Table of Contents

Index 0

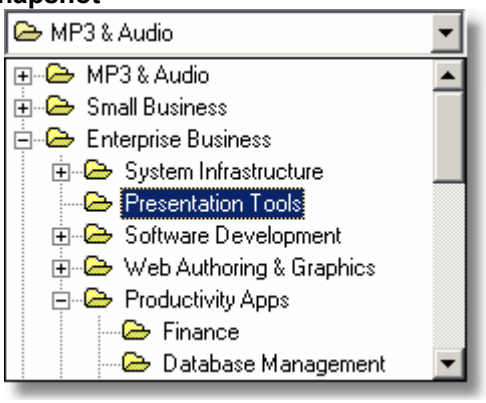
1 TreeComboBox - Overview

Overview

The TreeComboBox is the combo-box with the TreeView embedded to its drop-down window. So, instead of searching and selecting the necessary list item from huge list of drop-down window, user will be able to see an elegant tree-like structure, select the tree nodes, expand and collapse them, export and import the nodes to another TreeView controls and so on.

Like in standard TreeView, the TreeComboBox can display images for each tree node, show image in editor, accept the selection either on one or double click, contains the label attached to control, customizeable image for button + many other neat features, result of combination of ComboBox and TreeView.

Snapshot



How to use?

Drop tcTreeComboBox control onto your form and specify the tree nodes to Items property + if the list items should contain an icons — specify the image list to Images property. In general, the TreeComboBox are ready for work… But let's change some its behaviours… For example we want the tree nodes to be selected by double click intead of one, and don't like that selection hovers the nodes under mouse. Then just set AcceptOnDblClick property to True (the nodes will be selected on double click), and set TreeOptions.MouseTrack property to False (mouse will not track the selection in drop-down view).

To change the selected node at run- or design-time — modify ItemIndex property (this is the absolute index of selected node).

To change the width and height of the drop-down window — set DropDownCount and DropDownWidth properties. To let user to enter or modify the text using keyboard — set ShowEditor property to True.

The EmptyItemText and EmptyItemImageIndex properties can be used to specify the text and image which should be displayed in the box when no tree node currently selected.

To take some specific actions when the drop-down window appears or disappears — write OnDropDown, OnAfterDropDown and OnCloseUp event handlers.

Also the tree nodes of TreeComboBox can be exported or imported from another containers of TTreeNodes object, like standard TreeView control. Check out ExportToTreeNodes and ImportFromTreeNodes methods for more details.

2 Installation Instructions

Package without source code

to Delphi 2

1. Unzip files from "Delphi2" directory to your "Delphi 2\Lib" directory.
2. Start Delphi 2 IDE.
3. Select "Component \ Install..." menu item.
4. Press "Add" button and select "TreeComboBox.dcu" file.
5. Rebuild library.

to Delphi 3

1. Unzip files from "Delphi3" directory and copy them to "Delphi 3\Lib".
2. Start Delphi 3 IDE.
3. Open "TreeComboBoxD3.dpk" file.
4. Install package to the components palette ("Install" button).

to Delphi 4

1. Unzip files from "Delphi4" directory and copy them to "Delphi 4\Lib".
2. Start Delphi 4 IDE.
3. Open "TreeComboBoxD4.dpk" file.
4. Install package to the components palette ("Install" button).

to Delphi 5

1. Unzip files from "Delphi5" directory and copy them to "Delphi 5\Lib".
2. Start Delphi 5 IDE.
3. Open "TreeComboBoxD5.dpk" file.
4. Install package to the components palette ("Install" button).

to Delphi 6

1. Unzip files from "Delphi6" directory and copy them to "Delphi 6\Lib".
2. Start Delphi 6 IDE.
3. Open "TreeComboBoxD6.dpk" file.
4. Install package to the components palette ("Install" button).

to Delphi 7

1. Unzip files from "Delphi7" directory and copy them to "Delphi 7\Lib".
2. Start Delphi 7 IDE.
3. Open "TreeComboBoxD7.dpk" file.
4. Install package to the components palette ("Install" button).

to C++ Builder 1

1. Unzip files from "BCB1" directory to your "CBuilder\Lib" directory.
2. Start C++ Builder IDE.
3. Select "Component \ Install..." menu item.
4. Press "Add" button and select "TreeComboBox.dcu" file.
5. Rebuild library.

to C++ Builder 3

1. Unzip files from "BCB3" directory and copy them to "CBuilder3\Lib".
2. Start C++ Builder 3 IDE.
3. Open "TreeComboBoxCB3.bpk" file.
6. Select "Project \ Make TreeComboBoxCB3" menu item.
7. Select "Component \ InstallPackages" menu item.
8. Press "Add" button and select "TreeComboBoxCB3.bpl" file.

to C++ Builder 4
1. Unzip files from "BCB4" directory and copy them to "CBuilder4\Lib".
2. Start C++ Builder 4 IDE.
3. Open "TreeComboBoxCB4.bpk" file.
4. Install package to the components palette ("Install" button).

to C++ Builder 5
1. Unzip files from "BCB5" directory and copy them to "CBuilder5\Lib".
2. Start C++ Builder 5 IDE.
3. Open "TreeComboBoxCB5.bpk" file.
4. Install package to the components palette ("Install" button).

to C++ Builder 6
1. Unzip files from "BCB6" directory and copy them to "CBuilder6\Lib".
2. Start C++ Builder 6 IDE.
3. Open "TreeComboBoxCB6.bpk" file.
4. Install package to the components palette ("Install" button).

Source code
1. Uninstall / delete all previous (trial) instances of TreeComboBox.
2. Unzip files from "Sources" directory and copy them to "..\Lib" directory.
3. Run Delphi or ++ Builder IDE.
4. Select "Component \ Install..." menu item.
5. Press "Add" button and select "TreeComboBox.pas" file.
6. Rebuild library.

3 Registration Information

TreeComboBox component is SHAREWARE. This means that you can try it out for free, but if you like it and want to use it you have to register it with the author. Before continue read and accept license agreement please.

The only difference between the unregistered and registered versions is that the registered one has not message box with remind to register and works without Delphi (C++ Builder) running. You can also purchase the source code, if you would like to have it, and be able to compile or modify the TreeComboBox on any 32-bit version of Delphi or C++ Builder.

If you would like to use the TreeComboBox and receive full, unrestricted version, priority support or even source code — you have to purchase proper license.

All prices in US dollars. Registering entitles you to unlimited support via E-Mail, minor version updates indefinitely and major version updates for 6 month from date of purchase.

Registration types:

Full, unrestricted version without source code:
Single user license:
- **https://secure.element5.com/register.html?productid=185912 - $19,95**

Site license:
- https://secure.element5.com/register.html?productid=185914 - $69,95

Full version including 100% Source Code:
Single user license:

- **https://secure.element5.com/register.html?productid=185915 - $29,95**

Site license:

- https://secure.element5.com/register.html?productid=185916 - $99,95

Comments

1. ***Site license*** covers a single organisation in one location (building complex). If you buy a site license, you may use the software in unlimited number of your company's computers withing this area. Site license is very cost-effective if you have many computers (many software developers).

See license agreement for more details.

4 License Agreement

Copyright

The TreeComboBox component (software) is Copyright © 1999-2003, by Utilmind Solutions® (Utilmind). All rights reserved.
The authors - Utilmind Solutions® and Aleksey Kuznetsov (founder of Utilmind), exclusively own all copyrights to the Advanced Application Controls (AppControls) and all other products distributed by Utilmind Solutions®.

Liability disclaimer

THIS SOFTWARE IS DISTRIBUTED "AS IS" AND WITHOUT WARRANTIES AS TO PERFORMANCE OF MERCHANTABILITY OR ANY OTHER WARRANTIES WHETHER EXPRESSED OR IMPLIED. YOU USE IT AT YOUR OWN RISK. THE AUTHOR WILL NOT BE LIABLE FOR DATA LOSS, DAMAGES, LOSS OF PROFITS OR ANY OTHER KIND OF LOSS WHILE USING OR MISUSING THIS SOFTWARE.

Restrictions

You may not attempt to reverse compile, modify, translate or disassemble the software in whole or in part. You may not remove or modify any copyright notice or the method by which it may be invoked.

Operating license

Unregistered version

You may distribute the unregistered version of software freely, provided that all files are included and remain unmodified and that no extra files have been added to the package. You may not ask any money for the distribution. You may use the unregistered version of software free of charge for testing purposes, but if you want to use it for other purposes than testing - you have to register it with the author.

Registered version (single user license)

Once you have registered, you will receive a personal registered copy via email and login information to access your personal area at AppControls.com. This copy may not be copied or lend. You have the non-exclusive right to use registered version of the software only by a single person, on a single computer at a time. You may physically transfer the software from one computer to another, provided that the software is used only by a single person, on a single computer at a time. In group projects where multiple persons will use the software, you must purchase an individual license for each member of the group or purchase site license. Use over a "local area network" (within the same locale) is permitted provided that the software is used only by a single person, on a single computer at a time. Use over a "wide area network" (outside the same locale) is strictly prohibited under any and all circumstances.

Registered version (site/team license)
Once you have registered, you will receive a personal registered copy via email and login information to access your personal area at AppControls.com. This copy may not be copied or lend. You have the non-exclusive right to use and transfer registered version of software on any number of computers by your company or your team only in one location (building complex). If you purchase a site license, you may use the program in an unlimited number of your company's computers within this area.

Registered version (Educational site license)
Once you have registered, you will receive a personal registered copy via email and login information to access your personal area at AppControls.com. This copy may not be copied or lend. You have the non-exclusive right to use and transfer registered version of software on any number of computers by your educational organisation (school/college/university etc) in one location (building complex). If you buy a educational site license, you may use the program in an unlimited number of your edicational organisation's computers within this area.

Registered version (World-wide license)
Once you have registered, you will receive a personal registered copy via email and login information to access your personal area at AppControls.com. This copy may not be copied or lend. You have the non-exclusive right to use and transfer registered version of software on any number of computers by your company or your team world-wide. If your company has many branches even with thouthands of computers, world wide license covers them all.

Notes (clarification)
"Single-user license" means "single-developer license". "Site license" means that it can be used by any number of software developers within your company.
You can use purchased components in ANY number of your projects and deploy the "end-user" software to ANY number of your users/customers without any additional royalty fees. However you are not permitted to distribute the component itself (the source code or .dcu files of components).

Back-up and transfer
You may make one copy of the software solely for "back-up" purposes, as prescribed by international copyright laws. You must reproduce and include the copyright notice on the back-up copy.

Terms
This license is effective until terminated. You may terminate it by destroying the program, the documentation and copies thereof. This license will also terminate if you fail to comply with any terms or conditions of this agreement. You agree upon such termination to destroy all copies of the program and of the documentation, or return them to author.

Other rights and restrictions
All other rights and restrictions not specifically granted in this license are reserved by authors.

5 Properties

5.1 AcceptOnDblClick

Applies to
tcTreeComboBox component.

Declaration

```
property AcceptOnDblClick: Boolean;
```

Description

The AcceptOnDblClick property controls wether the tree node can be selected by double click or by single click.

Set AcceptOnDblClick to True, if you think that user should double click the node to select it, or leave it False if you would like to make it selectable by single mouse click.

See also

TreeOptions property.

5.2 AttachedLabel

Applies to

tcTreeComboBox component.

Unit

acAttachedLabel

Declaration

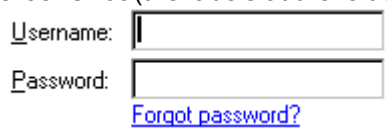
```
type
  TacAttachedLabelPosition = (lpAbove, lpBelow, lpLeft, lpRight);
  TacAttachedLabelSpacingKind = (skNearPoint, skFarPoint);
  TacAttachedLabel = class
    property Caption: TCaption;
    property Color: TColor default clBtnFace;
    property Cursor: TCursor default crDefault;
    property Enabled: Boolean;
    property FocusOnClick: Boolean default True;
    property Font: TFont;
    property Hint: String;
    property ParentColor: Boolean default True;
    property ParentFont: Boolean default True;
    property ParentShowHint: Boolean default True;
    property PopupMenu: TPopupMenu;
    property Position: TtcAttachedLabelPosition default lpLeft;
    property ShowAccelChar: Boolean default True;
    property ShowHint: Boolean default False;
    property Spacing: Integer default 4;
    property SpacingKind: TtcAttachedLabelSpacingKind default
skNearPoint;
    property Transparent: Boolean default False;
    property Visible: Boolean;
    property WordWrap: Boolean default False;
  end;
```

Description

The AttachedLabel structure lets you to operate with the label attached to the some side of control (at the left or right side, above or below). You can specify the behaviour of label, its position near the control, and accelerator key for the control (character in caption, after ampersand (&)).

The label always associated with the control. When user press the accelerator key (Alt + Character, specified after ampersand (&)), the input focus will be imediately switched to the labeled control. The input focus also will be moved to the control if user clicks the label with mouse.

Screenshot (the labels at the left side is attached to the edit control):



See also
OnLabelClick and OnLabelDblClick events.

5.2.1 Caption

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

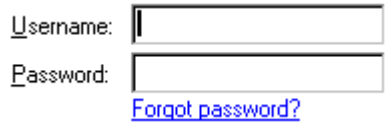
Declaration
```
property Caption: TCaption;
```

Description
The Caption property specified a text string which appers on label attached to control (attached label).

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

The label always associated with the control. When user press the accelerator key (Alt + Character, specified after ampersand (&)), the input focus will be imediately switched to the labeled control.

Screenshot (the labels at the left side is attached to the edit control):



See also
ShowAccelChar, Color, Font, Transparent, Visible and WordWrap properties.

5.2.2 Color

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
property Color: TColor;
```

Description
The Color property specifies the background color for label attached to control. Use Color to read or change the background color of the control.

If a control's ParentColor property is true, then changing the Color property of the control's parent automatically changes the Color property of the control. When the value of the Color property is changed, the control's ParentColor property is automatically set to False.

If you don't want to show background — set Transparent property to True.

See also
ParentColor and Transparent properties.

5.2.3 Cursor

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
property Cursor: TCursor;
```

Description
The Cursor property specifies the image used to represent the mouse pointer when it passes into the region covered by this label attached to control.

Change the value of Cursor to provide feedback to the user when the mouse pointer enters the control. The value of Cursor is the index of the cursor in the list of cursors maintained by the global variable, Screen. In addition to the built-in cursors provided by TScreen, applications can add custom cursors to the list.

5.2.4 Enabled

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
property Enabled: Boolean;
```

Description
The Enabled property controls whether the control responds to mouse, keyboard, and timer events.

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse, keyboard, and timer events.

To re-enable a control, set Enabled to true. The control is no longer dimmed, and the user can use the control.

See also
Caption and Color properties.

5.2.5 FocusOnClick

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
property FocusOnClick: Boolean; // True by default
```

Description
The FocusOnClick property controls whether the control, associated with the attached label, should accept focus when user clicks the label with mouse.

Set FocusOnClick to True if you want to switch focus to the control when user clicks the label, or set

it to False otherwise.

Note
Additionally you can use OnLabelClick and OnLabelDblClick events of the control.

5.2.6 Font

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
property Font: TFont;
```

Description
The Font property controls the attributes of text written on the attached label of control.

To change to a new font, specify a new TFont object. To modify a font, change the value of the Charset, Color, Height, Name, Pitch, Size, or Style of the TFont object.

See also
ParentFont property.

5.2.7 Hint

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
property Hint: String;
```

Description
The Hint property contains the text string that can appear when the user moves the mouse over the label attached control.

See also
ShowHint and ParentShowHint properties.

5.2.8 ParentColor

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
property ParentColor: Boolean;
```

Description
The ParentColor determines where an attached label looks for its color information.

To have a control use the same color as its parent control, set ParentColor to true If ParentColor is false the control uses its own Color property.

Set ParentColor to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentColor is true for all controls in a form, changing the background color of the form to gray causes all the controls on the form to also have a gray background.

When the value of a control's Color property changes, ParentColor becomes false automatically.

See also
Color and Transparent properties.

5.2.9 ParentFont

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
property ParentFont: Boolean;
```

Description
The ParentFont property determines where a control looks for its font information.

To have a control use the same font as its parent control, set ParentFont to true If ParentFont is false the control uses its own Font property.

Set ParentFont to true for all controls in order to ensure that all the controls on a form have a uniform appearance. For example, if ParentFont is true for all controls in a form, changing the form's Font property to 12-point Courier causes all controls on the form to use that font.

When the value of a control's Font property changes, ParentFont becomes false automatically.

When ParentFont is true for a form, the form uses the default font.

See also
Font property.

5.2.10 ParentShowHint

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
property ParentShowHint: Boolean;
```

Description
The ParentShowHint property determines where an attached label looks to find out if its Help Hint should be shown.

Use ParentShowHint to ensure that all the controls on a form either uniformly show their Help Hints or uniformly do not show them.

If ParentShowHint is true, the control uses the ShowHint property value of its parent. If ParentShowHint is false, the control uses the value of its own ShowHint property.

To provide Help Hints for only selected controls on a form, set the ShowHint property for those controls that should have Help Hints to true, and ParentShowHint becomes false automatically.

Note
Enable or disable all Help Hints for the entire application using the ShowHint property of the application object.

See also
ShowHint and Hint properties.

5.2.11 PopupMenu

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
property PopupMenu: TPopupMenu;
```

Description
The PopupMenu property Identifies the pop-up menu associated with the label attached to control.

Assign a value to PopupMenu to make a pop-up menu appear when the user selects the label and clicks the right mouse button. If the TPopupMenu's AutoPopup property is True, the pop-up menu appears automatically. If the menu's AutoPopup property is False, display the menu with a call to its Popup method from the control's OnContextPopup event handler.

5.2.12 Position

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
type
  TacAttachedLabelPosition = (lpAbove, lpBelow, lpLeft, lpRight);

property Position: TacAttachedLabelPosition;
```

Description
The Position property of the attached label specifies the layout of the label near the control.

You can make the label visible above, below, at the left or the right side of control.

See also
Spacing and SpacingKind properties.

5.2.13 ShowAccelChar

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
property ShowAccelChar: Boolean;
```

Description
The ShowAccelChar property determines how an ampersand in the label text is displayed.

Set ShowAccelChar to True to allow the label to display an underlined accelerator key value. When ShowAccelChar is True, any character preceded by an ampersand (&) appears underlined. If the FocusControl property is set, the windowed control specified by the FocusControl property receives input focus when the user types that underlined character. To display an ampersand when ShowAccelChar is True, use two ampersands (&&) to stand for the single ampersand that is displayed.

Set ShowAccelChar to False to display the label text with all ampersands appearing as ampersands.

Example
This example uses two labels on a form. The first label has a caption with an accelerator character in it. The second label also includes an ampersand, but it does not appear as an accelerator character.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  Label1.ShowAccelChar := True;
  Label1.Caption := 'An &Underlined character appears here';
  Label2.ShowAccelChar := False;
  Label2.Caption := 'An ampersand (&) appears here';
end;
```

See also
Caption property.

5.2.14 ShowHint

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
property ShowHint: Boolean;
```

Description
The ShowHint property determines whether the label attached to control displays a Help Hint when the mouse pointer rests momentarily on the label.

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true and either the control's own ShowHint property must be true, or the control's ParentShowHint property must be true and its parent's ShowHint property must be true.

For example, imagine a check box within a group box. If the ShowHint property of the group box is true and the ParentShowHint property of the check box is true, but the ShowHint property of the check box is false, the check box still displays its Help Hint.

Changing the ShowHint value automatically sets the ParentShowHint property to false.

See also
Hint property.

5.2.15 Spacing

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
property Spacing: Integer;
```

Description
The Spacing property determines the space (in pixels) between the control and attached label.

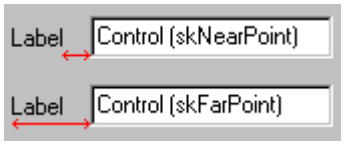
There is two spacing kinds which have an influence on distance between the control and label (see

SpacingKind property).

When SpacingKind is `skNearPoint`, the Spacing property signifies the distance between control and nearest point of attached label.

When SpacingKind is `skFarPoint`, the Spacing property signigies the distance between control and far point of label.

Screenshot



See also

SpacingKind and Position properties.

5.2.16 SpacingKind

Applies to

tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration

```
type
  TacAttachedLabelSpacingKind = (skNearPoint, skFarPoint);

property SpacingKind: TtcAttachedLabelSpacingKind;
```

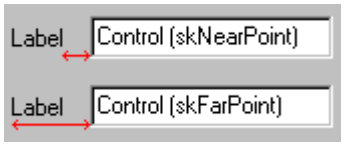
Description

The SpacingKind property specifies how to measure the spacing between the control and attached label.

There is two possible values for SpacingKind property:

Value	Meaning
`skNearPoint`	the Spacing property signifies the distance between control and nearest point of attached label;
`skFarPoint`	the Spacing property signigies the distance between control and far point of label.

Screenshot



See also

Spacing and Position properties.

5.2.17 Transparent

Applies to

tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration

```
property Transparent: Boolean;
```

Description
The Transparent property specifies whether controls that sit below the label on a form can be seen through the label.

Set Transparent to True to prevent the label from obscuring other controls on the form. For example, if the label is used to add text to a graphic, set Transparent to True so that the label does not stand out as a separate object.

Writing text so that the background is transparent is slower than writing text when Transparent is False. If the label is not obscuring a complicated image, performance can be improved by setting the background color of the label to match the object beneath it and setting Transparent to False.

See also
Color property.

5.2.18 Visible

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
property Visible: Boolean;
```

Description
The Visible property determines whether the label attached to the control appears on screen.

Use the Visible property to control the visibility of the control at runtime. If Visible is true, the control appears. If Visible is false, the control is not visible.

5.2.19 WordWrap

Applies to
tcTreeComboBox component as subproperty of **AttachedLabel** structure.

Declaration
```
property WordWrap: Boolean;
```

Description
The WordWrap property specifies whether the label text wraps when it is too long for the width of the label.

Set WordWrap to True to allow the label to display multiple line of text. When WordWrap is True, text that is too wide for the label control wraps at the right margin and continues in additional lines.

Set WordWrap to False to limit the label to a single line. When WordWrap is False, text that is too wide for the label appears truncated.

See also
Caption property.

5.3 AutoComplete

Applies to
acTreeComboBox component.

Declaration

```
property AutoComplete: Boolean;
```

Description

The AutoComplete property specifies whether the combo box automatically completes words that user typed by selecting the first item that begins with the currently typed string.

When AutoComplete is True, the combo box responds to user keystrokes by searching the Items property array for the first item that matches the string entered so far. If it finds an item that begins with the prefix typed so far, the combo box selects that item, "completing" the user's typing. If the user continues to type, selection may move to another, later, item, as the currently typed prefix no longer matches the initial AutoComplete selection. If the user types a string that is not the prefix of a string in the combo box, the string is taken as a unique entry and none of the items in the list are selected.

When AutoComplete is False, this feature is disabled. Strings that the user types are taken literally, with no attempt to match them to a string in the combo box. Setting AutoComplete to False is necessary if you want to allow the user to enter a value that is not in the Items list but that is a prefix on one of the items.

See also

AutoCompleteDelay, AutoCompleteIgnoreCase, Items, ShowEditor and SelectedNode properties.

5.4 AutoCompleteDelay

Applies to

acTreeComboBox component.

Declaration

```
property AutoCompleteDelay: Integer; // 100 milliseconds by default
```

Description

The AutoCompleteDelay property speicifes the delay, in milliseconds, which should pass between keypress and attempt to auto-complete the input.

By default AutoCompleteDelay = 100 milliseconds, that means that when user typed the character and delayed 1/10 of second before typing the next character, the combo box will try to auto-complete the input searching the item with the text which begins with the currently typed string.

See also

AutoComplete, AutoCompleteIgnoreCase, Items, ShowEditor and SelectedNode properties.

5.5 AutoCompleteIgnoreCase

Applies to

tcTreeComboBox component.

Declaration

```
property AutoCompleteIgnoreCase: Boolean; // True by default
```

Description

The AutoCompleteIgnoreCase property specifies whether the case of typed characters does not matter, and whether you would like to auto-complete the input disregarding the case of characters.

For example, the drop-down list contains the line "Apple". If the AutoCompleteIgnoreCase is True and user type the "a", the compobox will try to put missing part of line "pple", disregarding of the

charcase of the line in the history list. In case if the AutoComplet IgnoreCase is False, the combo box will try to find the line with exactly same case of characters.

See also
AutoComplete, AutoCompleteDelay, Items, ShowEditor and SelectedNode properties.

5.6 Button

Applies to
tcTreeComboBox component.

Declaration
```
type
  TacEditButton = class(TPersistent)
  published
    property Cursor: TCursor;
    property Flat: Boolean; // in Delphi 3 and later !!!
    property Glyph: TBitmap;
    property NumGlyphs: TNumGlyphs; // 1..4
    property Hint: String;
    property Width: Word;
    property Visible: Boolean; // False by default
  end;
```

Description
The tcTreeComboBox is able to have a custom button at the right edge of edit control. The Button structure intended for specify settings for this button. To make button visible on edit control — set Button.Visible to True.

You can specify the glyph image and number of images, hint, cursor and width for button.

The button which at the right of edit control is just usual TSpeedButton component. You can even gain direct access to all button's properties using hidden **EditButton** property.
For example, to specify caption for this button you may use following code:
```
acTreeComboBox1.EditButton.Caption := 'Test';
```

See also
OnButtonClick event.

5.6.1 Cursor

Applies to
tcTreeComboBox component as subproperty of Button structure.

Declaration
```
property Cursor: TCursor;
```

Description
The Cursor property specifies the image for mouse pointer when it passes over the custom button at the right side of edit control.

See also
CursorBorder property.

5.6.2 Flat

Applies to
tcTreeComboBox component as subproperty of Button structure.

Declaration
```
property Flat: Boolean;
```

Description
The Flat property determines whether the button has a 3D border that provides a raised or lowered look.

Set Flat to True to remove the raised border when the button is unselected and the lowered border when the button is clicked or selected. When Flat is True, use separate bitmaps for the different button states to provide visual feedback to the user about the button state.

Remarks
This property available in Delphi 3 and later.

5.6.3 Glyph

Applies to
tcTreeComboBox component as subproperty of Button structure.

Declaration
```
property Glyph: TBitmap;
```

Description
The Glyph property specifies the bitmap that appears on the custom button at the right side of edit control.

Set Glyph to a bitmap object that contains the image that should appear on the face of the button. Bring up the Open dialog box from the Object Inspector to choose a bitmap file (with a .BMP extension), or specify a bitmap file at runtime.

Glyph can provide up to four images within a single bitmap. All images must be the same size and next to each other in a horizontal row. Button displays one of these images depending on the state of the button.

Image position	Buttons state	Description
First	Up	This image appears when the button is unselected. If no other images exist in the bitmap, this image is used for all states.
Second	Disabled	This image usually appears dimmed to indicate that the button can't be selected.
Third	Clicked	This image appears when the button is clicked. The Up image reappears when the user releases the mouse button.
Fourth	Down	This image appears when the button stays down indicating that it remains selected.

If only one image is present, Button attempts to represent the other states by altering the image slightly for each state, although the Down state is always the same as the Up state.

If the bitmap contains multiple images, specify the number of images in the bitmap with the NumGlyphs property.

Note
The lower left hand pixel of the bitmap is reserved for the "transparent" color. Any pixel in the bitmap which matches that lower left hand pixel will be transparent.

See also
NumGlyphs property.

5.6.4 NumGlyphs

Applies to
tcTreeComboBox component as subproperty of Button structure.

Declaration
```
property NumGlyphs: TNumGlyphs; // 1..4
```

Description
The NumGlyphs is the number of images specified in the Glyph property.

Set NumGlyphs to the number of images provided by the bitmap assigned to the Glyph property. All images must be the same size and next to each other in a row. The Glyph property can provide up to four images.

See also
Glyph property.

5.6.5 Hint

Applies to
tcTreeComboBox component as subproperty of Button structure.

Declaration
```
property Hint: String;
```

Description
The Hint property contains the text string that can appear when the user moves the mouse pointer over the button at right side of edit control.

5.6.6 Kind

Applies to
tcTreeComboBox component as subproperty of Button structure.

Declaration
```
type
  TacEditButtonKind = (bkCustom, bkEllipsis, bkDropDown, bkFileOpen,
bkFileSave, bkBrowse, bkCalc);

property Kind: TacEditButtonKind;
```

Description
The Kind property determines the kind of bitmap button.

You can use custom image (Kind = bkCustom) and specify your sign to Glyph property, or use one of following images (), just switching the Kind property.

See also

Glyph and NumGlyphs properties.

5.6.7 Width

Applies to
tcTreeComboBox component as subproperty of Button structure.

Declaration
```
property Width: Integer;
```

Description
The Width property controls the horizontal size of the button in pixels.

The button's height controlled by height of tcTreeComboBox control.

5.6.8 Visible

Applies to
tcTreeComboBox component as subproperty of Button structure.

Declaration
```
property Visible: Boolean;
```

Description
The Visible property determines whether the button appears at the right side of tcTreeComboBox control.

Use the Visible property to control the visibility of the button at runtime. If Visible is True, the button appears. If Visible is False, the button is not visible.

5.7 ColorDisabled

Applies to
tcTreeComboBox component.

Declaration
```
property ColorDisabled: TColor; // clBtnFace by default
```

Description
The ColorDisabled property for tcTreeComboBox component is the background color of the edit control at disabled state (when Enabled is False). Use the ColorDisabled property to specify color for disabled edit control.

See also
Color property.

5.8 CursorBorder

Applies to
tcTreeComboBox component.

Declaration
```
property CursorBorder: TCursor;
```

Description
The CursorBorder property specifies the image used to represent the mouse pointer when it passes

over the non-client area of edit control.

5.9 DropDownCount

Applies to
tcTreeComboBox component.

Declaration
```
property DropDownCount: Integer;
```

Description
The DropDownCount property specifies the number of items (tree nodes) which can be visible in the drop down window simultaneously. You can increase this value if you wish to let user to see more tree nodes when the list is dropped down.

See also
DropDownWidth property.

5.10 DropDownWidth

Applies to
tcTreeComboBox component.

Declaration
```
property DropDownWidth: Integer;
```

Description
The DropDownWidth controls the width of the drop down window, when can be visible when user clicks on the TreeComboBox control or its button at the right side.

By default the width of drop down window is the same as the width of control (when the DropDownWidth equal to Width and you resize the control at design time, you will see that DropDownWidth will be changed accordingly to Width of control). But of course, you can modify this value, if you wish to make the width of popup window different than width of contol.

See also
DropDownCount property.

5.11 EmptyItemImageIndex

Applies to
tcTreeComboBox component.

Declaration
```
property EmptyItemImageIndex: Integer;
```

Description
The EmptyItemImageIndex property used to specify the index of image from image list. The image which you selected will appear in the combo box if no tree node is selected.

See also
EmptyItemText and Images properties.

5.12 EmptyItemText

Applies to
tcTreeComboBox component.

Declaration
```
property EmptyItemText: String;
```

Description
The EmptyItemText property used to specify the text which appears in the combo box if no tree node is selected (for example, you can specify "Please select something" text to this property if you don't want to show the blank box).

See also
EmptyItemImageIndex property.

5.13 Images

Applies to
tcTreeComboBox component.

Declaration
```
property Images: TImageList;
```

Description
The Images property determines which image list is associated with the TreeView embedded to the drop-down window of tcTreeComboBox.

Use Images to provide a customized list of bitmaps that can be displayed to the left of a node's label. Individual nodes specify the image from this list that should appear by setting their ImageIndex property (use Items property to specify the ImageIndex'es of the tree nodes at design-time).

See also
StateImages properties.

5.14 ItemIndex

Applies to
tcTreeComboBox component.

Declaration
```
property ItemIndex: Integer;
```

Description
The ItemIndex property determines the absolute index of selected node. You can read and specify this value both at run- and design-time to select the node.

See also
SelectedNode property;
OnCanSelectNode event.

5.15 Items

Applies to
tcTreeComboBox component.

Declaration
```
property Items: TTreeNodes;
```

Description
The Items property lists the individual nodes that appear in the TreeView control, embedded to the drop-down window of tcTreeComboBox.

Individual nodes in a tree view are TTreeNode objects. These individual nodes can be accessed by using the Items property along with the item's index into the tree view. For example, to access the second item in the tree view, you could use the following code.

MyTreeNode := tcTreeComboBox1.Items[1];

When setting this property at design-time in the Object Inspector the TreeView Items Editor appears. Use the New Item and New SubItem buttons to add items to the tree view. Use the Text property to modify what text is displayed in the label of the item.

At run-time nodes can be added and inserted by using the TTreeNodes methods AddChildFirst, AddChild, AddChildObjectFirst, AddChildObject, AddFirst, Add, AddObjectFirst, AddObject and Insert.

Notes
Accessing tree view items by index can be time-intensive, particularly when the tree view contains many items. For optimal performance, try to design your application so that it has as few dependencies on the tree view's item index as possible;

Alternatively you can access the tree nodes at run-time through TreeView property.

See also
TreeView, TreeOptions and SelectedNode properties.

5.16 SelectedNode

Applies to
tcTreeComboBox component.

Declaration
```
property SelectedNode: TTreeNode; // read-only!!
```

Description
The SelectedNode is the read-only property which returns the pointer to TTreeNode object, which currently selected in the TreeView control embedded to drop-down window.

Alternatively you can determinate and specify the selected tree node by its absolute index using ItemIndex property.

See also
ItemIndex property;
OnCanSelectNode event.

5.17 ShowEditor

Applies to
tcTreeComboBox component.

Declaration
```
property ShowEditor: Boolean;
```

Description
The ShowEditor property controls whether user should be able to enter of modify the text in the combo-box using keyboard.

Set ShowEditor to True if you would like to let user to modify the text with keyboard, otherwise, if user must select the tree node from drop-down list, set ShowEditor to False.

See also
SelectedNode and TreeOptions properties.

5.18 ShowTreePathInEdit

Applies to
tcTreeComboBox component.

Declaration
```
property ShowTreePathInEdit: Boolean;
```

Description
The ShowTreePathInEdit property determines whether the "edit box" should display the full path of the node.

For example, we have the following structure:
```
Test
    Test 1
    Test 2
        Test 1 (selected)
        Test 2
    Test 3
```

Then, in case if ShowTreePathInEdit = True, we will have "Test\Test 2\Test 1" shown in the edit box of the combobox.

Note
 This property will take effect only in case if ShowEditor property is False. You can't display the path when the combobox is editable.

See also
TreePathSeparator and ShowEditor properties.

5.19 StateImages

Applies to
tcTreeComboBox component.

Declaration
```
property StateImages: TImageList;
```

Description
The StateImages property determines which image list to use for state images.

Use StateImages to provide a set of bitmaps that reflect the state of tree view nodes. The state image appears as an additional image to the left of the item's icon.

See also
Images, TreeOptions and TreeView properties.

5.20 TreeOptions

Applies to
tcTreeComboBox component.

Declaration
```
type
  TacTreeViewOptions = class
  published
    property AutoExpand: Boolean default False;
    property FullExpand: Boolean default False;
    property MouseTrack: Boolean default True;
    property RowSelect: Boolean default False;
    property ShowButtons: Boolean default True;
    property ShowLines: Boolean default True;
    property ShowRoot: Boolean default True;
    property Tooltips: Boolean default True;
  end;

property TreeOptions: TacTreeViewOptions;
```

Description
The TreeOptions structure used to specify some behaviours of the TreeView embedded to drop-down window of tcTreeComboBox control.

There are possible properties of embedded TreeView control:

Property	Purpose
AutoExpand	Specifies whether the nodes in the tree view automatically expand and collapse depending on the selection;
FullExpand	Specifies whether the TreeComboBox should automatically expand all the nodes and their sub-nodes upon DropDown (alternativaly you can just expand nodes in OnDropDown event handler);
MouseTrack	Specifies whether the selection should always hovers the tree node under mouse pointer, even when user move mouse over tree view without clicking;
RowSelect	Specifies whether the entire row of the selected item is highlighted;
ShowButtons	Specifies whether to display plus (+) and minus (-) buttons to the left side of each parent item;
ShowLines	Specifies whether to display the lines that link child nodes to their corresponding parent nodes;
ShowRoot	Specifies whether lines connecting top-level nodes are displayed;
Tooltips	Specifies whether the items in the tree view have tooltips.

See also
AcceptOnDblClick, ShowEditor, SelectedNode and TreeView properties.

5.21 TreePathSeparator

Applies to
tcTreeComboBox component.

Declaration
```
property TreePathSeparator: Char; // '\' by default
```

Description
The TreePathSeparator specifies the separator character for the node path, in case if ShowTreePathInEdit = True.

See also
ShowTreePathInEdit and ShowEditor properties.

5.22 TreeView

Applies to
tcTreeComboBox component.

Declaration
```
property TreeView: TacPopupTreeView; // run-time and read-only!
```

Description
The TreeView is the *read-only* property which can be used *at run-time* to access the properties and methods of the TreeView control embedded to the drop-down window of tcTreeComboBox.


Alternatively you can just specify some behaviours of TreeView using TreeOptions property.

Example
```
procedure TForm1.tcTreeComboBox1DropDown(Sender: TObject);
begin
  tcTreeComboBox1.TreeView.FullExpand;
end;
```

See also
TreeOptions and SelectedNode properties;
ExportToTreeNodes, ImportFromTreeNodes and ImportFromTreeView methods;
OnDropDown, OnAfterDropDown and OnCloseUp events.

6 Methods

6.1 AddPath

Applies to
tcTreeComboBox component.

Declaration
```
function AddPath(Node: TTreeNode; const Path: String; const
PathSeparator: String = '\'): TTreeNode;
```

Description
The AddPath method adds or inserts to the TreeView the *string* or nested *group of strings* (pathes) specified by *Path* parameter and separated by string specifed in *PathSeparator* parameter.

The declaration can be
```
AddPath(nil, 'D:\Test\Test');
```

So, if you call method above, the component will add following structure to the root item of TreeComboBox:
```
D:
  Test
    Test
```

Return value is the last added child node.

See also
ShowTreePathInEdit, TreePathSeparator and TreeView properties.

6.2 ExportToTreeNodes

Applies to
tcTreeComboBox component.

Declaration
```
procedure ExportToTreeNodes(TargetTreeNodes: TTreeNodes;
  SourceRootNode: TTreeNode = nil; TargetRootNode: TTreeNode = nil);
```

Description
The ExportToTreeNodes method used to copy the tree nodes from current TreeCombBox to another container TTreeNodes object (i.e: standard TreeView or another TreeComboBox).

Example *(demonstrates how to fill the TreeView with nodes from TreeComboBox)*
tcTreeComboBox1.ExportToTreeNodes(acTreeView1.Items);

See also
ImportFromTreeNodes and ImportFromTreeView methods.

6.3 FindAbsoluteIndexByNode

Applies to
tcTreeComboBox component.

Declaration
```
function FindAbsoluteIndexByNode(Node: TTreeNode): Integer;
```

Description
The FindAbsoluteIndexByNode method retreives the absolute index from node specified in the *Node* parameter.

See also
FindNode and FindAbsoluteIndexByNode methods.

6.4 FindNode

Applies to
tcTreeComboBox component.

Declaration
```
function FindNode(const Text: String; IgnoreCase: Boolean = False;
  WholeText: Boolean = True): TTreeNode;
```

Description
The FindNode method search the tree node with the text specified in *Text* parameter.

The *IgnoreCase* parameter specified whether the search should be case sensitive.

The *WholeText* parameter specified whether it should find the node where the length of caption is the same as length of Text parameter (if *WholeText* is False, the method will try to find the node which caption begins with the text specified in *Text* parameter).

Function returns the pointer to tree node if it successfully found, or NIL (NULL) otherwise (if the node with specified Text not exists in the tree view).

See also
FindNodeByAbsoluteIndex and FindAbsoluteIndexByNode methods.

6.5 FindNodeByAbsoluteIndex

Applies to
tcTreeComboBox components.

Declaration
```
function FindNodeByAbsoluteIndex(AbsoluteIndex: Integer): TTreeNode;
```

Description
The FindNodeByAbsoluteIndex method search the tree node with absolute index equal to the value specified in the *AbsoluteIndex* parameter.

Function returns the pointer to tree node if it successfully found, or NIL (NULL) otherwise (if the node with specified absolute index not exists in the tree view).

See also
FindNode and FindAbsoluteIndexByNode methods.

6.6 ImportFromTreeNodes

Applies to
tcTreeComboBox component.

Declaration
```
procedure ImportFromTreeNodes(SourceTreeNodes: TTreeNodes;
  SourceRootNode: TTreeNode = nil; TargetRootNode: TTreeNode = nil);
```

Description
The ImportFromTreeNodes method used to copy the tree nodes from some container TTreeNodes object (i.e: standard TreeView or another TreeComboBox) to *this* TreeComboBox control.

Example *(demonstrates how to fill the tcTreeComboBox control with nodes from some TreeView)*
tcTreeComboBox1.ImportFromTreeNodes(acTreeView1.Items);

See also
ExportToTreeNodes and ImportFromTreeView methods.

6.7 ImportFromTreeView

Applies to
tcTreeComboBox component.

Declaration
```
procedure ImportFromTreeView(SourceTreeView: TCustomTreeView;
  SourceRootNode: TTreeNode = nil; TargetRootNode: TTreeNode = nil);
```

Description
The ImportFromTreeView method used to copy the tree nodes from some TreeView control to *this* TreeComboBox control.

Example *(demonstrates how to fill the tcTreeComboBox control with nodes from some TreeView)*
tcTreeComboBox1.ImportFromTreeView(acTreeView1);

See also
ExportToTreeNodes and ImportFromTreeNodes methods.

6.8 SelectNodeByText

Applies to
tcTreeComboBox component.

Declaration
```
function SelectNodeByText(const Text: String; IgnoreCase: Boolean =
False): TTreeNode;
```

Description
The SelectNodeByText method looking for the node in drop-down list, with the same text as specified in Text property. If it successfully found the node with specified text, it selects it.

The *IgnoreCase* parameter specified whether the search should be case sensitive.

Function returns the pointer to selected node if succeed, or NIL (NULL) otherwise.

See also
ShowEditor property.

6.9 SetSelection

Applies to
tcTreeComboBox component.

Declaration
```
procedure SetSelection(SelStart: Integer = 0; SelEnd: Integer = -1);
```

Description
The SetSelection method used to select a range of characters in an edit control.

SelStart parameter specifies the starting character position of the selection.

SelEnd parameter specifies the ending character position of the selection.

Remarks

If the *SelStart* parameter is 0 and the *SelEnd* parameter is -1, all the text in the edit control is selected. If *SelStart* is -1, any current selection is removed. The caret is placed at the end of the selection indicated by the greater of the two values *SelEnd* and *SelStart*.

7 Events

7.1 OnAfterDropDown

Applies to

tcTreeComboBox component.

Declaration

```
property OnAfterDropDown: TNotifyEvent;
```

Description

The OnAfterDropDown event occurs at once the drop-down window (wich displays the tree nodes) appears below the combo-box.

Write this event handler if you need to perform some specific actions at once after the drop-down window appears on screen. If you need to do something (for example, specify the width of drop-down window) *before* it appears on screen — write OnDropDown event handler.

See also

OnDropDown and OnCloseUp events;
DropDownWidth property.

7.2 OnButtonClick

Applies to

tcTreeComboBox component.

Declaration

```
property OnButtonClick: TNotifyEvent;
```

Description

The OnButtonClick event occurs when user clicks the custom button at the right side of edit control.

To make the button appear on edit control — set Button.Visible to True.

See also

Button structure.

7.3 OnButtonLeftMouseDown

Applies to

tcTreeComboBox component.

Declaration

```
property OnButtonLeftMouseDown: TNotifyEvent;
```

Description
The OnButtonLeftMouseDown event occurs when user presses the left mouse button over the button at the right side of edit control.

 To make the button appear on edit control — set Button.Visible to True.

See also
Button structure.

7.4 OnCanSelectNode

Applies to
tcTreeComboBox component.

Declaration
```
type
  TtcTreeComboBoxCanSelectNodeEvent = procedure(Sender: TObject; Node:
TTreeNode; var AllowSelect: Boolean) of object;

property OnCanSelectNode: TtcTreeComboBoxCanSelectNodeEvent;
```

Description
The OnCanSelectNode event

See also
SelectedNode and TreeView properties;
OnDropDown, OnAfterDropDown and OnCloseUp events.

7.5 OnCloseUp

Applies to
tcTreeComboBox component.

Declaration
```
type
  TacDropDownCloseUpEvent = procedure(Sender: TObject; Accept: Boolean)
of object;

property OnCloseUp: TacDropDownCloseUpEvent;
```

Description
The OnCloseUp event occurs when the drop-down window is about to disappear from screen.

The *Accept* parameter indicates whether the tree node has been selected by user or the drop-down window has been closed without selection (i.e: user has clicked somewhere else and moved the input focus to another control). When *Accept* is True, user has selected the node, otherwise — just canceled the drop-down window.

See also
OnDropDown, OnAfterDropDown and OnCanSelectNode events.

7.6 OnDropDown

Applies to
tcTreeComboBox component.

Declaration

```
property OnDropDown: TNotifyEvent;
```

Description

The OnDropDown event occurs when the user clicks the combo box (or its button at the right side), and the drop-down window with tree nodes is about to appear on screen.

Write the OnDropDown event handler to perform some specific actions (for example, resize the drop-down window) *before* the tree nodes appears on screen.

If you need to take some actions when the drop-down window already on screen — write OnAfterDropDown event handler.

Example (demonstrates how to change the width of drop-down window in OnDropDown event handler)

```
procedure TForm1.tcTreeComboBox1DropDown(Sender: TObject);
begin
  tcTreeComboBox1.DropDownWidth := Random(200) + 20;
  // also you can expand some nodes of embedded TreeView
  tcTreeComboBox1.TreeView.FullExpand;
end;
```

See also

OnAfterDropDown and OnCloseUp events;
DropDownWidth, TreeOptions and TreeView properties.

7.7 OnLabelClick

Applies to

tcTreeComboBox component.

Declaration

```
property OnLabelClick: TNotifyEvent;
```

Description

The OnLabelClick event occurs when user clicks the label attached to the control with a mouse.

To make the label appear near the edit control — set AttachedLabel.Visible to True, or to False if you'd like to hide the attached label.

See also

AttachedLabel structure.

7.8 OnLabelDblClick

Applies to

tcTreeComboBox component.

Declaration

```
property OnLabelDblClick: TNotifyEvent;
```

Description

The OnLabelDblClick event occurs when user double clicks the label attached to the control with a mouse.

To make the label appear near the edit control — set AttachedLabel.Visible to True, or to False if you'd like to hide the attached label.

See also
AttachedLabel structure.

7.9 OnMouseEnter

Applies to
tcTreeComboBox component.

Declaration
```
property OnMouseEnter: TNotifyEvent;
```

Description
The OnMouseEnter occurs when mouse pointer hover the edit control.

See also
OnMouseLeave event.

7.10 OnMouseLeave

Applies to
tcTreeComboBox component.

Declaration
```
property OnMouseLeave: TNotifyEvent;
```

Description
The OnMouseLeave occurs when mouse pointer leave the edit control.

See also
OnMouseEnter event.