



**TMS TAdvTreeView
DEVELOPERS GUIDE**








Index

| | |
|--|----|
| Introduction | 3 |
| Organization..... | 4 |
| Modes | 5 |
| Virtual | 5 |
| Collection-based | 8 |
| Columns..... | 10 |
| Configuration / Appearance..... | 10 |
| Autosizing and stretching..... | 12 |
| Groups | 16 |
| Configuration..... | 16 |
| Appearance | 17 |
| Nodes..... | 19 |
| Configuration / Appearance..... | 19 |
| Adding, inserting and removing nodes..... | 19 |
| Fixed vs variable node height..... | 22 |
| Checkbox / Radiobutton support | 27 |
| Extended nodes | 28 |
| Interaction..... | 31 |
| Clipboard | 32 |
| Reordering / Drag & Drop | 33 |
| Filtering..... | 34 |
| Sorting | 37 |
| Editing | 37 |
| Custom Editor | 39 |
| Customization..... | 41 |
| Demos | 43 |
| Overview | 43 |
| Directory | 43 |
| Properties | 45 |
| Public Properties..... | 52 |
| Events | 55 |
| Procedures and functions | 58 |
| TAdvDirectoryTreeView and TAdvCheckedTreeView..... | 66 |
| TMS Mini HTML rendering engine..... | 66 |

Introduction





The TMS Advanced TreeView offers a wide range of features to enhance your applications.

- High performance virtual and collection-based mode able to easily deal with millions of nodes
- Multi-line HTML formatted text
- Various built-in column editors
- Multi-column support
- Fixed and variable node height and node auto sizing
- Multiple events for custom drawing and customization of default drawing
- Multiple events for all kinds of interactions such as editing, expand / collapse and selection
- Auto-sizing and stretching of columns
- Mouse and keyboard interaction
- Nodes with checkbox, radiobutton, image, disabled nodes
- Nodes extending over multiple columns
- TAdvCheckedTreeView & TAdvDirectoryTreeView

| Item | Description | Price | Stock | Amount | Delivery method |
|---|--|---------|-------------------------------|----------------|------------------------------------|
| [-] Cakes | | | | | |
| [-] Decoration | | | | | |
|  | A candle is wax with an ignitable wick embedded that provides <ul style="list-style-type: none"> • light • fragrance It can also be used to provide heat, or as a method of keeping time. | \$1.30 | <input type="text" value=""/> | 602 | Pro (2-3 business days, + \$5) |
|  | A balloon is a flexible that can be inflated with a gas, such as helium, hydrogen, nitrous oxide, oxygen, or air. | \$8.62 | <input type="text" value=""/> | [Enter Amount] | |
| [-] Types | | | | | |
| <input type="checkbox"/> |  Cake is a form of sweet dessert that is typically baked. | \$34.00 | <input type="text" value=""/> | 992 | Pro (2-3 business days, + \$5) |
| <input checked="" type="checkbox"/> |  In its oldest forms, cakes were modifications of breads but now cover a wide range of preparations. | \$13.21 | <input type="text" value=""/> | [Enter Amount] | |
| <input type="checkbox"/> |  Typical cake ingredients are flour, sugar, eggs, and butter or oil. | \$3.07 | <input type="text" value=""/> | 551 | Exclusive (1 business day, + \$10) |
| <input checked="" type="checkbox"/> |  Cake is often served as a celebratory dish on ceremonial occasions, for example weddings , anniversaries , and birthdays. | \$1.98 | <input type="text" value=""/> | [Enter Amount] | |
| <input type="checkbox"/> |  There are countless cake recipes; some are bread-like, some rich and elaborate, and | \$12.51 | <input type="text" value=""/> | 66 | Pro (2-3 business days, + \$5) |

Organization

Below is a quick overview of the most important elements in the TreeView. This guide will cover all elements in different chapters.

| Item ¹⁾ | Description | Price | Stock | Amount | Delivery method |
|---|--|---------|--|----------------|------------------------------------|
| [-] Cakes | | | | | |
| [-] Decoration | | | | | |
|  | A candle is wax with an ignitable wick embedded that provides <ul style="list-style-type: none"> light fragrance It can also be used to provide heat, or as a method of keeping time. | \$1.30 | <div style="width: 50%;"><div style="width: 50%;"></div></div> | 602 | Pro (2-3 business days, + \$5) |
|  | A balloon is a flexible that can be inflated with a gas, such as helium, hydrogen, nitrous oxide, oxygen, or air. | \$8.62 | | [Enter Amount] | |
| [-] Types | | | | | |
| <input type="checkbox"/> | Cake is a form of sweet dessert that is typically baked. | \$34.00 | | 992 | Pro (2-3 business days, + \$5) |
| <input checked="" type="checkbox"/> | In its oldest forms, cakes were modifications of breads but now cover a wide range of preparations. | \$13.21 | | [Enter Amount] | |
| <input type="checkbox"/> | Typical cake ingredients are flour, sugar, eggs, and butter or oil. | \$3.07 | <div style="width: 30%;"><div style="width: 30%;"></div></div> | 551 | Exclusive (1 business day, + \$10) |
| <input checked="" type="checkbox"/> |  | \$1.98 | | [Enter Amount] | |
| <input type="checkbox"/> | Cake is often served as a celebratory dish on ceremonial occasions, for example weddings, anniversaries, and birthdays. | \$1.60 | <div style="width: 10%;"><div style="width: 10%;"></div></div> | 904 | Standard (5-10 business days) |
| <input checked="" type="checkbox"/> |  | \$26.07 | | [Enter Amount] | |
| <input type="checkbox"/> | There are countless cake recipes; some are bread-like, some rich and elaborate, and | \$12.51 | <div style="width: 40%;"><div style="width: 40%;"></div></div> | 66 | Pro (2-3 business days, + \$5) |

- 1) Columns / Column groups, which are added through the Columns / Groups collection. Columns based settings can override the default appearance for nodes. Via Columns a header and configure text alignment, wordwrapping and appearance can be specified.
- 2) Nodes: Holds a set of values such as the text, icon and check state that are represented in each column. Nodes can have child nodes and when child nodes are added, an expand/collapse icon is shown.
- 3) HTML formatted text: The node can display HTML formatted text for each column. Simply add the supported HTML tags and the TreeView will automatically switch to HTML.
- 4) Checkbox / Radiobutton support is added for each column. Additionally an icon can be specified for each column as well.
- 5) Customization: Each element in the TreeView can be fully customized. In the above sample a progressbar is drawn to indicate a certain level of stock instead of text.

Modes

The TreeView supports a collection-based and a virtual mode. Both modes will be explained in this chapter together with a small sample. You will notice that each programmatic call to manipulate / interact with nodes has a virtual and a collection-based version of the node. By default a collection-based TreeView is used. Below is a screenshot on how a default TAdvTreeView instance looks like when dropped on the form.

| Model | Year | Miles |
|---------------|------|---------|
| [-] Audi | | |
| A3 | 2010 | 32,300 |
| [-] A5 series | | |
| S5 | 2015 | 40,000 |
| RS5 | 2012 | 15,000 |
| A8 | 2005 | 80,000 |
| [-] Mercedes | | |
| SLS | 2000 | 300,000 |
| SLK | 2010 | 20,000 |
| GLA | 2012 | 14,500 |

Virtual

The TreeView will retrieve its values for each column and the number of nodes/child nodes through events. Each event that retrieves the node values passes an ANode parameter of type TAdvTreeViewVirtualNode. The most important event to start with is the OnGetNumberOfNodes event. This event retrieves the number of nodes during creation of the TreeView. The event is also called for child nodes after successfully retrieving the child count for each node. The first level of nodes is -1 (under root) which is the initial display of nodes and can be accessed with the ANode parameter in the OnGetNumberOfNodes event. Below is a sample that demonstrates this.

```

procedure TForm1.AdvTreeView1GetNumberOfNodes(Sender: TObject;
  ANode: TAdvTreeViewVirtualNode; var ANumberOfNodes: Integer);
begin
  if ANode.Level = -1 then

```


| Column 1 | Column 2 |
|---------------------|---------------------|
| Node 0 for Column 1 | Node 0 for Column 2 |
| Node 1 for Column 1 | Node 1 for Column 2 |
| Node 2 for Column 1 | Node 2 for Column 2 |
| Node 3 for Column 1 | Node 3 for Column 2 |
| Node 4 for Column 1 | Node 4 for Column 2 |
| Node 5 for Column 1 | Node 5 for Column 2 |
| Node 6 for Column 1 | Node 6 for Column 2 |
| Node 7 for Column 1 | Node 7 for Column 2 |
| Node 8 for Column 1 | Node 8 for Column 2 |
| Node 9 for Column 1 | Node 9 for Column 2 |

To add child nodes for each node the level of the nodes is identified with the level property on the ANode parameter. Note from the first sample that the level is -1 for the root nodes. For all root child nodes that are added the level is 0 or larger. Each node has an Index parameter and a Row parameter to uniquely identify each node. The following sample adds 3 root nodes and adds 5 child nodes for the first root node.

```

procedure TForm1.AdvTreeView1GetNumberOfNodes(Sender: TObject;
  ANode: TAdvTreeViewVirtualNode; var ANumberOfNodes: Integer);
begin
  if ANode.Level = -1 then
    ANumberOfNodes := 3
  else if (ANode.Level = 0) and (ANode.Index = 0) then
    ANumberOfNodes := 5;
end;

```

```

procedure TForm1.AdvTreeView1GetNodeText(Sender: TObject;
  ANode: TAdvTreeViewVirtualNode; AColumn: Integer;
  AMode: TAdvTreeViewNodeTextMode; var AText: string);
begin
  if ANode.Level = 0 then
    AText := 'Node ' + inttostr(ANode.Index) + ' for ' +
  AdvTreeView1.Columns[AColumn].Text
  else
    AText := 'Child Node ' + inttostr(ANode.Index)
end;

```

| Column 1 | Column 2 |
|-------------------------|---------------------|
| [-] Node 0 for Column 1 | Node 0 for Column 2 |
| Child Node 0 | Child Node 0 |
| Child Node 1 | Child Node 1 |
| Child Node 2 | Child Node 2 |
| Child Node 3 | Child Node 3 |
| Child Node 4 | Child Node 4 |
| Node 1 for Column 1 | Node 1 for Column 2 |
| Node 2 for Column 1 | Node 2 for Column 2 |

Each property that affects the node text, icon, check state, ... can be configured through the OnGetNode* events. Alternatively a collection-based approach can be used which is explained below. When using a virtual TreeView all virtual procedures, functions and properties need to be used. Below is a sample that expands all nodes in a virtual TreeView.

```
AdvTreeView1.ExpandAllVirtual;
```

Collection-based

A collection-based TreeView uses nodes from the Nodes collection property. Each node represents a set of values for each column that can be accessed through the Values property. Below is the same sample as in the Virtual mode, but then created through the Nodes collection.

```
var
  I: Integer;
  C: Integer;
  K: Integer;
  pn: TAdvTreeViewNode;
begin
  AdvTreeView1.BeginUpdate;
  AdvTreeView1.ClearColumns;
  AdvTreeView1.ClearNodes;
  AdvTreeView1.Columns.Add.Text := 'Column 1';
  AdvTreeView1.Columns.Add.Text := 'Column 2';

  for I := 0 to 2 do
```



```

begin
  pn := AdvTreeView1.AddNode;
  for C := 0 to AdvTreeView1.Columns.Count - 1 do
    pn.Text[C] := 'Node ' + inttostr(I) + ' for ' +
AdvTreeView1.Columns[C].Text;

    if I = 0 then
      begin
        for K := 0 to 4 do
          begin
            childn := AdvTreeView1.AddNode(pn);
            for C := 0 to AdvTreeView1.Columns.Count - 1 do
              childn.Text[C] := 'Child Node ' + inttostr(K);
            end;
          end;
        end;
      end;

  AdvTreeView1.EndUpdate;
end;

```

| Column 1 | Column 2 |
|---------------------|---------------------|
| Node 0 for Column 1 | Node 0 for Column 2 |
| Child Node 0 | Child Node 0 |
| Child Node 1 | Child Node 1 |
| Child Node 2 | Child Node 2 |
| Child Node 3 | Child Node 3 |
| Child Node 4 | Child Node 4 |
| Node 1 for Column 1 | Node 1 for Column 2 |
| Node 2 for Column 1 | Node 2 for Column 2 |

When using a collection-based TreeView the information of each node such as the position, height, level, ... is stored in the TAdvTreeViewVirtualNode object which is the same object being used in the virtual mode. Each collection-based node has a reference to the virtual node through the VirtualNode property. When using a collection-based TreeView the non-virtual procedures / functions and properties need to be used. Below is a sample that expands all nodes in a collection-based TreeView.

```
AdvTreeView1.ExpandAll;
```

Columns

Configuration / Appearance

The columns are configured through the Columns collection. Each column displays a set of values for each node such as the text, icon and check state. The most important property for a column is the UseDefaultAppearance property which is used to switch between the properties set at ColumnsAppearance level or the properties on the column collection item level for controlling the appearance of a column. Per column, horizontal, vertical text alignment as well as trimming and word wrapping can be configured. Fine-tuning is possible through a variety of events. Below is a sample that explains the difference between using the default appearance and customizing the appearance with the UseDefaultAppearance = false property per column.

In the following sample, we want to customize the font color and size of the header of the column and the font color of the nodes. For this we need to set the ColumnsAppearance.TopFont.Color, the ColumnsAppearance.TopFont and the NodesAppearance.Font.Color properties. Note that the NodesAppearance covers the nodes area while the ColumnsAppearance covers the columns area.

```
var
  n: TAdvTreeNode;
begin
  AdvTreeView1.BeginUpdate;
  AdvTreeView1.Nodes.Clear;
  AdvTreeView1.Columns.Clear;

  AdvTreeView1.Columns.Add.Text := 'Column 1';
  AdvTreeView1.Columns.Add.Text := 'Column 2';
  n := AdvTreeView1.AddNode;
  n.Text[0] := 'Node 0 for Column 1';
  n.Text[1] := 'Node 0 for Column 2';
  n := AdvTreeView1.AddNode;
  n.Text[0] := 'Node 1 for Column 1';
  n.Text[1] := 'Node 1 for Column 2';
  n := AdvTreeView1.AddNode;
  n.Text[0] := 'Node 2 for Column 1';
  n.Text[1] := 'Node 2 for Column 2';

  AdvTreeView1.ColumnsAppearance.TopFont.Size := 16;
  AdvTreeView1.ColumnsAppearance.TopFont.Color := gcOrange;
```

```
AdvTreeView1.NodesAppearance.Font.Color := gcSeagreen;
```

```
AdvTreeView1.EndUpdate;  
end;
```

| Column 1 | Column 2 |
|---------------------|---------------------|
| Node 0 for Column 1 | Node 0 for Column 2 |
| Node 1 for Column 1 | Node 1 for Column 2 |
| Node 2 for Column 1 | Node 2 for Column 2 |

Let's say we add a third column, and don't want to take on the default appearance, but instead use a different color for the header and nodes text and we don't change the font size. Additionally we also apply trimming. Below is a sample that demonstrates this.

```
var  
  n: TAdvTreeNode;  
begin  
  AdvTreeView1.BeginUpdate;  
  AdvTreeView1.Nodes.Clear;  
  AdvTreeView1.Columns.Clear;  
  
  AdvTreeView1.Columns.Add.Text := 'Column 1';  
  AdvTreeView1.Columns.Add.Text := 'Column 2';  
  AdvTreeView1.Columns.Add.Text := 'Column 3';  
  AdvTreeView1.Columns[2].UseDefaultAppearance := False;  
  AdvTreeView1.Columns[2].Trimming := tvttWord;  
  
  n := AdvTreeView1.AddNode;  
  n.Text[0] := 'Node 0 for Column 1';  
  n.Text[1] := 'Node 0 for Column 2';  
  n.Text[2] := 'Node 0 for Column 3';  
  n := AdvTreeView1.AddNode;  
  n.Text[0] := 'Node 1 for Column 1';  
  n.Text[1] := 'Node 1 for Column 2';  
  n.Text[2] := 'Node 1 for Column 3';  
  n := AdvTreeView1.AddNode;  
  n.Text[0] := 'Node 2 for Column 1';  
  n.Text[1] := 'Node 2 for Column 2';  
  n.Text[2] := 'Node 2 for Column 3';
```

```
AdvTreeView1.ColumnsAppearance.TopFont.Color := gcOrange;
AdvTreeView1.ColumnsAppearance.TopFont.Size := 16;
AdvTreeView1.NodesAppearance.Font.Color := gcSeagreen;
```

```
AdvTreeView1.EndUpdate;
end;
```

| Column 1 | Column 2 | Column 3 |
|-----------|---------------|---------------|
| Node 0 fo | Node 0 for Cc | Node 0 for... |
| Node 1 fo | Node 1 for Cc | Node 1 for... |
| Node 2 fo | Node 2 for Cc | Node 2 for... |

As you will notice, the default font color for both the header and the nodes is gray which can be set on column level with the properties `Column[Index].TopFont.Color` and `Column[Index].Font.Color`. The following sample adds 2 additional lines to the previous sample to configure this.

```
AdvTreeView1.Columns[2].TopFont.Color := clRed;
AdvTreeView1.Columns[2].Font.Color := clPurple;
```

| Column 1 | Column 2 | Column 3 |
|-----------|---------------|---------------|
| Node 0 fo | Node 0 for Cc | Node 0 for... |
| Node 1 fo | Node 1 for Cc | Node 1 for... |
| Node 2 fo | Node 2 for Cc | Node 2 for... |

Autosizing and stretching

When dropping a TreeView instance on the form, you will notice that it already has three columns and has default behavior of stretching those columns to fit the width of the control. The TreeView exposes the ability to stretch all columns, or a specific column. When turning off stretching completely each column has its own `Width` property that can be used to set a fixed width for a column.

Below is a sample of the default TreeView and a sample after the width of the TreeView has been changed.

default

| Model | Year | Miles |
|---------------|------|---------|
| [-] Audi | | |
| A3 | 2010 | 32,300 |
| [-] A5 series | | |
| S5 | 2015 | 40,000 |
| RS5 | 2012 | 15,000 |
| A8 | 2005 | 80,000 |
| [-] Mercedes | | |
| SLS | 2000 | 300,000 |
| SLK | 2010 | 20,000 |
| GLA | 2012 | 14,500 |

width changed

| Model | Year | Miles |
|---------------|------|---------|
| [-] Audi | | |
| A3 | 2010 | 32,300 |
| [-] A5 series | | |
| S5 | 2015 | 40,000 |
| RS5 | 2012 | 15,000 |
| A8 | 2005 | 80,000 |
| [-] Mercedes | | |
| SLS | 2000 | 300,000 |
| SLK | 2010 | 20,000 |
| GLA | 2012 | 14,500 |

As explained, the default behavior of the columns is to stretch. Below is a sample that turns off stretching for all columns except for a specific column and instead automatically uses the leftover width.

```
AdvTreeView1.ColumnsAppearance.StretchAll := False;
AdvTreeView1.ColumnsAppearance.StretchColumn := 1;
```

| Model | Year | Miles |
|---------------|------|---------|
| [-] Audi | | |
| A3 | 2010 | 32,300 |
| [-] A5 series | | |
| S5 | 2015 | 40,000 |
| RS5 | 2012 | 15,000 |
| A8 | 2005 | 80,000 |
| [-] Mercedes | | |
| SLS | 2000 | 300,000 |
| SLK | 2010 | 20,000 |
| GLA | 2012 | 14,500 |

Turning off stretching completely with the Stretch property will allow the TreeView to fall back on the width property of each column which is 100 by default.

```
AdvTreeView1.ColumnsAppearance.Stretch := False;
```

| Model | Year | Miles |
|---------------|------|---------|
| [-] Audi | | |
| A3 | 2010 | 32,300 |
| [-] A5 series | | |
| S5 | 2015 | 40,000 |
| RS5 | 2012 | 15,000 |
| A8 | 2005 | 80,000 |
| [-] Mercedes | | |
| SLS | 2000 | 300,000 |
| SLK | 2010 | 20,000 |
| GLA | 2012 | 14,500 |

Autosizing can be done only when the Stretch property is set to false. The ability is included to autosize on double-click on the column header splitter line, but this feature is explained in the Interaction chapter. When programmatically autosizing the visible nodes, column header for top and bottom layouts are take into calculation to determine the width for a column. Below is a sample that applies autosizing on all three columns, after turning off stretching.

```

var
  I: Integer;
begin
  AdvTreeView1.ColumnsAppearance.Stretch := False;
  for I := 0 to AdvTreeView1.Columns.Count - 1 do
    AdvTreeView1.AutoSizeColumn(I);
end;

```

| Model | Year | Miles |
|---------------|------|---------|
| [-] Audi | | |
| A3 | 2010 | 32,300 |
| [-] A5 series | | |
| S5 | 2015 | 40,000 |
| RS5 | 2012 | 15,000 |
| A8 | 2005 | 80,000 |
| [-] Mercedes | | |
| SLS | 2000 | 300,000 |
| SLK | 2010 | 20,000 |
| GLA | 2012 | 14,500 |

Note that autosizing is only applied to the visible nodes, so collapsed nodes and nodes that fall outside the visible region will not be taken into calculation. To support autosizing on expand/collapse and scrolling, events can be used to accomplish this.

Groups

Configuration

Groups are used to add additional information to columns. They can be added to multiple columns or simply cover one column. Below is a sample that adds 2 groups for 3 columns, one group that is used for the first column and a group that stretches of the last 2 columns.

```
var
  grp: TAdvTreeViewGroup;
begin
  grp := AdvTreeView1.Groups.Add;
  grp.StartColumn := 0;
  grp.EndColumn := 1;
  grp.Text := 'Important';

  grp := AdvTreeView1.Groups.Add;
  grp.StartColumn := 2;
  grp.EndColumn := 3;
  grp.Text := 'Less Important';
end;
```

| Important | | Less Important |
|---------------|------|----------------|
| Model | Year | Miles |
| [-] Audi | | |
| A3 | 2010 | 32,300 |
| [-] A5 series | | |
| S5 | 2015 | 40,000 |
| RS5 | 2012 | 15,000 |
| A8 | 2005 | 80,000 |
| [-] Mercedes | | |
| SLS | 2000 | 300,000 |

Note that in this case, the additional groups decrease the available space for nodes so a vertical scrollbar is needed to make sure all nodes are reachable.

Appearance

As with the columns, the groups have their own appearance control. The default appearance is stored under the GroupsAppearance property and can be overridden with the UseDefaultAppearance property per group. Below is a sample that demonstrates this.

```
var
  grp: TAdvTreeViewGroup;
begin
  AdvTreeView1.BeginUpdate;
  grp := AdvTreeView1.Groups.Add;
  grp.StartColumn := 0;
  grp.EndColumn := 1;
  grp.Text := 'Important';

  grp := AdvTreeView1.Groups.Add;
  grp.StartColumn := 2;
  grp.EndColumn := 3;
  grp.Text := 'Less Important';
  grp.UseDefaultAppearance := False;
  grp.TopFill.Color := clRed;
  grp.TopFill.Kind := tvbksolid;
  grp.TopFont.Color := clWhite;

  AdvTreeView1.GroupsAppearance.TopFont.Size := 16;
  AdvTreeView1.GroupsAppearance.TopFont.Style := [fsBold];
  AdvTreeView1.GroupsAppearance.TopFont.Color := gcSeagreen;

  AdvTreeView1.EndUpdate;
end;
```

| Important | | Less Important |
|------------------------------------|------|----------------|
| Model | Year | Miles |
| <input type="checkbox"/> Audi | | |
| A3 | 2010 | 32,300 |
| <input type="checkbox"/> A5 series | | |
| S5 | 2015 | 40,000 |
| RS5 | 2012 | 15,000 |
| A8 | 2005 | 80,000 |
| <input type="checkbox"/> Mercedes | | |
| SLS | 2000 | 300,000 |

Nodes

Configuration / Appearance

Nodes are the core data structure for the TreeView and as already explained in the Modes chapter, the TreeView can use a collection-based and virtual mode for displaying nodes. The virtual mode always starts by implementing the OnGetNumberOfNodes event and the collection-based mode starts with the Nodes collection property. Each collection-based node automatically generates a virtual node to hold the most important information such as the Index, Row, Level, Children and TotalChildren. For each event that is triggered, the virtual node is passed as a parameter, because when using only a virtual based TreeView, the values represented in each column need to be returned through events. When a collection-based TreeView is used, and events need to be implemented, each virtual node holds a reference to the collection item node (TAdvTreeViewVirtualNode.Node) that is used and vice versa (TAdvTreeViewNode.VirtualNode). Only when using a virtual TreeView the TAdvTreeViewVirtualNode .Node property will be nil.

Important to know is that each procedure, function and property has a collection-based and a virtual implementation. Generally, the procedures, functions and properties without virtual in the name are used for a collection-based TreeView.

The appearance of the nodes is configured through the NodesAppearance property. As explained in the columns chapter, the nodes appearance can be overridden per column with setting UseDefaultAppearance = false.

Adding, inserting and removing nodes

Adding, inserting and removing nodes are supported in both collection-based and virtual mode. As already explained, each mode has its own procedures, methods and events. In this chapter we start with an empty TreeView, so all nodes are removed from the collection which are added at designtime. Both collection-based and virtual add, insert and remove node methods will be explained here.

Each TreeView, whether it's collection-based or virtual will start without nodes and with a single column. The code to accomplish this is demonstrated below.

```
AdvTreeView1.BeginUpdate;  
AdvTreeView1.ClearColumns;  
AdvTreeView1.ClearNodes;
```

```
AdvTreeView1.Columns.Add.Text := 'Column 1';  
AdvTreeView1.EndUpdate;
```

Additionally for the virtual TreeView implementation the `OnGetNumberOfNodes` always needs to be implemented and return at least one node. With virtual mode the text is empty by default, so the `OnGetNodeText` event needs to be implemented as well. The code below demonstrates this. Please note that the code below is only added in case a virtual TreeView mode is chosen.

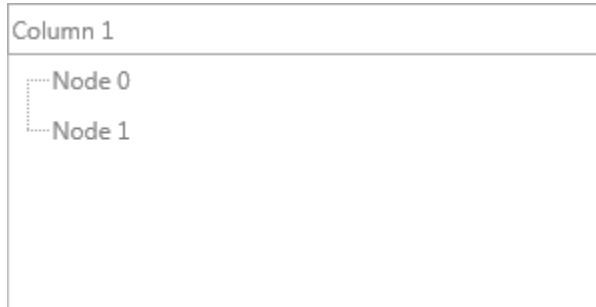
```
procedure TForm1.AdvTreeView1GetNodeText(Sender: TObject;  
  ANode: TAdvTreeViewVirtualNode; AColumn: Integer;  
  AMode: TAdvTreeViewNodeTextMode; var AText: string);  
begin  
  AText := 'Node ' + inttostr(ANode.Index);  
end;  
  
procedure TForm1.AdvTreeView1GetNumberOfNodes(Sender: TObject;  
  ANode: TAdvTreeViewVirtualNode; var ANumberOfNodes: Integer);  
begin  
  if ANode.Level = -1 then  
    ANumberOfNodes := 1;  
end;
```

Adding a new node (virtual)

A new node is added with the code `AdvTreeView1.AddVirtualNode;`. Note that the `OnGetNodeText` will be called returning a different text for the newly added node.

Adding a new node (collection-based)

In a collection-based TreeView, a node is added directly to the `Nodes` collection, or with the helper method `AdvTreeView1.AddNode;`. To get the same result as with the virtual implementation, we need to add 2 nodes, because in the virtual mode, the first node was added through the `OnGetNumberOfNodes`, which isn't used in a collection-based TreeView.



Adding child nodes (virtual)

Child nodes can be added with the same function, but instead passing the parent node as a parameter. The following sample demonstrates how to add a child node to the second root node added with the `AddVirtualNode` method. Additionally, the parent node that is added together with the child node is expanded to visually the newly added child node.

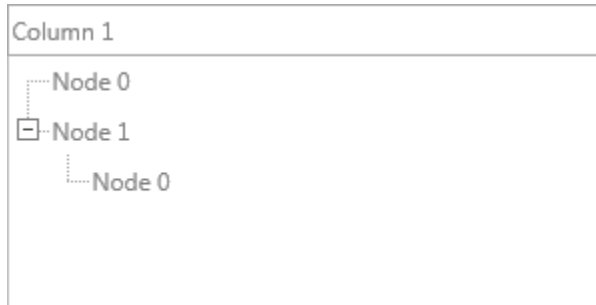
```
var
    pn, n: TAdvTreeViewVirtualNode;
begin
    pn := AdvTreeView1.AddVirtualNode;
    n := AdvTreeView1.AddVirtualNode(pn);
    AdvTreeView1.ExpandVirtualNode(pn);
end;
```

Adding child nodes (collection-based)

Child nodes can be added the same way as in the virtual mode, but with different method names. When we copy the above code and remove the `Virtual` keyword in the method name, the result output will be identical if we keep in mind that an additional node needs to be added in the collection to match the virtual node added with the `OnGetNumberOfNodes`.

```
var
    pn, n: TAdvTreeViewNode;
begin
    pn := AdvTreeView1.AddNode;
    pn.Text[0] := 'Node 0';
    pn := AdvTreeView1.AddNode;
    pn.Text[0] := 'Node 1';
    n := AdvTreeView1.AddNode(pn);
    n.Text[0] := 'Node 0';
    AdvTreeView1.ExpandNode(pn);
```

end;



Inserting a new node

Inserting nodes is done in the same way as adding nodes, but an additional parameter can be passed specifying the insert position of the new node. In virtual mode, there isn't any difference between inserting and adding new nodes because the `OnGetNodeText` will return text based on the index of the node.

Additionally, in a collection-based TreeView, the index parameter of the collection item node can be used to switch positions with an already existing node, creating a move node functionality.

Removing an existing node (virtual)

Removing an existing node can be done with the `RemoveVirtualNode` method. The parameter to pass is an existing node. The following sample retrieves the focused node and removes it.

```
AdvTreeView1.RemoveVirtualNode(AdvTreeView1.FocusedVirtualNode);
```

Removing an existing node (collection-based)

In a collection-based TreeView, removing a node can be done in a similar way but without the `Virtual` keyword in the method name. Additionally a node can also be removed by freeing the collection item. Below code output is identical and removes the focused node on both cases.

```
AdvTreeView1.RemoveNode(AdvTreeView1.FocusedNode);
```

```
AdvTreeView1.FocusedNode.Free;
```

Fixed vs variable node height

A key feature of the TreeView in both collection-based and virtual mode is support for fixed and variable node height. The simplest configuration is the fixed node height where each node has the same height, based on the `NodesAppearance.FixedHeight` property. Word wrapping the text of a node or specifying a node icon will be based on the fixed height and thus exceeding the node bounds when the height of the text or the node icon is larger than the fixed height.

To support auto-sizing of nodes, based on the node icon or text, the `NodesAppearance.HeightMode` property needs to change to `tnhmVariable`. The `NodesAppearance.VariableMinimumHeight` property is used to specify a minimum height for each node, so to initial total height for displaying a scrollbar can be calculated. The default value for this property is 25. Keep in mind that the TreeView needs to perform additional calculations during startup and during scrolling. Below is a sample that demonstrates the difference between a fixed and variable node height TreeView configuration. Both samples are demonstrated in a virtual TreeView implementation.

Fixed

```
AdvTreeView1.BeginUpdate;
AdvTreeView1.ClearNodes;
AdvTreeView1.ClearColumns;
AdvTreeView1.Columns.Add.Text := 'Fixed TreeView';
AdvTreeView1.EndUpdate;

procedure TForm1.AdvTreeView1GetNodeText(Sender: TObject;
  ANode: TAdvTreeViewVirtualNode; AColumn: Integer;
  AMode: TAdvTreeViewNodeTextMode; var AText: string);
begin
  AText := 'Node ' + inttostr(ANode.Index);
end;

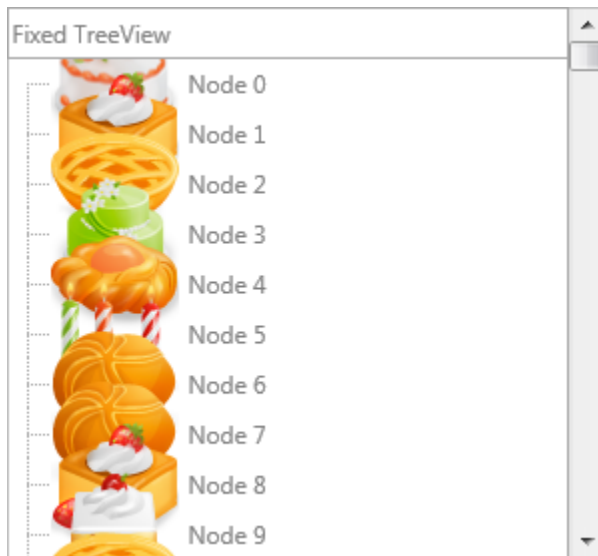
procedure TForm1.AdvTreeView1GetNumberOfNodes(Sender: TObject;
  ANode: TAdvTreeViewVirtualNode; var ANumberOfNodes: Integer);
begin
  if ANode.Level = -1 then
    ANumberOfNodes := 1000000;
end;

procedure TForm1.AdvTreeView1GetNodeIcon(Sender: TObject;
  ANode: TAdvTreeViewVirtualNode; AColumn: Integer; ALarge: Boolean;
  var AIcon: TGraphic);
begin
```

```

AIcon :=
PictureContainer1.Items [Random (PictureContainer1.Items.Count)].Picture
;
end;

```



Note that the icons specified through the `OnGetNodeIcon` event are too large to fit inside the fixed node height. The solution can be to specify a larger fixed height through the `NodesAppearance.FixedHeight` property, but when the values that need to be loaded are unknown, the fixed height approach is no longer valid. When switching to a variable row height mode you will notice that the node height will automatically take on the size of the icons.

```

AdvTreeView1.BeginUpdate;
AdvTreeView1.NodesAppearance.HeightMode := tnhmVariable;
AdvTreeView1.ClearNodes;
AdvTreeView1.ClearColumns;
AdvTreeView1.Columns.Add.Text := 'Variable TreeView';
AdvTreeView1.EndUpdate;

procedure TForm1.AdvTreeView1GetNodeIcon (Sender: TObject;
  ANode: TAdvTreeViewVirtualNode; AColumn: Integer; ALarge: Boolean;
  var AIcon: TGraphic);
begin
  AIcon :=
PictureContainer1.Items [Random (PictureContainer1.Items.Count)].Bitmap;
end;

```

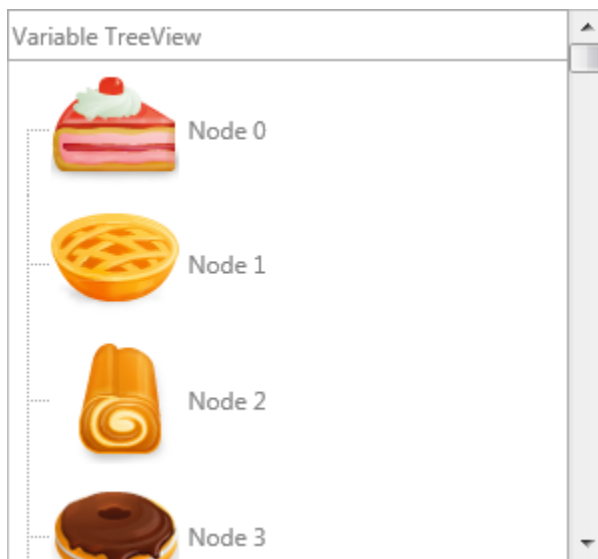


```

procedure TForm1.AdvTreeView1GetNodeText(Sender: TObject;
  ANode: TAdvTreeViewVirtualNode; AColumn: Integer;
  AMode: TAdvTreeViewNodeTextMode; var AText: string);
begin
  AText := 'Node ' + inttostr(ANode.Index);
end;

procedure TForm1.AdvTreeView1GetNumberOfNodes(Sender: TObject;
  ANode: TAdvTreeViewVirtualNode; var ANumberOfNodes: Integer);
begin
  if ANode.Level = -1 then
    ANumberOfNodes := 1000000;
end;

```



In case the text is larger than the node icon, the node height will automatically adapt as shown in the sample below.

```

AdvTreeView1.BeginUpdate;
AdvTreeView1.NodesAppearance.HeightMode := tnhmVariable;
AdvTreeView1.ClearNodes;
AdvTreeView1.ClearColumns;
AdvTreeView1.Columns.Add.Text := 'Variable TreeView';
AdvTreeView1.Columns[0].WordWrapping := True;
AdvTreeView1.EndUpdate;

```

```

procedure TForm1.AdvTreeView1GetNodeIcon(Sender: TObject;
  ANode: TAdvTreeViewVirtualNode; AColumn: Integer; ALarge: Boolean;
  var AIcon: TGraphic);
begin
  AIcon :=
  PictureContainer1.Items[Random(PictureContainer1.Items.Count)].Picture
  ;
end;

```

```

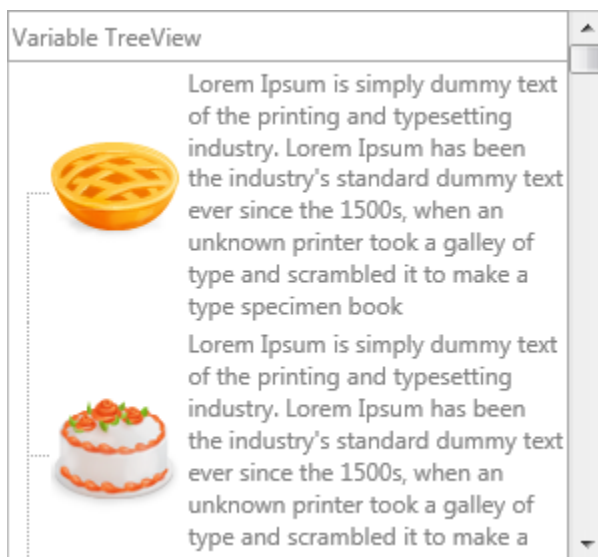
procedure TForm1.AdvTreeView1GetNodeText(Sender: TObject;
  ANode: TAdvTreeViewVirtualNode; AColumn: Integer;
  AMode: TAdvTreeViewNodeTextMode; var AText: string);
begin
  AText := 'Lorem Ipsum is simply dummy text of the printing and
  typesetting industry. Lorem Ipsum has been the industry's standard
  dummy text ever since the 1500s, when an unknown printer took a galley
  of type and scrambled it to make a type specimen book';
end;

```

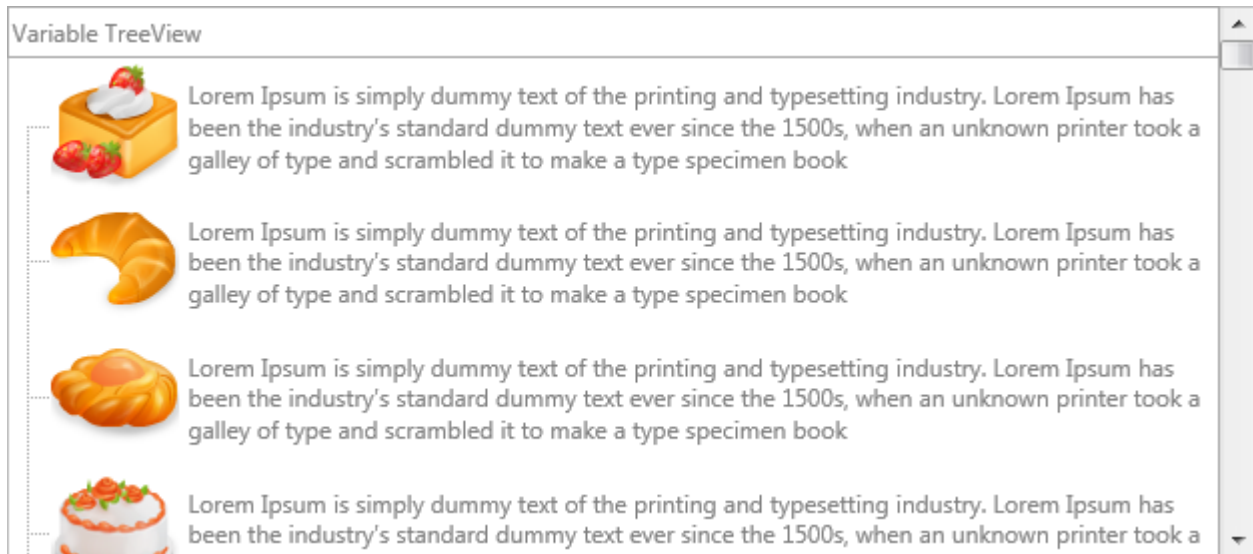
```

procedure TForm1.AdvTreeView1GetNumberOfNodes(Sender: TObject;
  ANode: TAdvTreeViewVirtualNode; var ANumberOfNodes: Integer);
begin
  if ANode.Level = -1 then
    ANumberOfNodes := 1000000;
end;

```



When resizing, the node heights will be recalculated, giving more space for the text, and thus decreasing the necessary height for a node to display all the contents.



Checkbox / Radiobutton support

The TreeView has radiobutton and checkbox support. When specifying a check type through the OnGetNodeCheckType event or through the collection-based property CheckTypes at node level a checkbox or radiobutton will be displayed. More information on interaction will be explained at the Interaction chapter.

```

var
  n: TAdvTreeNode;
  I: Integer;
begin
  AdvTreeView1.BeginUpdate;
  AdvTreeView1.ClearNodes;
  AdvTreeView1.ClearColumns;
  AdvTreeView1.Columns.Add.Text := 'Checkbox / radiobutton';
  for I := 0 to 3 do
  begin
    n := AdvTreeView1.Nodes.Add;
    n.Text[0] := 'CheckBox ' + IntToStr(I);
    n.CheckTypes[0] := tvntCheckBox;
    if Odd(I) then
      n.Checked[0] := True;
  end;
end;

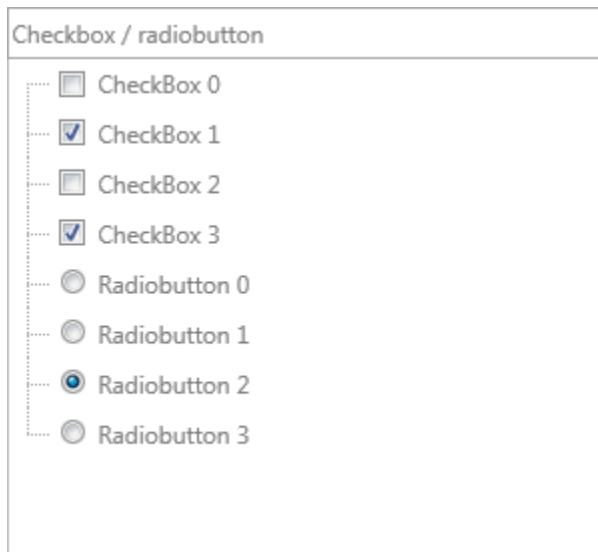
```

```

for I := 0 to 3 do
begin
  n := AdvTreeView1.Nodes.Add;
  n.Text[0] := 'Radiobutton ' + IntToStr(I);
  n.CheckTypes[0] := tvntRadioButton;
  if I = 2 then
    n.Checked[0] := True;
end;

AdvTreeView1.EndUpdate;
end;

```



Extended nodes

Extended nodes are nodes that are stretched over all columns and takes on the text of the first column. It is also styled with a different set of properties under NodesAppearance. An extended node is not editable and selectable by default. This behavior can be overridden in the Interaction property. To create an extended node, set the Extended property to true on a collection-based TreeView collection item node, or return True in the OnIsNodeExtended event. Below is a sample that demonstrates this.

```

var
  n, pn: TAdvTreeViewNode;
  I: Integer;
begin

```

```
AdvTreeView1.BeginUpdate;
AdvTreeView1.ClearNodes;
AdvTreeView1.ClearColumns;
AdvTreeView1.Columns.Add.Text := 'Column 1';
AdvTreeView1.Columns.Add.Text := 'Column 2';
pn := AdvTreeView1.Nodes.Add;
pn.Text[0] := 'Normal Node';

pn := AdvTreeView1.Nodes.Add;
pn.Text[0] := 'Extended Node';
pn.Extended := True;

for I := 0 to 3 do
begin
  n := AdvTreeView1.AddNode(pn);
  if I = 1 then
  begin
    n.Text[0] := 'Extended Node ' + IntToStr(I);
    n.Extended := True;
  end
  else
  begin
    n.Text[0] := 'Normal Node Column 1 ' + IntToStr(I);
    n.Text[1] := 'Normal Node Column 2 ' + IntToStr(I);
  end;
end;

AdvTreeView1.EndUpdate;
end;
```

| Column 1 | Column 2 |
|------------------------|------------------------|
| Normal Node | |
| Extended Node | |
| Normal Node Column 1 0 | Normal Node Column 2 0 |
| Extended Node 1 | |
| Normal Node Column 1 2 | Normal Node Column 2 2 |
| Normal Node Column 1 3 | Normal Node Column 2 3 |

Interaction

The TreeView supports interaction through mouse and keyboard. When clicking on a node that is selectable, the node is selected. When navigating with the keys up, down, home, end, page up or page down the selected node will be changed. Extended / disabled nodes are not selectable by default. The behaviour can be changed by changing the ExtendedSelectable and ExtendedEditable properties.

When the property MultiSelect is true, multiple nodes can be selected with the CTRL and SHIFT key with either the mouse or keyboard. The selected nodes can be retrieved with the SelectedNodeCount function and SelectedNodes property. Selection of nodes can be done with the SelectNode or SelectNodes method. The SelectNodes method takes an array of nodes. The above methods apply to a collection-based TreeView, but the same methods with the virtual method name are available for the virtual TreeView implementation.

When a node has children the left / right keys can expand or collapse the node and visualize or hide the children. Clicking on the expand / collapse node icon with the left mouse button will perform the same operation.

The keyboard and mouse can be used to edit the node text for each column when the column is configured to support editing. Additionally, when typing alphanumeric characters, the treeview will optionally search for the node that matches the lookup string and navigate to that node. To enable this feature, you need to set the Interaction.Lookup.Enabled property to true.

Clipboard

Cut, Copy and Paste is supported when setting the Interaction.ClipboardMode property to tcmTextOnly or tcmFull. The tcmTextOnly value only copies the text for each column and does not copy along other attributes such as the check and extended state, the node icon. The tcmFull clipboard mode copies all attributes of the node. Cut will first copy the node and then remove it from the treeview. When pasting, the focused node will act as the parent, if there is no node active the treeview will add the pasted node as a new node in the treeview. There are additional events that are triggered when performing a cut, copy or paste action.

Reordering / Drag & Drop

When setting `Interaction.Reorder` to `True`, clicking on an already selected node will duplicate the node and attach it while dragging. When releasing the node over another node on the same level it will reorder the node to the new location. Please note that touch scrolling is disabled when reordering is true on the selected node part. On the non-selected node parts, touch scrolling is still active.

| Model | Year | Miles |
|---------------|------|---------|
| [-] Audi | | |
| A3 | 2010 | 32,300 |
| [-] A5 series | | |
| S5 | 2016 | 40,000 |
| A3 | 2010 | 32,300 |
| A8 | 2005 | 80,000 |
| [-] Mercedes | | |
| SLS | 2000 | 300,000 |
| SLK | 2010 | 20,000 |
| GLA | 2012 | 14,500 |

When setting `Interaction.DragDropMode` to `tdmMove` or `tdmCopy` the same approach can be used as reordering, and will allow you to drop the node as a child node of the dropped node. Drag & drop takes precedence over reordering, and with drag & drop you cannot only move or copy nodes in the same treeview but also move nodes to another treeview.

Filtering

When setting `Columns[Index].Filtering.Enabled := True`; a filter dropdown button appears at the right side of the column. Clicking on the filter button will show a filter dropdown list with unique values from the node for that specific column. After clicking a value, the treeview shows a filtered list.

| Model | Year | Miles |
|---------------|------|---------|
| [-] Audi | | |
| A3 | | 32,300 |
| [-] A5 series | | |
| S5 | | 40,000 |
| RS5 | | 15,000 |
| A8 | 2005 | 80,000 |
| [-] Mercedes | | |
| SLS | 2000 | 300,000 |
| SLK | 2010 | 20,000 |
| GLA | 2012 | 14,500 |

After filtering, the node that matches the chosen filter is shown.

| Model | Year | Miles |
|----------|------|--------|
| [-] Audi | | |
| A8 | 2005 | 80,000 |

To clear filtering on a column, click the '(All)' entry in the filter list.

Note that filtering is also available programmatically. Below is a sample that filters the nodes with an A:

```
var
  f: TAdvTreeViewFilterData;
begin
  AdvTreeView1.Filter.Clear;
  f := AdvTreeView1.Filter.Add;
  f.Column := 0;
  f.Condition := '*A*';
  AdvTreeView1.ApplyFilter;
end;
```

| Model | Year | Miles |
|------------------------------------|------|--------|
| <input type="checkbox"/> Audi | | |
| <input type="checkbox"/> A3 | 2010 | 32,300 |
| <input type="checkbox"/> A5 series | | |
| <input type="checkbox"/> A8 | 2005 | 80,000 |
| <input type="checkbox"/> Mercedes | | |
| <input type="checkbox"/> GLA | 2012 | 14,500 |

Additionally we want to filter values from the year 2010 or greater:

```
var
  f: TAdvTreeViewFilterData;
begin
  AdvTreeView1.Filter.Clear;
  f := AdvTreeView1.Filter.Add;
  f.Column := 0;
  f.Condition := '*A*';

  f := AdvTreeView1.Filter.Add;
  f.Column := 1;
```

```
f.Condition := ' >= 2010';  
AdvTreeView1.ApplyFilter;  
end;
```

| Model | Year | Miles |
|-----------------------------------|------|--------|
| <input type="checkbox"/> Audi | | |
| <input type="checkbox"/> A3 | 2010 | 32,300 |
| <input type="checkbox"/> Mercedes | | |
| <input type="checkbox"/> GLA | 2012 | 14,500 |

To clear all filtering programmatically, you can use the following code:
`AdvTreeView1.RemoveFilters;`

Note that if a child node matches a filter condition, the parent tree is also added.

Sorting

Sorting can be performed on each column separately. When clicking on the column, the nodes are sorted and the treeview is updated. Sorting can be done for root nodes only, or recursive with an optional case sensitivity requirement. Below is a sample that sorts based on all nodes (recursive).

```
AdvTreeView1.Columns[0].Sorting := tcsRecursive;
```

| Model | Year | Miles |
|-----------|------|---------|
| Mercedes | | |
| SLS | 2000 | 300,000 |
| SLK | 2010 | 20,000 |
| GLA | 2012 | 14,500 |
| Audi | | |
| A8 | 2005 | 80,000 |
| A5 series | | |
| S5 | 2016 | 40,000 |
| RS5 | 2012 | 15,000 |
| A3 | 2010 | 32,300 |

Sorting can also be done programmatically, with the following code, which will show the same result as the screenshot above.

```
AdvTreeView1.Sort(0, True, False, nsmDescending);
```

Editing

The TreeView supports inplace editing of nodes per column. Each column has the ability to specify an editor through the EditorType property. When editing is started, either by clicking on the text, or by pressing F2 on the keyboard the OnGetNodeText event is called to retrieve the text that needs to be placed inside the editor. To know if the OnGetNodeText event is called for drawing/calculation or for editing the AMode parameter can be used. If the OnGetNodeText event isn't used to return a different text when editing, the text of the node is used.

Below is a sample that demonstrates this. (Note that the code above is applied on a default TreeView instance)

```
AdvTreeView1.Columns[2].EditorType := tcetEdit;
```

| A5 series | | |
|-----------|------|--------|
| S5 | 2015 | 40,000 |
| RS5 | 2012 | 15,000 |

When the editing is started, the OnGetNodeText event is called with a different mode. To initialize the editor with a different text the following code provides a sample to achieve this.

```
procedure TForm1.AdvTreeView1GetNodeText(Sender: TObject;
  ANode: TAdvTreeViewVirtualNode; AColumn: Integer;
  AMode: TAdvTreeViewNodeTextMode; var AText: string);
begin
  case AMode of
    tntmEditing: AText := 'Editor Started !';
  end;
end;
```

| A5 series | | |
|-----------|------|------------------|
| S5 | 2015 | Editor Started ! |
| RS5 | 2012 | 15,000 |

After editing is finished, the OnBeforeUpdateNode is called to allow making changes to the edited text or block updating the node if necessary. Additionally the OnCloseInplaceEditor event can be used to stop the editor from closing if the requirements of the text are not met.

Note that when editing is allowed on multiple columns, starting to edit a node will always start with the first not read-only column and then the tab key will jump to the next editable column.

Other than the default TEdit editor, a TMemo or TComboBox can be chosen to allow editing. A TMemo is typically used to allow a multi-line editor and a TComboBox to have a choice menu in case multiple values are possible. A sample that shows how to use the TComboBox as an inplace editor is shown in the sample below.

```
var
```

```

I: Integer;
begin
  AdvTreeView1.Columns[1].EditorType := tcetComboBox;
  for I := 0 to 19 do
    AdvTreeView1.Columns[1].EditorItems.Add(IntToStr(2000 + I));
  end;
end;

```

| Model | Year | Miles |
|---------------|------|---------|
| [-] Audi | | |
| A3 | 2010 | 32,300 |
| [-] A5 series | | |
| S5 | 2015 | 40,000 |
| RS5 | 2000 | 15,000 |
| A8 | 2001 | 80,000 |
| [-] Mercedes | | |
| SLS | 2004 | 300,000 |
| SLK | 2005 | 20,000 |
| GLA | 2006 | 14,500 |
| | 2007 | |

If the built-in editors are not sufficient, the TreeView supports using a custom editor as well, by setting the CustomEditor property to true.

Custom Editor

The code below demonstrates how to use a custom editor, in this case a TTrackBar.

```

AdvTreeView1.Columns[1].CustomEditor := True;

procedure TForm1.AdvTreeView1BeforeUpdateNode(Sender: TObject;
  ANode: TAdvTreeViewVirtualNode; AColumn: Integer; var AText: string;
  var ACanUpdate: Boolean);
begin
  AText := FloatToStr((AdvTreeView1.GetInplaceEditor as
  TTrackBar).Value);
end;

procedure TForm1.AdvTreeView1GetInplaceEditor(Sender: TObject;

```

```

ANode: TAdvTreeViewVirtualNode; AColumn: Integer;
var ATransparent: Boolean;
var AInplaceEditorClass: TAdvTreeViewInplaceEditorClass);
begin
  AInplaceEditorClass := TTrackBar;
end;

```

| Model | Year | Miles |
|---------------|------|---------|
| [-] Audi | | |
| A3 | 2010 | 32,300 |
| [-] A5 series | | |
| S5 | | 40,000 |
| RS5 | 2012 | 15,000 |
| A8 | 2005 | 80,000 |
| [-] Mercedes | | |
| SLS | 2000 | 300,000 |
| SLK | 2010 | 20,000 |
| GLA | 2012 | 14,500 |

After changing the value, the `OnBeforeUpdateNode` event is triggered which sets the value of the node text to the value of the trackbar.

| | | |
|-----|-----------------|--------|
| S5 | 37.209300994873 | 40,000 |
| RS5 | 2012 | 15,000 |

Customization

The TreeView supports a wide range of customization possibilities. Below is a sample how to implement the correct events for custom node drawing. Note that this sample starts from a default TreeView with nodes already added to the collection.

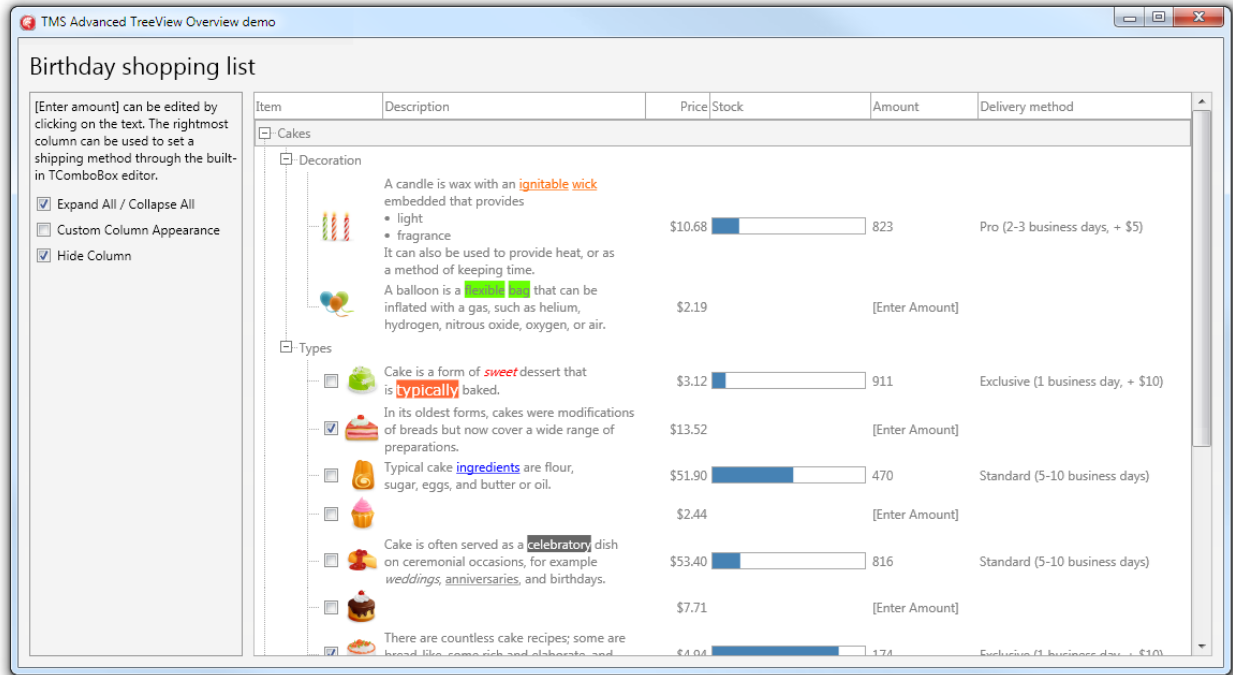
In the sample below the second column contains information on the build year of the car. To identify cars that are built in 2012 or later we want to draw a red ellipse in the top right corner of the text area for the year column.

```
procedure TForm1.AdvTreeView1AfterDrawNodeText(Sender: TObject;
  AGraphics: TAdvGraphics; ARect: TRectF; AColumn: Integer;
  ANode: TAdvTreeViewVirtualNode; AText: string);
var
  v: Integer;
begin
  if TryStrToInt(AText, v) then
  begin
    if (AColumn = 1) and (v >= 2012) then
    begin
      ACanvas.Brush.Style := bsSolid;
      ACanvas.Brush.Color := clRed;
      ACanvas.Ellipse(Round(ARect.Right - 12), Round(ARect.Top + 12),
Round(ARect.Right - 2), Round(ARect.Top + 2));
    end;
  end;
end;
```

| Model | Year | Miles |
|---------------|------|----------|
| [-] Audi | | |
| A3 | 2010 | 32,300 |
| [-] A5 series | | |
| S5 | 2015 | ● 40,000 |
| RS5 | 2012 | ● 15,000 |
| A8 | 2005 | 80,000 |
| [-] Mercedes | | |
| SLS | 2000 | 300,000 |
| SLK | 2010 | 20,000 |
| GLA | 2012 | ● 14,500 |

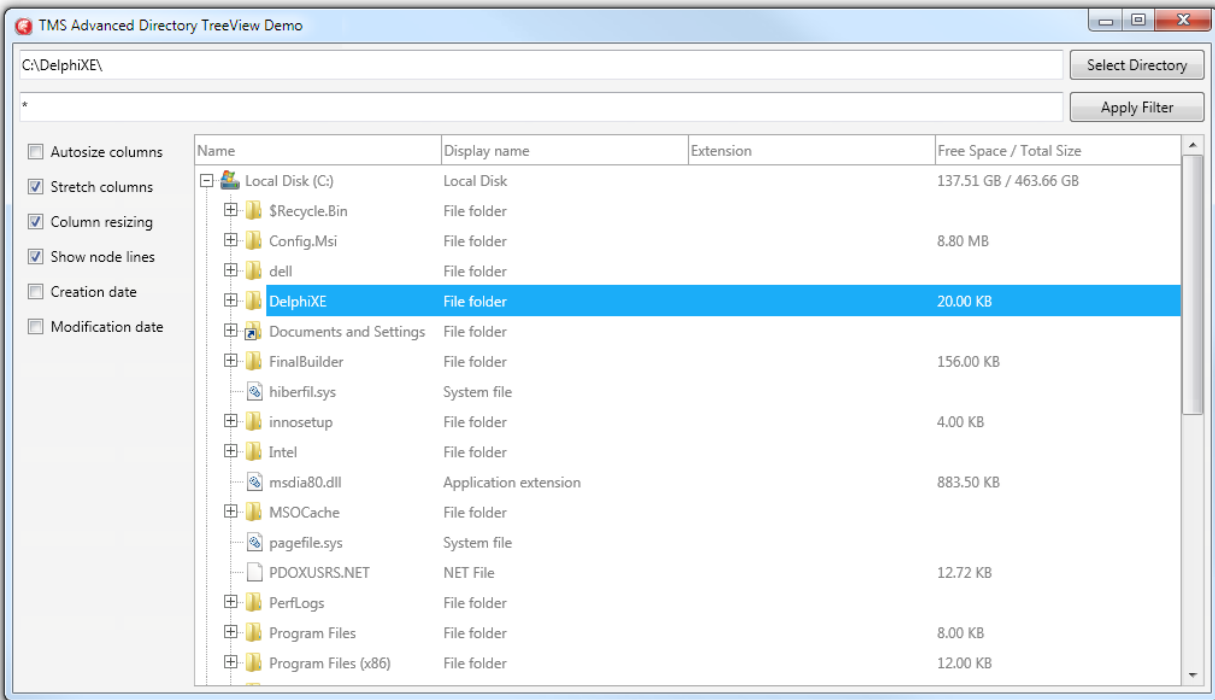
Demos

Overview



The overview demo demonstrates variable node heights, programmatically expand and collapse as well as custom column appearance and toggling column visibility. The node text in the description column is HTML formatted and a progressbar is custom drawn inside the stock column. The last 2 columns amount and delivery method show the capabilities of editing through TEdit and TComboBox.

Directory



The Directory treeview demo uses the TAdvDirectoryTreeView component that is capable of loading a drive, or a folder and apply a filter. Additionally the column sizing, auto sizing of columns and stretching is demonstrated.

Properties

| | |
|--------------------------------------|---|
| Columns | The collection of columns. Please note that each property that affects appearance is not applied unless UseDefaultAppearance is set to False. |
| Columns[Index] → BottomFill | The fill of the column when the Layout property is set to include tclBottom. |
| Columns[Index] → BottomFont | The font of the column text when the Layout property is set to include tclBottom. |
| Columns[Index] → BottomStroke | The stroke of the column when the Layout property is set to include tclBottom. |
| Columns[Index] → CustomEditor | Allows for a custom editor to be returned through the OnGetInplaceEditor event. |
| Columns[Index] → EditorItems | The items of the editor when using the tcetComboBox editor type. |
| Columns[Index] → EditorType | The type of editor to use for each node in a column. The editor can be customized per node. |
| Columns[Index] → Fill | The fill of a column. |
| Columns[Index] → Filtering | Configures filtering on a column. |
| Columns[Index] → Font | The font of the nodes of a column. |
| Columns[Index] → HorizontalTextAlign | The alignment of the text of the nodes of a column. |
| Columns[Index] → Name | The name of the column. |
| Columns[Index] → Sorting | Configures sorting on a column. |
| Columns[Index] → Stroke | The stroke of the column. |
| Columns[Index] → TopFill | The fill of the column when the Layout property is set to include tclTop. |
| Columns[Index] → TopFont | The font of the column text when the Layout property is set to include tclTop. |
| Columns[Index] → TopStroke | The stroke of the column when the Layout property is set to include |

| | |
|--|---|
| | tclTop. |
| Columns[Index] → Trimming | The trimming of nodes of a column. |
| Columns[Index] → UseDefaultAppearance | Allows overriding the default appearance of columns/nodes. |
| Columns[Index] → VerticalTextAlign | The vertical text alignment of nodes. |
| Columns[Index] → Visible | Sets the column visible / invisible. |
| Columns[Index] → Width | The width of the column. |
| Columns[Index] → WordWrapping | The word wrapping of nodes of a column. |
| ColumnsAppearance | The overall appearance of columns. Note that these properties are applied to all columns unless UseDefaultAppearance is set to False for a column. |
| ColumnsAppearance → BottomFill | The fill of the column when the Layout property is set to include tclBottom. |
| ColumnsAppearance → BottomFont | The font of the column text when the Layout property is set to include tclBottom. |
| ColumnsAppearance → BottomSize | The size of the bottom columns. |
| ColumnsAppearance → BottomStroke | The stroke of the column when the Layout property is set to include tclBottom. |
| ColumnsAppearance → BottomVerticalText | Allows displaying vertical text in the columns bottom layout. |
| ColumnsAppearance → FillEmptySpaces | Allows filling empty spaces at the right side of the columns when the StretchScrollBars property is set to False. |
| ColumnsAppearance → Layouts | The layout of the columns which include tclTop and tclBottom. |
| ColumnsAppearance → Stretch | Allows stretching of columns. |
| ColumnsAppearance → StretchAll | Stretches all columns. |
| ColumnsAppearance → StretchColumn | Calculates all columns except for the column that matches this property. The StretchColumn is automatically given the leftover width after calculation. |
| ColumnsAppearance → TopFill | The fill of the column when the |

| | |
|--------------------------------------|--|
| | Layout property is set to include tclTop. |
| ColumnsAppearance → TopFont | The font of the column text when the Layout property is set to include tclTop. |
| ColumnsAppearance → TopSize | The size of the top columns. |
| ColumnsAppearance → TopStroke | The stroke of the column when the Layout property is set to include tclTop. |
| ColumnsAppearance → TopVerticalText | Allows displaying vertical text in the columns top layout. |
| ColumnStroke | The stroke between columns of a specific column. |
| Groups | The collection of groups. Please note that each property that affects appearance is not applied unless UseDefaultAppearance is set to False. |
| Groups[Index] → BottomFill | The fill of the group when the Layout property is set to include tglBottom. |
| Groups[Index] → BottomFont | The font of the group text when the Layout property is set to include tglBottom. |
| Groups[Index] → BottomStroke | The stroke of the group when the Layout property is set to include tglBottom. |
| Groups[Index] → EndColumn | The column on which the group ends. Multiple groups can be added that cover multiple columns. |
| Groups[Index] → Name | The name of the group. |
| Groups[Index] → StartColumn | The column on which the group starts. |
| Groups[Index] → Text | The text of the group. |
| Groups[Index] → TopFill | The fill of the group when the Layout property is set to include tglTop. |
| Groups[Index] → TopFont | The font of the group text when the Layout property is set to include tglTop. |
| Groups[Index] → TopStroke | The stroke of the group when the Layout property is set to include tglTop. |
| Groups[Index] → UseDefaultAppearance | Allows overriding the default |

| | |
|--|---|
| | appearance of groups. |
| GroupsAppearance | The overall appearance of groups. Note that these properties are applied to all groups unless UseDefaultAppearance is set to False for a group. |
| GroupsAppearance → BottomFill | The fill of the group when the Layout property is set to include tglBottom. |
| GroupsAppearance → BottomFont | The font of the group text when the Layout property is set to include tglBottom. |
| GroupsAppearance → BottomHorizontalTextAlign | The horizontal alignment of the group text when the Layout property is set to include tglBottom. |
| GroupsAppearance → BottomSize | The size of the bottom columns. |
| GroupsAppearance → BottomStroke | The stroke of the column when the Layout property is set to include tclBottom. |
| GroupsAppearance → BottomVerticalText | Allows displaying vertical text in the columns bottom layout. |
| GroupsAppearance → BottomVerticalTextAlign | The vertical alignment of the group text when the Layout property is set to include tglBottom. |
| GroupsAppearance → FillEmptySpaces | Allows filling empty spaces at the right side of the columns when the StretchScrollBars property is set to False. |
| GroupsAppearance → Layouts | The layout of the groups which include tglTop and tglBottom. |
| GroupsAppearance → TopFill | The fill of the group when the Layout property is set to include tglTop. |
| GroupsAppearance → TopFont | The font of the group text when the Layout property is set to include tglTop. |
| GroupsAppearance → TopHorizontalTextAlign | The horizontal alignment of the group text when the Layout property is set to include tglTop. |
| GroupsAppearance → TopSize | The size of the top columns. |
| GroupsAppearance → TopStroke | The stroke of the column when the Layout property is set to include |

| | |
|---|--|
| | tglTop. |
| GroupsAppearance → TopVerticalText | Allows displaying vertical text in the columns top layout. |
| GroupsAppearance → TopVerticalTextAlign | The vertical alignment of the group text when the Layout property is set to include tglTop. |
| Interaction | Set of properties for configuring mouse and keyboard interaction. |
| Interaction → ClipboardMode | Sets the mode for clipboard support. |
| Interaction → ColumnAutoSizeOnDblClick | Allows auto sizing of a column on double-click. Please note that this will only apply auto sizing on the visible nodes. |
| Interaction → ColumnSizing | Allows for column sizing. |
| Interaction → DragDropMode | When true, the treeview supports drag & drop of nodes. |
| Interaction → ExtendedEditable | Allows extended nodes to be editable. |
| Interaction → ExtendedSelectable | Allows extended nodes to be selectable. |
| Interaction → KeyboardEdit | Allows keyboard editing when editing is supported. |
| Interaction → Lookup | When true, the treeview supports keyboard lookup. |
| Interaction → MouseEditMode | Sets the mouse edit mode when editing is supported. |
| Interaction → MultiSelect | Allows for multiple node selection with mouse and keyboard. |
| Interaction → ReadOnly | Sets the TreeView in readonly mode, which disables node editing on all columns. |
| Interaction → Reorder | When true, the treeview supports reordering of nodes. |
| Interaction → TouchScrolling | Allows/disallows touch scrolling. When True, scrolling can be done by flicking the mouse (finger) up / down on the TreeView. |
| Nodes | The nodes collection when a collection-based TreeView is being used. |
| Nodes[Index] → Enabled | When False, disables editing and |

| | |
|---|--|
| | selection. |
| Nodes[Index] → Expanded | When True and the node has children, expands the child nodes. When False, collapses the child nodes. |
| Nodes[Index] → Extended | When True, applies the extended properties under NodesAppearance and only uses and stretches the first column text over the number of columns. |
| Nodes[Index] → Nodes | The child nodes collection. |
| Nodes[Index] → Values | The values collection that is represented in a column for each node. |
| Nodes[Index] → Values[Index] → Checked | Sets whether the node value for a specific column is checked. |
| Nodes[Index] → Values[Index] → CheckType | Specifies the check type of a node value. The type can be a radiobutton or a checkbox. |
| Nodes[Index] → Values[Index] → CollapsedIcon | The icon in collapsed state. |
| Nodes[Index] → Values[Index] → CollapsedIconLarge | The icon in collapsed state for high DPI / retina screens. |
| Nodes[Index] → Values[Index] → CollapsedIconLargeName | The icon name linked to a PictureContainer in collapsed state for high DPI / retina screens. |
| Nodes[Index] → Values[Index] → CollapsedIconName | The icon name linked to a PictureContainer in collapsed state. |
| Nodes[Index] → Values[Index] → ExpandedIcon | The icon in expanded state. |
| Nodes[Index] → Values[Index] → ExpandedIconLarge | The icon in expanded state for high DPI / retina screens. |
| Nodes[Index] → Values[Index] → ExpandedIconLargeName | The icon name linked to a PictureContainer in expanded state for high DPI / retina screens. |
| Nodes[Index] → Values[Index] → ExpandedIconName | The icon name linked to a PictureContainer in expanded state. |
| Nodes[Index] → Values[Index] → Text | The Text of a node. |
| NodesAppearance | The appearance for each node. |
| NodesAppearance → CollapseNodeIcon | The icon for the ExpandColumn in collapsed state. |
| NodesAppearance → CollapseNodeIconLarge | The icon for the ExpandColumn in collapsed state for high DPI / retina |

| | |
|--|---|
| | screens. |
| NodesAppearance → ColumnStroke | The stroke between columns. |
| NodesAppearance → DisabledFill | The fill of a node in disabled state. |
| NodesAppearance → DisabledStroke | The stroke of a node in disabled state. |
| NodesAppearance → ExpandColumn | The column that shows the expand / collapse node icons and is used to expand / collapse the nodes. |
| NodesAppearance → ExpandHeight | The height of the expand / collapse node icon area. |
| NodesAppearance → ExpandNodeIcon | The icon for the ExpandColumn in expanded state. |
| NodesAppearance → ExpandNodeIconLarge | The icon for the ExpandColumn in expanded state. |
| NodesAppearance → ExpandWidth | The width of the expand / collapse node icon area. |
| NodesAppearance → ExtendedDisabledFill | The fill of an extended node in disabled state. |
| NodesAppearance → ExtendedDisabledStroke | The stroke of an extended node in disabled state. |
| NodesAppearance → ExtendedFill | The fill of an extended node. |
| NodesAppearance → ExtendedFont | The font of an extended node. |
| NodesAppearance → ExtendedSelectedFill | The fill of an extended node in selected state. |
| NodesAppearance → ExtendedSelectedStroke | The stroke of an extended node in selected state. |
| NodesAppearance → ExtendedStroke | The stroke of an extended node. |
| NodesAppearance → Fill | The fill of a node in normal state. |
| NodesAppearance → FixedHeight | The height of each node in case the HeightMode property is set to tnhmFixed. |
| NodesAppearance → Font | The font of a node. |
| NodesAppearance → HeightMode | The HeightMode of the nodes. In case the HeightMode property is set to tnhmFixed, the FixedHeight property is used to determine a fixed height for each node. When the HeightMode property is set to tnhmVariable, the minimum height of a node is 25 and depending on the text calculation and properties such as wordwrapping / |

| | |
|--------------------------------------|---|
| | trimming and alignment the treeview automatically calculates the real node height on the fly. Mode information can be found in the chapter Fixed vs variable node height under Nodes. |
| NodesAppearance → LevelIndent | The size of the indenting used for different node levels (child nodes). |
| NodesAppearance → LineStroke | The stroke of the line used when ShowLines is true. |
| NodesAppearance → SelectedFill | The fill of a node in selected state. |
| NodesAppearance → SelectedFont.Color | The color of the font of a node in selected state. |
| NodesAppearance → SelectedStroke | The stroke of a node in selected state. |
| NodesAppearance → SelectionArea | The area of selection indication. The selection area can be limited to the text only, include the icon and level indenting as well. |
| NodesAppearance → ShowFocus | Shows a focus border on the focused node. |
| NodesAppearance → ShowLines | Shows node and child node lines. |
| NodesAppearance → Stroke | The stroke of a node in normal state. |
| PictureContainer | Support for adding icons to nodes and to support image tags inside HTML formatted text. |
| StretchScrollBars | Allows stretching of scrollbars to enable a more integrated look and feel. |

Public Properties

TreeView

| | |
|---|--|
| FocusedNode: TAdvTreeViewNode | Returns the focused node (collection-based). |
| FocusedVirtualNode: TAdvTreeViewVirtualNode | Returns the focused node (virtual). |
| SelectedNodes[AIndex: Integer]: TAdvTreeViewNode | Gives access to the selected nodes based on the SelectedNodeCount property (collection-based). |
| SelectedVirtualNodes[AIndex: Integer]: TAdvTreeViewVirtualNode | Gives access to the selected nodes based on the SelectedVirtualNodeCount property (virtual). |

Node (Virtual)

| | |
|---------------|--|
| BitmapRects | An array of rectangles for node icons for each column. |
| Calculated | When the HeightMode is tnhmVariable, this property is set to true whenever a node height is calculated. The Height property contains the height of the node. |
| CheckRects | An array of rectangles for node check types for each column. |
| CheckStates | An array of Booleans for node check states for each column. |
| Children | The count of children of a node. |
| Expanded | Determines if the node is expanded / collapsed. |
| ExpandRects | An array of rectangles for expand / collapse node icons for each column. |
| Extended | Determines if the node is extended / normal. |
| Height | The height of the node. |
| Index | The index of the node relative to its parent. |
| Level | The level of the node. |
| Node | A reference to the collection-based node if a collection-based TreeView is used. |
| ParentNode | The row index of the parent node. |
| Row | The index of the node relative to the TreeView. |
| TextRects | An array of rectangles for the text of each column. |
| TotalChildren | The total count of children of a node. The total count includes the count of all levels of child nodes. |

Important notice: When using one of the array properties, the length of the array will always be the same as the column count, yet the values that are included will only be valid if the width & height are larger than 0. When using one of those array properties for custom drawing keep in mind that drawing is only valid when the above criteria is met.

Node (Collection-Based)

| | |
|---|--|
| Checked[AColumn: Integer]: Boolean | Property to set the checked state of a node for a specific column. |
| CheckTypes[AColumn: Integer]: TAdvTreeViewNodeCheckType | Property to set the check type of a node for a specific column. The check type of a node can |

| | |
|--|--|
| | be a radiobutton or a checkbox. |
| CollapsedIconNames[AColumn: Integer ALarge: Boolean]: String | The name of the icon in collapsed state of a node for a specific column. |
| CollapsedIcons[AColumn: Integer ALarge: Boolean]: TAdvTreeViewBitmap | The icon in collapsed state of a node for a specific column. |
| ExpandedIconNames[AColumn: Integer ALarge: Boolean]: String | The name of the icon in expanded state of a node for a specific column. |
| ExpandedIcons[AColumn: Integer ALarge: Boolean]: TAdvTreeViewBitmap | The icon in expanded state of a node for a specific column. |
| Text[AColumn: Integer]: String | The text of a node for a specific column. |
| VirtualNode: TAdvTreeViewVirtualNode | A reference to the virtual node. |

Events

Note that for each event, the TAdvTreeViewVirtualNode is being passed as a parameter. This class is used in virtual mode and in collection-based mode but has a property Node to easily access the collection item in case a collection-based TreeView is used.

| | |
|-----------------------------|--|
| OnAfterCheckNode | Event called after a node check state is changed. |
| OnAfterCollapseNode | Event called after a node is collapsed. |
| OnAfterCopyToClipboard | Event called after a copy operation is performed on the clipboard. |
| OnAfterCutToClipboard | Event called after a cut operation is performed on the clipboard. |
| OnAfterDrawColumn | Event called after a column is drawn. |
| OnAfterDrawColumnEmptySpace | Event called after an empty space next to the columns area is drawn. |
| OnAfterDrawColumnHeader | Event called after the header area of a column is drawn. |
| OnAfterDrawColumnText | Event called after the text of a column is drawn. |
| OnAfterDrawGroup | Event called after the group is drawn. |
| OnAfterDrawGroupEmptySpace | Event called after the empty space next to the groups area is drawn. |
| OnAfterDrawGroupText | Event called after the group text is drawn. |
| OnAfterDrawNode | Event called after a node is drawn. |
| OnAfterDrawNodeCheck | Event called after the check area of a node is drawn. |
| OnAfterDrawNodeColumn | Event called after the column area of the nodes is drawn. |
| OnAfterDrawNodeExpand | Event called after the expand / collapse area of a node is drawn. |
| OnAfterDrawNodeIcon | Event called after the icon of a node is drawn. |
| OnAfterDrawNodeText | Event called after the text of a node is drawn. |
| OnAfterDropNode | Event called after a node is dropped. |
| OnAfterExpandNode | Event called after a node is expanded. |
| OnAfterOpenInplaceEditor | Event called after the inplace editor is opened. |
| OnAfterPasteFromClipboard | Event called after a paste operation is performed on the clipboard. |
| OnAfterReorderNode | Event called after a node is reordered. |
| OnAfterSelectNode | Event called after a node is selected. |
| OnAfterSizeColumn | Event called after a column is sized. |

| | |
|------------------------------|--|
| OnAfterUnCheckNode | Event called after a node is UnChecked. |
| OnAfterUnSelectNode | Event called after a node is UnSelected. |
| OnAfterUpdateNode | Event called after a node is updated after editing. |
| OnBeforeCheckNode | Event called before a node check state is changed. |
| OnBeforeCollapseNode | Event called before a node is collapsed. |
| OnBeforeCopyToClipboard | Event called before a copy operation is performed on the clipboard. |
| OnBeforeCutToClipboard | Event called before a cut operation is performed on the clipboard. |
| OnBeforeDrawColumn | Event called before a column is drawn. |
| OnBeforeDrawColumnEmptySpace | Event called before an empty space next to the columns area is drawn. |
| OnBeforeDrawColumnHeader | Event called before the header area of a column is drawn. |
| OnBeforeDrawColumnText | Event called before the text of a column is drawn. |
| OnBeforeDrawGroup | Event called before the group is drawn. |
| OnBeforeDrawGroupEmptySpace | Event called before the empty space next to the groups area is drawn. |
| OnBeforeDrawGroupText | Event called before the group text is drawn. |
| OnBeforeDrawNode | Event called before a node is drawn. |
| OnBeforeDrawNodeCheck | Event called before the check area of a node is drawn. |
| OnBeforeDrawNodeColumn | Event called before the column area of the nodes is drawn. |
| OnBeforeDrawNodeExpand | Event called before the expand / collapse area of a node is drawn. |
| OnBeforeDrawNodeIcon | Event called before the icon of a node is drawn. |
| OnBeforeDrawNodeText | Event called before the text of a node is drawn. |
| OnBeforeDropNode | Event called before a node will be dropped. |
| OnBeforeExpandNode | Event called before a node is expanded. |
| OnBeforeOpenInplaceEditor | Event called before the inplace editor is opened. |
| OnBeforePasteFromClipboard | Event called before a paste operation is performed from the clipboard. |
| OnBeforeReorderNode | Event called before reordering a node. |
| OnBeforeSelectNode | Event called before a node is selected. |
| OnBeforeSizeColumn | Event called before a column is sized. |
| OnBeforeUnCheckNode | Event called before a node is UnChecked. |

| | |
|--------------------------------|---|
| OnBeforeUnselectNode | Event called before a node is UnSelected. |
| OnBeforeUpdateNode | Event called before a node is updated after editing. |
| OnCloseInplaceEditor | Event called when the inplace editor is closed. |
| OnCustomizeInplaceEditor | Event for customization of the inplace editor after it has been created. |
| OnFilterSelect | Event triggered when a value of the filter listbox is selected and the condition needs to be applied. In this event you can additionally customize the condition. |
| OnGetColumnHorizontalTextAlign | Event called to get the horizontal alignment of the text in a column. |
| OnGetColumnText | Event called to get the text of a column. |
| OnGetColumnTrimming | Event called to get the trimming of the text in a column. |
| OnGetColumnVerticalTextAlign | Event called to get the vertical alignment of the text in a column. |
| OnGetColumnWordWrapping | Event called to get the word wrapping of the text in a column. |
| OnGetGroupText | Event called to get the text for a specific group. |
| OnGetInplaceEditor | Event called to use a custom inplace editor. |
| OnGetInplaceEditorRect | Event called to get the inplace editor rectangle. |
| OnGetNodeCheckType | Event called to get the check type of a node. |
| OnGetNodeColor | Event called to get the color of a node in normal state. |
| OnGetNodeDisabledColor | Event called to get the color of a node in disabled state. |
| OnGetNodeDisabledTextColor | Event called to get the color of the text of a node in disabled state. |
| OnGetNodeHeight | Event called to get the height of a node in case the NodesAppearance.HeightMode is set to tnhmVariable. |
| OnGetNodeHorizontalTextAlign | Event called to get the horizontal text alignment of a node. |
| OnGetNodeIcon | Event called to get the icon of a node. |
| OnGetNodeSelectedColor | Event called to get the color of a node in selected state. |
| OnGetNodeSelectedTextColor | Event called to get the text color of a node in selected state. |
| OnGetNodeText | Event called to get the text of a node. |
| OnGetNodeTextColor | Event called to get the color of the text of a node. |
| OnGetNodeTrimming | Event called to get the trimming of the text of |

| | |
|----------------------------|--|
| | a node. |
| OnGetNodeVerticalTextAlign | Event called to get the vertical text alignment of a node. |
| OnGetNodeWordWrapping | Event called to get the word wrapping of the text of a node. |
| OnGetNumberOfNodes | Event called to get the number of nodes. |
| OnHScroll | Event called when the TreeView scrolls horizontally. |
| OnIsNodeChecked | Event called to determine if a node is checked or not. |
| OnIsNodeEnabled | Event called to determine if a node is enabled or not. |
| OnIsNodeExpanded | Event called to determine if a node is expanded or not. |
| OnIsNodeExtended | Event called to determine if a node is extended or not. |
| OnNeedFilterDropDownData | Event triggered when applying a column filter operation. In this event you can additionally change or add values you wish to see in the dropdown window. |
| OnNodeAnchorClick | Event called when an anchor is clicked in the HTML formatted text of a node. |
| OnNodeChanged | Event called when the node is changed after editing. |
| OnNodeClick | Event called when a node is clicked. |
| OnNodeDbClick | Event called when a node is double clicked. |
| OnVScroll | Event called when the TreeView scrolls vertically. |

Procedures and functions

TreeView

| | |
|---|---|
| AddNode(AParentNode: TAdvTreeNode = nil): TAdvTreeNode | Adds a node to the node collection (collection-based). An optional parent node parameter can be passed to add the node as a child node. |
| AddVirtualNode(AParentNode: TAdvTreeViewVirtualNode = nil): TAdvTreeViewVirtualNode | Adds a node to the virtual node list (virtual). An optional parent node parameter can be passed to add the node as a child node. |
| AutoSizeColumn(ACol: Integer) | Autosizes a column. |
| BeginUpdate | Blocks all updates to increase performance. Must be paired with an EndUpdate. |

| | |
|---|---|
| CancelEditing | Cancels editing when editing is active. |
| CheckNode(ANode: TAdvTreeNode; AColumn: Integer; ARecurse: Boolean = False) | Checks the node for a specific column (collection-based). |
| CheckVirtualNode(ANode: TAdvTreeViewVirtualNode; AColumn: Integer; ARecurse: Boolean = False) | Checks the node for a specific column (virtual). |
| ClearColumns | Removes all columns. |
| ClearNodeList | Clears the internal node list. (virtual) |
| ClearNodes | Removes all nodes from the node collection (collection-based). |
| CollapseAll | Collapses all nodes and child nodes (collection-based). |
| CollapseAllVirtual | Collapsed all nodes and child nodes (virtual). |
| CollapseNode(ANode: TAdvTreeNode; ARecurse: Boolean = False) | Collapses a specific node (collection-based). |
| CollapseVirtualNode(ANode: TAdvTreeViewVirtualNode; ARecurse: Boolean = False) | Collapse a specific node (virtual). |
| EditNode(ANode: TAdvTreeNode; AColumn: Integer) | Starts editing a specific node (collection-based). |
| EditVirtualNode(ANode: TAdvTreeViewVirtualNode; AColumn: Integer) | Starts editing a specific node (virtual). |
| EndUpdate | Bundles all updates into one update for performance. Needs to be paired with a BeginUpdate. |
| ExpandAll | Expands all nodes (collection-based). |
| ExpandAllVirtual | Expands all nodes (virtual). |
| ExpandNode(ANode: TAdvTreeNode; ARecurse: Boolean = False) | Expands a specific node (collection-based). |
| ExpandVirtualNode(ANode: TAdvTreeViewVirtualNode; ARecurse: Boolean = False) | Expands a specific node (virtual). |
| FindColumnName(AName: String): TAdvTreeViewColumn | Finds a column with a specific name. |
| FindColumnIndexByName(AName: String): Integer | Finds a column index with a specific name. |
| FindGroupName(AName: String): TAdvTreeViewGroup | Finds a group with a specific name. |
| FindGroupIndexByName(AName: String): Integer | Finds a group index with a specific name. |
| GetFirstChildNode(ANode: TAdvTreeNode) | Returns the first child node of a node |

| | |
|---|--|
| TAdvTreeNode): TAdvTreeNode | (collection-based). |
| GetFirstChildVirtualNode(ANode: TAdvTreeViewVirtualNode): TAdvTreeViewVirtualNode | Returns the first child node of a node (virtual). |
| GetFirstRootNode: TAdvTreeNode | Returns the first root node (collection-based). |
| GetFirstRootVirtualNode: TAdvTreeViewVirtualNode | Returns the first root node (virtual). |
| GetHorizontalScrollPosition: Double | Returns the horizontal scroll position. |
| GetInplaceEditor: TAdvTreeViewInplaceEditor | Returns the inplace editor when active. The GetInplaceEditor function will return nil when the editor is not active. |
| GetLastChildNode(ANode: TAdvTreeNode): TAdvTreeNode | Returns the last child node of a node (collection-based). |
| GetLastChildVirtualNode(ANode: TAdvTreeViewVirtualNode): TAdvTreeViewVirtualNode | Returns the last child node of a node (virtual). |
| GetLastNode: TAdvTreeNode | Returns the last node (collection-based). |
| GetLastRootNode: TAdvTreeNode | Returns the last root node (collection-based). |
| GetLastRootVirtualNode: TAdvTreeViewVirtualNode | Returns the last root node (virtual). |
| GetLastVirtualNode: TAdvTreeViewVirtualNode | Returns the last node (virtual). |
| GetNextChildNode(ANode: TAdvTreeNode; AStartNode: TAdvTreeNode): TAdvTreeNode | Returns the next child node starting from a parent node and the previous node (collection-based). |
| GetNextChildVirtualNode(ANode: TAdvTreeViewVirtualNode; AStartNode: TAdvTreeViewVirtualNode): TAdvTreeViewVirtualNode | Returns the next child node starting from a parent node and the previous node (virtual). |
| GetNextNode(ANode: TAdvTreeNode): TAdvTreeNode | Returns the next node starting from a node (collection-based). |
| GetNextSiblingNode(ANode: TAdvTreeNode): TAdvTreeNode | Returns the next sibling node starting from a node (collection-based). |
| GetNextSiblingVirtualNode(ANode: TAdvTreeViewVirtualNode): | Returns the next sibling node starting from a node (virtual). |

| | |
|---|---|
| TAdvTreeViewVirtualNode | |
| GetNextVirtualNode(ANode: TAdvTreeViewVirtualNode): TAdvTreeViewVirtualNode | Returns the next node starting from the previous node (virtual). |
| GetNodeChildCount(ANode: TAdvTreeViewNode): Integer | Returns the child count for a specific node (collection-based). |
| GetParentNode(ANode: TAdvTreeViewNode): TAdvTreeViewNode | Returns the parent node for a specific node (collection-based). |
| GetParentVirtualNode(ANode: TAdvTreeViewVirtualNode): TAdvTreeViewVirtualNode | Returns the parent node for a specific node (virtual). |
| GetPreviousChildNode(ANode: TAdvTreeViewNode; AStartNode: TAdvTreeViewNode): TAdvTreeViewNode | Returns the previous child node starting from a parent node and the previous node (collection-based). |
| GetPreviousChildVirtualNode(ANode: TAdvTreeViewVirtualNode; AStartNode: TAdvTreeViewVirtualNode): TAdvTreeViewVirtualNode | Returns the previous child node starting from a parent node and the previous node (virtual). |
| GetPreviousNode(ANode: TAdvTreeViewNode): TAdvTreeViewNode | Returns the previous node starting from a node (collection-based). |
| GetPreviousSiblingNode(ANode: TAdvTreeViewNode): TAdvTreeViewNode | Returns the previous sibling starting from a node (collection-based). |
| GetPreviousSiblingVirtualNode(ANode: TAdvTreeViewVirtualNode): TAdvTreeViewVirtualNode | Returns the previous sibling starting from a node (virtual). |
| GetPreviousVirtualNode(ANode: TAdvTreeViewVirtualNode): TAdvTreeViewVirtualNode | Returns the previous node starting from a node (virtual). |
| GetRootNodeByIndex(AIndex: Integer): TAdvTreeViewNode | Returns a root node by a specific index (collection-based). |
| GetRootVirtualNodeByIndex(AIndex: Integer): TAdvTreeViewVirtualNode | Returns a root node by a specific index (virtual). |
| GetTotalColumnWidth: Double | Returns the total column width. |
| GetTotalRowHeight: Double | Returns the total row height. |
| GetVerticalScrollPosition: Double | Returns the vertical scroll position. |

| | |
|---|---|
| GetVirtualNodeChildCount(ANode: TAdvTreeViewVirtualNode): Integer | Returns the child count of a specific node (virtual). |
| HorizontalScrollBar: TScrollBar | Returns the horizontal scrollbar. |
| InitSample | Initializes a sample (the same sample initialized at designtime when dropping a new instance of TAdvTreeView). |
| InsertNode(AIndex: Integer; AParentNode: TAdvTreeViewNode = nil): TAdvTreeViewNode | Inserts a new node on a specific index and parent node (collection based). |
| InsertVirtualNode(AIndex: Integer; AParentNode: TAdvTreeViewVirtualNode = nil): TAdvTreeViewVirtualNode | Inserts a new node on a specific index and parent node (virtual). |
| IsColumnVisible(ACol: Integer): Boolean | Returns if the specified column is visible or hidden. |
| IsEditing: Boolean | Returns if editing is active. |
| IsNodeSelectable(ANode: TAdvTreeViewNode): Boolean | Returns if a node is selectable (collection-based). |
| IsNodeSelected(ANode: TAdvTreeViewNode): Boolean | Returns if a node is selected (collection-based). |
| IsVirtualNodeSelectable(ANode: TAdvTreeViewVirtualNode): Boolean | Returns if a node is selectable (virtual). |
| IsVirtualNodeSelected(ANode: TAdvTreeViewVirtualNode): Boolean | Returns if a node is selected (virtual). |
| RemoveNodeChildren(ANode: TAdvTreeViewNode) | Removes all children of a specific node (collection-based). |
| RemoveSelectedNodes | Removes all selected nodes (collection-based). |
| RemoveSelectedVirtualNodes | Removes all selected nodes (virtual). |
| RemoveVirtualNode(ANode: TAdvTreeViewVirtualNode) | Removes a node (virtual). |
| RemoveVirtualNodeChildren(ANode: TAdvTreeViewVirtualNode) | Removes all children of a specific node (virtual). |
| RestoreScrollPosition | Restores scroll position after it has been saved with SaveScrollPosition. |
| SaveScrollPosition | Saves the scroll position. Restoring the scroll position is done with RestoreScrollPosition. |
| ScrollToNode(ANode: TAdvTreeViewNode; AScrollIfNotVisible: Boolean = False; AScrollPosition: TAdvTreeViewNodeScrollPosition = tvnspTop) | Scrolls to a specific node. Additional parameters can be passed to scroll only if not visible, and the scroll position when the node is found (collection-based). |
| ScrollToVirtualNode(ANode: TAdvTreeViewVirtualNode; | Scrolls to a specific node. Additional parameters can be passed to scroll only if not |

| | |
|---|---|
| AScrollIfNotVisible: Boolean = False; AScrollPosition: TAdvTreeViewNodeScrollPosition = tvnspTop) | visible, and the scroll position when the node is found (virtual). |
| SelectAllNodes | Selects all nodes (collection-based). |
| SelectAllVirtualNodes | Selects all nodes (virtual). |
| SelectedNodeCount: Integer | Selected node count (collection-based). |
| SelectedVirtualNodeCount: Integer | Selected node count (virtual). |
| SelectNode(ANode: TAdvTreeViewNode) | Selects a specific node (collection-based). |
| SelectNodes(ANodes: TAdvTreeViewNodeArray) | Selects an array of nodes (collection-based). |
| SelectVirtualNode(ANode: TAdvTreeViewVirtualNode) | Selects a specific node (virtual). |
| SelectVirtualNodes(ANodes: TAdvTreeViewVirtualNodeArray) | Selects an array of nodes (virtual). |
| StopEditing | Stops editing. |
| ToggleCheckNode(ANode: TAdvTreeViewNode; AColumn: Integer; ARecurse: Boolean = False) | Toggles the state of a checkbox or radiobutton when used in a node column (collection-based). |
| ToggleCheckVirtualNode(ANode: TAdvTreeViewVirtualNode; AColumn: Integer; ARecurse: Boolean = False) | Toggles the state of a checkbox or radiobutton when used in a node column (virtual). |
| ToggleVirtualNode(ANode: TAdvTreeViewVirtualNode; ARecurse: Boolean = False) | Toggles the expand/collapse state of a node (virtual). |
| UnCheckNode(ANode: TAdvTreeViewNode; AColumn: Integer; ARecurse: Boolean = False) | Unchecks a node (collection-based). |
| UnCheckVirtualNode(ANode: TAdvTreeViewVirtualNode; AColumn: Integer; ARecurse: Boolean = False) | Unchecks a node (virtual). |
| UnSelectAllNodes | Unselects all nodes (collection-based). |
| UnSelectAllVirtualNodes | Unselects all nodes (virtual). |
| UnSelectNode(ANode: TAdvTreeViewNode) | Unselects a specific node (collection-based). |
| UnSelectNodes(ANodes: TAdvTreeViewNodeArray) | Unselects an array of nodes (collection-based). |
| UnSelectVirtualNode(ANode: TAdvTreeViewVirtualNode) | Unselects a specific node (virtual). |
| UnSelectVirtualNodes(ANodes: TAdvTreeViewVirtualNodeArray) | Unselects an array of nodes (virtual). |
| VerticalScrollBar: TScrollBar | Returns the vertical scrollbar. |
| XYToColumnSize(X, Y: Single): Integer | Returns a column index at a specific X and Y |

| | |
|--|---|
| | coordinate. |
| XYToNode(X, Y: Double): TAdvTreeViewVirtualNode | Returns a node at a specific X and Y coordinate (virtual). |
| XYToNodeAnchor(ANode: TAdvTreeViewVirtualNode; X, Y: Single): TAdvTreeViewNodeAnchor | Returns a node anchor at a specific X and Y coordinate. |
| XYToNodeCheck(ANode: TAdvTreeViewVirtualNode; X, Y: Single): TAdvTreeViewNodeCheck | Returns a node checkbox or radiobutton area at a specific X and Y coordinate. |
| XYToNodeExpand(ANode: TAdvTreeViewVirtualNode; X, Y: Single): Boolean | Returns a node expand / collapse area at a specific X and Y coordinate. |
| XYToNodeTextColumn(ANode: TAdvTreeViewVirtualNode; X, Y: Single): Integer | Returns the column of the text of a specific node at a specific X and Y coordinate. |

Node (Virtual)

| | |
|---|------------------------------------|
| Collapse(ARecurse: Boolean = False) | Collapses the child nodes. |
| Expand(ARecurse: Boolean = False) | Expands the child nodes. |
| GetChildCount: Integer | Returns the count of child nodes. |
| GetFirstChild: TAdvTreeViewVirtualNode | Returns the first child node. |
| GetLastChild: TAdvTreeViewVirtualNode | Returns the last child node. |
| GetNext: TAdvTreeViewVirtualNode | Returns the next node. |
| GetNextChild(ANode: TAdvTreeViewVirtualNode): TAdvTreeViewVirtualNode | Returns the next child node. |
| GetNextSibling: TAdvTreeViewVirtualNode | Returns the next sibling node. |
| GetParent: TAdvTreeViewVirtualNode | Returns the parent node. |
| GetPrevious: TAdvTreeViewVirtualNode | Returns the previous node. |
| GetPreviousChild(ANode: TAdvTreeViewVirtualNode): TAdvTreeViewVirtualNode | Returns the previous child node. |
| GetPreviousSibling: TAdvTreeViewVirtualNode | Returns the previous sibling node. |
| RemoveChildren | Removes all children. |

Node (Collection-Based)

| | |
|-------------------------------------|----------------------------|
| Collapse(ARecurse: Boolean = False) | Collapses the child nodes. |
|-------------------------------------|----------------------------|

| | |
|---|------------------------------------|
| Expand(ARecurse: Boolean = False) | Expands the child nodes. |
| GetChildCount: Integer | Returns the count of child nodes. |
| GetFirstChild: TAdvTreeViewNode | Returns the first child node. |
| GetLastChild: TAdvTreeViewNode | Returns the last child node. |
| GetNext: TAdvTreeViewNode | Returns the next node. |
| GetNextChild(ANode: TAdvTreeViewNode): TAdvTreeViewNode | Returns the next child node. |
| GetNextSibling: TAdvTreeViewNode | Returns the next sibling node. |
| GetParent: TAdvTreeViewNode | Returns the parent node. |
| GetPrevious: TAdvTreeViewNode | Returns the previous node. |
| GetPreviousChild(ANode: TAdvTreeViewNode): TAdvTreeViewNode | Returns the previous child node. |
| GetPreviousSibling: TAdvTreeViewNode | Returns the previous sibling node. |
| RemoveChildren | Removes all children. |

TAdvDirectoryTreeView and TAdvCheckedTreeView

The TAdvDirectoryTreeView and the TAdvCheckedTreeView both inherit from TAdvTreeView and add additional functionality.

TAdvDirectoryTreeView

The TAdvDirectoryTreeView is capable of displaying drives, folders and files. There are three important methods to load this information: LoadDrives, LoadDrive and LoadDirectory. Additionally a filter can be applied for further fine-tuning.

By default there is only one column added which is the name of the file / folder or drive. When more information is needed, there are additional columns supported such as the creation date, modification date and the free space, total space in case of a drive. These columns can be added with the AddColumn function. The directory demo that is included in the distribution demonstrates this component.

TAdvCheckedTreeView

The TAdvCheckedTreeView adds a checkbox for each node by default. The behaviour is identical to the TAdvTreeView but saves the code for adding a checkbox to each node.

TMS Mini HTML rendering engine

Another core technology used among many components is a small fast & lightweight HTML rendering engine. This engine implements a childset of the HTML standard to display formatted text. It supports following tags :

B : Bold tag

 : start bold text

 : end bold text

Example : This is a test

U : Underline tag

<U> : start underlined text

</U> : end underlined text

Example : This is a <U>test</U>

I : Italic tag

<I> : start italic text

</I> : end italic text

Example : This is a <I>test</I>

S : Strikeout tag

<S> : start strike-through text

</S> : end strike-through text

Example : This is a <S>test</S>

A : anchor tag

 : text after tag is an anchor. The 'value' after the href identifier is the anchor. This can be an URL (with ftp,http,mailto,file identifier) or any text.

If the value is an URL, the shellexecute function is called, otherwise, the anchor value can be found in the OnAnchorClick event : end of anchor

Examples : This is a test

This is a test

This is a test

FONT : font specifier tag

 : specifies font of text after tag.

with

- face : name of the font
- size : HTML style size if smaller than 5, otherwise pointsize of the font
- color : font color with either hexadecimal color specification or color constant name, ie clRed,clYellow,clWhite ... etc
- bgcolor : background color with either hexadecimal color specification or color constant name : ends font setting

Examples : This is a test

This is a test

P : paragraph

<P align="alignvalue" [bgcolor="colorvalue"] [bgcolorto="colorvalue"]> : starts a new paragraph, with left, right or center alignment. The paragraph background color is set by the optional bgcolor parameter. If bgcolor and bgcolorto are specified,

a gradient is displayed ranging from begin to end color.
</P> : end of paragraph

Example : <P align="right">This is a test</P>

Example : <P align="center">This is a test</P>

Example : <P align="left" bgcolor="#ff0000">This has a red background</P>

Example : <P align="right" bgcolor="clYellow">This has a yellow background</P>

Example : <P align="right" bgcolor="clYellow" bgcolorto="clred">This has a gradient background</P>*

HR : horizontal line

<HR> : inserts linebreak with horizontal line

BR : linebreak

 : inserts a linebreak

BODY : body color / background specifier

<BODY bgcolor="colorvalue" [bgcolorto="colorvalue"] [dir="v|h"] background="imagefile specifier"> : sets the background color of the HTML text or the background bitmap file

Example : <BODY bgcolor="clYellow"> : sets background color to yellow

<BODY background="file://c:\test.bmp"> : sets tiled background to file test.bmp

<BODY bgcolor="clYellow" bgcolorto="clWhite" dir="v"> : sets a vertical gradient from yellow to white

IND : indent tag

This is not part of the standard HTML tags but can be used to easily create multicolumn text
<IND x="indent"> : indents with "indent" pixels

Example :

This will be <IND x="75">indented 75 pixels.

IMG : image tag

 : inserts an image at the location

specifier can be: name of image in a BitmapContainer

Optionally, an alignment tag can be included. If no alignment is included, the text alignment with respect to the image is bottom. Other possibilities are: align="top" and align="middle"

The width & height to render the image can be specified as well. If the image is embedded in anchor tags, a different image can be displayed when the mouse is in the image area through the Alt attribute.

Examples :

This is an image

CHILD : childscript tag

<CHILD> : start childscript text

</CHILD> : end childscript text

Example : This is ⁹/<CHILD>16</CHILD> looks like 9/16

SUP : superscript tag

<SUP> : start superscript text

</SUP> : end superscript text

UL : list tag

 : start unordered list tag

 : end unordered list

Example :

List item 1

List item 2

 Child list item A

 Child list item B

List item 3

LI : list item

<LI [type="specifier"] [color="color"] [name="imagenam"]>: new list item specifier can be "square", "circle" or "image" bullet. Color sets the color of the square or circle bullet.

Imagenam sets the PictureContainer image name for image to use as bullet

SHAD : text with shadow

<SHAD> : start text with shadow

</SHAD> : end text with shadow

Z : hidden text

<Z> : start hidden text

</Z> : end hidden text

Special characters

Following standard HTML special characters are supported :

< : less than : <

> : greater than : >

& : &

" : "

 : non breaking space

™ : trademark symbol

€ : euro symbol

§ : section symbol

© : copyright symbol

¶ : paragraph symbol