ComponentOne

TreeView 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 TreeView for WPF Overview

Get a hierarchical view of your data items with **ComponentOne TreeView for WPF**. It's similar to the standard TreeView controls available in WPF and Window Forms, but provides more features like keyboard-based search, drag and drop functionality, auto-search, hierarchical templates, and more.

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.

TreeView for WPF Key Features

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

- **Drag-and-drop Nodes within the TreeView**

  TreeView supports drag-and-drop operations within the tree. Simply set the AllowDragDrop property to true and users will be able to reorder nodes within the tree by dragging them with the mouse.

- **Customizable Drag-drop Behavior**

  TreeView fires events during drag-drop operations so you can customize their behavior. For example, you can prevent some nodes from being dragged or some nodes from acting as drop targets.

- **Auto-search**

  With our TreeView control, you can easily jump to a letter in the node with auto-search. Just type a letter to go to a specific tree node.

- **Hierarchical Templates**

  You can use different templates for different node types without having to subclass the C1TreeViewItem class.

- **Customizable Nodes**

  Node headers are content elements, so they can host any type of element. Add images, checkboxes, or whatever your application requires.

- **Keyboard Navigation**

  Use the cursor keys to navigate the nodes, expanding and collapsing them as you go. Or use the auto-search feature to find specific nodes quickly and easily.

- **Supports ClearStyle Technology**

  TreeView for WPF supports ComponentOne's ClearStyle technology, which allows you to easily change control colors without having to change control templates. By setting a few color properties, you can quickly style the **C1TreeView** control.

TreeView for WPF Quick Start

The following quick start guide is intended to get you up and running with **TreeView for WPF**. In this quick start, you'll start in Visual Studio to create a new project, add a C1TreeView control to your application, and then add content to the C1TreeView control's content area.

Step 1 of 3: Creating an Application with a C1TreeView Control

In this step, you'll begin in Visual Studio to create a WPF application using **TreeView 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** to create your project. The **MainWindow.xaml** file should open
3. In the XAML window of the project, place the cursor between the `<Grid>` and `</Grid>` tags and click once.

   Navigate to the Toolbox and double-click the **C1TreeView** icon to add the treeview control to **MainWindow.xaml**. Note that the C1.WPF namespace and `<c1:C1TreeView></c1:C1TreeView>` tags have been added to the project.
4. Remove any markup in or between the `<c1:C1TreeView></c1:C1TreeView>` tags. The XAML markup will now look similar to the following:

```
<Window x:Class="MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525"
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml">
  <Grid>
<c1:C1TreeView></c1:C1TreeView>
  </Grid>
</Window>
```

5. Give your grid a name by adding `x:Name="Tree"` to the `<c1:C1TreeView>` tag so that it appears similar to the following:

```
<c1:C1TreeView x:Name="Tree">
```

   By giving the control a unique identifier, you'll be able to access the **C1TreeView** control in code.

You've successfully created a WPF application containing a C1TreeView control. In the next step, you will customize the appearance and behavior of the C1TreeView control.

Step 2 of 3: Adding C1TreeView Items to C1TreeView

This lesson will show you how to add static **C1TreeView** items to the **C1TreeView** control in the XAML markup and in the code behind file.

In XAML

To add static **C1TreeViewItems** to the **C1TreeView** control in the XAML:

1. Add the **C1TreeViewItem** to create the top-level node called "Book List". Within the `<c1:C1TreeViewItem>` tag add `Header="Book List`. This will create a top-level node that at run time. The XAML markup appears as follows:

```
<c1:C1TreeViewItem Header="Book List"></c1:C1TreeViewItem>
```

2. Add two child **C1TreeViewItems** between the `<c1:C1TreeViewItem Header="Book List"></c1:C1TreeViewItem>` tags to create two child nodes beneath the **Book List** node. The XAML markup appears as follows:

```
<c1:C1TreeViewItem Header="Language Books"/>
<c1:C1TreeViewItem Header="Security Books"/>
```

3. Add another `<c1:C1TreeViewItem>` tag to create a new top level node that will hold two child nodes. The XAML markup appears as follows:

```
<c1:C1TreeViewItem Header="Classic Books">
  <c1:C1TreeViewItem Header="Catch-22"/>
  <c1:C1TreeViewItem Header="The Great Gatsby"/>
</c1:C1TreeViewItem>
```

You should have all of the following XAML markup now included in your MainWindow.xaml file:

```
<Window x:Class="MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525"
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml">
    <Grid>
        <c1:C1TreeView Name="Tree" >
            <c1:C1TreeViewItem Header="Book List">
                <c1:C1TreeViewItem Header="Language Books"/>
                <c1:C1TreeViewItem Header="Security Books"/>
                <c1:C1TreeViewItem Header="Classic Books">
                    <c1:C1TreeViewItem Header="Catch-22"/>
                    <c1:C1TreeViewItem Header="The Great Gatsby"/>
                </c1:C1TreeViewItem>
            </c1:C1TreeViewItem>
        </c1:C1TreeView>
    </Grid>
</Window>
```

4. Run the project and notice that the **Book** node is not expanded. You can expand it by clicking on the arrow image.

In Code

To add static **C1TreeView** items to the **C1TreeView** control in the code behind file instead, add the following code in the Code Editor:

- Visual Basic

```
Imports C1.WPF
Class MainWindow
    Public Sub New()
        InitializeComponent()
        InitializeTreeView()
    End Sub
    Private Sub InitializeTreeView()
        ' Remove items that were added at design time
        Tree.Items.Clear()
        Dim booklist As New C1TreeViewItem()
        booklist.Header = "Book List"
        Tree.Items.Add(booklist)

        ' Adding child items
        Dim language As New C1TreeViewItem()
        language.Header = "Language Books"
```

```
        booklist.Items.Add(language)

        ' Adding child items
        Dim security As New C1TreeViewItem()
        security.Header = "Security Books"
        booklist.Items.Add(security)

        ' Adding child items
        Dim classic As New C1TreeViewItem()
        classic.Header = "Classic Books"
        booklist.Items.Add(classic)

        ' Adding child items
        Dim subclassic As New C1TreeViewItem()
        subclassic.Header = "Catch-22"
        classic.Items.Add(subclassic)
        Dim subclassic2 As New C1TreeViewItem()
        subclassic2.Header = "The Great Gatsby"
        classic.Items.Add(subclassic2)
    End Sub
End Class
```

- C#

```
using C1.WPF;
public MainWindow()
{
    InitializeComponent();
    InitializeTreeView();
}
void InitializeTreeView()
{
    // Remove items that were added at design time
    Tree.Items.Clear();
    C1TreeViewItem booklist = new C1TreeViewItem();
    booklist.Header = "Book List";
    Tree.Items.Add(booklist);

   // Adding child items
    C1TreeViewItem language = new C1TreeViewItem();
    language.Header = "Language Books";
    booklist.Items.Add( language );

    // Adding child items
    C1TreeViewItem security = new C1TreeViewItem();
    security.Header = "Security Books";
    booklist.Items.Add(security);

    // Adding child items
    C1TreeViewItem classic = new C1TreeViewItem();
    classic.Header = "Classic Books";
    booklist.Items.Add(classic);

    // Adding child items
    C1TreeViewItem subclassic = new C1TreeViewItem();
    subclassic.Header = "Catch-22";
    classic.Items.Add(subclassic);
    C1TreeViewItem subclassic2 = new C1TreeViewItem();
```

```
    subclassic2.Header = "The Great Gatsby";
    classic.Items.Add(subclassic2);
}
```

In this step, you added several **C1TreeView** items to the C1TreeView control. In the next step, you will customize the behavior and appearance of the **C1TreeView** control.

Step 3 of 3: Customizing TreeView's Appearance and Behavior

In the previous step you worked in Visual Studio to create **C1TreeViewItems** in XAML. In this step you'll customize the **C1TreeView** control's appearance and behavior in Visual Studio using XAML code.

To customize **TreeView for WPF**, complete the following steps:

1. Place your cursor within the `<c1:C1TreeView>` tag. Within the `<c1:C1TreeView>` tag add `SelectionMode="Extended"`. This will create a top-level node that you will be able to select multiple tree items by holding the shift and control keys. The XAML markup appears as follows:

   ```
   <c1:C1TreeView x:Name="Tree" SelectionMode="Extended">
   ```

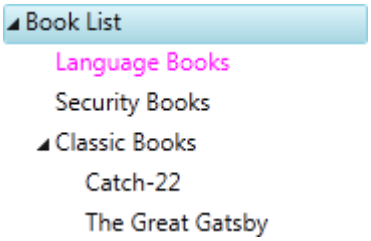
2. Place your cursor within the first `<c1:C1TreeViewItem>` tag. Within the `<c1:C1TreeViewItem>` add the tag `IsExpanded="True"`and `IsSelected="True"`. This will create a top-level node that appears selected and expanded at run time. The XAML markup appears as follows:

   ```
   <c1:C1TreeViewItem Header="Book List" IsExpanded="True" IsSelected="True">
   ```

3. Locate the tag that reads `<c1:C1TreeViewItem Header="Language Books">`. Within the `<c1:C1TreeViewItem Header="Language Books">` add `Foreground="Fuchsia" Background="LightPink"`. This will add a light pink background to the "Classic Books" tree item and a fuchsia color to the text. The XAML markup will resemble the following:

   ```
   <c1:C1TreeViewItem Header="Language Books" Foreground="Fuchsia"/>
   ```

4. From the **Debug** menu, select **Start Debugging** to view how your application will appear at run time. Observe the following behavioral and appearance changes for **C1TreeView**:



- The **C1TreeView** appears expanded.
- The first **C1TreeViewItem** appears selected.
- The second **C1TreeViewItem** has fuchsia colored text.

Congratulations! You have successfully completed the **TreeView for WPF** quick start. In this quick start, you've created and customized a **TreeView for WPF** application, added static **C1TreeViewItem**s, and observed several of the control's run-time features.

C1TreeView Structure

The **C1TreeView** class is a **StackPanel** with two elements:

- A header that represents the actual node, with a button to collapse and expand the children.
- A body that is another **StackPanel** and contains other nodes.

You can add images to a node by grabbing its first child (the header), casting that to a **StackPanel**, and inserting an image element at whatever position you prefer. For example:

- Visual Basic

```
Dim nodeHeader As StackPanel = TryCast(TreeNode.Children(0),
StackPanel)
nodeHeader.Children.Insert(0, myImage)
```

- C#

```
StackPanel nodeHeader = TreeNode.Children[0] as StackPanel;
nodeHeader.Children.Insert(0, myImage);
```

TreeView Creation

C1TreeViewItems can be added to the **C1TreeView** control as static items defined either in the XAML or in the code behind or can be defined on your page or user control by using any of the following methods:

- Static creation using XAML syntax or programmatically through the code behind file
- Dynamic creation using a constructor to create new instances of the **C1TreeViewItem** class.
- Data source creation through binding **C1TreeView** to a **SiteMapDataSource**, **XMLDataSource**, or an **AccessDataSource**.

Static TreeView Creation

Each node in the Tree is represented by a name/value pair, defined by the text and value properties of treenode, respectively. The text of a node is rendered, whereas the value of a node is not rendered and is typically used as additional data for handling postback events.

A static menu is the simplest way to create the treeview structure.

To display static C1TreeViewNodes using XAML syntax, first nest opening and closing <Nodes> tags between opening and closing tags of the C1TreeView control. Next, create the treeview structure by nesting <c1:C1TreeViewNode> elements between opening and closing <Nodes> tags. Each <c1:C1TreeViewNode> element represents a node in the control and maps to a **C1TreeViewNode** object.

Declarative syntax can be used to define the **C1TreeViewNodes** inline on your page.

For example:

```
<Grid x:Name="LayoutRoot">
    <c1:C1TreeView x:Name="Tree">
         <c1:C1TreeViewItem Header="Book List" IsExpanded="True"
IsSelected="True">
               <c1:C1TreeViewItem Header="Language Books"/>
               <c1:C1TreeViewItem Header="Security Books"/>
               <c1:C1TreeViewItem Header="Classic Books">
                   <c1:C1TreeViewItem Header="Catch-22"/>
                   <c1:C1TreeViewItem Header="The Great Gatsby"/>
               </c1:C1TreeViewItem>
```

```
        </c1:C1TreeViewItem>
    </c1:C1TreeView>
</Grid>
```

Dynamic TreeView Creation

Dynamic treeviews can be created on the server side or client side. When creating dynamic treeview on the server side, use a constructor to dynamically create a new instance of the **C1TreeView** class. For example:

- Visual Basic

```
Namespace TreeViewQuickStart
    Public Partial Class MainWindow
        Inherits UserControl
        Public Sub New()
            InitializeComponent()
            InitializeTreeView()
        End Sub
        Private Sub InitializeTreeView()

            ' Remove items that were added at design time

            Tree.Items.Clear()

            Dim booklist As New C1TreeViewItem()
            booklist.Header = "Book List"
            'booklist.Foreground = new SolidColorBrush( Colors.Brown);
            Tree.Items.Add(booklist)

            ' Adding child items
            Dim language As New C1TreeViewItem()
            language.Header = "Language Books"
            booklist.Items.Add(language)

            ' Adding child items
            Dim security As New C1TreeViewItem()
            security.Header = "Security Books"
            booklist.Items.Add(security)

            ' Adding child items
            Dim classic As New C1TreeViewItem()
            classic.Header = "Classic Books"
            booklist.Items.Add(classic)
            'Add checkbox
            classic.Header = New CheckBox()

            ' Adding child items
            Dim subclassic As New C1TreeViewItem()
            subclassic.Header = "Catch-22"
            classic.Items.Add(subclassic)
            Dim subclassic2 As New C1TreeViewItem()
            subclassic2.Header = "The Great Gatsby"
            classic.Items.Add(subclassic2)
        End Sub
    End Class
End Namespace
```

- C#

```
namespace TreeViewQuickStart
{
    public partial class MainWindow : UserControl
    {
        public MainWindow()
        {
            InitializeComponent();
            InitializeTreeView();

        }
       void InitializeTreeView()
       {

            // Remove items that were added at design time

            Tree.Items.Clear();

            C1TreeViewItem booklist = new C1TreeViewItem();
            booklist.Header = "Book List";
           //booklist.Foreground = new SolidColorBrush( Colors.Brown);
           Tree.Items.Add(booklist);

           // Adding child items
          C1TreeViewItem language = new C1TreeViewItem();
          language.Header = "Language Books";
          booklist.Items.Add( language );

         // Adding child items
         C1TreeViewItem security = new C1TreeViewItem();
         security.Header = "Security Books";
         booklist.Items.Add(security);

        // Adding child items
        C1TreeViewItem classic = new C1TreeViewItem();
        classic.Header = "Classic Books";
        booklist.Items.Add(classic);
        //Add checkbox
        classic.Header = new CheckBox();

       // Adding child items
       C1TreeViewItem subclassic = new C1TreeViewItem();
       subclassic.Header = "Catch-22";
       classic.Items.Add(subclassic);
       C1TreeViewItem subclassic2 = new C1TreeViewItem();
       subclassic2.Header = "The Great Gatsby";
       classic.Items.Add(subclassic2);
}
    }
}
```

Data Source TreeView Creation

TreeView items can be created from a hierarchal datasource control such as an **XMLDataSource** or **SiteMapDataSource**. This allows you to update the treeview items without having to edit code.

When using multi-level data as **ItemsSource** for the **C1TreeView**, you need to specify a **C1HierarchicalDataTemplate** for the items.

The template will tell the **C1TreeView** where to find the next level of data, this is done through the "ItemsSource" property of the **C1HierarchicalDataTemplate**.

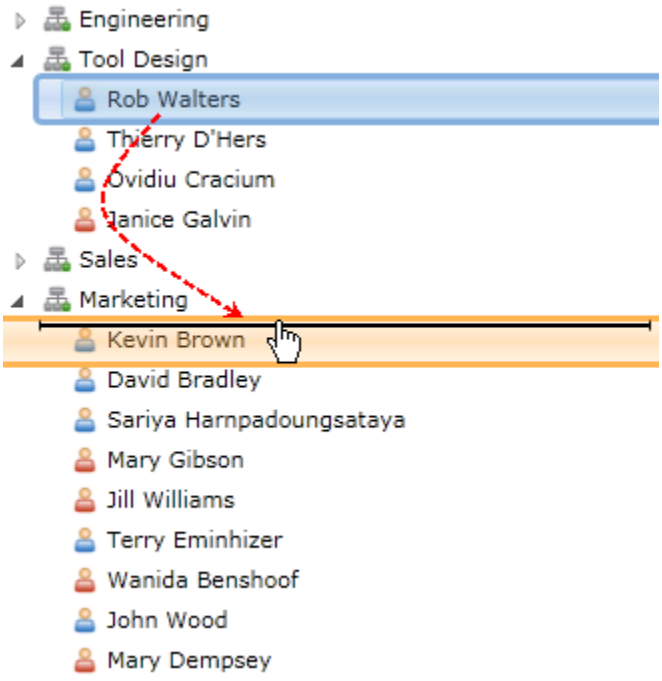
TreeView Behavior

The following topics describe the behavior for the **C1TreeView** control.

Drag-and-Drop Nodes

You can drag-and-drop **C1TreeViewNode**s on nodes, in between nodes, or from one tree to another tree when the AllowDragDrop property is set to **True**.

The following image shows a **C1TreeViewNode** being dragged from one **C1TreeView** to another **C1TreeView**. An arrow or vertical line can be used as a visual cue to show you where the **C1TreeViewItem** is going to be dropped when either the DragDropArrowMarker or DragDropLineMarker properties are applied.



Load on Demand

Instead of fully populating each node when the application starts, you can use a technique called "delayed loading" where nodes are populated on demand when the user expands them. This allows the application to load faster and use resources more efficiently

To implement the delayed loading nodes, use the following code:

```
public Page()
{
    InitializeComponent();
    // No changes here.
    // ...
```

```
    // Initialize the C1TreeView
    InitializeTreeView();
}
void InitializeTreeView()
{
  // Remove items that were added at design time
  _tv.Items.Clear();

   // Scan every type in the assembly
  foreach (Type t in _tv.GetType().Assembly.GetTypes())
  {
    if (t.IsPublic && !t.IsSpecialName && !t.IsAbstract)
    {
      // Add node for this type
      C1TreeViewItem node = new C1TreeViewItem();
      node.Header = t.Name;
      node.FontWeight = FontWeights.Bold;
      _tv.Items.Add(node);
      // Add subnodes for properties, events, and methods
      node.Items.Add(CreateMemberNode("Properties", t,
MemberTypes.Property));
      node.Items.Add(CreateMemberNode("Events", t, MemberTypes.Event));
      node.Items.Add(CreateMemberNode("Methods", t,
MemberTypes.Method));
    }
  }
}
C1TreeViewItem CreateMemberNode(string header, MemberTypes memberTypes)
{
  // Create the node
  C1TreeViewItem node = new C1TreeViewItem();
  node.Header = header;
  node.Foreground = new SolidColorBrush(Colors.DarkGray);
   // Hook up event hander to populate the node before expanding it
  node.Expanding += node_Expanding;
   // Save information needed to populate the node
  node.Tag = memberTypes;
   // Add a dummy node so this node can be expanded
  node.Items.Add(new C1TreeViewItem());
  node.IsExpanded = false;
   // Finish
  return node;
}
//populate the node
void node_Expanding(object sender, RoutedEventArgs e)
{
  // Get the node that fired the event
  C1TreeViewItem node = sender as C1TreeViewItem;
  // Unhook event handler (we'll populate the node and be done with it)
  node.Expanding -= node_Expanding;
   // Remove dummy node
  node.Items.Clear();
   // Populate the node
  Type type = (Type)node.Parent.Tag;
  MemberTypes memberTypes = (MemberTypes)node.Tag;
   BindingFlags bf = BindingFlags.Public | BindingFlags.Instance;
  foreach (MemberInfo mi in type.GetMembers(bf))
```

```
  {
    if (mi.MemberType == memberTypes)
    {
      if (!mi.Name.StartsWith("get_") && !mi.Name.StartsWith("set_"))
      {
        C1TreeViewItem item = new C1TreeViewItem();
        item.Header = mi.Name;
        item.FontSize = 12;
        node.Items.Add(item);
      }
    }
  }
}
```

This implementation hooks up an event hander for the **Expanding** event, so we can populate the node when the user tries to expand it. We also save the information we will need to populate the node in the **Tag** property. Finally, we add a dummy child node so the user will be able to expand this node and trigger the Expanding event that will populate the node.

Note that instead of using the **Tag** property, we could also have derived a custom class from **C1TreeViewNode** and built all the delay-load logic into that class. This would be a more elegant approach, but unfortunately WPF doesn't support template inheritance. If you derive a class from a class that has a template (such as **Button** or **C1TreeViewNode**), the template is not inherited, and you have to provide a template yourself or your derived class will be just an empty control.

Node Selection

When you click on a node at run time it is automatically marked as selected. Clicking a node will raise the SelectionChanged event to provide custom functionality. To have the nodes marked as selected without clicking them you can enable the IsSelected property.

When the user selects a new item, the **C1TreeView** fires the SelectionChanged event. You can then retrieve the item that was selected using the SelectedItem property.

There're several ways to do this. One is to assign additional data to the **Tag** property of each **C1TreeViewItem** as you create them. Later, you can inspect the **Tag** property to retrieve the information. For example:

- Visual Basic

```
' Create a node and assign some data to its Tag property
Dim item As New C1TreeViewItem()
item.Header = "Beverages"
item.Tag = beveragesID
```

- C#

```
// Create a node and assign some data to its Tag property
C1TreeViewItem item = new C1TreeViewItem();
item.Header = "Beverages";
item.Tag = beveragesID;
```

Later, use the information in whatever way you see fit:

- Visual Basic

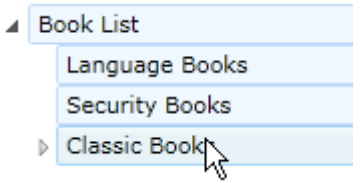
```
Dim item As C1TreeViewItem = _tv.SelectedItem
    ' Handle beverages node
If TypeOf item.Tag Is Integer AndAlso CInt(item.Tag) = beveragesID Then
End If
```

- C#

```
C1TreeViewItem item = _tv.SelectedItem;
if (item.Tag is int && (int)item.Tag == beveragesID)
```

```
{
    // Handle beverages node
}
```

If the SelectionMode property is set to Multiple then multiple nodes can be selected at one time by holding down the control key while mouse clicking multiple nodes. To unselect a node, click on it again. The nodes are marked as selected in the following **C1TreeView**:



Node Navigation

C1TreeView supports mouse and keyboard navigation.

Navigating C1TreeViewNodes using the mouse

The following table describes the actions and corresponding mouse commands when navigating through the **C1TreeViewNode**s:

Action	Mouse Command
Expand a node	Click on the plus sign at the left of the node's name.
Collapse a node	Click on the minus sign at the left of the node's name.
Select a node	Click on the node's name.

Navigating C1TreeViewNodes using the keyboard

The following table describes the actions and their associated keys to use when navigating through **C1TreeViewNode**s:

Action	Keyboard Command
Expand a node	+ KEY
Collapse a node	- KEY
Move up a node	UP ARROW KEY
Move down a down	DOWN ARROW KEY
Select multiple nodes	MOUSE + CTRL KEY

TreeView for WPF Layout and Appearance

The following topics detail how to customize the C1TreeView control's layout and appearance. You can use built-in layout options to lay your controls out in panels such as Grids or Canvases. Themes allow you to customize the appearance of the treeview and take advantage of WPF's XAML-based styling.

You can customize the appearance of your treeview item by using the **Header** property. You can set the **Header** to be any object such as a stack panel containing an image and some text:

```
<C1TreeViewItem>
   <C1:C1TreeViewItem.Header>
      <StackPanel Orientation="Horizontal">
         <Image Source="myImage,jpg"/>
         <TextBlock Text="My Text"/>
      </StackPanel>
   </C1:C1TreeViewItem.Header>
</C1:C1TreeViewItem>
```

TreeView for WPF Appearance Properties

ComponentOne TreeView 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 C1TreeView control.

Property	Description
FontFamily	Gets or sets the font family of the control. This is a dependency property.
FontSize	Gets or sets the font size. This is a dependency property.
FontStretch	Gets or sets the degree to which a font is condensed or expanded on the screen. This is a dependency property.
FontStyle	Gets or sets the font style. This is a dependency property.
FontWeight	Gets or sets the weight or thickness of the specified font. This is a dependency property.
C1HierarchicalPresenter.Header	Gets or sets the item that labels the control.

Content Positioning Properties

The following properties let you customize the position of header and content area content in the C1TreeView control.

Property	Description
Padding	Gets or sets the padding inside a control. This is a dependency property.
HorizontalAlignment	Gets or sets the horizontal alignment characteristics applied to this element when it is composed within a parent element, such as a panel or items control. This is a dependency property..
HorizontalContentAlignment	Gets or sets the horizontal alignment of the control's content. This is a dependency property.

VerticalAlignment	Gets or sets the vertical alignment characteristics applied to this element when it is composed within a parent element such as a panel or items control. This is a dependency property.
VerticalContentAlignment	Gets or sets the vertical alignment of the control's content. This is a dependency property.

Color Properties

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

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

Border Properties

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

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

Size Properties

The following properties let you customize the size of the **C1TreeView** control.

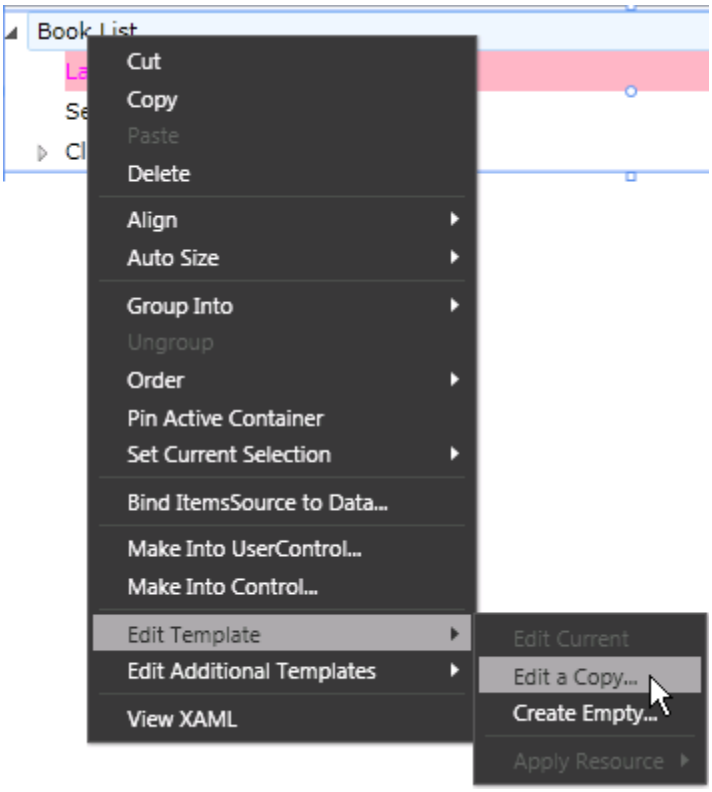
Property	Description
Height	Gets or sets the suggested height of the element. This is a dependency property.
MaxHeight	Gets or sets the maximum height constraint of the element. This is a dependency property.
MaxWidth	Gets or sets the maximum width constraint of the element. This is a dependency property.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

C1TreeView Templates

One of the main advantages to using a 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 TreeView 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 C1TreeView control and, in the menu, selecting **Edit Template**. Select **Edit a Copy** to create an editable copy of the current template or **Create Empty**, to create a new blank template.

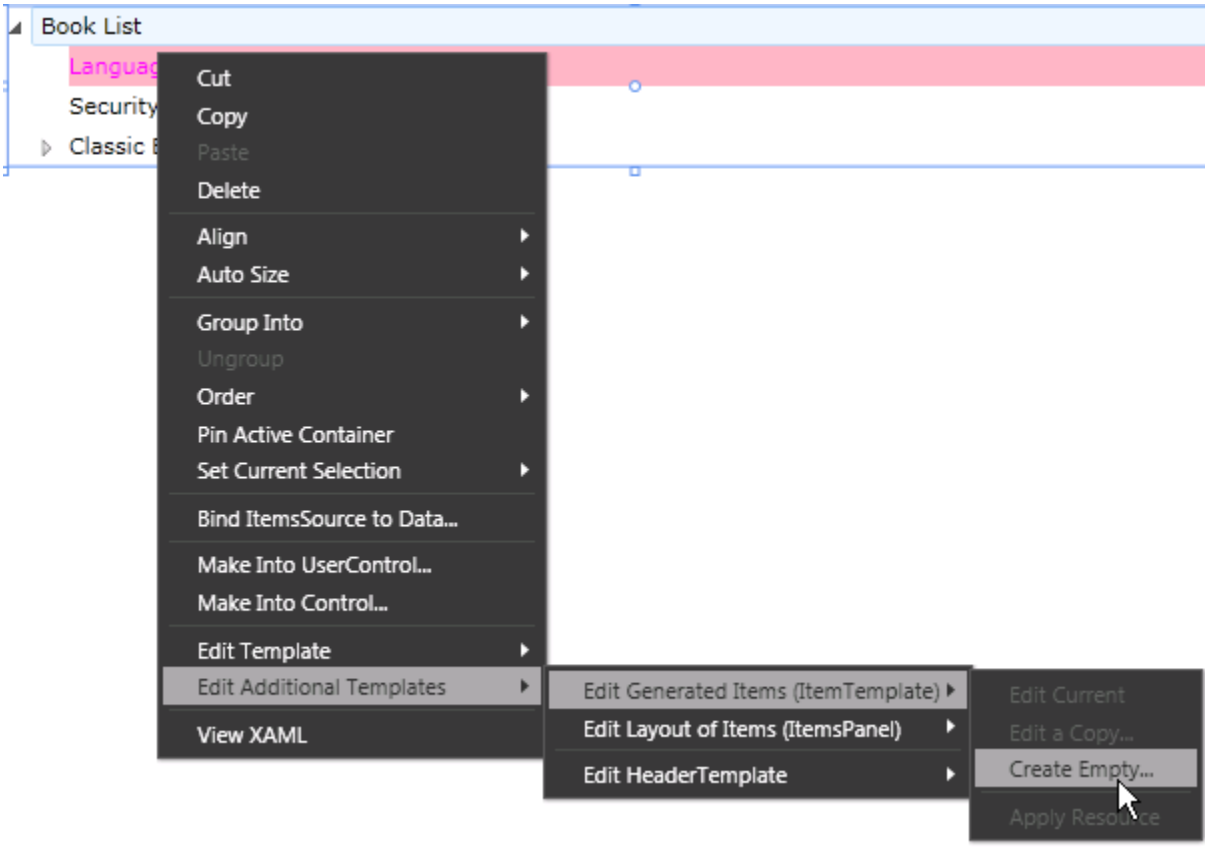


Once you've created a new template, the template will appear in the **Objects and Timeline** window. Note that you can use the Template property to customize the template.

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

Additional Templates

In addition to the default template, the C1TreeView control includes a few additional templates. These additional templates can also be accessed in Microsoft Expression Blend – in Blend select the C1TreeView control and, in the menu, select **Edit Additional Templates**. Choose a template, and select **Create Empty**:



C1TreeView Styles

ComponentOne TreeView for WPF's C1TreeView control provides several style properties that you can use to change the appearance of the control. Some of the included styles are described in the table below:

Style	Description
Style	Gets or sets the style used by this element when it is rendered. This is a dependency property.

TreeView ClearStyle Properties

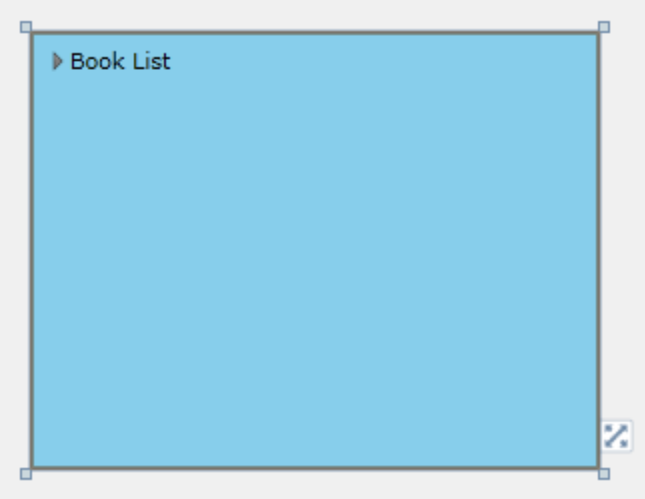
TreeView for WPF supports ComponentOne's ClearStyle technology, which allows you to easily change control colors without having to change control templates. By setting a few color properties, you can quickly style the **C1TreeView** control. The supported properties for **C1TreeView** are listed in the following table:

Property	Description
Background	Gets or sets the background used to fill the **C1DockControl**.
MouseOverBrush	Gets or sets the brush used to highlight the control when the mouse is hovering over it.
SelectedBackground	Gets or sets the background color of the selected item.

You can completely change the appearance of the **C1TreeView** by setting these properties. For example, if you set the **C1TreeView.Background** property to **SkyBlue** so the XAML markup appears similar to the following:

```
<c1:C1TreeView Background="SkyBlue" x:Name="Tree">
    <c1:C1TreeViewItem Header="Book List">
        <c1:C1TreeViewItem Header="Language Books"/>
        <c1:C1TreeViewItem Header="Security Books"/>
        <c1:C1TreeViewItem Header="Classic Books">
            <c1:C1TreeViewItem Header="Catch-22"/>
            <c1:C1TreeViewItem Header="The Great Gatsby"/>
        </c1:C1TreeViewItem>
    </c1:C1TreeViewItem>
</c1:C1TreeView>
```

The **C1TreeView** will look similar to the following:



Experiment with the other properties to quickly change the look of the **C1TreeView** elements. For example, the following XAML sets the **Background**, **MouseOverBrush**, and **SelectedBackground** properties:

```
<c1:C1TreeView Background="SkyBlue" MouseOverBrush="Gold"
SelectedBackground="Green" x:Name="Tree">
    <c1:C1TreeViewItem Header="Book List">
        <c1:C1TreeViewItem Header="Language Books"/>
        <c1:C1TreeViewItem Header="Security Books"/>
        <c1:C1TreeViewItem Header="Classic Books">
            <c1:C1TreeViewItem Header="Catch-22"/>
            <c1:C1TreeViewItem Header="The Great Gatsby"/>
        </c1:C1TreeViewItem>
    </c1:C1TreeViewItem>
</c1:C1TreeView>
```

The **C1TreeView** will look similar to the following when you run the project:

When you mouse over a header, it appears yellow. When you select an item, it appears green.

TreeView for 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 C1TreeView control in general. If you are unfamiliar with the **ComponentOne TreeView for WPF** product, please see the **TreeView for WPF** Quick Start first.

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

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

Adding Check Boxes to the TreeView

You can easily add check boxes to the **C1TreeView** control, check boxes can appear before text and allow users to select tree view items. The following XAML markup adds check boxes to the **C1TreeView**:

- XAML

```
<c1:C1TreeView Name="C1TreeView1" Height="300" Width="200" >
    <c1:C1TreeViewItem IsExpanded="True" Margin="10">
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
                    <TextBlock Text="Desktop" />
                </CheckBox.Content>
            </CheckBox>
        </c1:C1TreeViewItem.Header>
    <c1:C1TreeViewItem>
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
                    <TextBlock Text="User" />
                </CheckBox.Content>
            </CheckBox>
        </c1:C1TreeViewItem.Header>
    </c1:C1TreeViewItem>
    <c1:C1TreeViewItem>
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
                    <TextBlock Text="Public" />
                </CheckBox.Content>
            </CheckBox>
```

```
        </c1:C1TreeViewItem.Header>
        <c1:C1TreeViewItem>
            <c1:C1TreeViewItem.Header>
                <CheckBox>
                    <CheckBox.Content>
                    <TextBlock Text="Favorites" />
                    </CheckBox.Content>
                </CheckBox>
            </c1:C1TreeViewItem.Header>
    </c1:C1TreeViewItem>
        <c1:C1TreeViewItem>
            <c1:C1TreeViewItem.Header>
                <CheckBox>
                    <CheckBox.Content>
                    <TextBlock Text="Public Downloads" />
                    </CheckBox.Content>
                </CheckBox>
            </c1:C1TreeViewItem.Header>
    </c1:C1TreeViewItem>
        <c1:C1TreeViewItem>
            <c1:C1TreeViewItem.Header>
                <CheckBox>
                    <CheckBox.Content>
                    <TextBlock Text="Public Music" />
                    </CheckBox.Content>
                </CheckBox>
            </c1:C1TreeViewItem.Header>
    </c1:C1TreeViewItem>
        <c1:C1TreeViewItem>
            <c1:C1TreeViewItem.Header>
                <CheckBox>
                    <CheckBox.Content>
                    <TextBlock Text="Public Pictures" />
                    </CheckBox.Content>
                </CheckBox>
            </c1:C1TreeViewItem.Header>
    </c1:C1TreeViewItem>
        <c1:C1TreeViewItem>
            <c1:C1TreeViewItem.Header>
                <CheckBox>
                    <CheckBox.Content>
                    <TextBlock Text="Public Videos" />
                    </CheckBox.Content>
                </CheckBox>
            </c1:C1TreeViewItem.Header>
    </c1:C1TreeViewItem>
            </c1:C1TreeViewItem>
            <c1:C1TreeViewItem IsExpanded="True">
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
                    <TextBlock Text="Computer" />
                </CheckBox.Content>
            </CheckBox>
        </c1:C1TreeViewItem.Header>
                <c1:C1TreeViewItem IsExpanded="True">
            <c1:C1TreeViewItem.Header>
```

```
                <CheckBox>
                    <CheckBox.Content>
                    <TextBlock Text="Local Disk (C:)" />
                    </CheckBox.Content>
                </CheckBox>
            </c1:C1TreeViewItem.Header>
            <c1:C1TreeViewItem>
                <c1:C1TreeViewItem.Header>
                    <CheckBox>
                        <CheckBox.Content>
                        <TextBlock Text="Program Files" />
                        </CheckBox.Content>
                    </CheckBox>
                </c1:C1TreeViewItem.Header>
        </c1:C1TreeViewItem>
            <c1:C1TreeViewItem>
                <c1:C1TreeViewItem.Header>
                    <CheckBox>
                        <CheckBox.Content>
                            <TextBlock Text="Users" />
                        </CheckBox.Content>
                    </CheckBox>
                </c1:C1TreeViewItem.Header>
        </c1:C1TreeViewItem>
            <c1:C1TreeViewItem>
                <c1:C1TreeViewItem.Header>
                    <CheckBox>
                        <CheckBox.Content>
                            <TextBlock Text="Windows" />
                        </CheckBox.Content>
                    </CheckBox>
                </c1:C1TreeViewItem.Header>
        </c1:C1TreeViewItem>
    </c1:C1TreeViewItem>
        <c1:C1TreeViewItem>
            <c1:C1TreeViewItem.Header>
                <CheckBox>
                    <CheckBox.Content>
                        <TextBlock Text="DVD Drive (D:)" />
                    </CheckBox.Content>
                </CheckBox>
            </c1:C1TreeViewItem.Header>
        </c1:C1TreeViewItem>
    </c1:C1TreeViewItem>
    <c1:C1TreeViewItem>
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
                    <TextBlock Text="Network" />
                </CheckBox.Content>
            </CheckBox>
        </c1:C1TreeViewItem.Header>
            </c1:C1TreeViewItem>
    <c1:C1TreeViewItem>
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
```

```
                    <TextBlock Text="Control Panel" />
                </CheckBox.Content>
            </CheckBox>
        </c1:C1TreeViewItem.Header>
            </c1:C1TreeViewItem>
    <c1:C1TreeViewItem>
        <c1:C1TreeViewItem.Header>
            <CheckBox>
                <CheckBox.Content>
                    <TextBlock Text="Recycle Bin" />
                </CheckBox.Content>
            </CheckBox>
        </c1:C1TreeViewItem.Header>
        </c1:C1TreeViewItem>
    </c1:C1TreeViewItem>
</c1:C1TreeView>
```

Getting the Text or Value of the SelectedItem in a TreeView

The **Header** property will return the values contained in your **C1TreeViewItem**s, you can get the string value using the following code:

- Visual Basic

```
Dim item As C1TreeViewItem = _tree.SelectedItem
_textBlock.Text = item.Header.ToString()
```

- C#

```
C1TreeViewItem item = _tree.SelectedItem;
_textBlock.Text = item.Header.ToString();
```

Enabling Drag-and-Drop

C1TreeView supports drag-and-drop behavior. For more information see the Drag-and-Drop Nodes (page 10) topic. To enable drag-and-drop behavior, set the AllowDragDrop property to **True**:

- Visual Basic

```
C1TreeView.AllowDragDrop = True
```

- C#

```
C1TreeView.AllowDragDrop = true;
```

To enable visual drag-and-drop indicators you can set the DragDropArrowMarker and DragDropLineMarker properties.