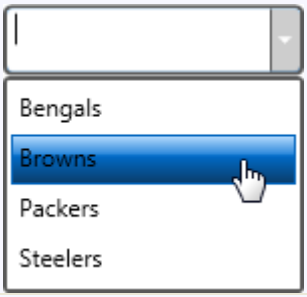
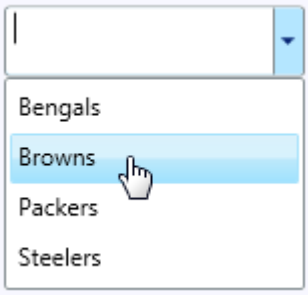
ComponentOne ComboBoxfor WPF incorporates several themes that allow you to customize the appearance of your grid. When you first add a C1ComboBox control to the page, it appears similar to the following image:



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	
C1ThemeOffice2007Black	
C1ThemeOffice2007Blue	
C1ThemeOffice2007Silver	

C1ThemeOffice2010Black	
C1ThemeOffice2010Blue	
C1ThemeOffice2010Silver	
C1ThemeShinyBlue	
C1ThemeWhistlerBlue	

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**.

ComboBox for WPF Appearance Properties

ComponentOne ComboBox 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 combo box control.

Property	Description
FontFamily	Gets or sets the font family of the control. This is a dependency property.
FontSize	Gets or sets the font size. This is a dependency property.
FontStretch	Gets or sets the degree to which a font is condensed or expanded on the screen. This is a dependency property.
FontStyle	Gets or sets the font style. This is a dependency property.
FontWeight	Gets or sets the weight or thickness of the specified font. This is a dependency property.
TextAlignment	Gets or sets how the text should be aligned in the drop-down list. This is a dependency property.

Content Positioning Properties

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

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

Color Properties

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

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

Border Properties

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

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

Size Properties

The following properties let you customize the size of the **C1ComboBox** control.

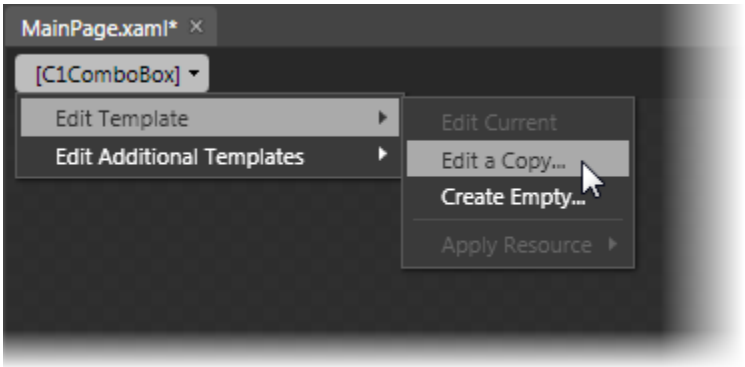
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.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

Templates

One of the main advantages to using a 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 ComboBox 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 C1ComboBox control and, in the menu, selecting **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or select **Create Empty** to create a new blank template.



If you want to edit the C1ComboBoxItem template, simply select the C1ComboBoxItem and, in the menu, select **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or **Create Empty**, to create a new blank template.

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

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

Additional ComboBox Templates

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

Item Templates

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

Accessing Templates

You can access these templates in Microsoft Expression Blend by selecting the C1ComboBox control and, in the menu, selecting **Edit Additional Templates**. Choose **Edit Generated Items (ItemTemplate)**, **Edit Layout of Items (ItemsPanel)**, or **Edit Generated Item Container (ItemStyleContainer)** and select **Create Empty** to create a new blank template or **Edit a Copy**.

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

ComboBox 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 C1ComboBox control in general. If you are unfamiliar with the **ComponentOne ComboBox for WPF** product, please see the **ComboBox for WPF** quick start first.

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

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

Working with ComboBox Items

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

Adding ComboBox Items in the Designer

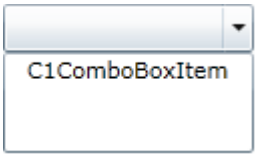
In this topic, you will learn how to add items to the C1ComboBox control in Expression Blend. This method is useful whenever you're creating a static combo box with just a few items.

Complete the following steps:

1. In the **Properties** window, click the **Items** ellipsis button  to open the **Collection Editor: Items** dialog box.
2. Click **Add** to add a **C1ComboBoxItem** to the C1ComboBox control.

This Topic Illustrates the Following:

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



Adding ComboBox Items in XAML

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

Complete the following steps:

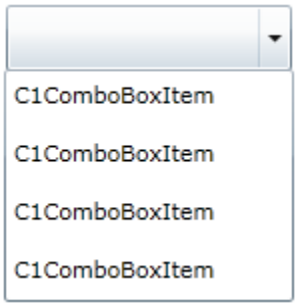
1. To add items to the C1ComboBox control, add the following XAML markup between the `<c1:C1ComboBox>` and `</c1:C1ComboBox>` tags:

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

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

This Topic Illustrates the Following:

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



Adding ComboBox Items in Code

In this topic, you will learn how to add items to the C1ComboBox control in C# and Visual Basic code. This method is useful when you're creating a static combo box with just a few items.

Complete the following steps:

To disable AutoComplete, complete the following:

1. Open the **MainPage.xaml.cs** page.
2. Import the following namespace into your project:
   - Visual Basic

     ```
     Imports C1.WPF
     ```

   - C#

     ```
     Using C1.WPF;
     ```

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

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

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

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

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

   - C#

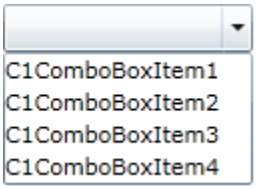
     ```
     c1ComboBox1.Items.Add(new C1ComboBoxItem() { Content =
     "C1ComboBoxItem1" });
     ```

```
c1ComboBox1.Items.Add(new C1ComboBoxItem() { Content =
"C1ComboBoxItem2" });

c1ComboBox1.Items.Add(new C1ComboBoxItem() { Content =
"C1ComboBoxItem3" });

c1ComboBox1.Items.Add(new C1ComboBoxItem() { Content =
"C1ComboBoxItem4" });
```

4. Run the program.

This Topic Illustrates the Following:

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



Adding ComboBox Items from a Collection

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

Complete the following steps:

1. Open the **MainPage.xaml.cs** page.
2. Import the following namespace into the project:
   - Visual Basic

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

   - C#

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

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

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

   - C#

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

```
dropDownList.Add("C1ComboBoxItem4");
```

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

     ```
     C1ComboBox1.ItemsSource = dropDownList
     ```

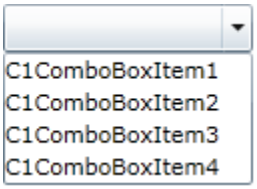
   - C#

     ```
     c1ComboBox1.ItemsSource = dropDownList;
     ```

5. Run the program.

This Topic Illustrates the Following:

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



Changing the Drop-Down List Direction

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

In the Designer

Complete the following steps:

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

In XAML

Complete the following steps:

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

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

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

In Code

Complete the following steps:

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

     ```
     C1ComboBox1.DropDownDirection = ForceAbove
     ```

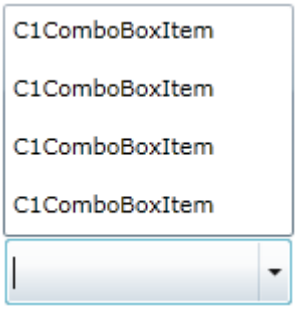
   - C#

     ```
     c1ComboBox1.DropDownDirection = ForceAbove;
     ```

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

This Topic Illustrates the Following:

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



Disabling AutoComplete

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

In the Designer

Complete the following steps:

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

In XAML

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

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

In Code

Complete the following steps:

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

     ```
     C1ComboBox1.AutoComplete = False
     ```

- C#

```
c1ComboBox1.AutoComplete = false;
```

3. Run the program.

This Topic Illustrates the Following:

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

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

You can specify the maximum height and maximum width of a combo box's drop-down list by setting its MaxDropDownHeight and MaxDropDownWidth properties. This topic assumes that the DropDownHeight and DropDownWidth properties are both set to **NaN**. For more information, see Drop-Down List Sizing (page 15).

In the Designer

Complete the following steps:

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

In XAML

Complete the following steps:

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

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

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

In Code

Complete the following steps:

1. Open the **MainPage.xaml.cs** page.
2. Add the following code beneath the **InitializeComponent()** method to set the DropDownHeight property :
   - Visual Basic

```
C1ComboBox1.MaxDropDownHeight = 150
```

   - C#

```
c1ComboBox1.MaxDropDownHeight = 150;
```

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

```
C1ComboBox1.MaxDropDownWidth = 350
```

- C#

```
c1ComboBox1.MaxDropDownWidth = 350;
```

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

This Topic Illustrates the Following:

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

Launching with the Drop-Down List Open

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

In the Designer

Complete the following steps:

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

In XAML

Complete the following steps:

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

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

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

In Code

Complete the following steps:

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

```
C1ComboBox1.IsDropDownOpen = True
```

   - C#

```
c1ComboBox1.IsDropDownOpen = true;
```

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

This Topic Illustrates the Following:

In this topic, you set the IsDropDownOpen property to **True** so that the drop-down list would be open at run time. You can also use this property to open the drop-down list when a user mouses over the C1ComboBox control (see Opening the Drop-Down List on MouseOver (page 32)).

Opening the Drop-Down List on MouseOver

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

Complete the following:

1. Click the C1ComboBox control to select it.
2. In the **Properties** window, click the **Events** button  to reveal the control's list of events.
3. Double-click inside of the **MouseEnter** text box. This will add the **C1ComboBox_MouseEnter** event handler to Code view.
4. Add the following code to the **C1ComboBox1_MouseEnter** event handler:
   - Visual Basic

     ```
     C1ComboBox1.IsDropDownOpen = True
     ```

   - C#

     ```
     c1ComboBox1.IsDropDownOpen = true;
     ```

5. Run the program.

This Topic Illustrates the Following:

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

Selecting an Item

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

In the Designer

Complete the following steps:

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

In XAML

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

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

In Code

Complete the following steps:

1. Open the **MainPage.xaml.cs** page.

2. Add the following code beneath the **InitializeComponent()** method:
   - Visual Basic

     ```
     C1ComboBoxItem1.SelectedIndex = 1
     ```
   - C#

     ```
     c1ComboBoxItem1.SelectedIndex = 1;
     ```
3. Run the program.

This Topic Illustrates the Following:

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