ComponentOne

TreeView for WinForms

GrapeCity US

GrapeCity
201 South Highland Avenue, Suite 301
Pittsburgh, PA 15206
Tel: 1.800.858.2739 | 412.681.4343
Fax: 412.681.4384
Website: https://www.grapecity.com/en/
E-mail: us.sales@grapecity.com

Trademarks

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $2 5 (to cover postage and handling) to ComponentOne.

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

Copying and Distribution

Table of Contents

Overview

ComponentOne Studio introduces **TreeView for WinForms**, a control to help users display a hierarchical list of items such as indexed entries, directories on a disk, headings in a document, etc. The TreeView control manages data through nodes that can be selected, edited, and used to display simple text, images, and other elements such as check boxes.

For more information on TreeView control, refer to the following links:

- Key Features
- Quick Start
- Design-Time Support
- Structure and Elements
- Columns
- Nodes
- Data Binding
- Styling and Appearance
- Import and Export XML
- API Reference

Getting Started with WinForms Edition

For information on installing ComponentOne Studio WinForms Edition, licensing, technical support, namespaces, and creating a project with the FlexChart control, visit Getting Started with WinForms Edition.

Key Features

TreeView for WinForms consists of the following key features:

- **Node presentation**

  Node items can be implemented as checkboxes, icons or multiple icons, or text.

- **Node editing**

  You can edit a node simply by setting the AllowEditing property to True and pressing F2 after selecting the node. You can also cancel editing by pressing the Escape key.

- **Unbound mode**

  You can manually create a tree of nodes at design-time as well as run-time, thereby representing any data in a hierarchical form.

- **Bound mode**

  You can bind TreeView for WinForms to multiple tables with hierarchical relations to display hierarchical data. Also, you can bind the control to a self referencing data like a single list instead of multiple related lists.

- **Themes support**

  TreeView for WinForms uses C1ThemeController and offers theme support.

- **Node selection and navigation**

  TreeView supports both keyboard and mouse navigation of nodes. In addition, the control supports single and multiple node selection. You can select a node at both design-time and run-time. And you can select multiple nodes in a contiguous or a non-contiguous manner.

- **Design-time support**

  TreeView lets you access collection editors right from the design form using Smart tag and context menu. The collection editors allow you to add, remove nodes and columns without writing code.

- **Node expansion**

  You can expand either a single node or all nodes in the tree as per the requirement. You can also fully expand the tree in one go using the ExpandAll method at the TreeView level. And you can even prevent end-users from collapsing the expanded tree by cancelling the Collapsing event.

Quick Start

This quick start illustrates how to create a simple TreeView application either by using designer or through programming in .NET.

Perform the following steps to walk through TreeView for WinForms quickly:

1. Adding TreeView to the Application
2. Creating Nodes in TreeView
3. Running the Application

Step 1: Adding TreeView to the Application

To add the TreeView control to your application, complete the following steps:

1. Create a new **Windows Forms Application**.
2. In the **Design** view, drag and drop the **C1TreeView** control to the form from the Toolbox.
   The **TreeView** control appears with **Column1** added by default.
3. Click the **Smart Tag**, and select **Dock in Parent Container**.
   You have successfully added the **TreeView** control to the application.

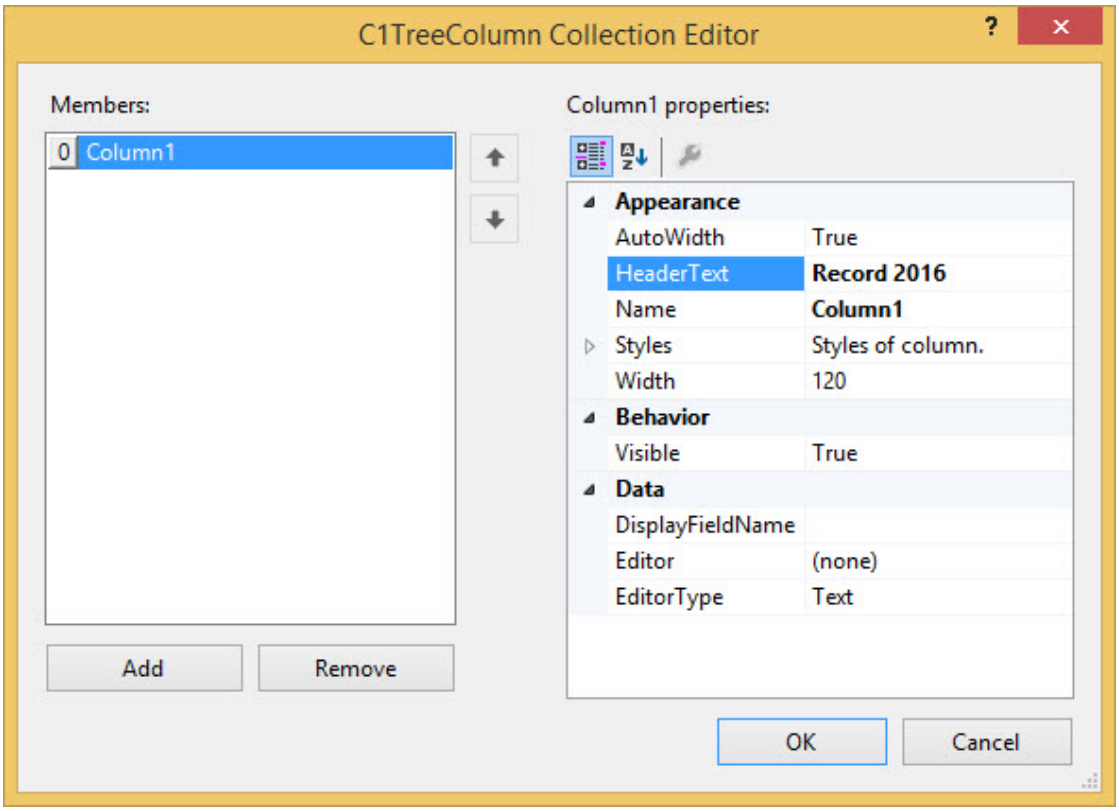
Step 2: Creating Nodes in TreeView

You can create nodes in TreeView either by using designer or through code in your application.

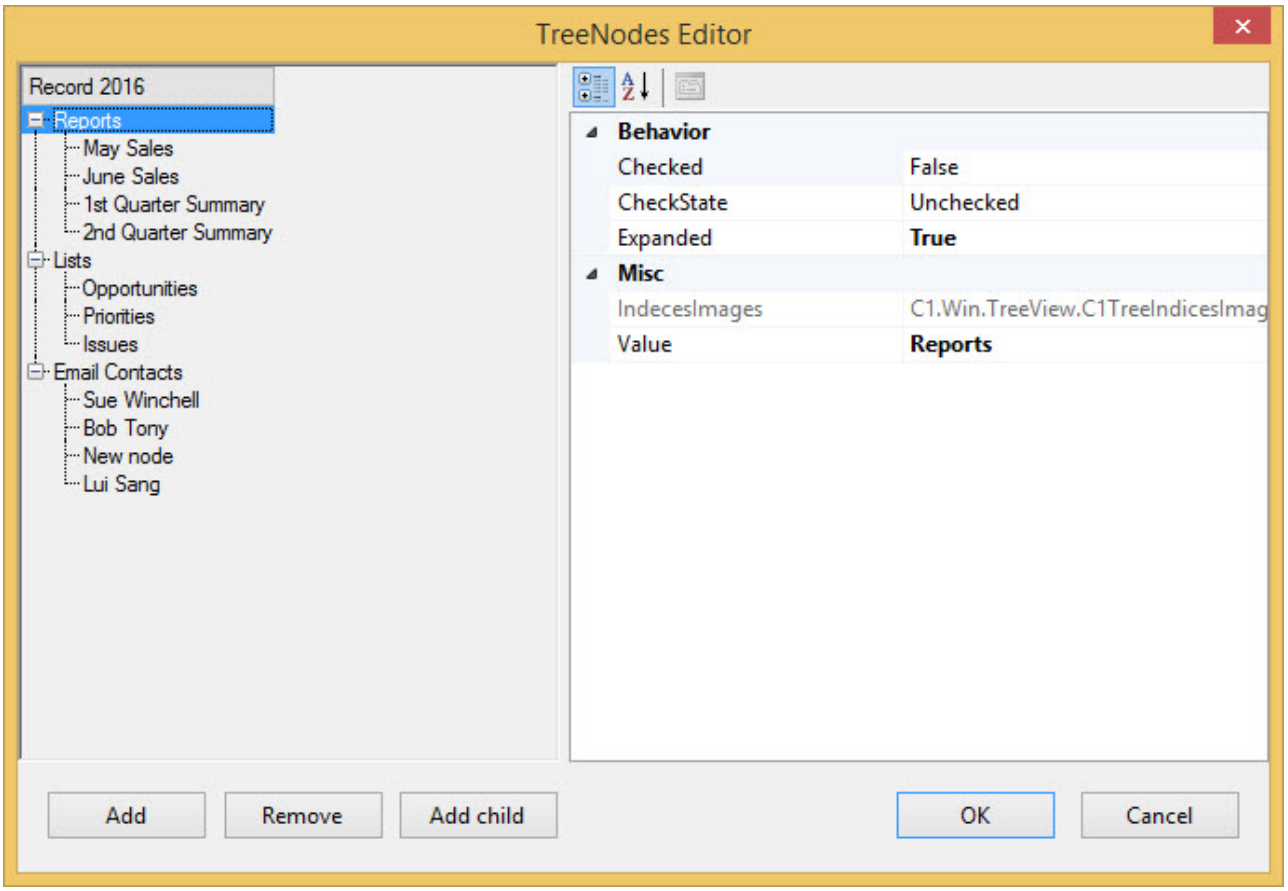
Using Designer

To create nodes in TreeView using designer, complete the following steps:

1. Click the **Smart Tag**, and select **Edit columns**.
   C1TreeColumn Collection Editor appears.
2. Enter the relevant name and header text in the **Name** and the **HeaderText** fields respectively.



3. Close **C1TreeColumn Collection Editor.**
4. Click the Smart Tag, and select **Edit nodes**.
   TreeNodes Editor appears.
5. Click **Add** to create a top-level node.
6. Specify a custom label for the node in the **Value** property.
7. Click **Add child** to add a child node to the selected node.
8. Specify a custom label for the child node in the **Value** property.
9. Repeat steps 7 to 10 as required in the application.



10. Click **OK** to save the changes.

Through Code

To create nodes in TreeView programmatically, complete the following steps:

1. Create new instances of a node.
2. Add the parent node to the TreeView nodes collection.
3. Set the value of the parent node.
    - **Visual Basic**

```
' create new instances of node
Dim node1 As New C1.Win.TreeView.C1TreeNode()
Dim node2 As New C1.Win.TreeView.C1TreeNode()

' add parent node to the TreeView nodes collection
C1TreeView1.Nodes.Add(node1)

' set the value of parent node
node1.SetValue("Reports")
```

    - **C#**

```
// create new instances of node
C1.Win.TreeView.C1TreeNode node1 = new C1.Win.TreeView.C1TreeNode();
C1.Win.TreeView.C1TreeNode node2 = new C1.Win.TreeView.C1TreeNode();

// add parent node to the TreeView nodes collection
c1TreeView1.Nodes.Add(node1);

// set the value of parent node
node1.SetValue("Reports");
```

4. Add the child node to the parent node.

5. Set the value of the child node.

    - **Visual Basic**

```
' add child node to parent node
node1.Nodes.Add(node2)

'set the value of child node
node2.SetValue("May Sales")
```

    - **C#**

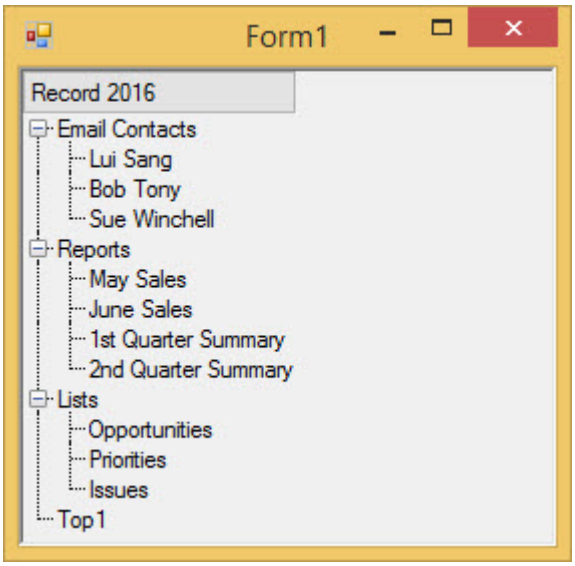
```
// add child node to parent node
node1.Nodes.Add(node2);

//set the value of child node
node2.SetValue("May Sales");
```

6. Repeat steps 3 to 7 to add more parent nodes and child nodes.

Step 3: Running the Application

Now that you have successfully created nodes in TreeView, you just need to run the application.

The following output appears once you have run the application.



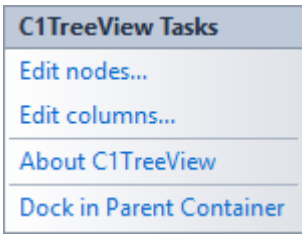
Design-Time Support

TreeView for WinForms provides design-time support that allows you to access collection editors right from the form. To know more about collection editors, refer to Collection Editors.

Smart Tag

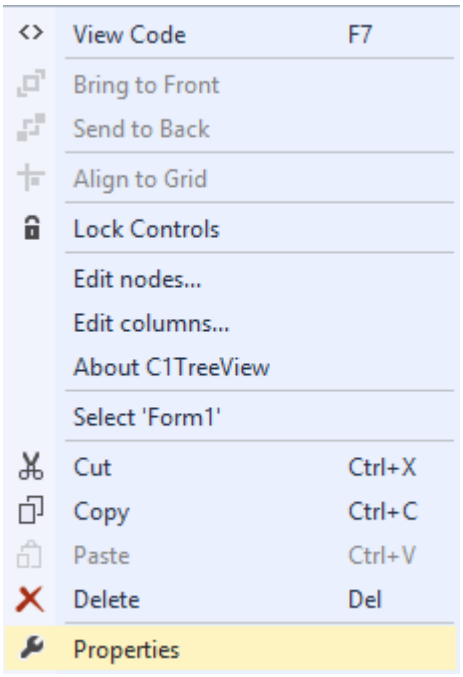
The smart tag for TreeView enables you to quickly access common **C1TreeView Tasks** involved with building TreeView elements.

- **Edit nodes:** Allows you to access **TreeNodes Editor** in just one click. For more information on TreeNodes Editor, refer to Collection Editors.
- **Edit columns:** Allows you to access **C1TreeColumn Collection Editor** in a single click. For more information on C1TreeColumn Collection Editor, refer to Collection Editors.
- **About C1.Win.TreeView:** Displays a dialog box that is helpful to find the version number.
- **Dock in Parent Container:** Docks TreeView into the parent container.



Context Menu

You can access the context menu by right-clicking on the TreeView control at design-time. The context menu, just like smart tag, provides shortcuts to access collection editors right from the design form.



Collection Editors

TreeView for WinForms includes two major collection editors- TreeNodes Editor and C1TreeColumn Collection Editor.

TreeNodes Editor

TreeNodes Editor enables you to add, remove parent nodes and child nodes in or from the Nodes collection of C1TreeView and C1TreeNode easily. Using TreeNodes Editor allows you to not only modify the behavior of the nodes, but also reorder them as per the requirement.

Following are the ways to access TreeNodes Editor:

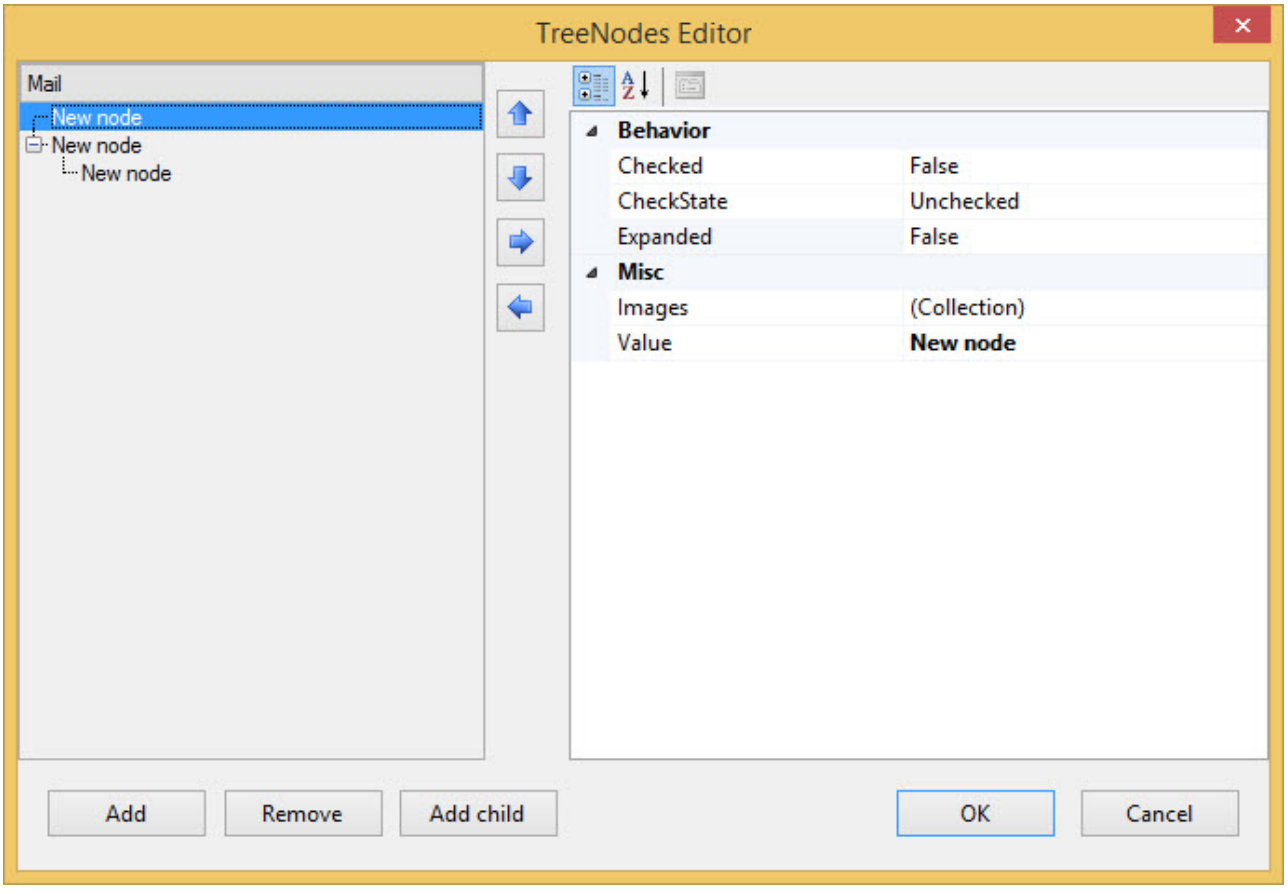
Using Smart Tag and Context Menu

Refer to Design-Time Support for more information.

Through the Properties Window

Perform the following steps to access TreeNodes Editor through the Properties window:

1. Right-click on the **TreeView** control.
   The context menu appears.
2. Select **Properties**.
3. Navigate to the **Nodes** field in the **Properties** window.
4. Click the **Ellipsis** button next to the **Nodes** field.
   TreeNodes Editor appears.



C1TreeColumn Collection Editor

C1TreeColumn Collection Editor allows you to add and remove columns in or from TreeView seamlessly. The editor lets you customize the appearance and modify the behavior and data for the added columns.

The editor can be accessed in the following ways:

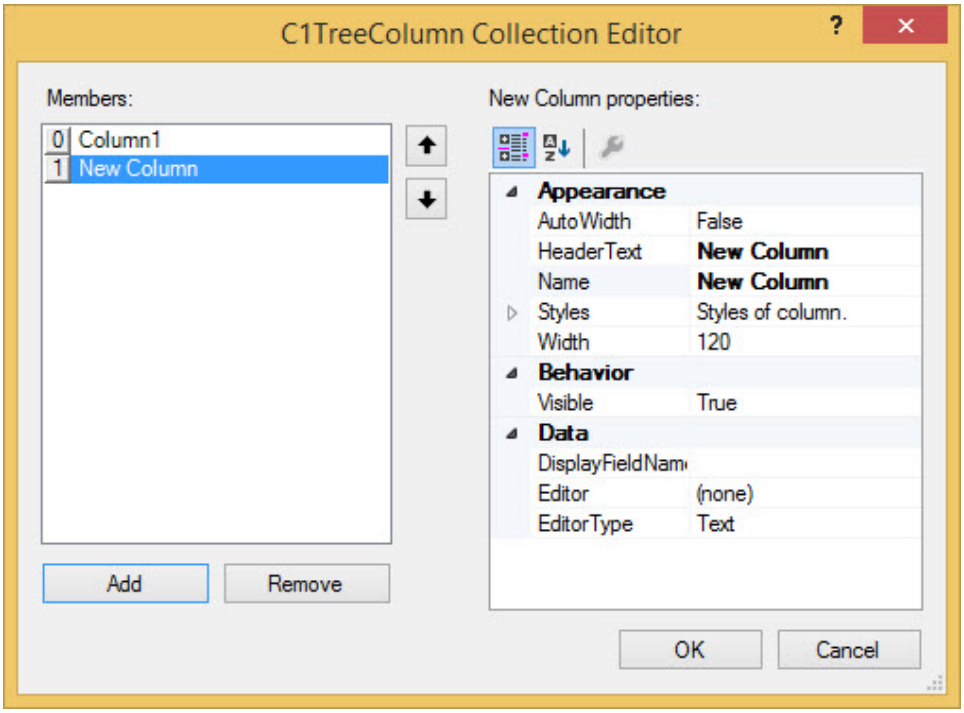
Using Smart Tag and Context Menu

Refer to Design-Time Support for more information.

Through the Properties Window

Perform the following steps to access TreeNodes Editor through the Properties window:

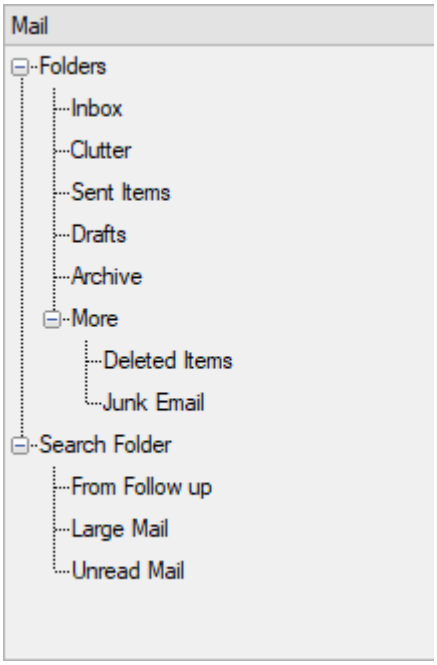
1. Right-click on the **TreeView** control.
   The context menu appears.
2. Select **Properties**.
3. Navigate to the **Columns** field in the **Properties** window.
4. Click the **Ellipsis** button next to the **Columns** field.
   C1TreeColumn Collection Editor appears.



Structure and Elements

TreeView is a tree type WinForms control that comprises one or more elements known as nodes. A node can be classified into the following three types:

- Parent node that contains other nodes
- Child node that is contained by another node
- Leaf node that does not contain child nodes



Columns

There are two types of columns in the TreeView control, bound and unbound. In the unbound mode, a column is not bound to the fields of the data source and displays the string representation of the node. In the bound mode, a column obtains its data from the TreeView's data source and displays the underlying data.

TreeView contains a single column by default. However, you can add multiple columns to the Columns collection of TreeView to use the control in different scenarios. For instance, you may want to present sales data of different regions for two consecutive years by using multiple columns.

The control allows you to add columns easily at design-time with C1TreeColumn Collection Editor. Refer to Collection Editors for more information.

To create columns through code, you need to create an instance of C1TreeColumn and add it to the Columns collection of C1TreeView. To remove all columns from the Columns collection of TreeView, you can use the Clear method of C1TreeColumnCollection. Also, you can remove the first occurrence of a specified column from the collection by using the Remove method that accepts the C1TreeColumn type instance as a parameter. And you can remove a column at the specified index by using the RemoveAt method that accepts the column index value of the Integer type as a parameter.

The following code snippet creates a column, specifies its header and name, and adds the column to the Columns collection of TreeView.

- **Visual Basic**

```
' clear the Columns collection
C1TreeView1.Columns.Clear()

' create a column
Dim Column1 As New C1.Win.TreeView.C1TreeColumn()

' name the column header
Column1.HeaderText = "Column1"

' name the column
Column1.Name = "Column1"

' add the column to the Columns collection of TreeView
C1TreeView1.Columns.Add(Column1)
```

- **C#**

```
// clear the Columns collection
c1TreeView1.Columns.Clear();

// create a column
C1.Win.TreeView.C1TreeColumn Column1 = new C1.Win.TreeView.C1TreeColumn();

// name the column header
Column1.HeaderText = "Column1";

// name the column
Column1.Name = "Column1";

// add the column to the Columns collection of TreeView
c1TreeView1.Columns.Add(Column1);
```

Nodes

In TreeView, nodes are arranged in a hierarchical form. Internally, the C1TreeNode class represents a node, provides access to its data, and allows many operations to be performed on it. A few of them include adding, removing, reordering, editing, navigating, and selecting nodes. Representing data in a tree-like structure, the TreeView control stores nodes as nested collections. The collection of nodes can be accessed through the Nodes property of C1TreeView. In the collection, each node can have its own child nodes accessible through the Nodes property of C1TreeNode.

The following topics explain different ways of working with nodes in **TreeView for WinForms**:

- Adding and Removing Nodes
- Reordering Nodes
- Selecting Nodes
- Navigating Nodes
- Using Checkboxes
- Editing Nodes
- Expanding and Collapsing Nodes
- Dragging and Dropping Nodes
- Custom Editors
- Custom Nodes

Adding and Removing Nodes

Adding Nodes

The C1TreeNodeCollection class provides the Add method and the Insert method for adding nodes in TreeView. To add a parent node, you can use either of the two methods with the Nodes collection of the C1TreeView class. Next, to add a child node in the parent node, or children nodes in a child node, you can either use the Add method or the Insert method with the Nodes collection of the C1TreeNode class.

The following code snippet demonstrates how to add nodes in the TreeView control using the Add method.

- **Visual Basic**

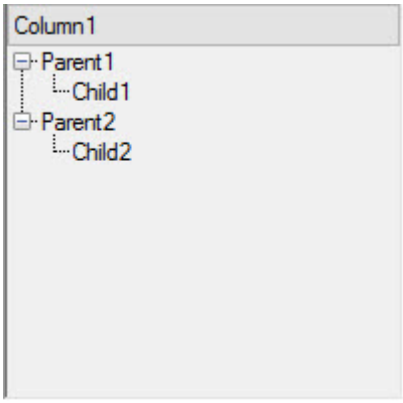
```
' create parent nodes
Dim parentNode1 As New C1.Win.TreeView.C1TreeNode()
Dim parentNode2 As New C1.Win.TreeView.C1TreeNode()

' add the parent nodes to the TreeView Nodes collection
C1TreeView1.Nodes.Add(parentNode1)
C1TreeView1.Nodes.Add(parentNode2)

' set the values of the parent nodes
parentNode1.SetValue("Parent1", 0)
parentNode2.SetValue("Parent2", 0)

' create child nodes
Dim childNode1 As New C1.Win.TreeView.C1TreeNode()
Dim childNode2 As New C1.Win.TreeView.C1TreeNode()

' add the child nodes to the parent nodes' Nodes collection
parentNode1.Nodes.Add(childNode1)
```

```
parentNode2.Nodes.Add(childNode2)

' set the values of the child nodes
childNode1.SetValue("Child1", 0)
childNode2.SetValue("Child2", 0)
```

- **C#**

```
// create parent nodes
C1.Win.TreeView.C1TreeNode parentNode1 = new C1.Win.TreeView.C1TreeNode();
C1.Win.TreeView.C1TreeNode parentNode2 = new C1.Win.TreeView.C1TreeNode();

// add the parent nodes to the TreeView Nodes collection
c1TreeView1.Nodes.Add(parentNode1);
c1TreeView1.Nodes.Add(parentNode2);

// set the values of the parent nodes
parentNode1.SetValue("Parent1", 0);
parentNode2.SetValue("Parent2", 0);

// create child nodes
C1.Win.TreeView.C1TreeNode childNode1 = new C1.Win.TreeView.C1TreeNode();
C1.Win.TreeView.C1TreeNode childNode2 = new C1.Win.TreeView.C1TreeNode();

// add the child nodes to the parent nodes' Nodes collection
parentNode1.Nodes.Add(childNode1);
parentNode2.Nodes.Add(childNode2);

// set the values of the child nodes
childNode1.SetValue("Child1", 0);
childNode2.SetValue("Child2", 0);
```

The following image shows the treeview after adding nodes.



The following code snippet demonstrates how to add nodes in the TreeView control using the Insert method.

- **Visual Basic**

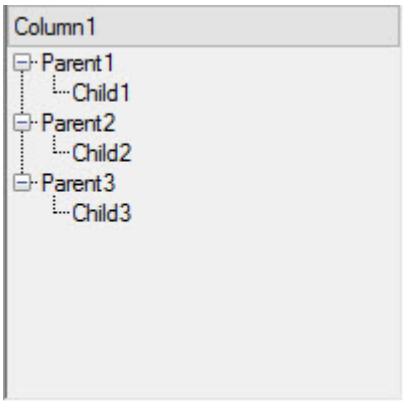
```
' create third parent node
Dim parentNode3 As New C1.Win.TreeView.C1TreeNode()

' insert the third parent node in the TreeView Nodes collection
C1TreeView1.Nodes.Insert(2, parentNode3)

' set the value of the third parent node
parentNode3.SetValue("Parent3", 0)

' create third child node
Dim childNode3 As New C1.Win.TreeView.C1TreeNode()
```

```
' insert the third child node in the third parent node's Nodes collection
parentNode3.Nodes.Insert(0, childNode3)

' set the value of the third child node
childNode3.SetValue("Child3", 0)
```

- **C#**

```
// create third parent node
C1.Win.TreeView.C1TreeNode parentNode3 = new C1.Win.TreeView.C1TreeNode();

// insert the third parent node in the TreeView Nodes collection
c1TreeView1.Nodes.Insert(2, parentNode3);

// set the value of the third parent node
parentNode3.SetValue("Parent3", 0);

// create third child node
C1.Win.TreeView.C1TreeNode childNode3 = new C1.Win.TreeView.C1TreeNode();

// insert the third child node in the third parent node's Nodes collection
parentNode3.Nodes.Insert(0, childNode3);

// set the value of the third child node
childNode3.SetValue("Child3", 0);
```

The final treeview appears, as shown below.



Removing Nodes

The C1TreeNodeCollection class provides the Remove method and the RemoveAt method for removing nodes from TreeView. You can use either of the two methods with the Nodes collection of C1TreeView to remove parent nodes from TreeView. To remove a child node from a parent node or children nodes from a child node, you can use either of the two methods with the Nodes collection of C1TreeNode.

The Remove method and the RemoveAt method removes a single node at a time. However, to remove all nodes simultaneously from the Nodes collection of TreeView, a parent node, or a child node, you can use the Clear method with the Nodes collection of C1TreeView or C1TreeNode, respectively.

Here is the code snippet demonstrating how to set these methods.

- **Visual Basic**

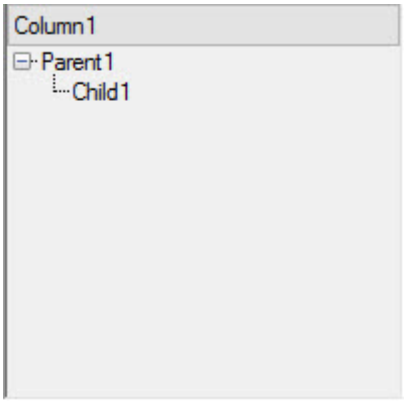
```
' remove the second parent node from the TreeView Nodes collection
C1TreeView1.Nodes.Remove(parentNode2)

' remove the node at index 1
```

```
C1TreeView1.Nodes.RemoveAt(1)
```

- **C#**

```
// remove the second parent node from the TreeView Nodes collection
c1TreeView1.Nodes.Remove(parentNode2);

// remove the node at index 1
c1TreeView1.Nodes.RemoveAt(1);
```

After removing the nodes, the treeview appears like the following.



Reordering Nodes

In TreeView, you can programmatically reorder nodes using the Move method of the C1TreeNodeCollection class. The Move method can be used with the Nodes collection of either C1TreeView or C1TreeNode. It allows you to move a node from its old position to a new position. The method accepts two parameters, oldIndex and newIndex of the Integer type. Here oldIndex and newIndex specify the old and the new position of the node respectively. The Move method, however, cannot be used to move a child node from one parent node to another.

The following code snippets moves Child1 from index 0 to index 2 in Parent1.

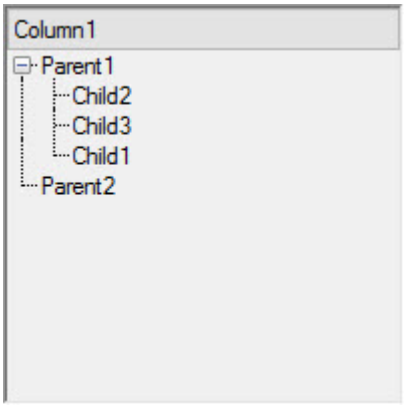
- **Visual Basic**

```
' move a node from its old position to a new position
parentNode1.Nodes.Move(0, 2)
```

- **C#**

```
// move a node from its old position to a new position
parentNode1.Nodes.Move(0, 2);
```

Here is an image showing the same.



Selecting Nodes

To select TreeView nodes in different ways, you need to set the SelectionMode property of C1TreeView. The SelectionMode property accepts the following values from the C1TreeViewSelectionMode enumeration:

- **Multi**: Allows you to select multiple nodes (parent or child) in TreeView. You can select multiple nodes either in a contiguous or a non-contiguous manner. To select multiple nodes in a contiguous manner, hold the SHIFT key while clicking on nodes. And to select nodes in a non-contiguous manner, hold the CTRL key while clicking on nodes.
- **MultiSibling**: Allows you to select multiple nodes in a contiguous or a non-contiguous manner using SHIFT or CTRL keys respectively, within a parent node only.
- **None**: Does not allow you to select any node in TreeView.
- **Single (default value)**: Allows you to select only one node (parent or child) in TreeView at a time.

Below is the sample code snippet setting the SelectionMode property.

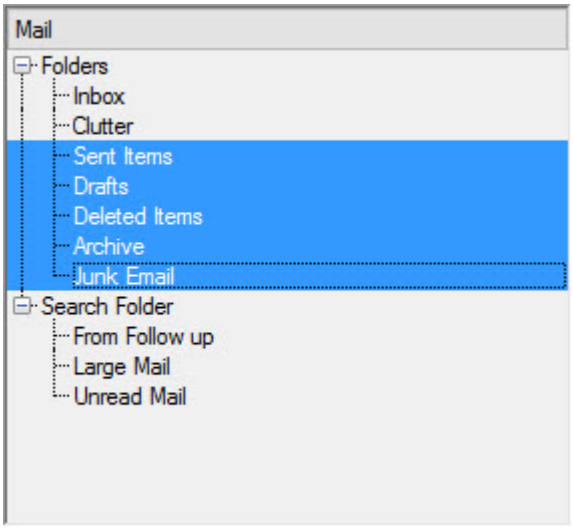
- **Visual Basic**

```
' set the SelectionMode property to Multi
C1TreeView1.SelectionMode = C1.Win.TreeView.C1TreeViewSelectionMode.Multi
```

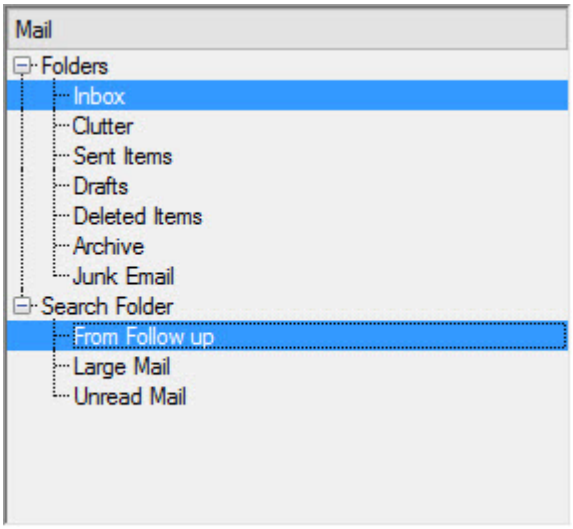
- **C#**

```
// set the SelectionMode property to Multi
c1TreeView1.SelectionMode = C1.Win.TreeView.C1TreeViewSelectionMode.Multi;
```

Contiguous Selection



Non-contiguous Selection



You can also select a node by setting the Selected property of C1TreeNode to **True**.

> The **Selected** property of **C1TreeNode** is not applicable when the **SelectedMode** property of **C1Treeview** is set to **None**.

The following code snippet demonstrates how to set the property.

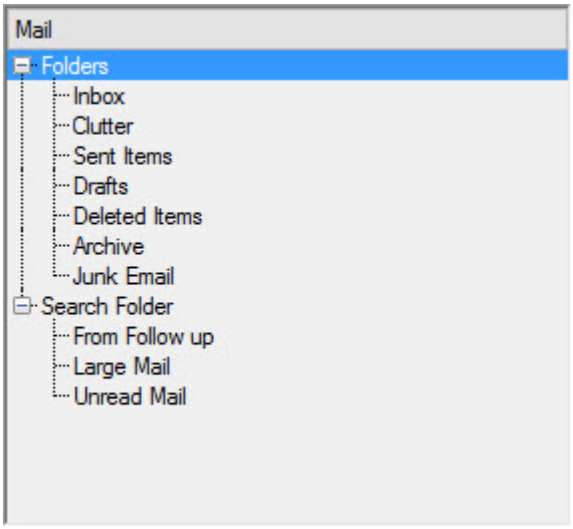
- **Visual Basic**

```
' enable the Selected property for the first parent node
C1TreeView1.Nodes(0).Selected = True
```

- **C#**

```
// enable the Selected property for the first parent node
c1TreeView1.Nodes[0].Selected = true;
```

The node for which the property is set gets selected as shown below.



Navigating Nodes

TreeView supports both mouse and keyboard navigation of nodes.

Mouse Navigation

The following table lists the actions and the corresponding Mouse commands for navigating nodes in TreeView:

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.

Keyboard Navigation

The following table describes the actions and their associated keys to use when navigating nodes in TreeView:

Action	Keyboard Command
Move up a node	Up Arrow Key
Move down a node	Down Arrow Key
Select multiple nodes	MOUSE + CTRL (distributed selection) / MOUSE + SHIFT (Continuous selection)

Using Check boxes

By default, TreeView does not display check boxes beside nodes. In order to display check boxes beside nodes in

TreeView, set the CheckBoxes property of the C1TreeView class to True.

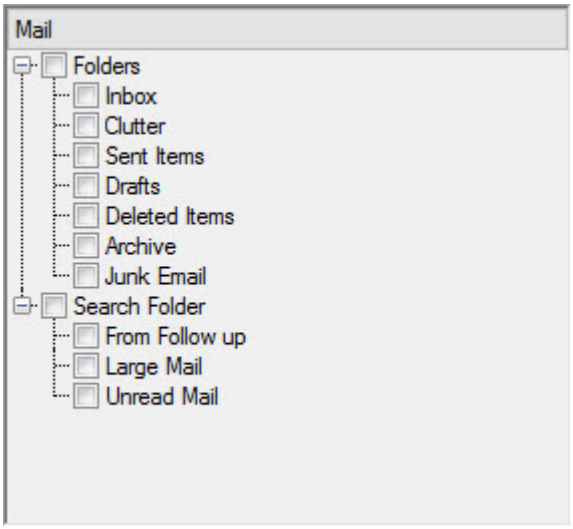
The following code snippet sets the CheckBoxes property.

- **Visual Basic**

```
' set the CheckBoxes property to enable checkboxes in TreeView
C1TreeView1.CheckBoxes = True
```

- **C#**

```
// set the CheckBoxes property to enable checkboxes in TreeView
c1TreeView1.CheckBoxes = true;
```



The check box next to a node can be checked by setting the Checked property of C1TreeNode to True. You can also alter the state of the check box of a node by setting the CheckState property of C1TreeNode to any of the following values from the **CheckState** enumeration of **System.Windows.Forms: Checked**, **Indeterminate**, and **Unchecked**.

In addition, you can change the state of the check box of a node by using the Check or the Uncheck method of C1TreeNode, respectively. These methods accept a parameter **allChildrens** of the **Boolean** type. The parameter **allChildrens** determines whether the check boxes of all child nodes within a node will be checked or unchecked. C1TreeView provides the CheckAll or the UncheckAll method using which you can check or uncheck all the check boxes beside the nodes. Moreover, using these methods, you can check nodes recursively in which if a parent node is checked, all the child nodes under it are recursively checked.

See the following code snippet for reference.

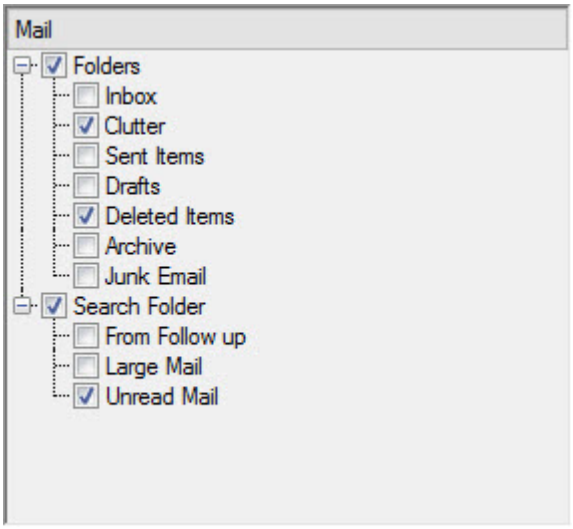
- **Visual Basic**

```
' set the parent nodes to the Checked state
C1TreeView1.Nodes(0).CheckState = CheckState.Checked
C1TreeView1.Nodes(1).CheckState = CheckState.Checked

' set the child nodes to the Checked state
parentNode1.Nodes(1).Check()
parentNode1.Nodes(4).Check()
parentNode2.Nodes(2).Check()
parentNode2.Nodes(2).CheckState = CheckState.Checked
```

- **C#**

```
// set the parent nodes to the Checked state
c1TreeView1.Nodes[0].CheckState = CheckState.Checked;
c1TreeView1.Nodes[1].CheckState = CheckState.Checked;
```

```
// set the child nodes to the Checked state
parentNode1.Nodes[1].Check();
parentNode1.Nodes[4].Check();
parentNode2.Nodes[2].Check();
parentNode2.Nodes[2].CheckState = CheckState.Checked;
```



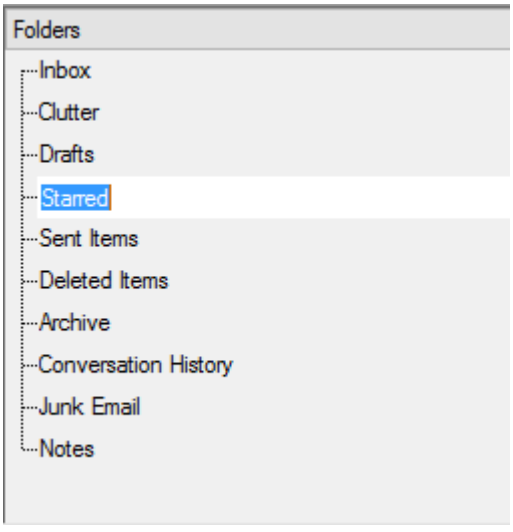
Editing Nodes

TreeView enables you to edit nodes in applications, you need to set the AllowEditing property of the C1TreeView class to **true**. The default value of the property is **false**.

You can start editing a node by selecting a node and pressing the Enter or F2 key, or simply double-clicking the node itself. In addition, you can edit a node programmatically by calling the BeginEdit method in code. To use any of these modes for editing nodes, you need to set the EditMode property of the C1TreeView class from the C1TreeViewEditMode enum.

> **Note**: If you want to determine whether the contents of a specific node can be changed, you can use the IsReadOnly method of C1TreeNode.

The following image shows a node being edited.



The given code snippet sets the AllowEditing property to allow node editing and specifies the mode of editing.

- **Visual Basic**

```
' set the AllowEditing property
C1TreeView1.AllowEditing = True

' set the EditMode property
C1TreeView1.EditMode = C1.Win.TreeView.C1TreeViewEditMode.EditOnEnter
```

- **C#**

```
// set the AllowEditing property
c1TreeView1.AllowEditing = true;

// set the EditMode property
c1TreeView1.EditMode = C1.Win.TreeView.C1TreeViewEditMode.EditOnEnter;
```

Expanding and Collapsing Nodes

Expanding Nodes

The C1TreeNode class provides the Expand method to expand a single node (parent or child). The Expand method accepts Boolean values to determine whether the child nodes within a particular node should expand or not. On setting this method to true, all the child nodes along with the selected node gets expanded. If set to false, only the node at which the method is called gets expanded. The following code snippet demonstrates how to expand a single node using the Expand method.

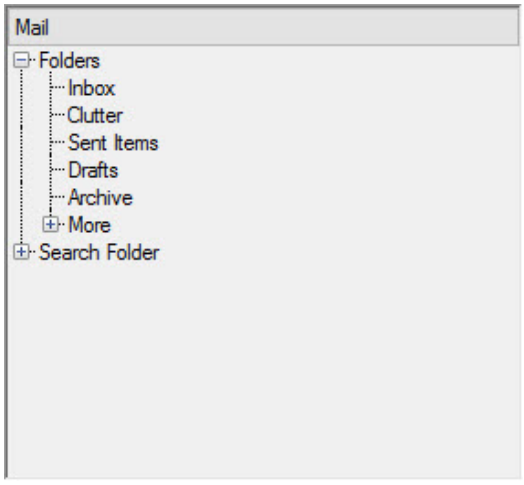
- **Visual Basic**

```
' expand the first parent node without expanding its child nodes
parentNode1.Expand(False)
```

- **C#**

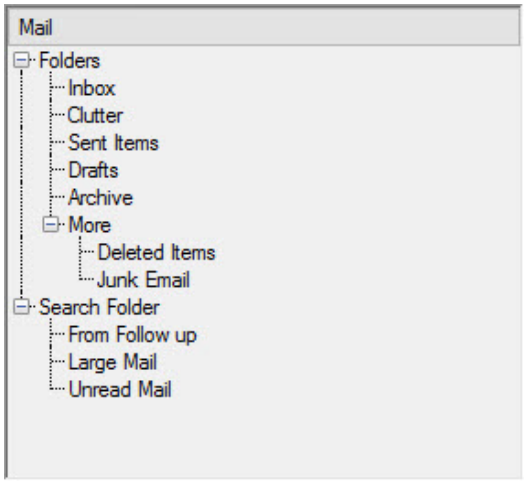
```
// expand the first parent node without expanding its child nodes
parentNode1.Expand(false);
```

The parent node gets expanded without expanding the child nodes, as shown below.



The C1TreeNode class also provides the Expanded property to expand a single node at a time.

Additionally, the C1TreeView class provides the ExpandAll method to fully expand a TreeView. In fully expanded state, the TreeView displays all the children nodes under a parent node as shown in the image below.



The following code snippet demonstrates how to fully expand a TreeView using the ExpandAll method.

- **Visual Basic**

```
' call the ExpandAll method
C1TreeView1.ExpandAll()
```

- **C#**

```
// call the ExpandAll method
c1TreeView1.ExpandAll();
```

Once the TreeView is in fully expanded state, you can prevent it from collapsing. This can be achieved by cancelling the Collapsing event of the C1TreeView class as demonstrated in the code below.

- **Visual Basic**

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' call the ExpandAll method
    C1TreeView1.ExpandAll()

    ' handle the Collapsing event
    AddHandler C1TreeView1.Collapsing, AddressOf C1TreeView1_Collapsing
End Sub

Private Sub C1TreeView1_Collapsing(sender As Object, e As C1TreeViewCancelEventArgs)

    ' cancel the event
    e.Cancel = True
End Sub
```

- **C#**

```
private void Form1_Load(object sender, EventArgs e)
{
    // call the ExpandAll method
    c1TreeView1.ExpandAll();

    // handle the Collapsing event
    c1TreeView1.Collapsing += C1TreeView1_Collapsing; ;
}

private void C1TreeView1_Collapsing(object sender, C1.Win.TreeView.C1TreeViewCancelEventArgs e)
{
    // cancel the event
    e.Cancel = true;
}
```

Collapsing Nodes

The C1TreeView class provides the CollapseAll method to fully collapse a TreeView. To collapse a single node (parent or child), the C1TreeNode class provides the Collapse method. This method accepts Boolean values to determine whether the child nodes within a particular node should collapse or not. On setting this method to true, all the child nodes along with the selected node gets collapsed. If set to false, only the node at which the method is called gets collapsed.

Dragging and Dropping Nodes

TreeView supports drag and drop operation within the same treeview as well as among multiple treeviews.

To perform drag and drop operation on nodes in a treeView, you need to set the EnableDragDrop property of C1TreeView to True. You need to set the **AllowDrop** property provided by **System.Windows.Forms.Control** to True for the treeview on which you need to perform the drop operation. Notice that you need to set the EnableDragDrop property to True for each of the multiple treeviews to enable drag and drop for them.

The C1TreeView class provides the ItemDrag event that occurs when a node is dragged. You can handle this event and use the DraggedData property provided by the C1TreeViewItemDragEventArgs class to get the dragged data. In addition, you can use the Node property to get the dragged node.

See the following code snippet for reference.

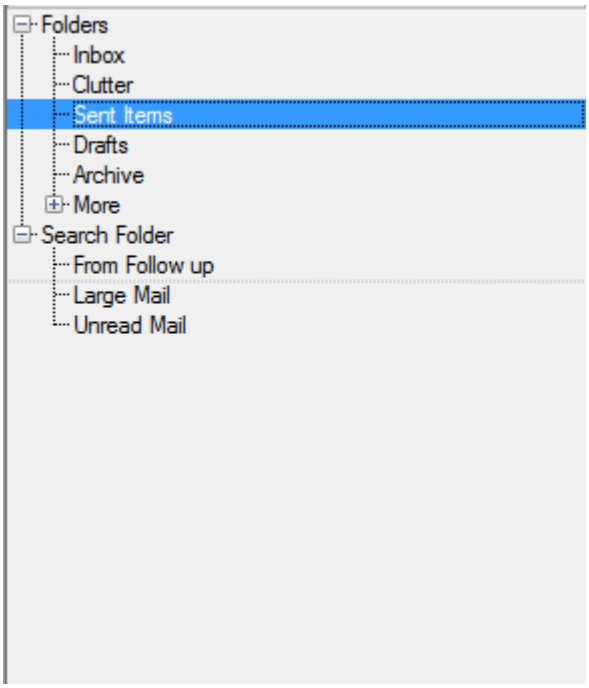
- **Visual Basic**

```
'set the EnableDragDrop property
C1TreeView1.EnableDragDrop = True

'set the AllowDrop property
C1TreeView1.AllowDrop = True
```

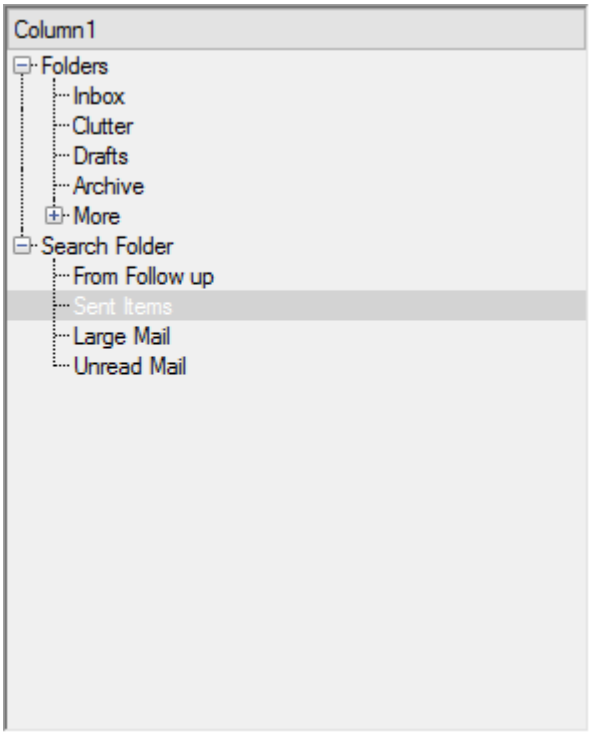
- **C#**

```
// set the EnableDragDrop property
c1TreeView1.EnableDragDrop = true;

// set the AllowDrop property
c1TreeView1.AllowDrop = false;
```

While Dragging



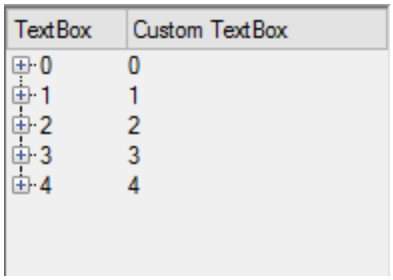
After Dropping



Custom Editors

TreeView, by default, uses a textbox editor for editing nodes. You can, however, replace the default editor with a custom editor, whenever needed. You can specify a custom editor for nodes for each column by using the Editor property of C1TreeColumn. In addition, you can set the type of the node editor by setting the EditorType property from the C1TreeViewEditorType enum.

The following image displays both the default textbox and the custom textbox editor.



The following code snippet creates a class EditorsData that provides sample data for the editors.

- **Visual Basic**

```
Imports System.ComponentModel
Public Class EditorsData
    Public Property TextBoxValue() As Object
        Get
            Return m_TextBoxValue
        End Get
        Set
            m_TextBoxValue = Value
```

```
        End Set
    End Property
    Private m_TextBoxValue As Object

    Public Property Collection() As BindingList(Of EditorsData)
        Get
            Return m_Collection
        End Get
        Set
            m_Collection = Value
        End Set
    End Property
    Private m_Collection As BindingList(Of EditorsData)

    Public Sub New()
        Collection = New BindingList(Of EditorsData)()
    End Sub

    Public Shared Function GetData() As BindingList(Of EditorsData)
        Dim data = New BindingList(Of EditorsData)()
        For i As Integer = 0 To 4
            data.Add(New EditorsData() With {
                .m_TextBoxValue = i
            })
            For j As Integer = 0 To 4
                data(i).Collection.Add(New EditorsData() With {
                .m_TextBoxValue = i + j
                })
            Next
        Next

        Return data
    End Function
End Class
```

- **C#**

```
using System.ComponentModel;

namespace SamplesData
{
    public class EditorsData
    {
        public object TextBoxValue
        {
            get; set;
        }

        public BindingList<EditorsData> Collection
        { get; set; }

        public EditorsData()
        {
            Collection = new BindingList<EditorsData>();
        }

        public static BindingList<EditorsData> GetData()
        {
            var data = new BindingList<EditorsData>();
            for (int i = 0; i < 5; i++)
            {
                data.Add(new EditorsData() {TextBoxValue = i,});
```

```
                for (int j = 0; j < 5; j++)
                {
                    data[i].Collection.Add(new EditorsData() {TextBoxValue = i + j});
                }
            }

            return data;
        }
    }
}
```

The following code snippet creates a class CustomTextBox to create a custom textbox editor on the basis of the default textbox editor.

- **Visual Basic**

```
Imports C1.Win.TreeView

Public Class CustomTextBox
    Inherits TextBox
    Implements IC1TreeEditor
    Public Function C1EditorGetValue() As Object
        Return Text
    End Function

    Public Sub C1EditorInitialize(value As Object, attrs As IDictionary)
        BorderStyle = BorderStyle.FixedSingle
        AutoSize = True
        TabStop = True

        If attrs.Contains("AcceptReturn") Then
            AcceptsReturn = CBool(attrs("AcceptReturn"))
        End If
        If attrs.Contains("AcceptTab") Then
            AcceptsTab = CBool(attrs("AcceptTab"))
        End If
        If attrs.Contains("BackColor") Then
            BackColor = DirectCast(attrs("BackColor"), Color)
        End If
        If attrs.Contains("Font") Then
            Font = DirectCast(attrs("Font"), Font)
        End If
        If attrs.Contains("ForeColor") Then
            ForeColor = DirectCast(attrs("ForeColor"), Color)
        End If
        If attrs.Contains("DisabledForeColor") Then
            ForeColor = DirectCast(attrs("DisabledForeColor"), Color)
        End If
        If attrs.Contains("MaxLength") Then
            MaxLength = CInt(attrs("MaxLength"))
        End If
        If attrs.Contains("ReadOnly") Then
            [ReadOnly] = CBool(attrs("ReadOnly"))
        End If
        If attrs.Contains("WordWrap") Then
            WordWrap = CBool(attrs("WordWrap"))
        End If

        Text = value.ToString()
    End Sub

    Public Function C1EditorKeyDownFinishEdit(e As KeyEventArgs) As Boolean
```

```
        If e.KeyData = Keys.Enter Then
            Return True
        End If
        Return False
    End Function

    Public Sub C1EditorUpdateBounds(rc As Rectangle)
        Bounds = rc
    End Sub

    Public Function C1EditorValueIsValid() As Boolean
        Return True
    End Function

    Private Function IC1TreeEditor_C1EditorValueIsValid()
        As Boolean Implements IC1TreeEditor.C1EditorValueIsValid
        Throw New NotImplementedException()
    End Function

    Private Sub IC1TreeEditor_C1EditorUpdateBounds(rc As Rectangle)
        Implements IC1TreeEditor.C1EditorUpdateBounds
        Throw New NotImplementedException()
    End Sub

    Private Function IC1TreeEditor_C1EditorKeyDownFinishEdit(e As KeyEventArgs)
        As Boolean Implements IC1TreeEditor.C1EditorKeyDownFinishEdit
        Throw New NotImplementedException()
    End Function

    Private Sub IC1TreeEditor_C1EditorInitialize(value As Object, attrs As IDictionary)
        Implements IC1TreeEditor.C1EditorInitialize
        Throw New NotImplementedException()
    End Sub

    Private Function IC1TreeEditor_C1EditorGetValue()
        As Object Implements IC1TreeEditor.C1EditorGetValue
        Throw New NotImplementedException()
    End Function
End Class
```

- **C#**

```
using C1.Win.TreeView;
using System.Collections;
using System.Drawing;
using System.Windows.Forms;

namespace CustomEditors
{
    public class CustomTextBox : TextBox, IC1TreeEditor
    {
        public object C1EditorGetValue()
        {
            return Text;
        }

        public void C1EditorInitialize(object value, IDictionary attrs)
        {
            BorderStyle = BorderStyle.FixedSingle;
            AutoSize = true;
            TabStop = true;
```

```
            if (attrs.Contains("AcceptReturn"))
                AcceptsReturn = (bool)attrs["AcceptReturn"];
            if (attrs.Contains("AcceptTab"))
                AcceptsTab = (bool)attrs["AcceptTab"];
            if (attrs.Contains("BackColor"))
                BackColor = (Color)attrs["BackColor"];
            if (attrs.Contains("Font"))
                Font = (Font)attrs["Font"];
            if (attrs.Contains("ForeColor"))
                ForeColor = (Color)attrs["ForeColor"];
            if (attrs.Contains("DisabledForeColor"))
                ForeColor = (Color)attrs["DisabledForeColor"];
            if (attrs.Contains("MaxLength"))
                MaxLength = (int)attrs["MaxLength"];
            if (attrs.Contains("ReadOnly"))
                ReadOnly = (bool)attrs["ReadOnly"];
            if (attrs.Contains("WordWrap"))
                WordWrap = (bool)attrs["WordWrap"];

            Text = value.ToString();
        }

        public bool C1EditorKeyDownFinishEdit(KeyEventArgs e)
        {
            if (e.KeyData == Keys.Enter)
                return true;
            return false;
        }

        public void C1EditorUpdateBounds(Rectangle rc)
        {
            Bounds = rc;
        }

        public bool C1EditorValueIsValid()
        {
            return true;
        }
    }
}
```

The following code snippet sets the default textbox and the custom textbox as editors in the first and the second columns, respectively.

- **Visual Basic**

```
Dim textBox1 As System.Windows.Forms.TextBox = New TextBox()
Dim textBox2 As New CustomTextBox()
C1TreeView1.AllowEditing = True

C1TreeView1.Columns.Clear()

Dim column1 As New C1.Win.TreeView.C1TreeColumn()
Dim column2 As New C1.Win.TreeView.C1TreeColumn()
Dim column3 As New C1.Win.TreeView.C1TreeColumn()
Dim column4 As New C1.Win.TreeView.C1TreeColumn()

C1TreeView1.Columns.Add(column1)
C1TreeView1.Columns.Add(column2)

column1.Name = "clnTextBox"
column1.HeaderText = "TextBox"
```

```
column1.Width = 60
column1.DisplayFieldName = "TextBoxValue\TextBoxValue"
column1.Editor = textBox1
column1.EditorType = C1.Win.TreeView.C1TreeViewEditorType.Text

column2.Name = "cnlCustomTextBox"
column2.HeaderText = "Custom TextBox"
column2.Width = 90
column2.DisplayFieldName = "TextBoxValue\TextBoxValue"
column2.Editor = New CustomTextBox()
column2.EditorType = C1.Win.TreeView.C1TreeViewEditorType.Text

C1TreeView1.DataMember = "Collection\Collection"
C1TreeView1.DataSource = EditorsData.GetData()
```

- **C#**

```
System.Windows.Forms.TextBox textBox1 = new TextBox();
CustomTextBox textBox2 = new CustomTextBox();
c1TreeView1.AllowEditing = true;

C1.Win.TreeView.C1TreeColumn column1 = new C1.Win.TreeView.C1TreeColumn();
C1.Win.TreeView.C1TreeColumn column2 = new C1.Win.TreeView.C1TreeColumn();
C1.Win.TreeView.C1TreeColumn column3 = new C1.Win.TreeView.C1TreeColumn();
C1.Win.TreeView.C1TreeColumn column4 = new C1.Win.TreeView.C1TreeColumn();

c1TreeView1.Columns.Add(column1);
c1TreeView1.Columns.Add(column2);

column1.Name = "clnTextBox";
column1.HeaderText = "TextBox";
column1.Width = 60;
column1.DisplayFieldName = "TextBoxValue\\TextBoxValue";
column1.Editor = textBox1;
column1.EditorType = C1.Win.TreeView.C1TreeViewEditorType.Text;

column2.Name = "cnlCustomTextBox";
column2.HeaderText = "Custom TextBox";
column2.Width = 90;
column2.DisplayFieldName = "TextBoxValue\\TextBoxValue";
column2.Editor = new CustomTextBox();
column2.EditorType = C1.Win.TreeView.C1TreeViewEditorType.Text;

c1TreeView1.DataMember = "Collection\\Collection";
c1TreeView1.DataSource = EditorsData.GetData();
```

Custom Nodes

TreeView allows creating and displaying custom nodes in place of the default nodes. It is possible to add custom properties and show various customizations, for instance, including, scaling, and aligning images, displaying borders, customizing font styles and colors, and changing dimensions etc. To set custom nodes in TreeView, set the CustomContentPresenter property of C1TreeColumn.

The following image displays custom nodes in TreeView.

To create custom nodes in TreeView, create custom classes to define properties and methods that apply to custom nodes and override the default ones. After creating custom classes, initialize their instances and set custom nodes using the CustomContentPresenter property.

The following code snippets show how to create custom classes to define main elements for custom nodes.

- **Visual Basic**

```
Public Class CategoryCustomNode
    Inherits CustomContentPresenter
    ' level 0
    Private _name As TextElement
    Private _description As TextElement
    Private _img As ImageElement
    Private _rw As RowPanel
    ' level 1
    Private _product As TextElement

    Public Sub New()
        ' level 0
        ' init text elements
        ' name
        _name = New TextElement()
        _name.Style = New Style()
        ' description
        _description = New TextElement()
        _description.Style = New Style()
        _description.Width = 120
        ' init image element
        _img = New ImageElement()
```

```
        _img.Style = New Style()
        _img.Size = New Size(70, 50)
        ' init a grid for text elements
        Dim cp = New ColumnPanel()
        cp.Children.Add(_name)
        cp.Children.Add(_description)
        ' init panel for image
        _rw = New RowPanel()
        _rw.Children.Add(cp)
        _rw.Children.Add(_img)
        _rw.Style = New Style()
        _rw.Style.VerticalAlignment = Alignment.Center
        ' level 1
        _product = New TextElement()
        _product.Style = New Style()
    End Sub

    Public Overrides ReadOnly Property ToolTipText() As String
        Get
            Return _name.Text
        End Get
    End Property

    Public Overrides Sub SetStyle(styles As TreeNodeCellStyles)
        ' level 0
        ' name
        _name.Style.Margins = New Thickness(1)
        _name.Style.Font = New Font("Calibri", 10, FontStyle.Bold)
        ' description
        _description.Style.Margins = New Thickness(1)
        _description.Style.Font = New Font("Calibri", 9, FontStyle.Italic)
        _description.Style.WordWrap = True
        ' img
        _img.Style.ImageScaling = ImageScaling.Scale
        _img.Style.ImageAlignment = ImageAlignment.CenterCenter
        ' level 1
        _product.Style.Font = New Font("Calibri", 10, FontStyle.Bold)
        _product.Style.Margins = New Thickness(2)
        _product.Style.HorizontalAlignment = Alignment.Center
        _product.Style.VerticalAlignment = Alignment.Center
    End Sub

    Public Overrides Sub SetValue(value As Object)
        If Node.Level = 0 Then
            Dim row = DirectCast(Node.GetValue(), DataSet1.CategoriesRow)
            _name.Text = row.CategoryName
            _description.Text = row.Description
            Dim converter = New ImageConverter()
            If _img.Image IsNot Nothing Then
                _img.Image.Dispose()
                _img.Image = Nothing
            End If
            _img.Image = DirectCast(converter.ConvertFrom(row.Picture), Image)
            ' set root panel
            Child = _rw
        Else
            _product.Text = value.ToString()
            ' set root element
            Child = _product
        End If
    End Sub
End Class
```

```
Public Class ProductCustomNode
    Inherits CustomContentPresenter
    ' level 0
    Private _count As TextElement
    ' level 1
    Private _quantityPerUnit As TextElement
    Private _unitPrice As TextElement
    Private _unitInStock As TextElement
    Private _unitsOnOrder As TextElement
    Private _reorderLevel As TextElement
    Private _gp As GridPanel
    Private _eStyle As Style

    Public Sub New()
        ' level 0
        ' count in category
        _count = New TextElement()
        _count.Style = New Style()
        _count.Style.Margins = New Thickness(2)
        ' level 1
        ' lable style
        Dim lStyle = New Style()
        lStyle.HorizontalAlignment = Alignment.Far
        lStyle.Margins = New Thickness(1)
        lStyle.Font = New Font("Calibri", 9, FontStyle.Regular)
        ' elements style
        _eStyle = New Style()
        _eStyle.Margins = New Thickness(1)
        _eStyle.Font = New Font("Calibri", 9, FontStyle.Regular)
        ' init elements
        _quantityPerUnit = New TextElement(_eStyle.Clone())
        _unitPrice = New TextElement(_eStyle.Clone())
        _unitInStock = New TextElement(_eStyle.Clone())
        _unitsOnOrder = New TextElement(_eStyle.Clone())
        _reorderLevel = New TextElement(_eStyle.Clone())
        ' init a grid for text elements
        _gp = New GridPanel()
        _gp.Columns.Add()
        ' labels
        _gp.Columns(0).Width = 100
        _gp.Columns.Add()
        ' text
        _gp.Columns(1).Width = 120

        _gp.Rows.Add()
        ' ReorderLevel
        _gp(0, 0).Element = New TextElement(lStyle, "Reorder level:")
        _gp(0, 1).Element = _reorderLevel
        _gp.Rows.Add()
        ' UnitPrice
        _gp(1, 0).Element = New TextElement(lStyle, "Unit price:")
        _gp(1, 1).Element = _unitPrice
        _gp.Rows.Add()
        ' QuantityPerUnit
        _gp(2, 0).Element = New TextElement(lStyle, "Quantity per unit:")
        _gp(2, 1).Element = _quantityPerUnit

        _gp.Columns.Add()
        ' labels
        _gp.Columns(2).Width = 100
        _gp.Columns.Add()
```

```
        ' text
        ' UnitsInStock
        _gp(0, 2).Element = New TextElement(lStyle, "Units in stock:")
        _gp(0, 3).Element = _unitInStock
        ' UnitsOnOrder
        _gp(1, 2).Element = New TextElement(lStyle, "Units on order:")

        _gp(1, 3).Element = _unitsOnOrder
    End Sub

    Public Overrides ReadOnly Property ToolTipText() As String
        Get
            Return String.Empty
        End Get
    End Property

    Public Overrides Sub SetStyle(styles As TreeNodeCellStyles)
        ' level 0
        _count.Style.HorizontalAlignment = Alignment.Center
        _count.Style.VerticalAlignment = Alignment.Center
        _count.Style.Font = New Font("Calibri", 11, FontStyle.Bold)
        ' level 1
        _unitPrice.Style.Font = New Font(_eStyle.Font, FontStyle.Bold)
        If _unitInStock.Text = "0" Then
            _unitInStock.Style.ForeColor = Color.Red
            _unitInStock.Style.Font = New Font(_eStyle.Font, FontStyle.Bold)
        End If
        If _unitsOnOrder.Text = "0" Then
            _unitsOnOrder.Style.ForeColor = Color.Red
            _unitsOnOrder.Style.Font = New Font(_eStyle.Font, FontStyle.Bold)
        End If
    End Sub

    Public Overrides Sub SetValue(value As Object)
        If Node.Level = 0 Then
            Dim count = If(Node.HasChildren, Node.Nodes.Count, 0)
            _count.Text = count.ToString()
            Child = _count
        Else
            Dim row = DirectCast(Node.GetValue(), DataSet1.ProductsRow)
            _quantityPerUnit.Text = row.QuantityPerUnit
            _unitPrice.Text = row.UnitPrice.ToString("C")
            _unitInStock.Text = row.UnitsInStock.ToString()
            _unitsOnOrder.Text = row.UnitsOnOrder.ToString()
            _reorderLevel.Text = row.ReorderLevel.ToString()
            Child = _gp
        End If
    End Sub
End Class
```

- **C#**

```
public class CategoryCustomNode : CustomContentPresenter
{
    // level 0
    private TextElement _name;
    private TextElement _description;
    private ImageElement _img;
    private RowPanel _rw;
    // level 1
    private TextElement _product;
```

```
public CategoryCustomNode()
{
    // level 0
    // init text elements
    // name
    _name = new TextElement();
    _name.Style = new Style();
    // description
    _description = new TextElement();
    _description.Style = new Style();
    _description.Width = 120;
    // init image element
    _img = new ImageElement();
    _img.Style = new Style();
    _img.Size = new Size(70, 50);
    // init a grid for text elements
    var cp = new ColumnPanel();
    cp.Children.Add(_name);
    cp.Children.Add(_description);
    // init panel for image
    _rw = new RowPanel();
    _rw.Children.Add(cp);
    _rw.Children.Add(_img);
    _rw.Style = new Style();
    _rw.Style.VerticalAlignment = Alignment.Center;
    // level 1
    _product = new TextElement();
    _product.Style = new Style();
}

public override string ToolTipText
{
    get
    {
        return _name.Text;
    }
}

public override void SetStyle(TreeNodeCellStyles styles)
{
    // level 0
    // name
    _name.Style.Margins = new Thickness(1);
    _name.Style.Font = new Font("Calibri", 10, FontStyle.Bold);
    // description
    _description.Style.Margins = new Thickness(1);
    _description.Style.Font = new Font("Calibri", 9, FontStyle.Italic);
    _description.Style.WordWrap = true;
    // img
    _img.Style.ImageScaling = ImageScaling.Scale;
    _img.Style.ImageAlignment = ImageAlignment.CenterCenter;
    // level 1
    _product.Style.Font = new Font("Calibri", 10, FontStyle.Bold);
    _product.Style.Margins = new Thickness(2);
    _product.Style.HorizontalAlignment = Alignment.Center;
    _product.Style.VerticalAlignment = Alignment.Center;
}

public override void SetValue(object value)
{
    if (Node.Level == 0)
    {
```

```
            var row = (DataSet1.CategoriesRow)Node.GetValue();
            _name.Text = row.CategoryName;
            _description.Text = row.Description;
            var converter = new ImageConverter();
            if (_img.Image != null)
            {
                _img.Image.Dispose();
                _img.Image = null;
            }
            _img.Image = (Image)converter.ConvertFrom(row.Picture);
            // set root panel
            Child = _rw;
        }
        else
        {
            _product.Text = value.ToString();
            // set root element
            Child = _product;
        }
    }
}

public class ProductCustomNode : CustomContentPresenter
{
    // level 0
    private TextElement _count;
    // level 1
    private TextElement _quantityPerUnit;
    private TextElement _unitPrice;
    private TextElement _unitInStock;
    private TextElement _unitsOnOrder;
    private TextElement _reorderLevel;
    private GridPanel _gp;
    private Style _eStyle;

    public ProductCustomNode()
    {
        // level 0
        // count in category
        _count = new TextElement();
        _count.Style = new Style();
        _count.Style.Margins = new Thickness(2);
        // level 1
        // lable style
        var lStyle = new Style();
        lStyle.HorizontalAlignment = Alignment.Far;
        lStyle.Margins = new Thickness(1);
        lStyle.Font = new Font("Calibri", 9, FontStyle.Regular);
        // elements style
        _eStyle = new Style();
        _eStyle.Margins = new Thickness(1);
        _eStyle.Font = new Font("Calibri", 9, FontStyle.Regular);
        // init elements
        _quantityPerUnit = new TextElement(_eStyle.Clone());
        _unitPrice = new TextElement(_eStyle.Clone());
        _unitInStock = new TextElement(_eStyle.Clone());
        _unitsOnOrder = new TextElement(_eStyle.Clone());
        _reorderLevel = new TextElement(_eStyle.Clone());
        // init a grid for text elements
        _gp = new GridPanel();
        _gp.Columns.Add(); // labels
        _gp.Columns[0].Width = 100;
```

```
        _gp.Columns.Add(); // text
        _gp.Columns[1].Width = 120;

        _gp.Rows.Add(); // ReorderLevel
        _gp[0, 0].Element = new TextElement(lStyle, "Reorder level:");
        _gp[0, 1].Element = _reorderLevel;
        _gp.Rows.Add(); // UnitPrice
        _gp[1, 0].Element = new TextElement(lStyle, "Unit price:");
        _gp[1, 1].Element = _unitPrice;
        _gp.Rows.Add(); // QuantityPerUnit
        _gp[2, 0].Element = new TextElement(lStyle, "Quantity per unit:");
        _gp[2, 1].Element = _quantityPerUnit;

        _gp.Columns.Add(); // labels
        _gp.Columns[2].Width = 100;
        _gp.Columns.Add(); // text
        // UnitsInStock
        _gp[0, 2].Element = new TextElement(lStyle, "Units in stock:");
        _gp[0, 3].Element = _unitInStock;
        // UnitsOnOrder
        _gp[1, 2].Element = new TextElement(lStyle, "Units on order:");
        _gp[1, 3].Element = _unitsOnOrder;

    }

    public override string ToolTipText
    {
        get
        {
            return string.Empty; ;
        }
    }

    public override void SetStyle(TreeNodeCellStyles styles)
    {
        // level 0
        _count.Style.HorizontalAlignment = Alignment.Center;
        _count.Style.VerticalAlignment = Alignment.Center;
        _count.Style.Font = new Font("Calibri", 11, FontStyle.Bold);
        // level 1
        _unitPrice.Style.Font = new Font(_eStyle.Font, FontStyle.Bold);
        if (_unitInStock.Text == "0")
        {
            _unitInStock.Style.ForeColor = Color.Red;
            _unitInStock.Style.Font = new Font(_eStyle.Font, FontStyle.Bold);
        }
        if (_unitsOnOrder.Text == "0")
        {
            _unitsOnOrder.Style.ForeColor = Color.Red;
            _unitsOnOrder.Style.Font = new Font(_eStyle.Font, FontStyle.Bold);
        }
    }

    public override void SetValue(object value)
    {
        if (Node.Level == 0)
        {
            var count = Node.HasChildren ? Node.Nodes.Count : 0;
            _count.Text = count.ToString();
            Child = _count;
        }
        else
```

```
        {
            var row = (DataSet1.ProductsRow)Node.GetValue();
            _quantityPerUnit.Text = row.QuantityPerUnit;
            _unitPrice.Text = row.UnitPrice.ToString("C");
            _unitInStock.Text = row.UnitsInStock.ToString();
            _unitsOnOrder.Text = row.UnitsOnOrder.ToString();
            _reorderLevel.Text = row.ReorderLevel.ToString();
            Child = _gp;
        }
    }
}
```

The following code snippet shows how to set custom nodes using the CustomContentPresenter property.

- **Visual Basic**

```
' set custom nodes
C1TreeView1.Columns(0).CustomContentPresenter = New CategoryCustomNode()
C1TreeView1.Columns(1).CustomContentPresenter = New ProductCustomNode()
```

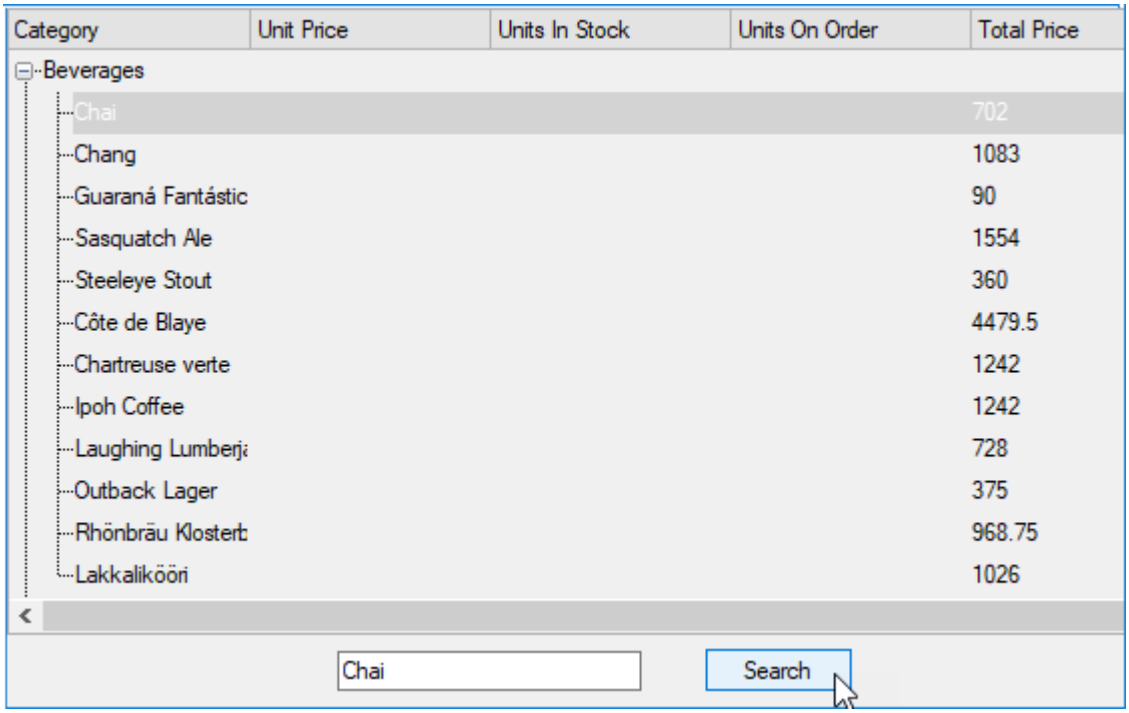
- **C#**

```
// set custom nodes
c1TreeView1.Columns[0].CustomContentPresenter = new CategoryCustomNode();
c1TreeView1.Columns[1].CustomContentPresenter = new ProductCustomNode();
```

Searching Nodes

TreeView enables you to search for the nodes matching the specified string. To search for a node in TreeView, you need to use Search method of C1TreeView class. The **Search** method takes string as a value, searches for the nodes using depth-first search and returns the node containing the searched string.

The following image shows the searched node containing “Chai” as input string.



To search for the nodes containing the string you input, use the following code. In this example, we have taken a text box where you can provide a string to be searched on a button click.

C#

```
private void Searchbutton_Click(object sender, EventArgs e)
{
      c1TreeView1.Search(textBox1.Text);
}
```

Data Binding

The process of connecting a data consumer with a data source is known as data binding. TreeView for WinForms supports data binding in two modes: unbound mode and bound mode.

The unbound mode means that the TreeView control is not bound to any data source. In the unbound mode, you need to create columns and nodes manually either by using designer or programmatically. To create nodes in the unbound mode at design-time, you can simply add the nodes in **TreeNodes Editor.** For more details on how to access TreeNodes Editor, refer to Collection Editors. If you want to create nodes in the unbound mode programmatically, you can do so by using the Add and the Insert methods. For more information, refer to Adding and Removing Nodes.

In the bound mode, the TreeView control uses data from a data source and displays it in the form of parent and child nodes in a hierarchical manner. TreeView supports a number of data source objects, namely, BindingSource, List, DataView, DataTable, DataSet, and objects implementing particular interfaces like IBindingList and IList.

To bind TreeView to a data source object, you need to use the DataSource property of C1TreeView. You can specify the type of the bound mode for TreeView by setting the TreeBoundMode property to one of the following values from the TreeBoundMode enumeration:

- **OneWay**: Does not update the data source after the treeview hierarchy changes.
- **TwoWay**: Updates the data source after the treeview hierarchy changes

Once you have set the data source, you can set the list of fields to display in columns by using the DisplayFieldName property of C1TreeColumn.

Refer to the following sections to know how you can bind TreeView to different objects:

- Binding TreeView to Lists
- Binding TreeView to Hierarchical Data

Binding TreeView to Lists

You can bind TreeView to a list containing objects that are accessed by using index values.

Here is a sample code that first creates a class **Products** and creates an instance of **List<T>** belonging to **System.Collections.Generic.** Then it binds the TreeView control to the list and displays the data from the list in a hierarchical form when the **Data Generate** button is clicked.

The following code snippet creates a class Products.cs.

- **Visual Basic**

```
' create a class
Public Class Products

    Private _id As Integer
    Private _description As String
    Private _price As Single
    Public Property ID() As Integer
        Get
            Return _id
        End Get
        Set
            _id = Value
        End Set
    End Property
    Public Property Description() As String
        Get
            Return _description
        End Get
        Set
            _description = Value
        End Set
    End Property
    Public Property Price() As Single
        Get
            Return _price
        End Get
        Set
            _price = Value
        End Set
    End Property
    Public Sub New(id As Integer, description As String, price As Single)
        _id = id
        _description = description
        _price = price
    End Sub
End Class
```

- **C#**

```
// create a class
public class Products
{

    private int _id;
    private string _description;
    private float _price;
    public int ID
    {
        get { return _id; }
```

```
        set { _id = value; }
    }
    public string Description
    {
        get { return _description; }
        set { _description = value; }
    }
    public float Price
    {
        get { return _price; }
        set { _price = value; }
    }
    public Products(int id, string description, float price)
    {
        _id = id;
        _description = description;
        _price = price;
    }
}
```

The following code snippets binds TreeView to the list.

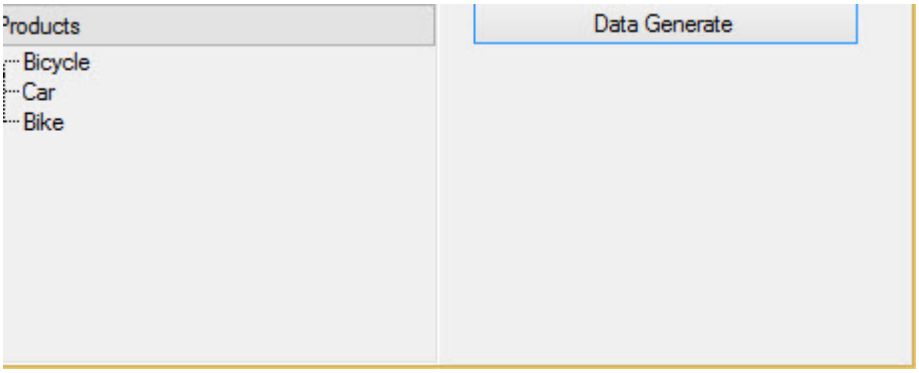
- **Visual Basic**

```
' to generate data when the button is clicked
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click

    ' create a list
    Dim product As New List(Of Products)()

    ' add the objects to the list
    product.Add(New Products(567, "Bicycle", 5))
    product.Add(New Products(456, "Car", 5000))
    product.Add(New Products(789, "Bike", 1500))

    ' bind TreeView to the list
    C1TreeView1.DataSource = product

    ' set the field to be displayed in the column
    C1TreeView1.Columns(0).DisplayFieldName = "Description\Description"
End Sub
```

- **C#**

```
// to generate data when the button is clicked
private void button1_Click(object sender, EventArgs e)
{
    // create a list
    List<Products> product = new List<Products>();

    // add the objects to the list
    product.Add(new Products(567, "Bicycle", 5));
    product.Add(new Products(456, "Car", 5000));
    product.Add(new Products(789, "Bike", 1500));

    // bind TreeView to the list
    c1TreeView1.DataSource = product;

    // set the field to be displayed in the column
    c1TreeView1.Columns[0].DisplayFieldName = "Description\\Description";
}
```

Once you have clicked the Data Generate button after running the code, the following output appears.



Binding TreeView to Hierarchical Data

An important feature of TreeView is its ability to bind to hierarchical data and display it as relationship between parent and child nodes.

The C1TreeView class provides the DataSource property to specify the data source for the TreeView control. The class also provides the DataMember property to specify the name of a specific record within the data source.

Here is a sample code that binds the TreeView control to a hierarchical data source that is created by using different classes.

The following code snippet creates a class **Product.**

- **Visual Basic**

```
' create a class
Public Class Product
    Public Property ID() As Guid
        Get
            Return m_ID
        End Get
        Set
            m_ID = Value
        End Set
    End Property
    Private m_ID As Guid
    Public Property Name() As String
        Get
            Return m_Name
        End Get
        Set
            m_Name = Value
        End Set
    End Property
    Private m_Name As String
    Public Property Price() As Double
        Get
            Return m_Price
        End Get
        Set
            m_Price = Value
        End Set
    End Property
    Private m_Price As Double
```

```
    Public Sub New(name__1 As String, price__2 As Double)
        ID = Guid.NewGuid()
        Name = name__1
        Price = price__2
    End Sub

    Public Overrides Function ToString() As String
        Return Name
    End Function
End Class
```

- **C#**

```
// create a class
public class Product
{
    public Guid ID { get; set; }
    public string Name { get; set; }
    public double Price { get; set; }

    public Product(string name, double price)
    {
        ID = Guid.NewGuid();
        Name = name;
        Price = price;
    }
    public override string ToString()
    {
        return Name;
    }
}
```

Here is a code snippet creating a class **ProductsGroup.**

- **Visual Basic**

```
' create a class
Public Class ProductsGroup

    Public Property ID() As Guid
        Get
            Return m_ID
        End Get
        Set
            m_ID = Value
        End Set
    End Property
    Private m_ID As Guid
    Public Property Name() As String
        Get
            Return m_Name
        End Get
        Set
            m_Name = Value
        End Set
    End Property
    Private m_Name As String
    Public Property Products() As BindingList(Of Product)
        Get
            Return m_Products
        End Get
        Set
```

```
            m_Products = Value
        End Set
    End Property
    Private m_Products As BindingList(Of Product)
    Public ReadOnly Property CountOfProducts() As Integer
        Get
            Return Products.Count
        End Get
    End Property

    Public Sub New(name__1 As String)
        ID = Guid.NewGuid()
        Name = name__1
        Products = New BindingList(Of Product)()
    End Sub

    Public Overrides Function ToString() As String
        Return Name
    End Function
End Class
```

- **C#**

```
// create a class
public class ProductsGroup
{
    public Guid ID { get; set; }
    public string Name { get; set; }
    public BindingList<Product> Products { get; set; }
    public int CountOfProducts
    {
        get { return Products.Count; }
    }

    public ProductsGroup(string name)
    {
        ID = Guid.NewGuid();
        Name = name;
        Products = new BindingList<Product>();
    }

    public override string ToString()
    {
        return Name;
    }
}
```

Below is the code creating a class **Store.**

- **Visual Basic**

```
' create a class
Public Class Store

    Public Property ID() As Guid
        Get
            Return m_ID
        End Get
        Set
            m_ID = Value
        End Set
    End Property
```

```
    Private m_ID As Guid
    Public Property Name() As String
        Get
            Return m_Name
        End Get
        Set
            m_Name = Value
        End Set
    End Property
    Private m_Name As String
    Public Property ProductsGroups() As BindingList(Of ProductsGroup)
        Get
            Return m_ProductsGroups
        End Get
        Set
            m_ProductsGroups = Value
        End Set
    End Property
    Private m_ProductsGroups As BindingList(Of ProductsGroup)
    Public ReadOnly Property CountOfProducts() As Integer
        Get
            Dim count As Integer = 0
            For Each pg As Object In ProductsGroups
                count += pg.CountOfProducts
            Next
            Return count
        End Get
    End Property

    Public Sub New(name__1 As String)
        ID = Guid.NewGuid()
        Name = name__1
        ProductsGroups = New BindingList(Of ProductsGroup)()
    End Sub

    Public Overrides Function ToString() As String
        Return Name
    End Function
End Class
```

- **C#**

```
// create a class
public class Store
{
    public Guid ID { get; set; }
    public string Name { get; set; }
    public BindingList<ProductsGroup> ProductsGroups { get; set; }
    public int CountOfProducts
    {
        get
        {
            int count = 0;
            foreach (var pg in ProductsGroups)
                count += pg.CountOfProducts;
            return count;
        }
    }

    public Store(string name)
    {
        ID = Guid.NewGuid();
```

```
        Name = name;
        ProductsGroups = new BindingList<ProductsGroup>();
    }

    public override string ToString()
    {
        return Name;
    }
}
```

The following code snippet creates a class **StoreCollection**.

- **Visual Basic**

```
' create a class
Public Class StoreCollection
    Inherits BindingList(Of Store)
    Public Shared Function GetData() As StoreCollection
        Dim stores = New StoreCollection()

        stores.Add(New Store("Pear Inc."))
        stores(0).ProductsGroups.Add(New ProductsGroup("Mobile phones"))
        stores(0).ProductsGroups(0).Products.Add(New Product("fPhone 34", 999.99))
        stores(0).ProductsGroups(0).Products.Add(New Product("fPhone 34Z", 9999.99))
        stores(0).ProductsGroups(0).Products.Add(New Product("fPhone 34XX", 100000))
        stores(0).ProductsGroups.Add(New ProductsGroup("Notebooks"))
        stores(0).ProductsGroups(1).Products.Add(New Product("DuckBook S", 9999.99))
        stores(0).ProductsGroups(1).Products.Add(New Product("DuckBook Ultra", 14000))
        stores(0).ProductsGroups(1).Products.Add(New Product("DuckBook Pro", 20000))
        stores(0).ProductsGroups.Add(New ProductsGroup("Computers"))
        stores(0).ProductsGroups(2).Products.Add(New Product("DuckPC 3", 10000.99))
        stores(0).ProductsGroups(2).Products.Add(New Product("DuckPro X", 15000))
        stores(0).ProductsGroups(2).Products.Add(New Product("DuckPro Ultra", 19000))

        stores.Add(New Store("Space Inc."))
        stores(1).ProductsGroups.Add(New ProductsGroup("Mobile phones"))
        stores(1).ProductsGroups(0).Products.Add(New Product("Rocket 1A", 900))
        stores(1).ProductsGroups(0).Products.Add(New Product("Rocket 2X", 3999))
        stores(1).ProductsGroups(0).Products.Add(New Product("Rocket 3E", 20000))
        stores(1).ProductsGroups.Add(New ProductsGroup("Notebooks"))
        stores(1).ProductsGroups(1).Products.Add(New Product("Shuttle 1A", 9999.99))
        stores(1).ProductsGroups(1).Products.Add(New Product("Shuttle 1X", 14000))
        stores(1).ProductsGroups(1).Products.Add(New Product("Shuttle Pro", 20000))
        stores(1).ProductsGroups.Add(New ProductsGroup("Computers"))
        stores(1).ProductsGroups(2).Products.Add(New Product("IssPC 2D", 10000.99))
        stores(1).ProductsGroups(2).Products.Add(New Product("IssPro 2X", 15000))
        stores(1).ProductsGroups(2).Products.Add(New Product("IssPro Pro", 19000))

        stores.Add(New Store("Fruit Inc."))
        stores(2).ProductsGroups.Add(New ProductsGroup("Mobile phones"))
        stores(2).ProductsGroups(0).Products.Add(New Product("Pineapple 1", 2900))
        stores(2).ProductsGroups(0).Products.Add(New Product("Mango 1", 3099))
        stores(2).ProductsGroups(0).Products.Add(New Product("Orange 1", 5000))
        stores(2).ProductsGroups.Add(New ProductsGroup("Notebooks"))
        stores(2).ProductsGroups(1).Products.Add(New Product("Mandarin X", 9999.99))
        stores(2).ProductsGroups(1).Products.Add(New Product("Lemon X", 14000))
        stores(2).ProductsGroups(1).Products.Add(New Product("Lemon Pro", 20000))
        stores(2).ProductsGroups.Add(New ProductsGroup("Computers"))
        stores(2).ProductsGroups(2).Products.Add(New Product("Plum X", 10000.99))
        stores(2).ProductsGroups(2).Products.Add(New Product("Plum Z", 15000))
        stores(2).ProductsGroups(2).Products.Add(New Product("Plum Pro", 19000))
```

```
        Return stores
    End Function
End Class
```

- **C#**

```
// create a class
public class StoreCollection : BindingList<Store>
{
    public static StoreCollection GetData()
    {
        var stores = new StoreCollection();

        stores.Add(new Store("Pear Inc."));
        stores[0].ProductsGroups.Add(new ProductsGroup("Mobile phones"));
        stores[0].ProductsGroups[0].Products.Add(new Product("fPhone 34", 999.99));
        stores[0].ProductsGroups[0].Products.Add(new Product("fPhone 34Z", 9999.99));
        stores[0].ProductsGroups[0].Products.Add(new Product("fPhone 34XX", 100000));
        stores[0].ProductsGroups.Add(new ProductsGroup("Notebooks"));
        stores[0].ProductsGroups[1].Products.Add(new Product("DuckBook S", 9999.99));
        stores[0].ProductsGroups[1].Products.Add(new Product("DuckBook Ultra", 14000));
        stores[0].ProductsGroups[1].Products.Add(new Product("DuckBook Pro", 20000));
        stores[0].ProductsGroups.Add(new ProductsGroup("Computers"));
        stores[0].ProductsGroups[2].Products.Add(new Product("DuckPC 3", 10000.99));
        stores[0].ProductsGroups[2].Products.Add(new Product("DuckPro X", 15000));
        stores[0].ProductsGroups[2].Products.Add(new Product("DuckPro Ultra", 19000));

        stores.Add(new Store("Space Inc."));
        stores[1].ProductsGroups.Add(new ProductsGroup("Mobile phones"));
        stores[1].ProductsGroups[0].Products.Add(new Product("Rocket 1A", 900));
        stores[1].ProductsGroups[0].Products.Add(new Product("Rocket 2X", 3999));
        stores[1].ProductsGroups[0].Products.Add(new Product("Rocket 3E", 20000));
        stores[1].ProductsGroups.Add(new ProductsGroup("Notebooks"));
        stores[1].ProductsGroups[1].Products.Add(new Product("Shuttle 1A", 9999.99));
        stores[1].ProductsGroups[1].Products.Add(new Product("Shuttle 1X", 14000));
        stores[1].ProductsGroups[1].Products.Add(new Product("Shuttle Pro", 20000));
        stores[1].ProductsGroups.Add(new ProductsGroup("Computers"));
        stores[1].ProductsGroups[2].Products.Add(new Product("IssPC 2D", 10000.99));
        stores[1].ProductsGroups[2].Products.Add(new Product("IssPro 2X", 15000));
        stores[1].ProductsGroups[2].Products.Add(new Product("IssPro Pro", 19000));

        stores.Add(new Store("Fruit Inc."));
        stores[2].ProductsGroups.Add(new ProductsGroup("Mobile phones"));
        stores[2].ProductsGroups[0].Products.Add(new Product("Pineapple 1", 2900));
        stores[2].ProductsGroups[0].Products.Add(new Product("Mango 1", 3099));
        stores[2].ProductsGroups[0].Products.Add(new Product("Orange 1", 5000));
        stores[2].ProductsGroups.Add(new ProductsGroup("Notebooks"));
        stores[2].ProductsGroups[1].Products.Add(new Product("Mandarin X", 9999.99));
        stores[2].ProductsGroups[1].Products.Add(new Product("Lemon X", 14000));
        stores[2].ProductsGroups[1].Products.Add(new Product("Lemon Pro", 20000));
        stores[2].ProductsGroups.Add(new ProductsGroup("Computers"));
        stores[2].ProductsGroups[2].Products.Add(new Product("Plum X", 10000.99));
        stores[2].ProductsGroups[2].Products.Add(new Product("Plum Z", 15000));
        stores[2].ProductsGroups[2].Products.Add(new Product("Plum Pro", 19000));

        return stores;
    }
}
```

The below-mentioned code snippet binds the TreeView control to these classes.

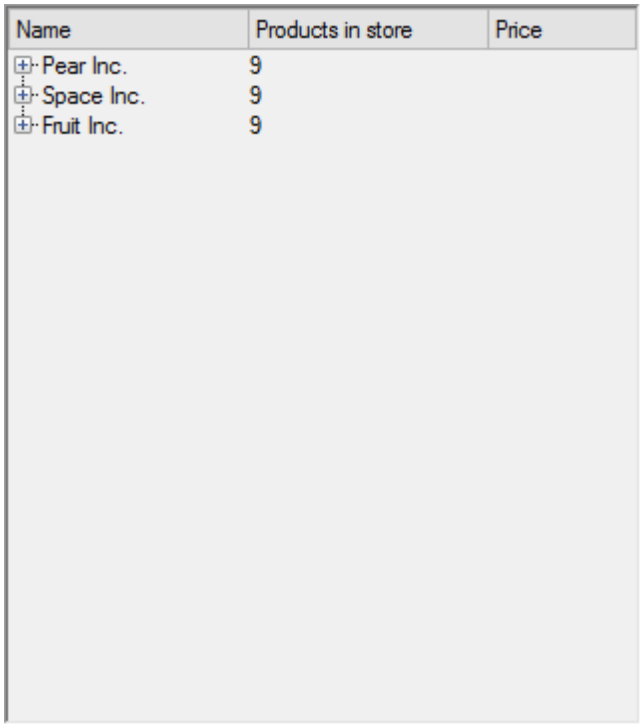
- **Visual Basic**

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

    C1TreeView1.Columns.Clear()
    C1TreeView1.DataSource = Nothing
    C1TreeView1.DataMember = "\ProductsGroups\Products"
    Dim column = New C1.Win.TreeView.C1TreeColumn()
    column.HeaderText = "Name"
    C1TreeView1.Columns.Add(column)
    column = New C1.Win.TreeView.C1TreeColumn()
    column.DisplayFieldName = "CountOfProducts\CountOfProducts\"
    column.HeaderText = "Products in store"
    C1TreeView1.Columns.Add(column)
    column = New C1.Win.TreeView.C1TreeColumn()
    column.DisplayFieldName = "\\Price"
    column.HeaderText = "Price"
    C1TreeView1.Columns.Add(column)
    C1TreeView1.DataSource = StoreCollection.GetData()
End Sub
```

- **C#**

```
public Form1()
{
    InitializeComponent();

    c1TreeView1.Columns.Clear();
    c1TreeView1.DataSource = null;
    c1TreeView1.DataMember = "\\ProductsGroups\\Products";
    var column = new C1.Win.TreeView.C1TreeColumn();
    column.HeaderText = "Name";
    c1TreeView1.Columns.Add(column);
    column = new C1.Win.TreeView.C1TreeColumn();
    column.DisplayFieldName = "CountOfProducts\\CountOfProducts\\";
    column.HeaderText = "Products in store";
    c1TreeView1.Columns.Add(column);
    column = new C1.Win.TreeView.C1TreeColumn();
    column.DisplayFieldName = "\\\\Price";
    column.HeaderText = "Price";
    c1TreeView1.Columns.Add(column);
    c1TreeView1.DataSource = StoreCollection.GetData();
}
```

Run the code to observe the following output.



Styling and Appearance

This section provides you with information on customizing the appearance of the TreeView control and its elements. You can customize the control in a number of ways, as explained in the following sections:

- Themes
- Tree Lines
- Right-to-Left Support
- Custom Button Images for Nodes
- Icons for Expand Button and Checkbox

Themes

You can apply a number of built-in or custom themes to present TreeView with a consistent and customized look by using **Themes for WinForms**. For more information, refer to the Themes for WinForms documentation.

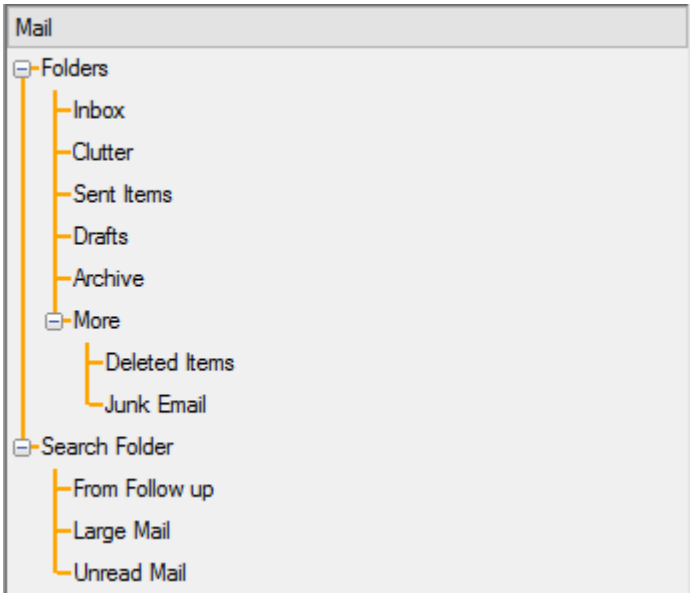
Tree Lines

Tree lines, as the name suggests, are the lines connecting the nodes of a treeview displaying the relationship between parent and child nodes clearly.

TreeView allows you to display or hide the tree lines by setting the ShowLines property provided by the TreeViewStyles class to **true** or **false**. In addition, the control enables you to perform customizations, as follows:

- **Color**: Change the color of the connected lines between the nodes by setting the LinesColor property of the TreeViewStyles class.
- **Width**: Set the width of the tree lines by setting the LinesWidth property of TreeViewStyles to integer values.
- **Style**: Specify the style of the tree lines by setting the LinesStyle property of TreeViewStyles from the DashStyle enumeration.

The following image shows TreeView with the tree lines customized with respect to their color, width, and style.



The following code snippet enables the visibility of the tree lines and customizes them in terms of their width, color, and style.

- **Visual Basic**

```
'set the ShowLines property
C1TreeView1.Styles.ShowLines = True

'set the LinesWidth property
C1TreeView1.Styles.LinesWidth = 2

'set the LinesColor proprerty
C1TreeView1.Styles.LinesColor = System.Drawing.Color.Orange

' set the LinesStyle property
C1TreeView1.Styles.LinesStyle = Drawing2D.DashStyle.Solid
```

- **C#**

```
// set the ShowLines property
c1TreeView1.Styles.ShowLines = true;

// set the LinesWidth property
c1TreeView1.Styles.LinesWidth = 2;

// set the LinesColor proprerty
c1TreeView1.Styles.LinesColor = System.Drawing.Color.Orange;

// set the LinesStyle property
c1TreeView1.Styles.LinesStyle = System.Drawing.Drawing2D.DashStyle.Solid;
```

Right-to-Left Support

TreeView provides support for languages that follow Right-to-Left scripts. The control enables you to present all treeview contents in the right-to-left direction by using the **RightToLeft** property of **System.Windows.Forms.Control**. You can set the property to any of the values in the **RightToLeft** enumeration of **Systems.Windows.Forms** as follows:

- **No**: Sets the direction from left to right, which is the default direction.
- **Yes**: Sets the direction from right to left.
- **Inherit**: Inherits the direction from the parent control.

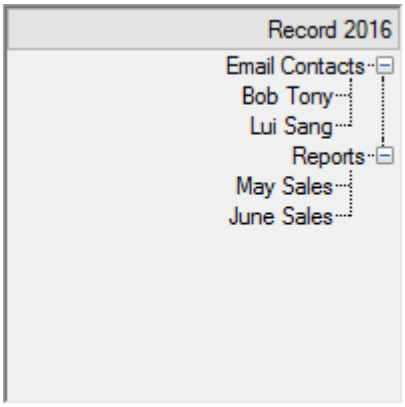
The following code snippets illustrate how to set these properties in code.

- **Visual Basic**

```
' set the RightToLeft property
C1TreeView1.RightToLeft = System.Windows.Forms.RightToLeft.Yes
```

- **C#**

```
// set the RightToLeft property
c1TreeView1.RightToLeft = System.Windows.Forms.RightToLeft.Yes;
```



Custom Button Images for Nodes

TreeView allows you to easily display custom button images for nodes by using an image list.

Create an instance of the **Systems.Windows.Forms.ImageList** class and add images to the list by using the Add method with the **Images** collection of the list. Set this instance as the image list of TreeView by using the ImageList property of the C1TreeView class.

To access the collection of images for a particular node, you need to use the Images property of C1TreeNode. And to add a specific image from the image list to a specific node, add the index of the image by using the **Add** method of **System.Collections.ObjectModel** with the Images property of C1TreeNode.

To see the implementation, refer to the following code snippets.

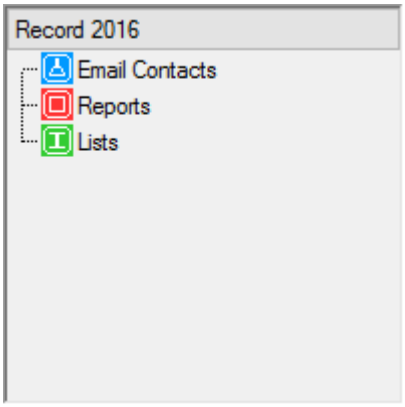
- **Visual Basic**

```
' create an instance of ImageList
Dim imageList As New ImageList()

' add images to the image list
imageList.Images.Add(Image.FromFile("C:\Resources\1.png"))
imageList.Images.Add(Image.FromFile("C:\Resources\2.png"))
imageList.Images.Add(Image.FromFile("C:\Resources\3.png"))

' set the image list instance as the TreeView image list
C1TreeView1.ImageList = imageList

' specify image indices for nodes
C1TreeView1.Nodes(0).Images.Add(0)
C1TreeView1.Nodes(1).Images.Add(1)
C1TreeView1.Nodes(2).Images.Add(2)
```

- **C#**

```
// create an instance of ImageList
ImageList imageList = new ImageList();

// add images to the image list
imageList.Images.Add(Image.FromFile("C:\\Resources\\1.png"));
imageList.Images.Add(Image.FromFile("C:\\Resources\\2.png"));
imageList.Images.Add(Image.FromFile("C:\\Resources\\3.png"));

// set the image list instance as the TreeView image list
c1TreeView1.ImageList = imageList;

// specify image indices for nodes
c1TreeView1.Nodes[0].Images.Add(0);
```

```
c1TreeView1.Nodes[1].Images.Add(1);
c1TreeView1.Nodes[2].Images.Add(2);
```



Icons for Expand Button and Checkbox

In TreeView, you can customize the way icons appear for expand buttons and check boxes.

To customize the icons for expand buttons, you need to set the ExpandButtonStyle property provided by the TreeViewStyles class. The property accepts values from the ExpandButtonStyle enumeration. By using the values, you can set any of standard System, VS2015, or Windows10 views for the expand buttons. To customize the icons for check boxes, set the CheckBoxStyle property of TreeViewStyles, which accepts values from the CheckBoxStyle enumeration. Using those values, you can set any of the standard System, MS Office, or Windows10 views for the check boxes.

Notice that you can use ExpandButtonStyle and CheckBoxStyle properties only after accessing TreeView styles by using the Styles property of C1TreeView.

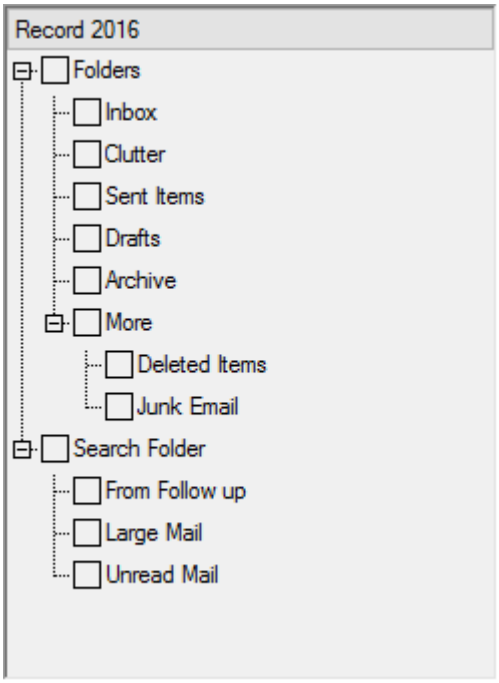
The following code snippets demonstrate the implementation.

- **Visual Basic**

```
' set the CheckBoxStyle property
C1TreeView1.Styles.CheckBoxStyle = C1.Win.TreeView.CheckBoxStyle.Windows10

' set the ExpandButtonStyle property
C1TreeView1.Styles.ExpandButtonStyle = C1.Win.TreeView.ExpandButtonStyle.Windows10
```

- **C#**

```
// set the CheckBoxStyle property
c1TreeView1.Styles.CheckBoxStyle = C1.Win.TreeView.CheckBoxStyle.Windows10;

// set the ExpandButtonStyle property
c1TreeView1.Styles.ExpandButtonStyle = C1.Win.TreeView.ExpandButtonStyle.Windows10;
```



Styling a Single Node or Node Cell

TreeView allows you to change the style of a single node or node cell by using the **ApplyNodeStyles** and the **ApplyNodeCellStyles** event of **C1TreeView**. The **ApplyNodeStyles** event occurs when you apply styles to a node, while the **ApplyNodeCellStyles** event occurs when you apply styles to a cell of a node.

To customize a node or a node cell, you first need to specify the node level by using the **Level** property of the **C1TreeNode** class. And then you can use the **NodeStyles** or the **NodeCellStyles** property provided by the **C1TreeViewNodeStylesEventArgs** and the **C1TreeViewNodeCellStylesEventArgs** class respectively, to access properties of **NodeCellStyle** and **TreeNodeCellStyles** classes. Using various properties provided by these classes, such as **BackColor**, **ForeColor**, **Default**, and others, you can customize a specific node or node cell.

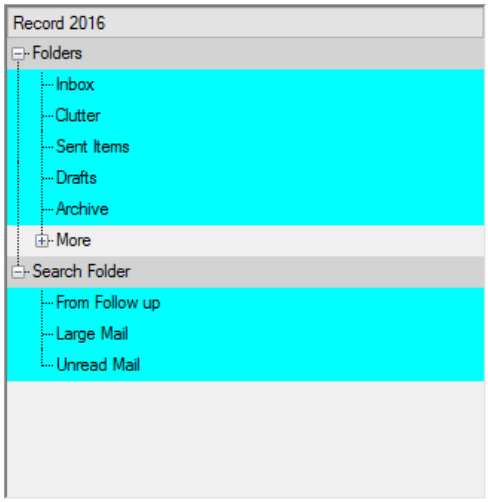
The following code snippet shows the implementation.

- **Visual Basic**

```
Private Sub C1TreeView1_ApplyNodeStyles(sender As Object, e As C1TreeViewNodeStylesEventArgs)

    If (e.Node.Level = 1) OrElse (e.Node.Level = 2) Then
        e.NodeStyles.[Default].BackColor = Color.Aqua
    End If
End Sub
Private Sub C1TreeView1_ApplyNodeCellStyles(sender As Object, e As C1.Win.TreeView.C1TreeViewNodeCellStylesEventArgs)

    If e.Node.Level = 0 AndAlso e.ColumnIndex = 0 Then
        e.NodeCellStyles.[Default].BackColor = Color.LightGray
    End If
End Sub
```

- **C#**

```
private void C1TreeView1_ApplyNodeStyles(object sender, C1.Win.TreeView.C1TreeViewNodeStylesEventArgs e)
{
    if ((e.Node.Level == 1)||(e.Node.Level == 2))
        e.NodeStyles.Default.BackColor = Color.Aqua;
}

private void C1TreeView1_ApplyNodeCellStyles(object sender, C1.Win.TreeView.C1TreeViewNodeCellStylesEventArgs e)
{

    if (e.Node.Level == 0 && e.ColumnIndex == 0)
        e.NodeCellStyles.Default.BackColor = Color.LightGray;
}
```



Import and Export XML

TreeView supports both exporting and importing data to and from an XML document.

The C1TreeView class provides the ReadXml method that allows you to load the treeView with the data of the XML document. The method accepts the XML file path as the parameter to determine the file from which the data is to be loaded.

In addition, the C1TreeView class provides the WriteXml method that enables you to save the treeview contents to an XML document. Just like the ReadXml method, this method also accepts the file path as the parameter to determine the file to which the contents of the treeview are to be saved.

The following code snippet first imports data from an XML document (**Doc.XML**) to the TreeView control, and then exports data from the TreeView control to another XML document (**TreeView.XML**).

- **Visual Basic**

```
' call the ReadXml method
C1TreeView1.ReadXml("C:/Users/GPCTAdmin/Desktop/Doc.XML")

' call the WriteXml method
C1TreeView1.WriteXml("C:/Users/GPCTAdmin/Desktop/TreeView.XML")
```

- **C#**

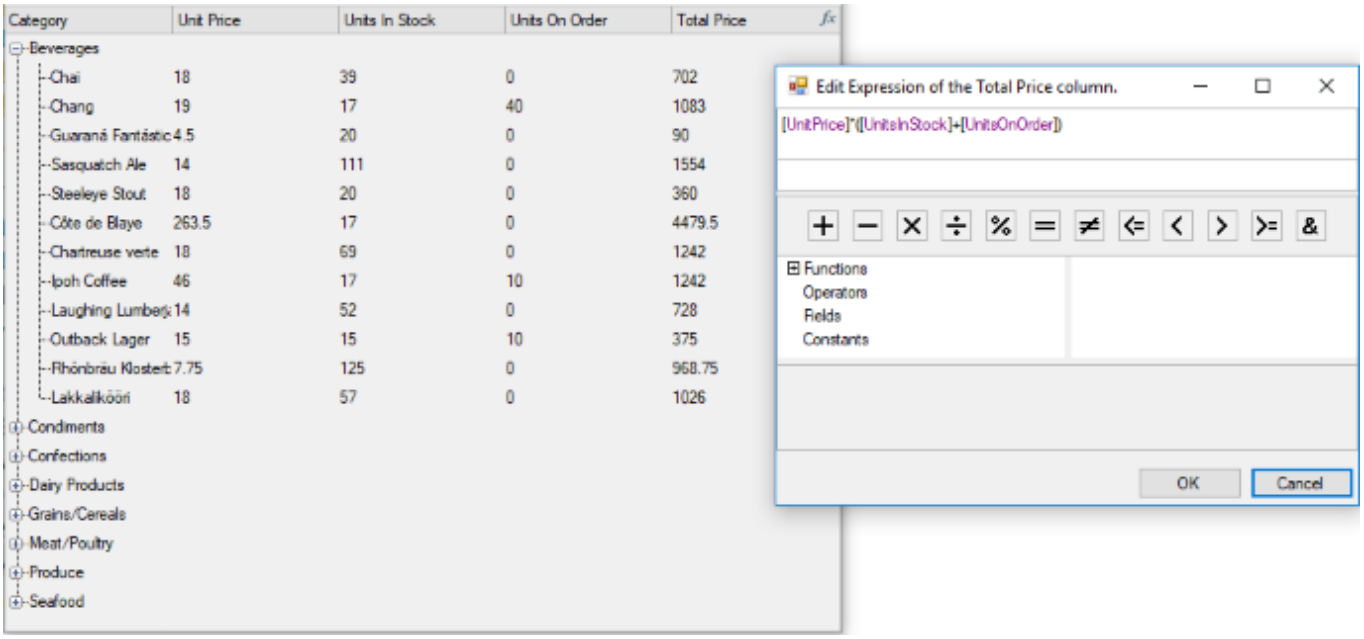
```
// call the ReadXml method
c1TreeView1.ReadXml("C:/Users/GPCTAdmin/Desktop/Doc.XML");

// call the WriteXml method
c1TreeView1.WriteXml("C:/Users/GPCTAdmin/Desktop/TreeView.XML");
```

Integration with Expression Editor

TreeView supports integration with the Expression Editor control. When integrated with Expression Editor, expressions can be used to perform operations such as column calculation over the data in TreeView. These expressions can be used on all the node levels available in the TreeView control.

To integrate Expression Editor with the TreeView, you need to use ExpressionInfo property of C1TreeColumn class, which contains information about expressions. These expressions can be set for columns using Expressions property of **ExpressionInfo** class.

The following image shows the TreeView control integrated with Expression Editor.



The ExpressionInfo class provides various other properties listed below that can be used to manipulate expressions in TreeView control. These properties can be used to add, edit, or delete expressions on any of the node levels available in TreeView.

Property	Description
AllowAddNew	Indicates the ability to add node level expression
AllowDelete	Indicates the ability to remove node level expression
AllowEdit	Indicates whether the expressions of this column can be edited by clicking the Expression Editor icon in the column header
ExpandLastExpression	Indicates whether the last expression expand to all subsequent levels of the TreeView even if the expression is not defined for them

The following code demonstrates integration of TreeView with Expression Editor. In this code, expression is used on the fifth column of the TreeView control which contains two levels. Here, we have added an expression for the second level only.For this, we set our expression to the second item of the Expressions array, and set the first item as empty.

C#

```
private void Form1_Load(object sender, EventArgs e)
{
    this.categoriesTableAdapter.Fill(this.c1NWindDataSet.Categories);
    this.productsTableAdapter.Fill(this.c1NWindDataSet.Products);
```

```
    c1TreeView1.BindingInfo.DataSource = c1NWindDataSetBindingSource;

    c1TreeView1.Columns[4].ExpressionInfo.Expressions = new
              string[] { "", "[UnitPrice]*([UnitsInStock]+[UnitsOnOrder])" };

    c1TreeView1.Columns[4].ExpressionInfo.AllowEdit = new bool[] { true, true };
    c1TreeView1.Columns[4].ExpressionInfo.AllowAddNew = true;
    c1TreeView1.Columns[4].ExpressionInfo.AllowDelete = true;
}
```