ComponentOne

ListBox 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 ListBox for WPF Overview

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

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

- **Horizontal or Vertical Orientation**

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

- **Display Items as Tiles**

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

- **Optical Zoom**

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

- **UI Virtualization**

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

- **Preview State**

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

C1ListBox Quick Start

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

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

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

   ```
   <StackPanel x:Name="loading" VerticalAlignment="Center">
       <TextBlock Text="Retrieving data from Flickr..."
   TextAlignment="Center"/>
       <ProgressBar IsIndeterminate="True" Width="200"  Height="4"/>
   </StackPanel>
   <Button x:Name="retry" HorizontalAlignment="Center"
   VerticalAlignment="Center" Content="Retry" Visibility="Collapsed"
   Click="Retry_Click"/>
   ```

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

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

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

   ```
   <c1:C1ListBox.PreviewItemTemplate>
       <DataTemplate>
           <Grid Background="Gray">
               <Image Source="{Binding Thumbnail}"
   Stretch="UniformToFill"/>
   ```

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

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

What You've Accomplished

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

Step 2 of 3: Adding Data to the ListBox

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

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

1. Right-click the **Window** and select **View Code** to switch to the Code Editor.
2. Add the following imports statements to the top of the page:
   - Visual Basic
     ```
     Imports System.Linq
     Imports System.Windows.Controls
     Imports System.Windows
     Imports System.Collections.Generic
     Imports System.Net
     Imports System.Xml.Linq
     Imports System
     Imports C1.WPF
     ```
   - C#
     ```
     using System.Linq;
     using System.Windows.Controls;
     using System.Windows;
     using System.Collections.Generic;
     using System.Net;
     using System.Xml.Linq;
     using System;
     using C1.WPF;
     ```
3. Add the following code inside the initial event handler:
   - Visual Basic
     ```
     DataContext = Enumerable.Range(0, 100)
     AddHandler Loaded, AddressOf ListBoxSample_Loaded
     ```
   - C#
     ```
     DataContext = Enumerable.Range(0, 100);
     Loaded += new System.Windows.RoutedEventHandler(ListBoxSample_Loaded);
     ```

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

- Visual Basic

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

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

        Dim photos = New List(Of Photo)()
        Dim client = New WebClient()
        AddHandler client.OpenReadCompleted, Function(s, e)
                                                 Try
                                                     '#Region "** parse
flickr data"
                                                     Dim doc =
XDocument.Load(e.Result)
                                                     For Each entry In
doc.Descendants(XName.[Get]("entry", AtomNS))
                                                         Dim title =
entry.Element(XName.[Get]("title", AtomNS)).Value
                                                         Dim enclosure
= entry.Elements(XName.[Get]("link", AtomNS)).Where(Function(elem)
elem.Attribute("rel").Value = "enclosure").FirstOrDefault()
                                                         Dim contentUri
= enclosure.Attribute("href").Value
                                                         photos.Add(New
Photo() With { _
                                                           .Title =
title, _
                                                           .Content =
contentUri, _
                                                           .Thumbnail =
contentUri.Replace("_b", "_m") _
                                                         })
                                                     Next
                                                     '#End Region

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

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

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

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

#Region "** public properties"

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

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

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

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

- C#

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

        private void LoadPhotos()
        {
            var flickrUrl =
"http://api.flickr.com/services/feeds/photos_public.gne";
```

```
            var AtomNS = "http://www.w3.org/2005/Atom";
            loading.Visibility = Visibility.Visible;
            retry.Visibility = Visibility.Collapsed;

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

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

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

        #region ** public properties

        public Orientation Orientation
        {
            get
            {
                return listBox.Orientation;
            }
```

```
            set
            {
                listBox.Orientation = value;
            }
        }

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

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

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

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

   - Visual Basic

```
Public Class Photo
    Public Property Title() As String
        Get
            Return m_Title
        End Get
        Set(value As String)
            m_Title = Value
        End Set
    End Property
```

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

- C#

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

What You've Accomplished

You have successfully added data to **C1TileListBox**. In the next step, Step 3 of 3: Running the ListBox Application (page 8), you'll examine the **ListBox for WPF** features.

Step 3 of 3: Running the ListBox Application

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

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

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

What You've Accomplished

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

C1TileListBox Quick Start

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

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

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

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

   This gives the control a name, sets the **ItemsSource** property (you'll customize this in code in a later step), and sets the size of the control.
6. Add markup between the `<c1:C1TileListBox>` and `</c1:C1TileListBox>` tags so it looks like the following:

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

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

What You've Accomplished

You've successfully created a WPF application containing a **C1TileListBox** control. In the next step, Step 2 of 3: Adding Data to the TileListBox (page 9), you will add data for **C1TileListBox**.

Step 2 of 3: Adding Data to the TileListBox

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

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

1. Right-click the page and select **View Code** to open the Code Editor.
2. Add the following imports statements to the top of the page:
   - Visual Basic
     ```
     Imports System
     Imports System.Collections.Generic
     Imports System.Diagnostics
     Imports System.Linq
     Imports System.Net
     Imports System.Windows;
     Imports System.Windows.Controls;
     Imports System.Xml.Linq;
     Imports C1.WPF
     ```
   - C#
     ```
     using System;
     using System.Collections.Generic;
     using System.Diagnostics;
     using System.Linq;
     using System.Net;
     using System.Windows;
     using System.Windows.Controls;
     using System.Xml.Linq;
     using C1.WPF;
     ```
3. Add the following code inside the initial event handler within the **MainPage** class:
   - Visual Basic
     ```
     DataContext = Enumerable.Range(0, 100).[Select](Function(i) New Item()
     With {.Title = i.ToString()})
     ```
   - C#
     ```
     DataContext = Enumerable.Range(0, 100).Select(i => new Item { Title =
     i.ToString() });
     ```
4. Add the following code below the initial event handler but within the **MainPage** class:
   - Visual Basic
     ```
     #Region "** public properties"

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

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

         Public Property ItemHeight() As Double
     ```

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

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

- C#

```
#region ** public properties

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

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

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

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

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

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

- Visual Basic

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

- C#

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

What You've Accomplished

You have successfully added data to **C1TileTileListBox**. In the next step, Step 3 of 3: Running the TileListBox Application (page 12), you'll examine the **TileListBox for WPF** features.

Step 3 of 3: Running the TileListBox Application

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

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

The application will appear, displaying a tiled list of numbers.

2. Use the scroll bar on the right of the control, to scroll through the numbered squares.
3. If you have touch capabilities, try pinching to zoom an image.

What You've Accomplished

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

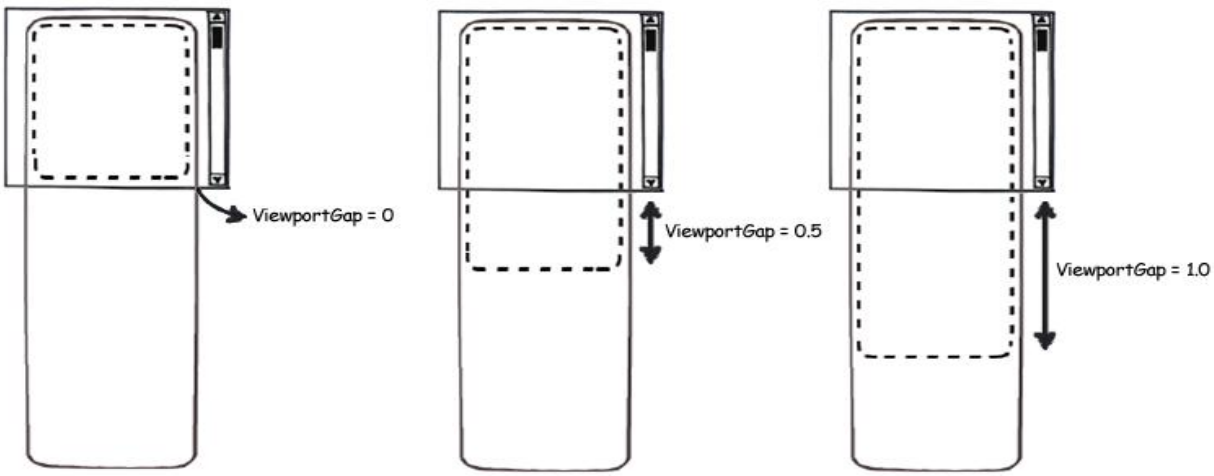
Top Tips

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

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

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

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



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

Working with ListBox for WPF

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

Basic Properties

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

The following properties let you customize the C1ListBox control:

Property	Description
ActualMaxZoom	Gets the actual maximum zoom.
ActualMinZoom	Gets the actual minimum zoom.
ActualZoom	Gets the actual zoom.
IsScrolling	Gets a value indicating whether the list is scrolling.
IsZooming	Gets a value indicating whether this list is zooming.
ItemHeight	Gets or sets the height of each item.
Items	Gets the collection used to generate the content of the control. (Inherited from ItemsControl.)
ItemsPanel	Gets or sets the template that defines the panel that controls the layout of items. (Inherited from ItemsControl.)
ItemsSource	Gets or sets a collection used to generate the content of

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

The following properties let you customize the C1ListBoxItem:

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

Optical Zoom

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

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

UI Virtualization

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

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

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

Orientation

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

Preview State

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