IntegralUI TreeView

v4.0 for .NET

Rich hierarchical data presentation control

User Guide

for IntegralUI TreeView v4.0

Table of contents

Introduction

This guide will provide you with information on how to work with IntegralUI TreeView control and help you to successfully include it in your applications.

With IntegralUI TreeView you can present your data in very customizable way. This control allows you to create hierarchical structure of your data in standard way, node contains an icon and a label, or by using XML tags which results in nodes with rich content.

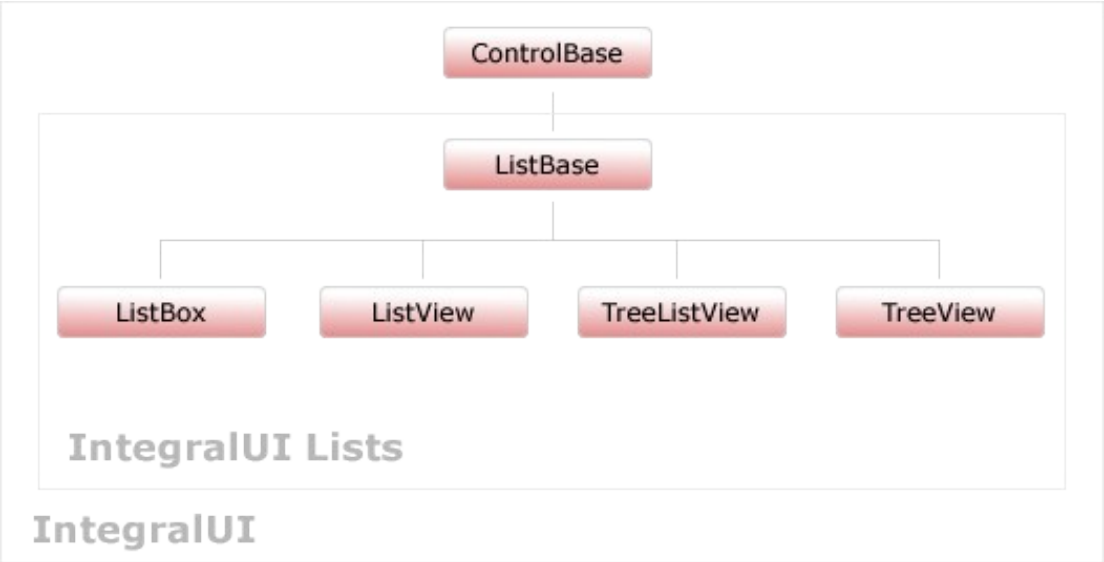
Here are some of the main features:

- Highly customizable appearance
- Rich content: Text, Images, Hyperlinks, Controls, CheckBox, Flags; can be included in every node
- Arrange node content in custom layouts
- Advanced Drag&Drop operations
- Built-in sorting and option to add custom sort operations
- Fast loading along with custom progress presentation
- XML encoding & serialization
- Theme support

Architecture

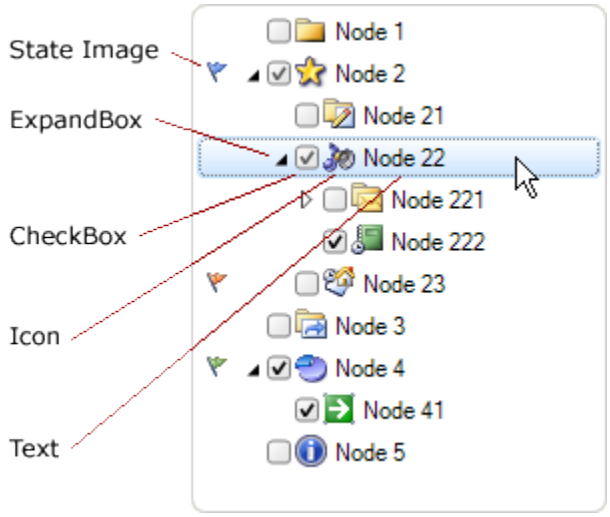
IntegralUI TreeView control is part of IntegralUI family of products, and as such uses common infrastructure shared among other controls. The TreeView class is inherited from ListBase class which is a parent class for all list controls in IntegralUI class library. Because of this, you will find many similarities between other controls like ListBox, ListView and TreeListView, which all belongs to the IntegralUI Lists class library.

Object model

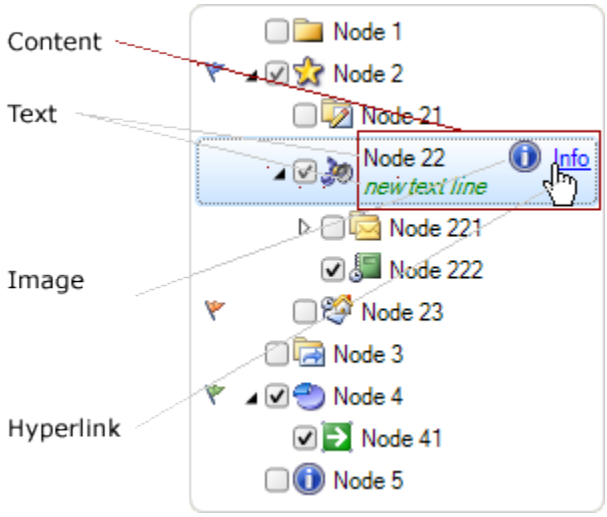


To fill the TreeView control with data you need to use TreeNodeCollection which holds a collection of nodes. Every node can contain standard objects like StateImage, CheckBox, Icon, Text arranged in single line. But also it can contain custom Content, which can have custom layouts in multiple lines.

If you want to present not just Text in the node, then the best is to use the XML tags. By using XML tags you can create various objects like Text, Images, Hyperlinks, Controls, CheckBox, Icons, Animated gifs etc. and arrange them in custom layouts by using Tables and paragraphs.

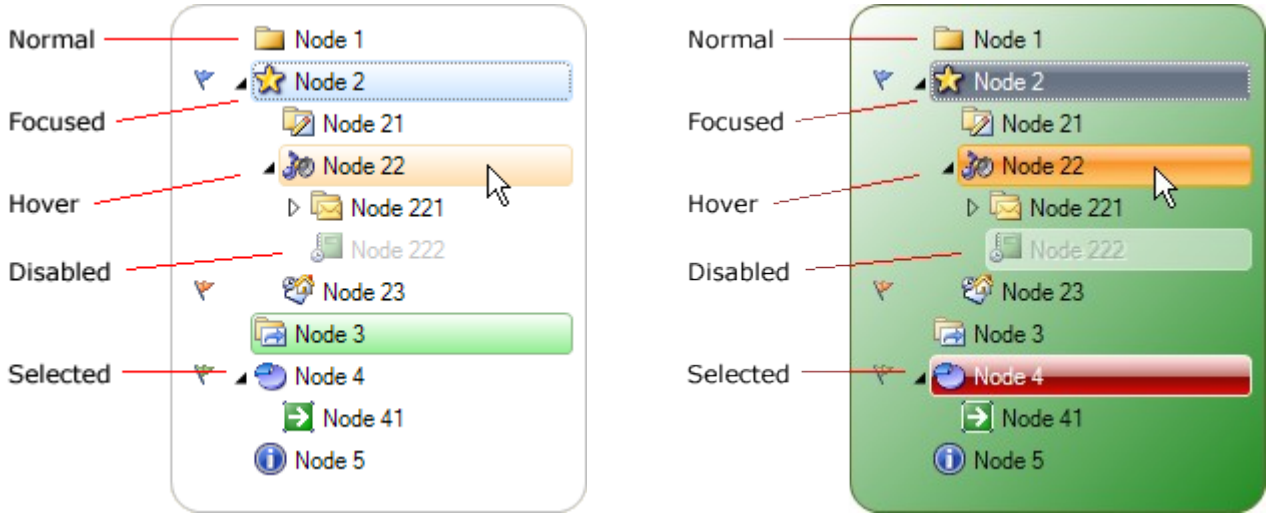


Normal mode



Using XML Encoding

In order to create custom look you can use many color and format styles for TreeView control and for every node individually. Additionally, in further customization the styles can be changed for every part of the node content by using special XML tags.



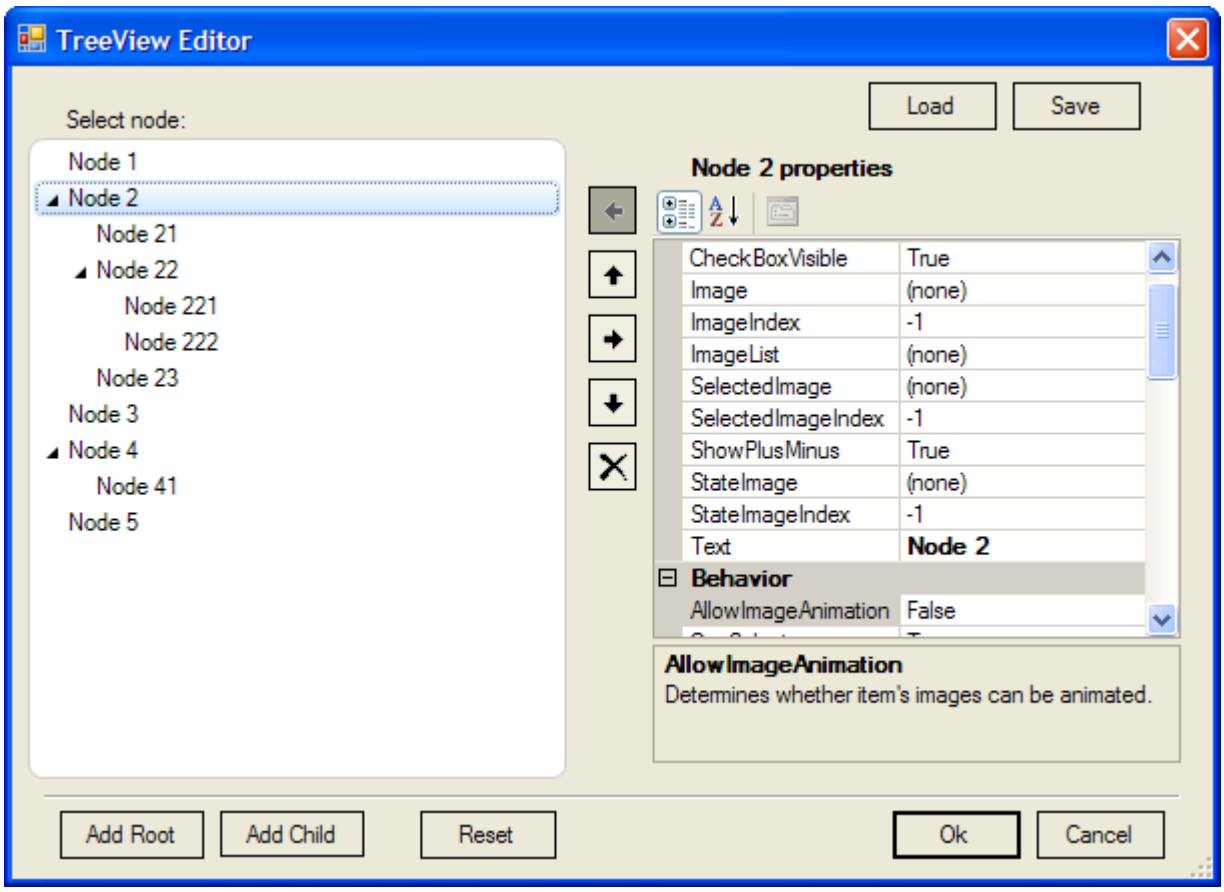
Event model

There are many events which can help you to determine the current state of TreeView control. Here is a complete list of events:

- AfterCheck — Occurs after the node's check box is checked
- AfterCollapse — Occurs after the node is collapsed
- AfterExpand — Occurs after the node is expanded
- AfterLabelEdit — Occurs after the node text is edited
- AfterSelect — Occurs after the node is selected
- BeforeCheck — Occurs before the node's check box is checked
- BeforeCollapse — Occurs before the node is collapsed
- BeforeExpand — Occurs before the node is expanded
- BeforeLabelEdit — Occurs before the node text is edited
- BeforeSelect — Occurs before the node is selected
- FocusedNodeChanged — Occurs after the focused node is changed
- FocusedNodeChanging — Occurs before the focused node is changed
- NodeAdded — Occurs after new node is added
- NodeAdding — Occurs before new node is added
- ItemDrag — Occurs when the user begins dragging an node
- ItemMouseHover — Occurs when the mouse cursor hovers for some time over an node space
- ItemObjectClicked — Occurs after the user clicks an custom object (placed with XML encoding) in the control space
- ItemObjectClicking — Occurs before the user clicks an custom object (placed with XML encoding) in the control space
- NodeRemoved — Occurs after node is removed from collection
- NodeRemoving — Occurs before node is removed from collection
- ScrollPosChanged — Occurs when position of scrollbar has changed
- SelectionModeChanged — Occurs when selection type changes

Editor

There are two ways to create tree nodes, during design-time or programmatically.
In design-time you can use the TreeView Editor by right-clicking on the TreeView control. The form will appear in which you can add, remove, change the node order in collection or set the properties for each node.



In order to create nodes programmatically, you can use the following code:

```
[VB]
' Assuming that TreeView control exist
Dim node As New LidorSystems.IntegralUI.Lists.TreeNode("Node")
Me.treeView1.Nodes.Add(node)
Me.treeView1.UpdateLayout()

[C#]
// Assuming that TreeView control exist
LidorSystems.IntegralUI.Lists.TreeNode node = new
LidorSystems.IntegralUI.Lists.TreeNode("Node");
this.treeView1.Nodes.Add(node);
this.treeView1.UpdateLayout();
```

Or, you can create a node collection and add it to the node:

```
[Visual Basic]
' Assuming that TreeView control exist
' Suspend the layout logic for the TreeView control
Me.TreeView1.SuspendUpdate()

' Create small array of child nodes
Dim children(2) As LidorSystems.IntegralUI.Lists.TreeNode
Dim i As Integer = 0

While i < 3
   children(i) = New LidorSystems.IntegralUI.Lists.TreeNode("ChildNode " +
i.ToString)
   System.Math.Min(System.Threading.Interlocked.Increment(i), i - 1)
End While

Dim node As LidorSystems.IntegralUI.Lists.TreeNode = New
LidorSystems.IntegralUI.Lists.TreeNode("Node", children)
Me.TreeView1.Nodes.Add(node)

' Resume the layout logic for the TreeView control
Me.TreeView1.ResumeUpdate()

[C#]
// Assuming that TreeView control exist
// Suspend the layout logic for the TreeView control
this.treeView1.SuspendUpdate();

// Create small array of child nodes
LidorSystems.IntegralUI.Lists.TreeNode[] children = new
LidorSystems.IntegralUI.Lists.TreeNode[3];

for (int i = 0; i < 3; i++)
   children[i] = new LidorSystems.IntegralUI.Lists.TreeNode("ChildNode " +
i.ToString());

LidorSystems.IntegralUI.Lists.TreeNode node = new
LidorSystems.IntegralUI.Lists.TreeNode("Node", children);
this.treeView1.Nodes.Add(node);

// Resume the layout logic for the TreeView control
this.treeView1.ResumeUpdate();
```

Appearance

Working with styles

The IntegralUI TreeView is very customizable control. You can customize every part of the control, starting from the background, border, nodes etc. This is done by numerous styles with which you can set colors, fonts, alignment, rendering mode. Here is the list of color and format styles:

- **CheckBoxStyle** - The drawing style of the node's check box
- **ColorStyle** - The drawing style of the control
- **ExpandBoxStyle** - The drawing style of the node expand/collapse button
- **FormatStyle** - The format style of the control
- **ScrollBarStyle** - The drawing style of horizontal and vertical scrollbar
- **ToolTipStyle** - The drawing style of node's tooltip

To simplify control over node's appearance, there are general styles which are used for all nodes:

- **DisabledNodeStyle** - The drawing style of node when it's disabled
- **FocusedNodeStyle** - The drawing style of node with input focus
- **HoverNodeStyle** - The drawing style of node when mouse hovers over it
- **NodeFormatStyle** - Style by which the node content is formatted
- **NormalNodeStyle** - The default drawing style of the node
- **SelectedNodeStyle** - The drawing style of node when it's selected

Furthermore, every node have their own set of the above styles with witch you can customize their appearance individually. These styles are:

- **DisabledStyle** - The drawing style of node when it's disabled
- **FocusedStyle** - The drawing style of node with input focus
- **HoverStyle** - The drawing style of node when mouse hovers over it
- **FormatStyle** - Style by which the node content is formatted
- **NormalStyle** - The default drawing style of the node
- **SelectedStyle** - The drawing style of node when it's selected

How to change the appearance of control border

In some cases you may want to change the border of the TreeView to appear differently then the standard rectangular form. To achieve this goal several properties of the control FormatStyle needs to be changed. Here is a list of properties which controls a different part of the border:

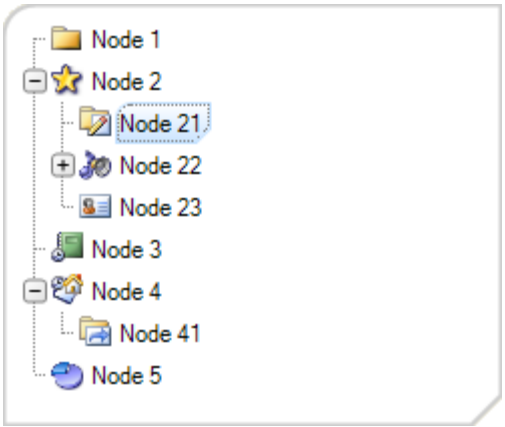
BorderCornerRadius – holds the value by which the border corner is rounded
BorderCornerShape – responsible for changing the shape of every border corner
BorderLineStyle – determines the thickness of border line
BorderVisibility – determines which side of the border is visible

These properties also exist in format style of every node.

Here is an example where you can see how border can be customized:

```
[VB]
Imports LidorSystems.IntegralUI.Style
. . .

' Changing the border of the TreeView control
Me.treeView1.FormatStyle.BorderCornerRadius = 15
Me.treeView1.FormatStyle.BorderCornerShape.BottomLeft = CornerShape.Squared
```

```
Me.treeView1.FormatStyle.BorderCornerShape.BottomRight = CornerShape.Chamfered
Me.treeView1.FormatStyle.BorderCornerShape.TopRight = CornerShape.Squared
Me.treeView1.FormatStyle.BorderLineStyle = LineStyle.Double

' Changing the border of the nodes
Me.treeView1.NodeFormatStyle.BorderCornerRadius = 7
Me.treeView1.NodeFormatStyle.BorderCornerShape.BottomLeft = CornerShape.Squared
Me.treeView1.NodeFormatStyle.BorderCornerShape.BottomRight =
CornerShape.Chamfered
Me.treeView1.NodeFormatStyle.BorderCornerShape.TopRight = CornerShape.Squared
Me.treeView1.NodeFormatStyle.Padding = New Padding(3, 2, 3, 2)

[C#]
using LidorSystems.IntegralUI.Style;
. . .

// Changing the border of the TreeView control
this.treeView1.FormatStyle.BorderCornerRadius = 15;
this.treeView1.FormatStyle.BorderCornerShape.BottomLeft = CornerShape.Squared;
this.treeView1.FormatStyle.BorderCornerShape.BottomRight = CornerShape.Chamfered;
this.treeView1.FormatStyle.BorderCornerShape.TopRight = CornerShape.Squared;
this.treeView1.FormatStyle.BorderLineStyle = LineStyle.Double;

// Changing the border of the nodes
this.treeView1.NodeFormatStyle.BorderCornerRadius = 7;
this.treeView1.NodeFormatStyle.BorderCornerShape.BottomLeft =
CornerShape.Squared;
this.treeView1.NodeFormatStyle.BorderCornerShape.BottomRight =
CornerShape.Chamfered;
this.treeView1.NodeFormatStyle.BorderCornerShape.TopRight = CornerShape.Squared;
this.treeView1.NodeFormatStyle.Padding = new Padding(3, 2, 3, 2);
```



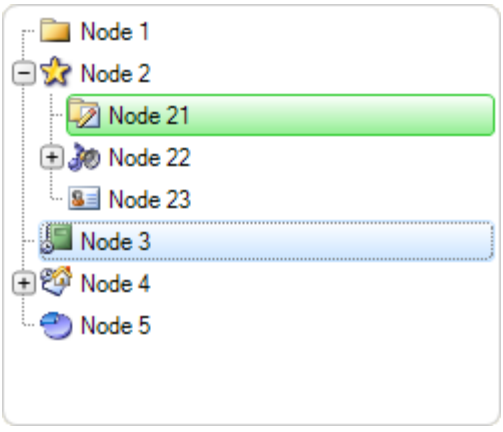
Use of individual styles for nodes

By default the appearance of nodes is controlled by set of styles from their parent TreeView control.

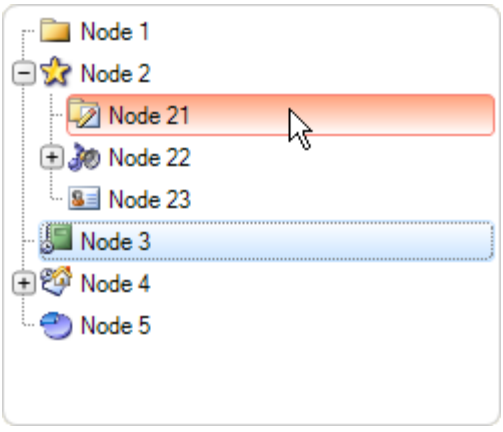
If you want to have separate look for a specific node, you can customize it from their individual styles. Before changing any of these styles, the StyleFromParent property for this node must be set to False. In the following example we will change the look of the node in its normal and hovered state:

```
[VB]
' Set the StyleFromParent to False, so that
' a specific style changes be applied
node.StyleFromParent = False

' Change the look of the node, when it is in normal state
node.NormalStyle.BackColor = Color.LightGreen
node.NormalStyle.BorderColor = Color.LimeGreen

' Change the look of the node, when it is in hovered state
node.HoverStyle.BackColor = Color.LightSalmon
node.HoverStyle.BorderColor = Color.Salmon
node.HoverStyle.FillStyle = FillStyle.Vertical

' Repaing the control
Me.treeView1.Invalidate()

[C#]
// Set the StyleFromParent to False, so that
// a specific style changes be applied
node.StyleFromParent = false;

// Change the look of the node, when it is in normal state
node.NormalStyle.BackColor = Color.LightGreen;
node.NormalStyle.BorderColor = Color.LimeGreen;

// Change the look of the node, when it is in hovered state
node.HoverStyle.BackColor = Color.LightSalmon;
node.HoverStyle.BorderColor = Color.Salmon;
node.HoverStyle.FillStyle = FillStyle.Vertical;

// Repaing the control
this.treeView1.Invalidate();
```

In the code above the node is the third node in the collection. Here is the result:



Changes in normal state



Changes in hover state

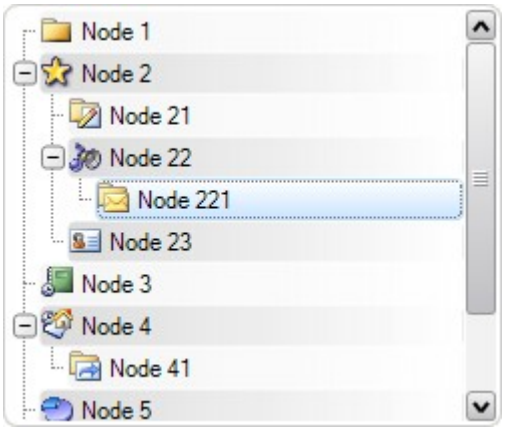
How to create alternate look of nodes

You can create standalone styles which in appropriate conditions can be applied to specific nodes. For example, every node with an even index can have one appearance and every node with odd index can have another appearance. But instead of changing the colors for every node, you need only to create two color styles, one style for the even rows and other for the odd rows.

Here is a sample code that creates node list with alternate look:

```
[VB]
Imports LidorSystems.IntegralUI.Lists
Imports LidorSystems.IntegralUI.Lists.Style

. . .

' Create the even color style
Dim evenNodeStyle As New ListItemColorStyle()
evenNodeStyle.BackColor = Color.WhiteSmoke
evenNodeStyle.FillStyle = FillStyle.Horizontal

' Create the odd color style
Dim oddNodeStyle As New ListItemColorStyle()
oddNodeStyle.BackColor = Color.Gainsboro
oddNodeStyle.FillStyle = FillStyle.Horizontal

' Cycle through all nodes and apply style changes
For Each node As LidorSystems.IntegralUI.Lists.TreeNode In Me.treeView1.FlatNodes
    ' When you use individual styles for an node
    ' the StyleFromParent needs to be set to False
    node.StyleFromParent = False
    If node.FlatIndex Mod 2 = 0 Then
        node.NormalStyle = evenNodeStyle
    Else
        node.NormalStyle = oddNodeStyle
    End If
Next

' Repaing the control
Me.treeView1.Invalidate()

[C#]
using LidorSystems.IntegralUI.Lists;
using LidorSystems.IntegralUI.Lists.Style;

. . .

// Create the even color style
ListItemColorStyle evenNodeStyle = new ListItemColorStyle();
evenNodeStyle.BackColor = Color.WhiteSmoke;
evenNodeStyle.FillStyle = FillStyle.Horizontal;

// Create the odd color style
ListItemColorStyle oddNodeStyle = new ListItemColorStyle();
oddNodeStyle.BackColor = Color.Gainsboro;
oddNodeStyle.FillStyle = FillStyle.Horizontal;

// Cycle through all nodes and apply style changes
foreach (LidorSystems.IntegralUI.Lists.TreeNode node in this.treeView1.FlatNodes)
```

```
{
   // When you use individual styles for an node
   // the StyleFromParent needs to be set to False
   node.StyleFromParent = false;
   if (node.FlatIndex % 2 == 0)
      node.NormalStyle = evenNodeStyle;
   else
      node.NormalStyle = oddNodeStyle;
}

// Repaing the control
this.treeView1.Invalidate();
```



As you can see from the code we have used the FlatNodes collection to cycle through nodes. This is a collection which holds linear structure of TreeView control, instead hierarchical structure. Furthermore, the position and level of all nodes are persisted. It is best to use this collection if you have large set of nodes, and you need to perform some operation like searching. The performance is great.

Style serialization and reuse

If you want to create some predefined color schemes and apply it to the control at specific conditions, one way to do that is by serializing the control layout. You only would need to set the color and format styles and then save the layout in external xml file or database. After that, you only need to load the file which corresponds to the target color scheme.

You can read more on how serialization is done at the end of this document in Serialization section.

Visual Styles

There are three predefined visual styles:

- **Classic** - Used to render controls in Windows Classic control style
- **XP** - Used to render controls in Windows XP control style
- **Vista** - Used to render controls in Windows Vista control style

By setting the VisualStyle property to some of the above options, different appearance is applied to the control.

Themes

If you want controls to adjust to the current theme and color scheme of windows operating system you need to set **UseTheme** property value to **True**. Depending of the current **VisualStyle** a predefined color scheme is used which corresponds to the current operating system color scheme.

Note For creation of more advanced appearance, you can read in XML encoding section.

Behavior

Working with Node collection

Add/Remove operations

The nodes displayed in the TreeView control are stored in Nodes property. This is an object from TreeNodeCollection class, which has several methods and events you can use to add, remove, clear or make any other operation that will change the collection. Here is a list of methods that you can use:

- Add – adds a new node at the end of the collection
- AddRange – add a set of nodes to the end of the collection
- Clear – empties the list
- Remove – deletes the node from the collection
- RemoveAt – deletes the node that is located at specified location in the collection

Here is an example how to add a new node programmatically to the collection:

```
[VB]
Dim node As New TreeNode("New node")
Me.treeView1.Node.Add(node)

[C#]
TreeNode node = new TreeNode("New node");
this.treeView1.Nodes.Add(node);
```

To insert this node at a specified position in the collection, use the Insert method:

```
[VB]
' Adds the node at sixth position in the collection
Me.treeView1.Nodes.Insert(5, node)

[C#]
// Adds the node at sixth position in the collection
this.treeView1.Nodes.Insert(5, node);
```

When you add a new node to the TreeView control, this process is accompanied with two events:

- NodeAdding – it is fired before node is added and can cancel the add operation
- NodeAdded – it is fired after the node is added

To remove a node from the collection, two methods can be used:

```
[VB]
' Removes the node from the collection
Me.treeView1.Nodes.Remove(node)

' Removes the node located at the specified location from the collection
Me.treeView1.Nodes.RemoveAt(5)

[C#]
// Removes the node from the collection
this.treeView1.Nodes.Remove(node);

// Removes the node located at the specified location from the collection
this.treeView1.Nodes.RemoveAt(5);
```

If you want to remove all nodes, use the Clear method

```
[VB]
Me.treeView1.Nodes.Clear();

[C#]
this.treeView1.Nodes.Clear();
```

How to add existing node collection

If you have some predefined set of nodes, and you want this set as a whole to be added to the TreeView control, you can use AddRange method:

```
[VB]
Imports LidorSystems.IntegralUI.Lists

. . .

Dim nodes As TreeNode() = New Node(6) {}
For i As Integer = 0 To 6
   nodes(i) = New TreeNode("Item " & i.ToString())
Next

Me.treeView1.Nodes.AddRange(nodes)

[C#]
using LidorSystems.IntegralUI.Lists;

. . .

TreeNode[] nodes = new TreeNode[7];
for (int i = 0; i < 7; i++)
   nodes[i] = new TreeNode("Item " + i.ToString());

this.treeView1.Nodes.AddRange(nodes);
```

How to edit the node text

During runtime you can allow to users to edit the node text and change it. This can be done by setting the **LabelEdit** property to **True**. After that whenever the user clicks on the node and releases the mouse button, a TextBox will appear in which the node text can be edited.

If you want to start editing process in other way, you need to use **BeginEdit** and **EndEdit** methods. In the following example we will show how to programmatically start editing process:

```
[VB]
If Me.TreeView1.SelectedNode IsNot Nothing Then
   Me.TreeView1.LabelEdit = True
   Me.TreeView1.SelectedNode.BeginEdit()
End If

[C#]
```

```
if (this.treeView1.SelectedNode != null)
{
   this.treeView1.LabelEdit = true;
   this.treeView1.SelectedNode.BeginEdit();
}
```

The edit process is accompanied by two events:

- **BeforeLabelEdit** – it is fired before the TextEditor is shown and editing beggins
- **AfterLabelEdit** – it is fired when TextEditor closes and just before the new text is applied to the node Text property

In order to cancel the current edit process, you can simply press the ESC key or by handle the AfterLabelEdit event. In the following example the edit is allowed only for labels which don't contain some characters:

```
[VB]
Imports LidorSystems.IntegralUI.Lists

. . .

Private Sub treeView1_AfterLabelEdit(ByVal sender As Object, ByVal e As
LidorSystems.IntegralUI.ObjectEditEventArgs)
   If TypeOf e.[Object] Is LidorSystems.IntegralUI.Lists.TreeNode Then
      Dim node As TreeNode = DirectCast(e.[Object], TreeNode)

      If e.Label IsNot Nothing Then
         If e.Label.Length > 0 Then
            If e.Label.IndexOfAny(New Char() {"@"c, "."c, ","c, "!"c}) >= 0 Then
               ' Cancel the label edit action, inform the user, and
               ' place the node in edit mode again.
               e.Cancel = True

               MessageBox.Show("Invalid node label." & vbLf & "The invalid
characters are: '@','.', ',', '!'", "Node Label Edit")
               node.BeginEdit()
            End If
         Else
            ' Cancel the label edit action, inform the user, and
            ' place the node in edit mode again.
            e.Cancel = True

            MessageBox.Show("Invalid node label." & vbLf & "The label cannot be
blank", "Node Label Edit")
            node.BeginEdit()
         End If
      End If
   End If
End Sub

[C#]
using LidorSystems.IntegralUI.Lists;

. . .

private void treeView1_AfterLabelEdit(object sender,
LidorSystems.IntegralUI.ObjectEditEventArgs e)
```

```
{
   if (e.Object is LidorSystems.IntegralUI.Lists.TreeNode)
   {
      LidorSystems.IntegralUI.Lists.TreeNode node =
(LidorSystems.IntegralUI.Lists.TreeNode)e.Object;

      if (e.Label != null)
      {
         if (e.Label.Length > 0)
         {
            if (e.Label.IndexOfAny(new char[] { '@', '.', ',', '!' }) >= 0)
            {
                  // Cancel the label edit action, inform the user, and
                  // place the node in edit mode again.
                  e.Cancel = true;

                  MessageBox.Show("Invalid node label.\n" + "The invalid
characters are: '@','.', ',', '!'", "Node Label Edit");
                  node.BeginEdit();
            }
         }
         else
         {
            // Cancel the label edit action, inform the user, and
            // place the node in edit mode again.
            e.Cancel = true;

            MessageBox.Show("Invalid node label.\nThe label cannot be blank",
"Node Label Edit");
            node.BeginEdit();
         }
      }
   }
}
```

Checked nodes

In order to display checkboxes to the nodes, the **CheckBoxes** property of the TreeView control must be set to **True**. Every node has a check box shown to the left side of the node space. Their visibility is controlled by CheckBoxVisible property. In this way, you can decide which nodes will have checkbox. This is useful for example when you want some node to behave like a header for other nodes.

The check box can display two or three state values. By default the two-state (checked and unchecked) behavior is active. If you want to have three-state behavior, at first you need to set the **CheckMode** to **ThreeState** value. Whenever user clicks inside the check box space, it will cycle the state of the check box through Unchecked, Indeterminate and Checked values.

In some cases you will also want to change the check box state by node selection. In order to do that, set the **AllowSelectionCheck** property to **True**.

The set of checked nodes (only those with CheckState set to Checked) is stored in **CheckedItems** collection. This collection is very useful when you want to examine only nodes that are currently checked.

How to preserve checked nodes

In some cases you may want to preserve the collection of checked nodes unchanged. For example, there may be custom operation which will indirectly change the CheckState of the nodes, and in this way the CheckedItems collection. In order to prevent this from happening, we have added a **PreserveCheckState** property. With it you can preserve the current collection, run your custom operation without side effects, and then set back this property to its original value. For example:

```
[VB]
Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs)
   Me.treeView1.PreserveCheckState = True
   Me.treeView1.SelectedNode = Me.treeView1.Nodes(1)
   Me.treeView1.PreserveCheckState = False
End Sub

Private Sub treeView1_AfterSelect(ByVal sender As Object, ByVal e As
LidorSystems.IntegralUI.ObjectEventArgs)
   If TypeOf e.[Object] Is LidorSystems.IntegralUI.Lists.TreeNode Then
   Dim node As LidorSystems.IntegralUI.Lists.TreeNode = DirectCast(e.[Object],
LidorSystems.IntegralUI.Lists.TreeNode)

      Me.treeView1.CheckedNodes.Clear()
      node.Checked = True
   End If
End Sub

[C#]
private void button1_Click(object sender, EventArgs e)
{
   this.treeView1.PreserveCheckState = true;
   this.treeView1.SelectedNode = this.treeView1.Nodes[1];
   this.treeView1.PreserveCheckState = false;
}

private void treeView1_AfterSelect(object sender,
LidorSystems.IntegralUI.ObjectEventArgs e)
{
   if (e.Object is LidorSystems.IntegralUI.Lists.TreeNode)
   {
      LidorSystems.IntegralUI.Lists.TreeNode node =
(LidorSystems.IntegralUI.Lists.TreeNode)e.Object;

      this.treeView1.CheckedNodes.Clear();
      node.Checked = true;
   }
}
```

In this example we have a TreeView with some nodes in it. Whenever a button is clicked the second node is selected and checked. But along with this by calling the Clear method, we try to remove all checked nodes from CheckedNodes collection. Because we have set **PreserveCheckState** to **True**, this collection will remain intact.

Selected nodes

Multiple selections

The control supports four different modes for node selection. They are controlled from the SelectionMode property, which can have following values:

- **None** – there is no selection
- **One** – only one node can be selected
- **MultiSimple** – selection is performed with single mouse click
- **MultiExtended** – selection is perfomed with use of CTRL or SHIFT keys

In order to have multiple node selection, the SelectionMode must be set either to MultiSimple or MultiExtended value.

Whenever there is a selection, nodes that are currently selected are stored in SelectedNodes collection. This collection is very useful when you want to examine only nodes that are currently selected.

Hover selection

In some cases you may want to allow nodes to be selected while mouse cursor hovers over them. This functionality is built-in the code, and in order to allow it, just set the **HoverSelection** property to **True**.

How to preserve selection

In some cases you may want to preserve the collection of selected nodes unchanged. For example, there may be custom operation which will indirectly change the Selected state of the node, and in this way the SelectedNodes collection. In order to prevent this from happening, we have added a **PreserveSelection** property. With it you can preserve the current collection, run your custom operation without side effects, and then set back this property to its original value. For example:

```
[VB]
Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs)
   Me.treeView1.PreserveSelection = True
   Me.treeView1.SelectedNodes.Clear()
   Me.treeView1.SelectedNode = Me.treeView1.Nodes(1)
   Me.treeView1.PreserveSelection = False
End Sub

[C#]
private void button1_Click(object sender, EventArgs e)
{
   this.treeView1.PreserveSelection = true;
   this.treeView1.SelectedNodes.Clear();
   this.treeView1.SelectedNode = this.treeView1.Nodes[1];
   this.treeView1.PreserveSelection = false;
}
```

In this example we have a TreeView with some nodes in it. Whenever a button is clicked the second node is selected. But along with this by calling the Clear method, we try to remove all selected nodes from SelectedNodes collection. Because we have set **PreserveSelection** to **True**, this collection will remain intact. This only works in multiple selection mode.

Drag&Drop operations

The IntegralUI TreeView control has built-in support for standard drag&drop operations like: drag and drop of single node, copy/move the node/s from one TreeView to other controls etc. In the following sections we will give you information on how you can use various permissions during drag&drop operations, and how to extend this by creating your own custom operation.

Use of permissions

Here is a brief description on various properties that are used during drag&drop:

- **AllowDrag** – Gives permission to node/s to be dragged
- **AllowDrop** – Gives permission to the control to accept node/s during drag&drop
- **DragDropMode** – Determines the type of drag&drop operation
- **DragDropReorder** – Determines whether a reordering of nodes is allowed during drag&drop
- **ShowDropMarker** – Determines whether the marker is shown that represents the current drop position

By default drag&drop is disabled. In order to start dragging of node/s, the **AllowDrag** property must be set to **True**. However, in order for dragged nodes to be dropped, the **AllowDrop** property must also be set to **True**. We have deliberately separated the control over drag&drop operation in two states, because in some cases you may want to have one TreeView control from which nodes can only be dragged, and other TreeView control to which nodes can only be dropped.

By default there is a predefined drag&drop operation which handles all of standard cases. In order to create your own operation, the **DragDropMode** must be set to **Custom**. This case is described in detail in the next section.

If you don't want to appear the drop marker during drag&drop, the best is to disable it from the **ShowDropMarker** property. In other case, this marker will be shown. You can change the color of this marker from the ColorStyle of the TreeView, by changing the **NodePosColor** property.

How to create a custom drag&drop operation

If the default drag&drop operaton doesn't give you the solution you want, you can always create your own custom operation.

In order to do that, you need to do the following:

1. At first, you need to set the **DragDropMode** to **Custom**
2. Handle the **ItemDrag** event and start the drag&drop operation
3. Handle the **DragOver** and **DragDrop** events
4. Optionally, handle other drag&drop events, like: **DragEnter**, **DragLeave**, **QueryContinueDrag**, etc.

Here is some code that you can use:

```vb
[VB]
Imports LidorSystems.IntegralUI.Collections
Imports LidorSystems.IntegralUI.Lists
Imports LidorSystems.IntegralUI.Lists.Collections

. . .

Private Sub treeView1_ItemDrag(ByVal sender As Object, ByVal e As
ItemDragEventArgs)
   If e.Button = MouseButtons.Left Then
      Me.treeView1.DoDragDrop(e.Item, DragDropEffects.All)
   End If
End Sub

Private Sub treeView1_DragOver(ByVal sender As Object, ByVal e As DragEventArgs)
   If e.Data.GetDataPresent(GetType(TreeViewItem)) Then
      ' Depending od the control key pressed change the drag effect
      If (e.KeyState And 8) = 8 AndAlso (e.AllowedEffect And DragDropEffects.Copy)
= DragDropEffects.Copy Then
         e.Effect = DragDropEffects.Copy
      Else
         e.Effect = DragDropEffects.Move
      End If
   Else
      e.Effect = DragDropEffects.None
   End If
End Sub

Private Sub treeView1_DragDrop(ByVal sender As Object, ByVal e As DragEventArgs)
   ' Get the current mouse position
   Dim mousePos As Point = Me.treeView1.PointToClient(New Point(e.X, e.Y))

   If e.Data.GetDataPresent(GetType(LidorSystems.IntegralUI.Lists.TreeNode)) Then
      ' Get the dragged node
      Dim node As LidorSystems.IntegralUI.Lists.TreeNode =
DirectCast(e.Data.GetData(GetType(LidorSystems.IntegralUI.Lists.TreeNode)),
LidorSystems.IntegralUI.Lists.TreeNode)

      ' Get the target node (the one that is currently hovered)
      Dim targetNode As LidorSystems.IntegralUI.Lists.TreeNode =
Me.treeView1.GetNodeAt(mousePos)

      ' Suspend the treeview layout to increase performance
      Me.treeView1.SuspendUpdate()

      ' In Copy operation, create a clone node and then add it to the target
      If e.Effect = DragDropEffects.Copy Then
         node = DirectCast(node.Clone(), LidorSystems.IntegralUI.Lists.TreeNode)
      Else
         ' Remove the node from its current parent node collection
         node.Remove()
      End If

      ' If there is no target node, then add the dragged node at the end
      If targetNode Is Nothing Then
         Me.treeView1.Nodes.Add(node)
      Else
         ' Get the index of the dropped node in target TreeView control
         Dim newIndex As Integer = Me.treeView1.GetDropPos(targetNode, mousePos)
```

```
         If newIndex >= 0 Then
            If targetNode.Parent IsNot Nothing Then
               targetNode.Parent.Nodes.Insert(newIndex, node)
            Else
               Me.treeView1.Nodes.Insert(newIndex, node)
            End If
         Else
            targetNode.Nodes.Add(node)
         End If
      End If

      ' Resume the treeview layout and update the control
      Me.treeView1.ResumeUpdate()
   End If
End Sub

Private Sub treeView1_QueryContinueDrag(ByVal sender As Object, ByVal e As
QueryContinueDragEventArgs)
   Dim mousePos As Point = Me.treeView1.PointToClient(Control.MousePosition)

   ' Cancel the drag operation when the mouse cursor leaves the TreeView space
   If Not Me.treeView1.ClientRectangle.Contains(mousePos) Then
      e.Action = DragAction.Cancel
   End If
End Sub

[C#]
using LidorSystems.IntegralUI.Collections;
using LidorSystems.IntegralUI.Lists;
using LidorSystems.IntegralUI.Lists.Collections;

. . .

private void treeView1_ItemDrag(object sender, ItemDragEventArgs e)
        {
            if (e.Button == MouseButtons.Left)
                this.treeView1.DoDragDrop(e.Item, DragDropEffects.All);
        }

        private void treeView1_DragOver(object sender, DragEventArgs e)
        {
            if
(e.Data.GetDataPresent(typeof(LidorSystems.IntegralUI.Lists.TreeNode)))
            {
                // Depending on the control key pressed change the drag effect
                if ((e.KeyState & 8) == 8 && (e.AllowedEffect &
DragDropEffects.Copy) == DragDropEffects.Copy)
                    e.Effect = DragDropEffects.Copy;
                else
                    e.Effect = DragDropEffects.Move;
            }
            else
                e.Effect = DragDropEffects.None;
        }

        private void treeView1_DragDrop(object sender, DragEventArgs e)
        {
            // Get the current mouse position
            Point mousePos = this.treeView1.PointToClient(new Point(e.X, e.Y));
```

```
            if
(e.Data.GetDataPresent(typeof(LidorSystems.IntegralUI.Lists.TreeNode)))
            {
                // Get the dragged node
                LidorSystems.IntegralUI.Lists.TreeNode node =
(LidorSystems.IntegralUI.Lists.TreeNode)e.Data.GetData(typeof(LidorSystems.Integra
lUI.Lists.TreeNode));

                // Get the target node (the one that is currently hovered)
                LidorSystems.IntegralUI.Lists.TreeNode targetNode =
this.treeView1.GetNodeAt(mousePos);

                // Suspend the treeview layout to increase performance
                this.treeView1.SuspendUpdate();

                // In Copy operation, create a clone node and then add it to the
target
                if (e.Effect == DragDropEffects.Copy)
                    node = (LidorSystems.IntegralUI.Lists.TreeNode)node.Clone();
                // Remove the node from its current parent node collection
                else
                    node.Remove();

                // If there is no target node, then add the dragged node at the
end
                if (targetNode == null)
                    this.treeView1.Nodes.Add(node);
                else
                {
                    // Get the index of the dropped node in target TreeView
control
                    int newIndex = this.treeView1.GetDropPos(targetNode,
mousePos);
                    if (newIndex >= 0)
                    {
                        if (targetNode.Parent != null)
                            targetNode.Parent.Nodes.Insert(newIndex, node);
                        else
                            this.treeView1.Nodes.Insert(newIndex, node);
                    }
                    else
                        targetNode.Nodes.Add(node);
                }

                // Resume the treeview layout and update the control
                this.treeView1.ResumeUpdate();
            }
        }

        private void treeView1_QueryContinueDrag(object sender,
QueryContinueDragEventArgs e)
        {
            Point mousePos = this.treeView1.PointToClient(Control.MousePosition);

            // Cancel the drag operation when the mouse cursor leaves the TreeView
space
            if (!this.treeView1.ClientRectangle.Contains(mousePos))
                e.Action = DragAction.Cancel;
```

```
        }
```

How to perform drag&drop for collection of nodes

Instead of dragging a single node, you can also make dragging of collection of nodes. In the following example is presented how this can be accomplished for selected nodes:

At first we need to set the SelectionMode to MultiSimple or MultiExtended.

```
[VB]
Imports LidorSystems.IntegralUI.Lists

. . .

Private Sub treeView1_ItemDrag(ByVal sender As Object, ByVal e As
ItemDragEventArgs)
   If e.Button = MouseButtons.Left Then
      Dim nodeCollection As New
LidorSystems.IntegralUI.Lists.Collections.TreeNodeCollection()
      For Each node As LidorSystems.IntegralUI.Lists.TreeNode In
Me.treeView1.SelectedNodes
         nodeCollection.Add(node)
      Next

      Me.treeView1.DoDragDrop(nodeCollection, DragDropEffects.All)
   End If
End Sub

Private Sub treeView1_DragOver(ByVal sender As Object, ByVal e As DragEventArgs)
   If
e.Data.GetDataPresent(GetType(LidorSystems.IntegralUI.Lists.Collections.TreeNodeCo
llection)) Then
      ' Depending od the control key pressed change the drag effect
      If (e.KeyState And 8) = 8 AndAlso (e.AllowedEffect And DragDropEffects.Copy)
= DragDropEffects.Copy Then
         e.Effect = DragDropEffects.Copy
      Else
         e.Effect = DragDropEffects.Move
      End If
   Else
      e.Effect = DragDropEffects.None
   End If
End Sub

Private Sub treeView1_DragDrop(ByVal sender As Object, ByVal e As DragEventArgs)
   ' Get the current mouse position
   Dim mousePos As Point = Me.treeView1.ContentPanel.PointToClient(New Point(e.X,
e.Y))

   If
```

```
e.Data.GetDataPresent(GetType(LidorSystems.IntegralUI.Lists.Collections.TreeNodeCollection)) Then
      ' Get the dragged node collection
      Dim nodeCollection As LidorSystems.IntegralUI.Lists.Collections.TreeNodeCollection = DirectCast(e.Data.GetData(GetType(LidorSystems.IntegralUI.Lists.Collections.TreeNodeCollection)), LidorSystems.IntegralUI.Lists.Collections.TreeNodeCollection)

      ' Get the target node (the one that is currently hovered)
      Dim targetNode As LidorSystems.IntegralUI.Lists.TreeNode = Me.treeView1.GetNodeAt(mousePos)

      ' Suspend the treeview layout to increase performance
      Me.treeView1.SuspendUpdate()

      ' Get the collection at which nodes will be added
      If targetNode Is Nothing Then
         ' In Copy operation, create a clone node and then add it to the target
         If e.Effect = DragDropEffects.Copy Then
            For Each node As LidorSystems.IntegralUI.Lists.TreeNode In nodeCollection
               Me.treeView1.Nodes.Add(node.Clone())
            Next
            ' In Move operation, it's best to use the while cycle, because when moving existing node
            ' it will alter the source node collection which can cause errors.
            ' The best approach is the code bellow.
         ElseIf e.Effect = DragDropEffects.Move Then
            For Each node As LidorSystems.IntegralUI.Lists.TreeNode In nodeCollection
               node.Remove()

               ' Add the node to the TreeView
               Me.treeView1.Nodes.Add(node)
            Next
         End If
      Else
         ' Get the index of the dropped node in target TreeView control
         Dim newIndex As Integer = Me.treeView1.GetDropPos(targetNode, mousePos)

         ' In Copy operation, create a clone node and then add it to the target
         If e.Effect = DragDropEffects.Copy Then
            For Each node As LidorSystems.IntegralUI.Lists.TreeNode In nodeCollection
               If newIndex >= 0 Then
                  If targetNode.Parent IsNot Nothing Then
                     targetNode.Parent.Nodes.Insert(newIndex, node.Clone())
                  Else
                     Me.treeView1.Nodes.Insert(newIndex, node.Clone())
                  End If
               Else
                  targetNode.Nodes.Add(node.Clone())
               End If
            Next
            ' In Move operation, it's best to use the while cycle, because when moving existing node
            ' it will alter the source node collection which can cause errors.
            ' The best approach is the node bellow.
         ElseIf e.Effect = DragDropEffects.Move Then
```

```
            For Each node As LidorSystems.IntegralUI.Lists.TreeNode In
nodeCollection
               node.Remove()

               ' Place the node at correct position
               If newIndex >= 0 Then
                  If targetNode.Parent IsNot Nothing Then
                     targetNode.Parent.Nodes.Insert(newIndex, node)
                  Else
                     Me.treeView1.Nodes.Insert(newIndex, node)
                  End If
               Else
                  targetNode.Nodes.Add(node)
               End If
            Next
         End If
      End If

      ' Resume the treeview layout and update the control
      Me.treeView1.ResumeUpdate()
   End If
End Sub

[C#]
using LidorSystems.IntegralUI.Lists;

. . .

private void treeView1_ItemDrag(object sender, ItemDragEventArgs e)
{
   if (e.Button == MouseButtons.Left)
   {
      LidorSystems.IntegralUI.Lists.Collections.TreeNodeCollection nodeCollection
= new LidorSystems.IntegralUI.Lists.Collections.TreeNodeCollection();
      foreach (LidorSystems.IntegralUI.Lists.TreeNode node in
this.treeView1.SelectedNodes)
         nodeCollection.Add(node);

      this.treeView1.DoDragDrop(nodeCollection, DragDropEffects.All);
   }
}

private void treeView1_DragOver(object sender, DragEventArgs e)
{
   if
(e.Data.GetDataPresent(typeof(LidorSystems.IntegralUI.Lists.Collections.TreeNodeCo
llection)))
   {
      // Depending od the control key pressed change the drag effect
      if ((e.KeyState & 8) == 8 && (e.AllowedEffect & DragDropEffects.Copy) ==
DragDropEffects.Copy)
         e.Effect = DragDropEffects.Copy;
      else
         e.Effect = DragDropEffects.Move;
   }
   else
      e.Effect = DragDropEffects.None;
}
```

```
private void treeView1_DragDrop(object sender, DragEventArgs e)
{
   // Get the current mouse position
   Point mousePos = this.treeView1.ContentPanel.PointToClient(new Point(e.X,
e.Y));

   if
(e.Data.GetDataPresent(typeof(LidorSystems.IntegralUI.Lists.Collections.TreeNodeCo
llection)))
   {
      // Get the dragged node collection
      LidorSystems.IntegralUI.Lists.Collections.TreeNodeCollection nodeCollection
=
(LidorSystems.IntegralUI.Lists.Collections.TreeNodeCollection)e.Data.GetData(typeo
f(LidorSystems.IntegralUI.Lists.Collections.TreeNodeCollection));

      // Get the target node (the one that is currently hovered)
      LidorSystems.IntegralUI.Lists.TreeNode targetNode =
this.treeView1.GetNodeAt(mousePos);

      // Suspend the treeview layout to increase performance
      this.treeView1.SuspendUpdate();

      // Get the collection at which nodes will be added
      if (targetNode == null)
      {
         // In Copy operation, create a clone node and then add it to the target
         if (e.Effect == DragDropEffects.Copy)
         {
            foreach (LidorSystems.IntegralUI.Lists.TreeNode node in
nodeCollection)
               this.treeView1.Nodes.Add(node.Clone());
         }
         // In Move operation, it's best to use the while cycle, because when
moving existing node
         // it will alter the source node collection which can cause errors.
         // The best approach is the code bellow.
         else if (e.Effect == DragDropEffects.Move)
         {
            foreach (LidorSystems.IntegralUI.Lists.TreeNode node in
nodeCollection)
            {
               node.Remove();

               // Add the node to the TreeView
               this.treeView1.Nodes.Add(node);
            }
         }
      }
      else
      {
         // Get the index of the dropped node in target TreeView control
         int newIndex = this.treeView1.GetDropPos(targetNode, mousePos);

         // In Copy operation, create a clone node and then add it to the target
         if (e.Effect == DragDropEffects.Copy)
         {
            foreach (LidorSystems.IntegralUI.Lists.TreeNode node in
nodeCollection)
```

```
            {
               if (newIndex >= 0)
               {
                  if (targetNode.Parent != null)
                     targetNode.Parent.Nodes.Insert(newIndex, node.Clone());
                  else
                     this.treeView1.Nodes.Insert(newIndex, node.Clone());
               }
               else
                  targetNode.Nodes.Add(node.Clone());
            }
         }
         // In Move operation, it's best to use the while cycle, because when
moving existing node
         // it will alter the source node collection which can cause errors.
         // The best approach is the node bellow.
         else if (e.Effect == DragDropEffects.Move)
         {
            foreach (LidorSystems.IntegralUI.Lists.TreeNode node in
nodeCollection)
            {
               node.Remove();

               // Place the node at correct position
               if (newIndex >= 0)
               {
                  if (targetNode.Parent != null)
                     targetNode.Parent.Nodes.Insert(newIndex, node);
                  else
                     this.treeView1.Nodes.Insert(newIndex, node);
               }
               else
                  targetNode.Nodes.Add(node);
            }
         }
      }

      // Resume the treeview layout and update the control
      this.treeView1.ResumeUpdate();
   }
}
```

How to perform drag&drop of a node to other controls

Other controls can accept nodes during drag&drop operation, if their AllowDrop property is set to True. In the following example we will demonstrate how node dragged from the TreeView control can be dropped in TextBox control:

```
[VB]
Private Sub textBox1_DragOver(ByVal sender As Object, ByVal e As DragEventArgs)
   If e.Data.GetDataPresent(GetType(LidorSystems.IntegralUI.Lists.TreeNode)) Then
      e.Effect = DragDropEffects.Move
   Else
      e.Effect = DragDropEffects.None
   End If
End Sub

Private Sub textBox1_DragDrop(ByVal sender As Object, ByVal e As DragEventArgs)
```

```
   If e.Data.GetDataPresent(GetType(LidorSystems.IntegralUI.Lists.TreeNode)) Then
      ' Get the dragged node
      Dim node As LidorSystems.IntegralUI.Lists.TreeNode =
DirectCast(e.Data.GetData(GetType(LidorSystems.IntegralUI.Lists.TreeNode)),
LidorSystems.IntegralUI.Lists.TreeNode)

      ' Set the Text property of TextBox control to contain the node text
      Me.textBox1.Text = node.Text
   End If
End Sub

[C#]
private void textBox1_DragOver(object sender, DragEventArgs e)
{
   if (e.Data.GetDataPresent(typeof(LidorSystems.IntegralUI.Lists.TreeNode)))
      e.Effect = DragDropEffects.Move;
   else
      e.Effect = DragDropEffects.None;
}

private void textBox1_DragDrop(object sender, DragEventArgs e)
{
   if (e.Data.GetDataPresent(typeof(LidorSystems.IntegralUI.Lists.TreeNode)))
   {
      // Get the dragged node
      LidorSystems.IntegralUI.Lists.TreeNode node =
(LidorSystems.IntegralUI.Lists.TreeNode)e.Data.GetData(typeof(LidorSystems.Integra
lUI.Lists.TreeNode));

      // Set the Text property of TextBox control to contain the node text
      this.textBox1.Text = node.Text;
   }
}
```

Sorting

When you have large list of nodes, the best to search through them is if they are sorted. The TreeView control supports predefined sorting of nodes based on some predefined types. Also, you can customize the sorting by creating your own implementation of IComparer interface.

Typically nodes are sorted using the **Sorting** property, which can have three values: None, Ascending and Descending. By default, the Sorting property has value None, which means that automatically sorting of nodes is disabled.

Use of predefined sorting method

When the Sorting property is set to other value then None, automatically sorting is active. So whenever a new node is added or removed, the list will be sorted. The nodes will be sorted depending of current value of ComparerObjectType, which can have one of these values:

- Double
- Integer
- String

Additionally, the sorting will only work for those nodes that have their **SortTag** set to some value. Nodes with their SortTag set to null, will be excluded from sorting.

In the following example the nodes will be sorted by their Integer value from the largest to lowest number:

```
[VB]
' At first set the SortTag for each node to some Integer value
For i As Integer = 0 To Me.treeView1.FlatNodes.Count - 1
    Me.treeView1.FlatNodes(i).SortTag = i
Next

' Change the ComparerObjectType to Integer, so the sorting will be used comparing
Integer values
Me.treeView1.ComparerObjectType =
LidorSystems.IntegralUI.Lists.ComparerObjectType.Integer

' Arrange nodes and subnodes from largest to lowest number
Me.treeView1.Sorting = SortOrder.Descending


[C#]
// At first set the SortTag for each node to some Integer value
for (int i = 0; i < this.treeView1.FlatNodes.Count; i++)
   this.treeView1.FlatNodes[i].SortTag = i;

// Change the ComparerObjectType to Integer, so the sorting will be used
comparing Integer values
this.treeView1.ComparerObjectType =
LidorSystems.IntegralUI.Lists.ComparerObjectType.Integer;

// Arrange nodes and subnodes from largest to lowest number
this.treeView1.Sorting = SortOrder.Descending;
```

Create custom sorting

To customize the sort order, you must write a class that implements the **IComparer** interface and set the **ListItemSorter** property to an object of that class. This is useful, for example, when you want to sort nodes by DateTime value added to the node Tag property.

Here is an example of custom sort operation. At first create a list of nodes:

```
[VB]
Imports LidorSystems.IntegralUI.Lists

. . .

Private Sub button1_Click(ByVal sender As Object, ByVal e As EventArgs)
   ' Suspend the treeview layout to increase performance
   Me.treeView1.SuspendUpdate()
   Me.treeView1.Nodes.Clear()

   Dim sampleDate As New DateTime(2008, 1, 1)
   Dim node As LidorSystems.IntegralUI.Lists.TreeNode = Nothing
   For i As Integer = 0 To 4
       ' Create a new node and add the current DateTime value to the node Tag
       node = New LidorSystems.IntegralUI.Lists.TreeNode("Node " & i.ToString())

       ' Create random date and add it to the node Tag
       sampleDate = CreateRandomDate(sampleDate)
```

```
        node.Tag = sampleDate

        ' Create a child nodes for this node
        CreateChildNodes(node, 0, sampleDate)

        ' Add the node as the root node
        Me.treeView1.Nodes.Add(node)
    Next

    ' Resume the treeview layout and update the control
    Me.treeView1.ResumeUpdate()
End Sub

Private Sub CreateChildNodes(ByVal parentNode As
LidorSystems.IntegralUI.Lists.TreeNode, ByVal level As Integer, ByVal sampleDate
As DateTime)
    'In this example only 2 levels in hierarchy are allowed
    If level = 2 Then
        Exit Sub
    Else
        Dim node As LidorSystems.IntegralUI.Lists.TreeNode = Nothing
        For i As Integer = 0 To 2
            ' Create a new child node and add the current DateTime value to the node
Tag
            node = New LidorSystems.IntegralUI.Lists.TreeNode(("Node " &
level.ToString()) + i.ToString())
            sampleDate = CreateRandomDate(sampleDate)
            node.Tag = sampleDate

            ' Create a child nodes for this node
            CreateChildNodes(node, level + 1, sampleDate)

            ' Add the node to the parent node
            parentNode.Nodes.Add(node)
        Next
    End If
End Sub

' Use this method to create random dates
Private Function CreateRandomDate(ByVal value As DateTime) As DateTime
    Dim gen As New Random()
    Dim range As Integer = DirectCast((DateTime.Today - value), TimeSpan).Days
    Return value.AddDays(gen.[Next](range))
End Function


[C#]
using LidorSystems.IntegralUI.Lists;

. . .

private void button1_Click(object sender, EventArgs e)
{
    // Suspend the treeview layout to increase performance
    this.treeView1.SuspendUpdate();
    this.treeView1.Nodes.Clear();

    DateTime sampleDate = new DateTime(2008, 1, 1);
    LidorSystems.IntegralUI.Lists.TreeNode node = null;
```

```
    for (int i = 0; i < 5; i++)
    {
        // Create a new node and add the current DateTime value to the node Tag
        node = new LidorSystems.IntegralUI.Lists.TreeNode("Node " + i.ToString());

        // Create random date and add it to the node Tag
        sampleDate = CreateRandomDate(sampleDate);
        node.Tag = sampleDate;

        // Create a child nodes for this node
        CreateChildNodes(node, 0, sampleDate);

        // Add the node as the root node
        this.treeView1.Nodes.Add(node);
    }

    // Resume the treeview layout and update the control
    this.treeView1.ResumeUpdate();
}

private void CreateChildNodes(LidorSystems.IntegralUI.Lists.TreeNode parentNode,
int level, DateTime sampleDate)
{
    //In this example only 2 levels in hierarchy are allowed
    if (level == 2)
        return;
    else
    {
        LidorSystems.IntegralUI.Lists.TreeNode node = null;
        for (int i = 0; i < 3; i++)
        {
            // Create a new child node and add the current DateTime value to the
node Tag
            node = new LidorSystems.IntegralUI.Lists.TreeNode("Node " +
level.ToString() + i.ToString());
            sampleDate = CreateRandomDate(sampleDate);
            node.Tag = sampleDate;

            // Create a child nodes for this node
            CreateChildNodes(node, level + 1, sampleDate);

            // Add the node to the parent node
            parentNode.Nodes.Add(node);
        }
    }
}

// Use this method to create random dates
private DateTime CreateRandomDate(DateTime value)
{
    Random gen = new Random();
    int range = ((TimeSpan)(DateTime.Today - value)).Days;
    return value.AddDays(gen.Next(range));
}
```

Then create a new class that implements the IComparer interface:

```
[VB]
Public Class CustomItemComparer
    Implements System.Collections.IComparer
    Private sortType As System.Windows.Forms.SortOrder =
System.Windows.Forms.SortOrder.Ascending

    Public Sub New()
    End Sub

    Public Sub New(ByVal order As System.Windows.Forms.SortOrder)
        sortType = order
    End Sub

    Public Function Compare(ByVal x As Object, ByVal y As Object) As Integer
        If sortType = System.Windows.Forms.SortOrder.Descending Then
            Dim temp As Object = x
            x = y
            y = temp
        End If

        Dim firstDate As Object = TryCast(x,
LidorSystems.IntegralUI.Lists.TreeNode).Tag
        Dim secondDate As Object = TryCast(y,
LidorSystems.IntegralUI.Lists.TreeNode).Tag

        If firstDate IsNot Nothing AndAlso secondDate IsNot Nothing Then
            If firstDate.[GetType]() Is GetType(DateTime) AndAlso secondDate.
[GetType]() Is GetType(DateTime) Then
                Return DateTime.Compare(DirectCast(firstDate, DateTime),
DirectCast(secondDate, DateTime))
            End If
        End If

        Return 0
    End Function
End Class


[C#]
public class CustomItemComparer : System.Collections.IComparer
{
   private System.Windows.Forms.SortOrder sortType =
System.Windows.Forms.SortOrder.Ascending;

   public CustomItemComparer()
   {
   }

   public CustomItemComparer(System.Windows.Forms.SortOrder order)
   {
      sortType = order;
   }

   public int Compare(object x, object y)
   {
      if (sortType == System.Windows.Forms.SortOrder.Descending)
      {
         object temp = x;
         x = y;
```

```
            y = temp;
        }

        object firstDate = (x as LidorSystems.IntegralUI.Lists.TreeNode).Tag;
        object secondDate = (y as LidorSystems.IntegralUI.Lists.TreeNode).Tag;

        if (firstDate != null && secondDate != null)
        {
            if (firstDate.GetType() == typeof(DateTime) && secondDate.GetType() ==
typeof(DateTime))
                return DateTime.Compare((DateTime)firstDate, (DateTime)secondDate);
        }

        return 0;
    }
}
```

At the end, add an object from this class to the ListItemSorter property:

```
[VB]
Me.treeView1.Sorting = SortOrder.Descending
Me.treeView1.ListItemSorter = New CustomItemComparer(Me.treeView1.Sorting)

[C#]
this.treeView1.Sorting = SortOrder.Descending;
this.treeView1.ListItemSorter = new CustomItemComparer(this.treeView1.Sorting);
```

Search

How to find a node by using specific criteria

If you want to find a node with specific value, the best is to use the FindNode method. The search criteria can be byKey, byPath or byTagString. Here is an example:

```
[VB]
Dim node As LidorSystems.IntegralUI.Lists.TreeNode = Me.treeView1.FindNode
("ITM", LidorSystems.IntegralUI.Lists.ListSearchCriteria.byKey)

[C#]
LidorSystems.IntegralUI.Lists.TreeNode node = this.treeView1.FindNode("ITM",
LidorSystems.IntegralUI.Lists.ListSearchCriteria.byKey);
```

The method will search through nodes, and compare their Key property value to match the "ITM" value. If some node is found it will return their reference, otherwise a null will be returned.

How to get a node reference from mouse position

Sometimes, while hovering inside the TreeView control client area, you may want to know which node the actual mouse cursor points at. To receive the node reference, you need to use the GetNodeAt method, here is how:

```
[VB]
' Convert the screen coordinates of the mouse cursor position to client
coordinates of the TreeView control
Dim mousePos As Point =
```

```
Me.treeView1.ContentPanel.PointToClient(Control.MousePosition)

' Get the node reference (if there is any), at the specified position
Dim node As LidorSystems.IntegralUI.Lists.TreeNode =
Me.treeView1.GetNodeAt(mousePos)

[C#]
// Convert the screen coordinates of the mouse cursor position to client
coordinates of the TreeView control
Point mousePos =
this.treeView1.ContentPanel.PointToClient(Control.MousePosition);

// Get the node reference (if there is any), at the specified position
LidorSystems.IntegralUI.Lists.TreeNode node = this.treeView1.GetNodeAt(mousePos);
```

Keyword search

There is a built-in search operations by pressing keys. For this behavior, the **KeySearchMode** property is used with following values:

- None – The search is disabled.
- Partial – The search is started whenever there is a key(s) pressed.
- Full – The same as Partial search. In other list controls, this mode has different purpose.

The search process begins whenever there is a single key pressed or a string with multiple characters in it. The search begins always from index 0. For example: whenever the 'A' key is pressed, the search is started. Also if you pressed keys in following order 'ABC' the search is also started. However, you must press the consecutive keys in time less then **500 miliseconds** between **each** pressed key. This is enough time to enter a search string with large length. If the node is found it is selected and positioned to the center of control visible area.

You can also manually search for nodes, by using FindNode methods. Here is how:

1. Set the **KeySearchMode** to **None**
2. Handle the **KeyDown** event or use some other way to start the search (like Button click event)
3. Use the **FindNode** methods

```
[VB]
Private Sub treeView1_KeyDown(ByVal sender As Object, ByVal e As KeyEventArgs)
   ' Search for an node with matching Text starting from index 0
   Dim node As LidorSystems.IntegralUI.Lists.TreeNode =
Me.treeView1.FindNode(e.KeyCode.ToString(), 0)

   If node IsNot Nothing Then
      ' Select the found node
      Me.treeView1.SelectedNode = node

      If Me.treeView1.IsVScrollVisible() Then
         ' If the node is not in visible area, set the scrollbar position to show
         ' the found node in center of ListView control
         If node.Bounds.Y > Me.treeView1.ContentPanel.ClientRectangle.Height / 2
Then
            Me.treeView1.SetScrollPos(New Point(0, node.Bounds.Y -
Me.treeView1.ContentPanel.ClientRectangle.Height / 2))
```

```
         Else
            Me.treeView1.SetScrollPos(New Point(0, node.Bounds.Y -
node.Bounds.Height))
         End If
      End If
   End If
End Sub

Protected Overridable Function GetAlphaNumValue(ByVal keyData As Keys) As String
   Select Case keyData
      Case Keys.A, Keys.B, Keys.C, Keys.D, Keys.E, Keys.F, _
            Keys.G, Keys.H, Keys.I, Keys.J, Keys.K, Keys.L, _
            Keys.M, Keys.N, Keys.O, Keys.P, Keys.Q, Keys.R, _
            Keys.S, Keys.T, Keys.U, Keys.V, Keys.W, Keys.X, _
            Keys.Y, Keys.Z
         Return keyData.ToString()

      Case Keys.D0, Keys.NumPad0
         Return "0"

      Case Keys.D1, Keys.NumPad1
         Return "1"

      Case Keys.D2, Keys.NumPad2
         Return "2"

      Case Keys.D3, Keys.NumPad3
         Return "3"

      Case Keys.D4, Keys.NumPad4
         Return "4"

      Case Keys.D5, Keys.NumPad5
         Return "5"

      Case Keys.D6, Keys.NumPad6
         Return "6"

      Case Keys.D7, Keys.NumPad7
         Return "7"

      Case Keys.D8, Keys.NumPad8
         Return "8"

      Case Keys.D9, Keys.NumPad9
         Return "9"

      Case Keys.Space
         Return " "
   End Select

   Return String.Empty
End Function

[C#]
private void treeView1_KeyDown(object sender, KeyEventArgs e)
{
   // Search for an node with matching Text starting from index 0
   LidorSystems.IntegralUI.Lists.TreeNode node =
```

```
this.treeView1.FindNode(GetAlphaNumValue(e.KeyCode), 0);

   if (node != null)
   {
      // Select the found node
      this.treeView1.SelectedNode = node;

      if (this.treeView1.IsVScrollVisible())
      {
         // If the node is not in visible area, set the scrollbar position to
show
         // the found node in center of ListView control
         if (node.Bounds.Y > this.treeView1.ContentPanel.ClientRectangle.Height /
2)
            this.treeView1.SetScrollPos(new Point(0, node.Bounds.Y -
this.treeView1.ContentPanel.ClientRectangle.Height / 2));
         else
            this.treeView1.SetScrollPos(new Point(0, node.Bounds.Y -
node.Bounds.Height));
      }
   }
}

protected virtual string GetAlphaNumValue(Keys keyData)
{
   switch (keyData)
   {
      case Keys.A:
      case Keys.B:
      case Keys.C:
      case Keys.D:
      case Keys.E:
      case Keys.F:
      case Keys.G:
      case Keys.H:
      case Keys.I:
      case Keys.J:
      case Keys.K:
      case Keys.L:
      case Keys.M:
      case Keys.N:
      case Keys.O:
      case Keys.P:
      case Keys.Q:
      case Keys.R:
      case Keys.S:
      case Keys.T:
      case Keys.U:
      case Keys.V:
      case Keys.W:
      case Keys.X:
      case Keys.Y:
      case Keys.Z:
         return keyData.ToString();

      case Keys.D0:
      case Keys.NumPad0:
         return "0";
```

```
      case Keys.D1:
      case Keys.NumPad1:
         return "1";

      case Keys.D2:
      case Keys.NumPad2:
         return "2";

      case Keys.D3:
      case Keys.NumPad3:
         return "3";

      case Keys.D4:
      case Keys.NumPad4:
         return "4";

      case Keys.D5:
      case Keys.NumPad5:
         return "5";

      case Keys.D6:
      case Keys.NumPad6:
         return "6";

      case Keys.D7:
      case Keys.NumPad7:
         return "7";

      case Keys.D8:
      case Keys.NumPad8:
         return "8";

      case Keys.D9:
      case Keys.NumPad9:
         return "9";

      case Keys.Space:
         return " ";
   }

   return String.Empty;
}
```

How to maintain scroll position

In order to manually change the position of scrollbar, there are two methods for this purpose:

- GetScrollPos – returns the current position of vertical and horizontal scrollbar
- SetScrollPos – sets the position of vertical and horizontal scrollbar to a specified value

Every node has Bounds property which holds the area in which the node content is drawn. This property has their position in absolute coordinates related to the origin of the control client area. So, if you want to have some specific node placed at the top of visible area, you can do that by using this code:

```
[VB]
Me.treeView1.SetScrollPos(New Point(0, node.Bounds.Y))

[C#]
this.treeView1.SetScrollPos(new Point(0, node.Bounds.Y));
```

The Top position of the node is directly related to the position of vertical scrollbar.

XML Encoding

By default the TreeView shows nodes ordered in simplified way, a hierarchical structure where each node contains image and a label. This is not much useful when you want to present your data in more custom layout.

By using XML tags you can create various different templates which will contain custom objects placed in custom location in node space. These custom objects represents: text, images, custom controls, hyperlinks, animated gifs etc. There are a set of XML tags with which you can design the layouts for every node. These templates can be applied to all nodes (to keep uniform look for every node), or separate template can be applied for every specific node.

The result is a very rich and customizable presentation of your data.

XML Tags that are supported

The use of XML tags is very like the using of standard HTML tags. In the following section the list of supported XML tags is presented in alphabetical order:

Tag	Atribute	Description
<a>		Defines an anchor with which you can create a link to another document
	href	The target URL of the link
	id	The identifier of the hyperlink
	style	An inline style definition
	underlined	Determines whether the link text is underlined. Default value is true.
<b>		The text is rendered as bold
	id	The identifier of the text enclosed with this tag
	style	An inline style definition
 		inserts a single line break
<control>		Defines a custom control
	height	Sets the height of a control
	index	Specifies the position of the control in a collection
	style	An inline style definition
	width	Sets the width of a control
		Defines the main section of the content
	style	An inline style definition
<font>		specifies the font of text
	color	Defines the color of the text
	face	Defines the font name of the text
	id	The identifier of the text enclosed with this tag
	size	Defines the size of the text
	style	An inline style definition
<i>		The text is rendered as italic

	id	The identifier of the text enclosed with this tag
	style	An inline style definition
<img>		Defines an image
	assemblypath	Specifies the path of an assembly used as a resource file
	height	Sets the height of an image
	id	The identifier of the image
	index	Specifies the position of the image in a collection
	resource	The resource object to be retrieved from assembly
	src	The URL of the image to display
	style	An inline style definition
	width	Sets the width of an image
<p>		Defines a new section (paragraph) in the content
	indent	Specifies a space (in pixels) by which paragraph's content is moved to the right
	style	An inline style definition
<table>		defines a table
	cellpadding	Specifies the space between the table cell border and content
	cellspacing	Specifies the space between table cells
	style	An inline style definition
	width	Specifies the width of the table. Can be specified in % or pixels.
<td>		Defines a cell in a table
	align	Specifies the horizontal alignment of cell content
	colspan	Indicates the number of columns this cell should span
	height	Specifies the height of the table cell. Can be specified in % or pixels.
	rowspan	Indicates the number of rows this cell should span
	style	An inline style definition
	valign	Specifies the vertical alignment of cell content
	width	Specifies the width of the table cell. Can be specified in % or pixels.
<tr>		Defines a new section (paragraph) in the content
	align	Specifies the horizontal alignment of cell content
	height	Specifies the height of the table row. Can be specified in % or pixels.
	style	An inline style definition
	valign	Specifies the vertical alignment of cell content
<r>		Defines regular text
	id	The identifier of the text enclosed with this tag
	style	An inline style definition
<s>		Defines strikethrough text
	id	The identifier of the text enclosed with this tag

	style	An inline style definition
<style>		Defines a style used for rendering and formatting of the content
	align	Specifies the content alignment
	backcolor	Specifies the color of the object background
	backfadecolor	Specifies the fading color of the object background
	bordercolor	Specifies the color of the object border
	fillstyle	Specifies the type of gradient fill of the object background
	font	Specifies the color of the text
	hovercolor	Specifies the color of the object background while mouse cursor hovers over it
	hoverfadecolor	Specifies the fading color of the object background while mouse cursor hovers over it
	hovertextcolor	Specifies the color of the text while mouse cursor hovers over it
	id	The identifier of the style
	margin	Defines the space around an object's border
	padding	Defines the space between the border and the content of an object
	rendering	Specifies the rendering mode for text
	selectedtextcolor	Specifies the color of the text when it is selected
	textcolor	Specifies the color of the text
	transparency	Specifies the percentage of transparency
<u>		defines underlined text
	id	The identifier of the text enclosed with this tag
	style	An inline style definition

In the following section we will present you how to use these tags in various combinations. As a result there would be created custom layouts for node's content.

Working with containers: and <p> tags

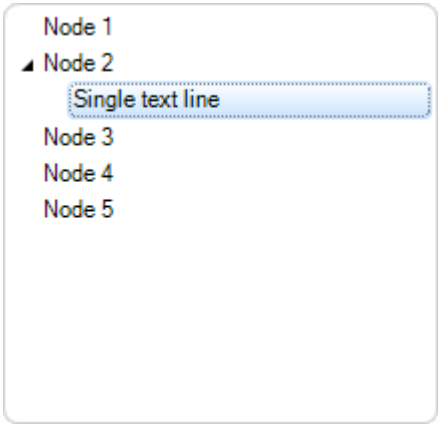
The simplest form of formatted content is constructed by using the tag and sample text. Here is an example:

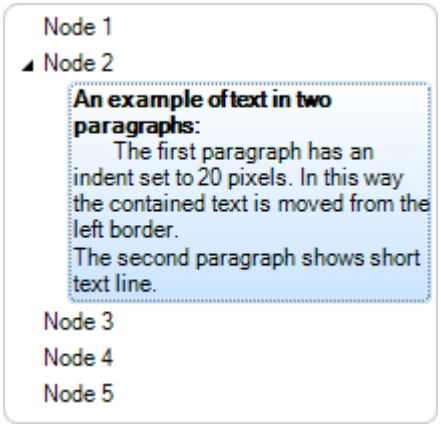
```
[VB]
node.Content = "<div>Single text line</div>"

[C#]
node.Content = "<div>Single text line</div>";
```

In some cases you want to display text in multiple lines or in different paragraphs. For this situation the <p> tag is used. Here is an example:

```
[VB]
node.Content = "<div><b>An example of text in two paragraphs:</b><p
indent=""20"">The first paragraph has an indent set to 20 pixels. In this way the
contained text is moved from the left border.</p><p>The second paragraph shows
short text line.</p></div>"

[C#]
node.Content = "<div><b>An example of text in two paragraphs:</b><p
indent=\"20\">The first paragraph has an indent set to 20 pixels. In this way the
contained text is moved from the left border.</p><p>The second paragraph shows
short text line.</p></div>";
```

The result is shown in following pictures:

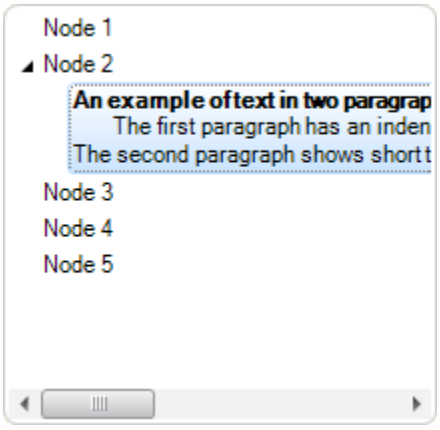




From the pictures you can also notice that the text is wrapped in multiple lines when words are reaching the end of the node content space. This happens because the WordWrap property by default is set to True for all nodes in the TreeView control.

If for example we set the WordWrap property for this node to False, the resulting look is different.

```
[VB]
node.WordWrap = False
node.Content = "<div><b>An example of text in two paragraphs:</b><p
indent=""20"">The first paragraph has an indent set to 20 pixels. In this way the
contained text is moved from the left border.</p><p>The second paragraph shows
short text line.</p></div>"

[C#]
node.WordWrap = false;
node.Content = "<div><b>An example of text in two paragraphs:</b><p
indent=\"20\">The first paragraph has an indent set to 20 pixels. In this way the
contained text is moved from the left border.</p><p>The second paragraph shows
short text line.</p></div>";
```



More information about word wrapping can be found in "Using word wrap" section of this document.

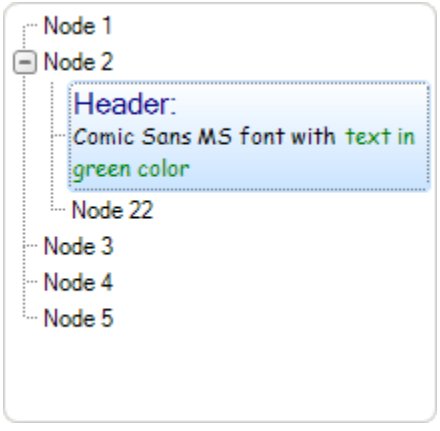
Working with <font> tag

Normally some parts of the text shown in a single content can have different font or text size. In order to do that the <font> tag is used.

In following example we will present you how to create content with header, text in multiple lines with some parts set to different font and color.

```
[VB]
node.Content = "<div><font size=""11"" color=""#000080"" >Header:</font><p><font
face=""Comic Sans MS"">Comic Sans MS font with<font color=""Green""> text in
green color</font></font></p></div>"

[C#]
node.Content = "<div><font size=\"11\" color=\"#000080\" >Header:</font><p><font
face=\"Comic Sans MS\">Comic Sans MS font with<font color=\"Green\"> text in
green color</font></font></p></div>";
```

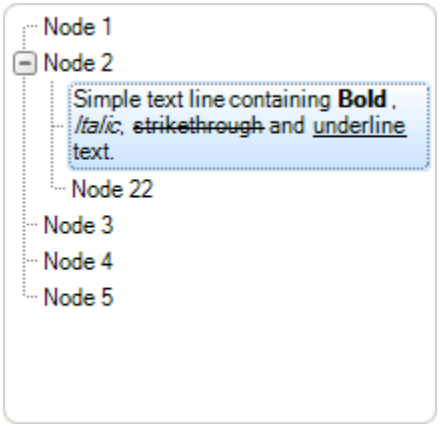
The result is a text in two lines. First line is shown in **Microsoft Sans Serif 11pt** in blue color, and the second line is shown in **Comic Sans MS** font containing parts in green color:



Working with font style tags

In font style tags group are <b>, <i>, <s> and <u>. With these tags you can change the style of font for some part or whole text. Here is an example:

```
[VB]
node.Content = "<div>Simple text line containing <b>Bold</b>, <i>Italic</i>,
<s>strikethrough </s>and <u>underline</u> text.</div>"

[C#]
node.Content = "<div>Simple text line containing <b>Bold</b>, <i>Italic</i>,
<s>strikethrough </s>and <u>underline</u> text.</div>";
```



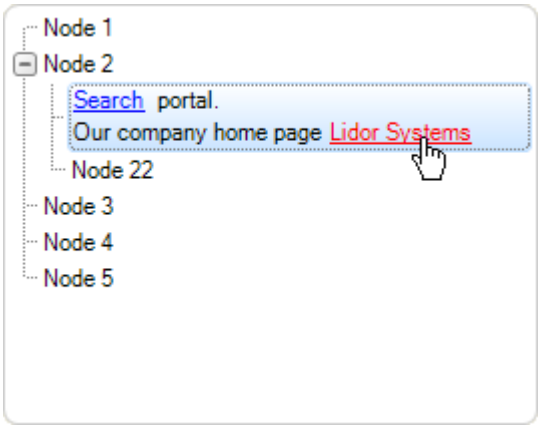
Working with <a> tag

One of objects that you can create in content is a hyperlink object. The link you are entering can be some file or page located on the web or on local hard disk.

If the link points to some application file like Notepad.exe, you can open it by simple clicking on the hyperlink. However in order this to work, you will also need to handle ItemObjectClicked event (see the "How to handle events" section in the beginning of this document).

Here is an example of content with two hyperlinks:

```
[VB]
node.Content = "<div><a href=""www.google.com"" style=""textcolor:Blue"">Search</
a> portal. <br></br>Our company home page <a href=""www.lidorsystems.com""
style=""textcolor:Red"">Lidor Systems</a></div>";

[C#]
node.Content = "<div><a href=\"www.google.com\" style=\"textcolor:Blue\">Search</
a> portal. <br></br>Our company home page <a href=\"www.lidorsystems.com\"
style=\"textcolor:Red\">Lidor Systems</a></div>";
```



As you can see from the picture, the hyperlinks have also different colors. Also, the hand cursor is automatically shown when the mouse hovers over the hyperlink. In future versions we may extend this to be able to add custom cursors.

In this example we have used the style attribute in which you can set different sub attributes for the object to which this style is applied. See below for more information how to use styles.

Working with <img> tag

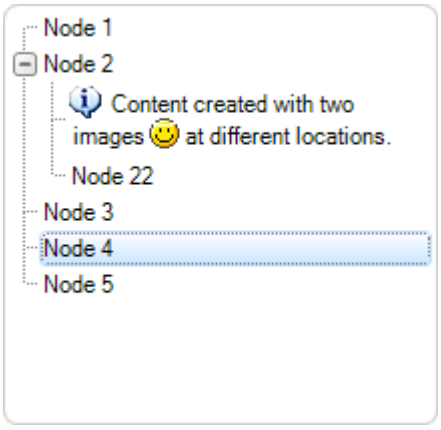
With this tag you can add images located on some local location or there is an option to create ImageList and extract the image from the list by using specified index.

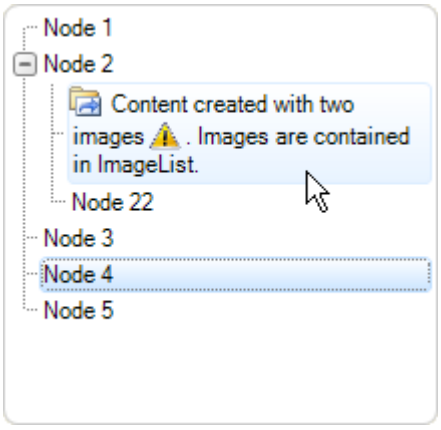
An example how to create content with two images located on local hard disk:

```
[VB]
node.Content = "<div><img src=""C:\\images\\info.ico""></img> Content created
with two images  <img src=""C:\\images\\smiley.ico""></img>at different
locations.</div>"

[C#]
node.Content = "<div><img src=\"C:\\images\\info.ico\"></img> Content created
with two images  <img src=\"C:\\images\\smiley.ico\"></img>at different
locations.</div>";
```

An example how to create content with two images located in ImageList referenced by TreeView control:

```
[VB]
node.Content = "<div><img index=""0""></img> Content created with two images
<img index=""1""></img>. Images are contained in ImageList.</div>";

[C#]
node.Content = "<div><img index=\"0\"></img> Content created with two images
<img index=\"1\"></img>. Images are contained in ImageList.</div>";
```





By default the images have their original size. You can change their size by setting the width and height attributes of the <img> tag. Also you can apply margin and padding attributes by setting the style attribute values (see below how to use styles).

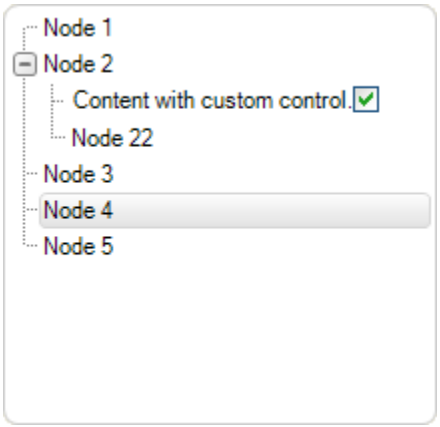
Working with <control> tag

With text, images and hyperlinks you can create very rich content. However, placing custom controls will give more interaction to the user.

The <control> tag is specifically added to meet this task. Every custom control, standard or third party, can be placed in single content of the node. You can add even complex controls like Grid.

Here is an example on how to add a button and a checkbox to node content:

```
[VB]
' At first the control must be created and added to
' the node Controls collection
Dim cBox As New CheckBox()
cBox.Size = New Size(14, 14)
Me.treeView1.SelectedNode.Controls.Add(cBox)

' In <control> tag use the index to reference the control
Me.treeView1.SelectedNode.Content = "<div>Content with custom control. <control
index=""0""></control></div>"

[C#]
// At first the control must be created and added to
// the node Controls collection
CheckBox cBox = new CheckBox();
cBox.Size = new Size(14, 14);
this.treeView1.SelectedNode.Controls.Add(cBox);

// In <control> tag use the index to reference the control
this.treeView1.SelectedNode.Content = "<div>Content with custom control. <control
index=\"0\"></control></div>";
```



The use of controls in simple format is not very useful. The controls like text or any other object, can also we wrapped and shown in next line, but that is all. Mainly in simple form there are no attributes by which you can place the control at specific place or aligned it to the right side of the content space, for example.

This can be done by using table formatting which is explained in next section.

Working with <table>, <tr> and <td> tags

The best way to create custom layouts with different objects placed at different locations in the content is by using table formatting.

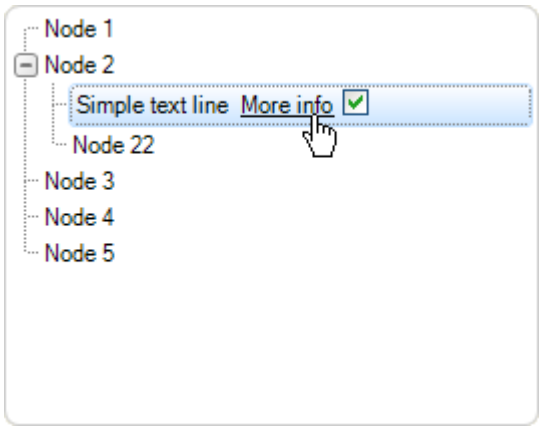
One of the reasons the table is used is presentation of data in ordered way for all nodes. In the following example we will show you how to place a text, hyperlink and custom control in a content using <table> tag:

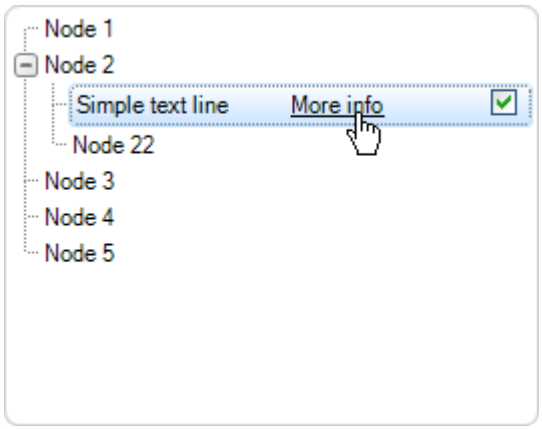
```
[VB]
Dim cBox As New CheckBox()
cBox.Size = New Size(14, 14)
Me.treeView1.SelectedNode.Controls.Add(cBox)

node.Content = "<div><table cellspacing=""2"" cellpadding=""1""><tr><td>Simple
text line</td><td><a href=""www.lidorsystems.com"">More info</a></td><td><control
index=""0""></control></td></tr></table></div>"

[C#]
CheckBox cBox = new CheckBox();
cBox.Size = new Size(14, 14);
this.treeView1.SelectedNode.Controls.Add(cBox);

node.Content = "<div><table cellspacing=\"2\" cellpadding=\"1\"><tr><td>Simple
text line</td><td><a href=\"www.lidorsystems.com\">More info</a></td><td><control
index=\"0\"></control></td></tr></table></div>";
```

In this example all three objects are placed next to each other, which doesn't give the best look. Also, it acts like there is not table present. We can change that by specifying the width for each table cell. If the width is not specified the table cell will have width as the minimum width of their content.

```
[VB]
```

```
Dim cBox As New CheckBox()
cBox.Size = New Size(14, 14)
Me.treeView1.SelectedNode.Controls.Add(cBox)

node.Content = "<div><table cellspacing=""2"" cellpadding=""1""
width=""100%""><tr><td width=""50%"">Simple text line</td><td width=""50%""><a
href=""www.lidorsystems.com"">More info</a></td><td><control
index=""0""></control></td></tr></table></div>"

[C#]
CheckBox cBox = new CheckBox();
cBox.Size = new Size(14, 14);
this.treeView1.SelectedNode.Controls.Add(cBox);

node.Content = "<div><table cellspacing=\"2\" cellpadding=\"1\"
width=\"100%\"><tr><td width=\"50%\">Simple text line</td><td width=\"50%\"><a
href=\"www.lidorsystems.com\">More info</a></td><td><control index=\"0\">
</control></td></tr></table></div>";
```





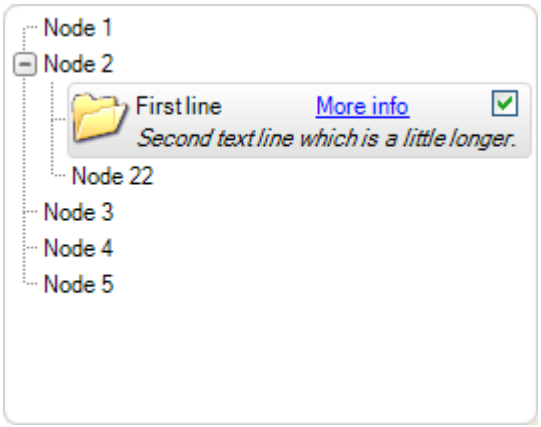
The change is obvious. By specifying the width of table cell in percents we have achieved the cell to shrink or expand depending of the current width of the node content. For the last cell the width is not specified. This is done to make sure that the CheckBox will remain always docked at the right side of the content.

Here is a more complex layout, which includes column and row span:

```
[VB]
' Use a temprary variable for storing large string
Dim content As String = "<div><table width=""100%"">"
' Add the first row with four cells
content += "<tr><td rowspan=""2""><img index=""1""></img></td><td
width=""50%"">First line</td><td width=""50%""><a href=""www.lidorsystems.com""
style=""textcolor:Blue"">More info</a></td><td><control
index=""0""></control></td></tr>"
' Add the second row and span the cell over three other cells
content += "<tr><td colspan=""3""><i>Second text line which is a little longer.</
i></td></tr>"
' Close the table and div tags
content += "</table></div>"

' Add the content string to the node Content property
node.Content = content
```

```
[C#]
// Use a temprary variable for storing large string
String content = "<div><table width=\"100%\">";
// Add the first row with four cells
content += "<tr><td rowspan=\"2\"><img index=\"1\"></img></td><td
width=\"50%\">First line</td><td width=\"50%\"><a href=\"www.lidorsystems.com\"
style=\"textcolor:Blue\">More info</a></td><td><control
index=\"0\"></control></td></tr>";
// Add the second row and span the cell over three other cells
content += "<tr><td colspan=\"3\"><i>Second text line which is a little longer.</
i></td></tr>";
// Close the table and div tags
content += "</table></div>";

// Add the content string to the node Content property
node.Content = content;
```

The first cell is spanned across two rows. The second cell of the second row is spanned across three columns. The result is shown in following picture:



Working with <style> tag

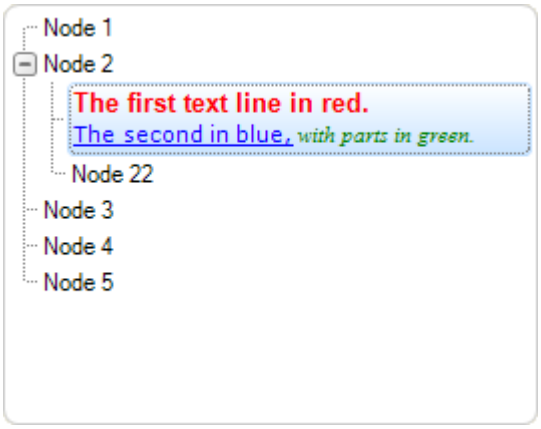
The styles are very powerful tool if you use them correctly. You can create several styles in beginning and later used them for specific parts of the code only by calling their Id. Also you can use inline coding to apply specific style to some part of the content.

Here is how to create styles at beginning of the code:

Here is how to create styles at beginning of the code:

```
[VB]
Dim content As String = "<div><style id=""1"" textcolor=""Red""
font=""name:Arial;size:10;style:b""></style>"
content += "<style id=""2"" textcolor=""Blue""
font=""name:Verdana;size:8;style:u""></style>"
content += "<style id=""3"" textcolor=""Green"" font=""name:Times New
Roman;size:8;style:i""></style>"
content += "<p style=""id:1"">The first text line in red.</p>"
content += "<p style=""id:2"">The second in blue, <font style=""id:3"">with parts
in green.</font></p>"
content += "</div>"

node.Content = content
```

```
[C#]
String content = "<div><style id=\"1\" textcolor=\"Red\"
font=\"name:Arial;size:10;style:b\"></style>";
content += "<style id=\"2\" textcolor=\"Blue\"
font=\"name:Verdana;size:8;style:u\"></style>";
content += "<style id=\"3\" textcolor=\"Green\" font=\"name:Times New
Roman;size:8;style:i\"></style>";
content += "<p style=\"id:1\">The first text line in red.</p>";
content += "<p style=\"id:2\">The second in blue, <font style=\"id:3\">with parts
in green.</font></p>";
content += "</div>";

node.Content = content;
```



Here is how to use inline style definition:

```
[VB]
Dim content As String = "<div>";
content += "<p style=""textcolor:Red;font:Arial,10,b"">The first text line in
red.</p>";
content += "<p style=""textcolor:Blue;font:Verdana,8,u"">The second in blue,
<font style=""textcolor:Green;font:Times New Roman,8,i"">with parts in
green.</font></p>";
content += "</div>";

node.Content = content

[C#]
String content = "<div>";
content += "<p style=\"textcolor:Red;font:Arial,10,b\">The first text line in
red.</p>";
content += "<p style=\"textcolor:Blue;font:Verdana,8,u\">The second in blue,
<font style=\"textcolor:Green;font:Times New Roman,8,i\">with parts in
green.</font></p>";
content += "</div>";

node.Content = content;
```

The result is the same as in previous picture, only the use of styles has changed. Instead of using predefined styles, we have used inline style definitions to change different parts of the text.

Styles can be applied for every object, not just text. For example the Margin and Padding attributes of the style, has more value when paragraphs or tables are used. The same applies for content alignment which has more value when it is used with tables.

Using word wrap

By default the WordWrap for the TreeView control and all nodes is set to True. This means that whenever some object reach the end of the line, if there is not enough space for the object to be placed at the line end, it will be placed in beginning of a new line. In this way by changing the width of the TreeView control the content of their nodes will expand or collapse , but always will show their full content in visible area of the control.

If you don't want to use word wrapping, then the node content will remain in exact place as node layout is constructed. Meaning, some objects may be placed outside the visible area of the TreeView control. This will trigger appearance of horizontal or vertical scrollbar, which allows you to scroll the visible area in order to show these objects.

As stated above, all nodes can have their independent setting of word wrap. This is very useful to have, when designing different layouts for different nodes. Some of them can have word wrapping while others don't. You only need to change the WordWrap property for specific nodes.

Serialization

You can use serialization to memorize the current state of TreeView. Also, this feature is very useful in creation of color schemes.

How to serialize the control content in Streams

In order to serialize the control content in streams, there are two methods:

- LoadFromStream – load the control content from Stream
- SaveToStream – save the control content in Stream

Here is how you can use these methods:

```
[VB]
Imports LidorSystems.IntegralUI.Serialization

. . .

' Create a MemoryStream object
Private memStream As New MemoryStream()

' Click on the btnLoad button to load the content of MemoryStream object
' to TreeView control
Private Sub btnLoad_Click(ByVal sender As Object, ByVal e As System.EventArgs)
   Dim serializer As New TreeViewSerializer()
   memStream.Seek(0, SeekOrigin.Begin)
   serializer.LoadFromStream(Me.treeView1, memStream)
End Sub

' Click on the btnSave button to save the control content to
' MemoryStream object
Private Sub btnSave_Click(ByVal sender As Object, ByVal e As System.EventArgs)
   Dim serializer As New TreeViewSerializer()
   serializer.SaveToStream(Me.treeView1, memStream, False)
End Sub

[C#]
using LidorSystems.IntegralUI.Serialization;

. . .

// Create a MemoryStream object
MemoryStream memStream = new MemoryStream();

// Click on the btnLoad button to load the content of MemoryStream object
// to TreeView control
private void btnLoad_Click(object sender, System.EventArgs e)
{
   TreeViewSerializer serializer = new TreeViewSerializer();
   memStream.Seek(0, SeekOrigin.Begin);
   serializer.LoadFromStream(this.treeView1, memStream);
}

// Click on the btnSave button to save the control  content to
// MemoryStream object
private void btnSave_Click(object sender, System.EventArgs e)
```

```
{
   TreeViewSerializer serializer = new TreeViewSerializer();
   serializer.SaveToStream(this.treeView1, memStream, false);
}
```

How to serialize the control content in files

In order to serialize the control content in files, there are two ways:

- By using file streams
- By using file names

Serialization using file streams

```
[VB]
Imports LidorSystems.IntegralUI.Serialization

. . .

' Click on the btnLoad button to load the content of FileStream object
' to TreeView control
Private Sub btnLoad_Click(ByVal sender As Object, ByVal e As System.EventArgs)
   Dim serializer As New TreeViewSerializer()
   Dim fs As FileStream = File.OpenRead("C:\fs.xml")
   serializer.LoadFromStream(Me.treeView1, fs)
End Sub

' Click on the btnSave button to save the control content to
' FileStream object
Private Sub btnSave_Click(ByVal sender As Object, ByVal e As System.EventArgs)
   Dim serializer As New TreeViewSerializer()
   Dim fs As FileStream = File.Create("C:\fs.xml")
   serializer.SaveToStream(Me.treeView1, fs, True)
End Sub

[C#]
using LidorSystems.IntegralUI.Serialization;

. . .

// Click on the btnLoad button to load the content of FileStream object
// to TreeView control
private void btnLoad_Click(object sender, System.EventArgs e)
{
   TreeViewSerializer serializer = new TreeViewSerializer();
   FileStream fs = File.OpenRead("C:\\fs.xml");
   serializer.LoadFromStream(this.treeView1, fs);
}

// Click on the btnSave button to save the control  content to
// FileStream object
private void btnSave_Click(object sender, System.EventArgs e)
{
   TreeViewSerializer serializer = new TreeViewSerializer();
   FileStream fs = File.Create("C:\\fs.xml");
   serializer.SaveToStream(this.treeView1, fs, true);
}
```

Serialization using file names

In order to serialize the control content in files, there are two methods:

- LoadFromFile – load the control content from file
- SaveToFile – save the control content in file

```
[VB]
' Click on the btnLoad button to open the Load dialog, by which you can
' choose the destination of the file
Private Sub btnLoad_Click(ByVal sender As Object, ByVal e As System.EventArgs)
   Dim openDlg As New OpenFileDialog()
   openDlg.Filter = "XML files (*.xml)|*.xml"
   openDlg.InitialDirectory = Application.ExecutablePath
   If openDlg.ShowDialog() = DialogResult.OK Then
      Me.treeView1.SuspendUpdate()

      Dim serializer As New
LidorSystems.IntregralUI.Serialization.TreeViewSerializer()
      serializer.LoadFromXml(Me.treeView1, openDlg.FileName)

      Me.treeView1.ResumeUpdate()
   End If
End Sub

' Click on the btnSave button to open the Save dialog, by which you can
' choose the destination of the file
Private Sub btnSave_Click(ByVal sender As Object, ByVal e As System.EventArgs)
   Dim saveDlg As New SaveFileDialog()
   saveDlg.Filter = "XML files (*.xml)|*.xml"
   saveDlg.AddExtension = True
   saveDlg.DefaultExt = ".xml"
   saveDlg.InitialDirectory = Application.ExecutablePath

   If saveDlg.ShowDialog() = DialogResult.OK Then
      Dim serializer As New
LidorSystems.IntregralUI.Serialization.TreeViewSerializer()
      serializer.SaveToXml(Me.treeView1, saveDlg.FileName)
   End If
End Sub

[C#]
using LidorSystems.IntegralUI.Serialization;

. . .

// Click on the btnLoad button to open the Load dialog, by which you can
// choose the destination of the file
private void btnLoad_Click(object sender, System.EventArgs e)
{
   OpenFileDialog openDlg = new OpenFileDialog();
   openDlg.Filter = "XML files (*.xml)|*.xml";
   openDlg.InitialDirectory = Application.ExecutablePath;
   if (openDlg.ShowDialog() == DialogResult.OK)
   {
      this.treeView1.SuspendUpdate();

      LidorSystems.IntregralUI.Serialization.TreeViewSerializer serializer = new
LidorSystems.IntregralUI.Serialization.TreeViewSerializer();
```

```
      serializer.LoadFromXml(this.treeView1, openDlg.FileName);

      this.treeView1.ResumeUpdate();
   }
}

// Click on the btnSave button to open the Save dialog, by which you can
// choose the destination of the file
private void btnSave_Click(object sender, System.EventArgs e)
{
   SaveFileDialog saveDlg = new SaveFileDialog();
   saveDlg.Filter = "XML files (*.xml)|*.xml";
   saveDlg.AddExtension = true;
   saveDlg.DefaultExt = ".xml";
   saveDlg.InitialDirectory = Application.ExecutablePath;

   if (saveDlg.ShowDialog() == DialogResult.OK)
   {
      LidorSystems.IntregralUI.Serialization.TreeViewSerializer serializer =
LidorSystems.IntregralUI.Serialization.new TreeViewSerializer();
      serializer.SaveToXml(this.treeView1, saveDlg.FileName);
   }
}
```

How to serialize the control content in SQL database

When you want to save the control content in some SQL database, at first you need a XML field in your table. The process of adding the control content to this field is the same as for streams, explained previously.

Loading content from a database and filling the TreeView control with it is also the same as using streams. But here, at first you need to create a MemoryObject:

```
[VB]
' Create a MemoryStream object
Dim memStream As New MemoryStream()

' sampleString is temporary varaiable that holds the database XMl field content
' In this demonstration we are using some XML encoded text.
Dim sampleString As String = "<?xml version=""1.0"" encoding=""us-ascii""?
><TreeView Version=""2.0""><Nodes><Node Text=""Node
1""></Node></Nodes></TreeView>"

' Write the sampleString to the MemoryStream object
Dim sw As New StreamWriter(memStream, System.Text.Encoding.ASCII)
sw.Write(sampleString)
sw.Flush()

[C#]
// Create a MemoryStream object
MemoryStream memStream = new MemoryStream();

// sampleString is temporary varaiable that holds the database XMl field content
// In this demonstration we are using some XML encoded text.
string sampleString = "<?xml version=\"1.0\" encoding=\"us-ascii\"?><TreeView
Version=\"2.0\"><Nodes><Node Text=\"Node 1\"></Node></Nodes></TreeView>";
```

```
// Write the sampleString to the MemoryStream object
StreamWriter sw = new StreamWriter(memStream, System.Text.Encoding.ASCII);
sw.Write(sampleString);
sw.Flush();
```

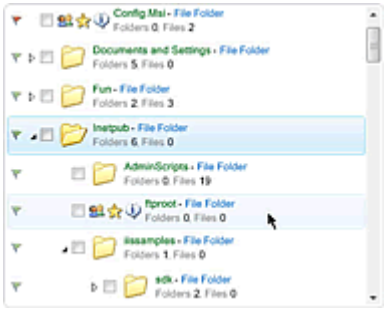
After that, you need to create a **TreeViewSerializer** object**:**

```
[VB]
Dim serializer As New TreeViewSerializer()
memStream.Seek(0, SeekOrigin.Begin)
serializer.LoadFromStream(Me.treeView1, memStream)

[C#]
TreeViewSerializer serializer = new TreeViewSerializer();
memStream.Seek(0, SeekOrigin.Begin);
serializer.LoadFromStream(this.treeView1, memStream);
```

As a result the control will be populated from the string placed in the MemoryStream object.

Sample projects

More information on IntegralUI TreeView control can be found by examining the source code of sample projects available on our web site:



Explorer

Demonstrates how IntegralUI TreeView control can be used to create a Windows Explorer like application. It includes various appearances, three visual styles (Classic, XP, Vista), Advanced Drag&Drop operations and Multiple selections. Among the advanced features is the XML encoding of nodes with which you can add hyperlinks, text in multiple paragraphs with different colors and fonts.

Custom Controls

Demonstrates how custom controls can be included in IntegralUI TreeView.