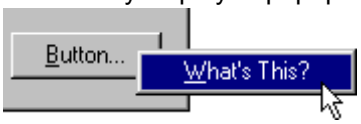
Table of Contents

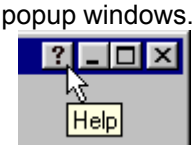
1 TFormHelp - Overview

Overview

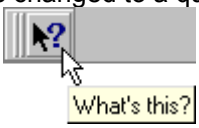
The FormHelp component adds the context-sensitive help features to your Delphi/C++ Builder forms without any bulky help files. It traps the context-sensitive help calls and creates its own popup windows from a control's hint. You can choose whether to interpret the hint string as plain text or as kind of rich text allowing you to apply different fonts colors, styles and line breaks. Don't worry about your hints — FormHelp uses the secondary part of a control's hint that is separated by a vertical bar "|". Mouse hints still works as well. With FormHelp, neither help context numbers nor extra help files are required to display context sensitive help. FormHelp's popup windows looks and feels like native context help in standard Microsoft's applications. Even if you're using regular help files, the FormHelp will be great addition and vice versa.

The FormHelp can automatically apply the "What's This?" menu item to every control with context-sensitive help in the secondary part of hint, and invokes the context help either after pressing the "Help" button on the title bar or after selecting "What's This?" menu item. This popup menu is displayed dynamically. TacFormHelp recognizes controls that have their own popup menu and even regards manually displayed popup menus or popup windows.





If the "What's this?" popup menu and button on the title bar of your form still is not enough for you, you can also point the toolbar button and menu item in the main menu or even system menu which can handle the context-help as well. When user clicks this button or menu item, the cursor will be changed to a question mark with a pointer.



If the user then clicks a control in the form, the control receives a WM_HELP message to show the context-sensitive help taken from secondary part of the Hint property.

For easement of context-help authoring, component contains a WYSIWYG help designer (drop FormHelp onto your form and try to edit secondary part of Hint property of any visible control) and may have an additional button on form's title bar.

How to use ?

Just drop FormHelp component onto your form and edit the Hint property of any visual control (like TButton, TCheckBox or TGroupBox). Type any text in the secondary part of Hint, and make additional button on form's title bar visible (if needed). Recompile and execute your application. Now your form will traps all context-sensitive help calls and when user clicks help button on form's title bar and clicks on control, popup window with context-sensitive help will be displayed. To arrange the width of popup window - use PopupWidth property and OnShow event.

You can also show help-window "manually", using ShowHelp, ShowHelpFromControl and ShowHelpFromPoint methods.

Properties

Active	Activity flag of the component;
AdjustPopupWidth	Whether the width of the popup window with help-message should be adjusted accordingly to the width of visible text in the the popup window;
CaptionButton	Additional button on the form's title bar. Works even if form style don't

	allow this button (see snapshot below);
Color	Background color of popup window;
Cursor	Cursor image for popup window with help message;
DelayInterval	Specifies the interval before auto-hiding of the popup window;
Font	Font of context-sensitive help text;
PopupWidth	Controls the width of popup window;
SystemMenu	Appies the "What's This?" menu item to the system menu;
TextStyle	Specifies the text style (formatted or plain) and tag brackets ('**[**' and '**]**' by default)
WhatsThis	Whether the controls contains the "What's This?" menu item.

Notes

Avoiding the Application.Hint problem.

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 "_FormHelpReg.pas" 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 "FormHelpD3.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 "FormHelpD4.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 "FormHelpD5.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 "FormHelpD6.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 "FormHelpD7.dpk" file.
4. Install package to the components palette ("Install" button).

to Delphi 2005
1. Download "formhelp.zip" file.
2. Create "..\Lib\FormHelp" directory.
3. Unzip files and copy them to "..\Lib\FormHelp".
4. Start Delphi 2005 IDE.
5. Open "FormHelpD2005.dpk" file.
6. Install package to the components palette (right-click on "FormHelpD2005.bpl" node in the Project Manager and select "Install" menu item).

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 "_FormHelpReg.pas" 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 "FormHelpCB3.bpk" file.
6. Select "Project \ Make FormHelpCB3" menu item.
7. Select "Component \ InstallPackages" menu item.
8. Press "Add" button and select "FormHelpCB3.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 "FormHelpCB4.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 "FormHelpCB5.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 "FormHelpCB6.bpk" file.
4. Install package to the components palette ("Install" button).

Source code

1. Uninstall / delete all previous (trial) instances of FormHelp.
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 "_FormHelpReg.pas" file.
6. Rebuild library.

3 Registration Information

FormHelp 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 FormHelp on any 32bit version of Delphi or C++ Builder.

If you would like to use the FormHelp 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=140751 - $24,95**

Site license:

- https://secure.element5.com/register.html?productid=140752 **- $99,95**

Full version including 100% Source Code:

Single user license:

- **https://secure.element5.com/register.html?productid=140753 - $44,95**

Site license:

- https://secure.element5.com/register.html?productid=140754 **- $199,95**

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 FormHelp component (software) is Copyright © 1998-2002, 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 Active

Applies to

FormHelp component.

Declaration

```
property Active: Boolean;
```

Description

The Active property controls whether the FormHelp component is Active and can display the context sensitive help by hooking the WM_HELP application message. If Active is True, then when you click  button on the title bar then clicks any control with specified hint, FormHelp will show popup window which looks and feels like native Windows help.

5.2 AdjustPopupWidth

Applies to

FormHelp component.

Declaration

```
property AdjustPopupWidth: Boolean;
```

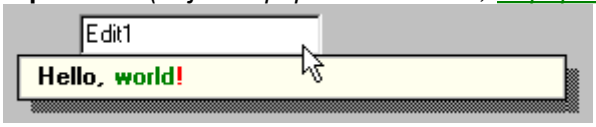
Description

The AdjustPopupWidth property controls whether the width of the popup window with help-message should be adjusted accordingly to the width of visible text in the the popup window.

When the AdjustPopupWidth is True, the width of right margin (between the right edge of popup window and text) will be equal to the width of left margin.

Snapshoot 1 *(AdjustPopupWidth = True)*

Snapshoot 2 *(AdjustPopupWidth = False, PopupWidth = 280)*



See also
PopupWidth property.

5.3 CaptionButton

Applies to
FormHelp component.

Declaration
```
property CaptionButton: TfhACaptionButton;
```

Description
The CaptionButton is the set of properties for managing an additonal button on the title bar:

Cursor	specifies the cursor image when button released;
CursorDown	specifies the cursor image when button pressed;
Enabled	enables or disables button;
Hint	specifies the button's tooltip;
ShowHint	enables or disables the tooltip (hint);
Visible	shows or hides the button from the title bar.

5.3.1 Cursor

Applies to
FormHelp component as subproperty of CaptionButton.

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

Description
The Cursor property controls the mouse cursor shape used when the mouse moves over CaptionButton in released state.

See also
CursorDown property.

5.3.2 CursorDown

Applies to
FormHelp component as subproperty of CaptionButton.

Declaration
```
property CursorDown: TCursor;
```

Description
The **CursorDown** property controls the mouse cursor shape used when the mouse moves over CaptionButton in pressed state. Value of CursorDown property for the FormHelp component are always the same as CursorHelp property

See also
CursorHelp property.

5.3.3 Enabled

Applies to
FormHelp component as subproperty of CaptionButton.

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

Description
The Enabled property controls whether the button on the title bar responds to mouse and keyboard messages. If Enabled is True, button responds normally. If Enabled is False, button become disabled  and user can not press that button.

See also
Visible property.

5.3.4 Hint

Applies to
FormHelp component as subproperty of CaptionButton.

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

Description
The Hint property is the text string that can appear when mouse pointer moves over the caption button.

If the CaptionButton's ShowHint property is False, the Help Hint won't appear, but the other hints still will.

Default Hint string for FormHelp component is 'Help'. You may change this string at run- or design-time if you're writting multilingual program.

See also
ShowHint property.

5.3.5 ShowHint

Applies to
FormHelp component as subproperty of CaptionButton.

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

Description
The ShowHint property controls whether CaptionButton can show the Hint string when mouse pointer moves over button. If ShowHint is True, Hint will appears.

See also
Hint property.

5.3.6 Visible

Applies to
FormHelp component as subproperty of CaptionButton.

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

Description
The Visible property determines whether the caption button currently visible on the title bar. Make Visible property True if you would like to show the button on title bar, or False if you would like to hide the button.

See also
Enabled property.

5.4 Color

Applies to
FormHelp component.

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

Description
The Color property specifies the background color of popup window with context-sensitive help. Default value is clContextHelp.

See also
ShadowColor property.

5.5 Cursor

Applies to
FormHelp component.

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

Description
The Cursor property is the image used when the mouse pointer moves over the popup window with context-sensitive help.

See also
Cursor and CursorDown properties of caption button.

5.6 CursorHelp

Applies to
FormHelp component *(in general, to all components, successors of TControl)*.

Declaration

```
property CursorHelp: TCursor;
```

Description
The CursorHelp property specifies the cursor image used to point the control to invoke the help for it. Value of CursorHelp are always the same as value of CursorDown property of CaptionButton structure.

See also
Cursor and CursorDown properties of CaptionButton structure.

5.7 DelayInterval

Applies to
FormHelp component.

Declaration
```
property DelayInterval: Word; // in milliseconds
```

Description
The DelayInterval property specifies the delay interval before popup window with context-sensitive help will automatically disappered. After expiration of specified interval, the popup window will be automatically closed, even if user did not clicked the mouse buttons.

5.8 Font

Applies to
FormHelp component.

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

Description
The Font property is a font object that controls the attributes of context-sensitive help text, written in secondary part of the Hint property of any control on current form.

5.9 Margins

Applies to
FormHelp component.

Declaration
```
type
  TFormHelpMargins = class
  published
    property Horizontal: Byte default 10;
    property Vertical: Byte default 4;
  end;
```

Description
The Margins specifies width and height (in pixels) for space between the border and the text in the help window.

5.9.1 Horizontal

Applies to
FormHelp component.

Declaration

```
property Horizontal: Byte;
```

Description

The Horizontal property specifies the width (in pixels) for space between the border and the text of help window.

See also

Vertical property.

5.9.2 Vertical

Applies to

FormHelp component.

Declaration

```
property Vertical: Byte;
```

Description

The Vertical property specifies the height (in pixels) for space between the border and the text of help window.

See also

Horizontal property.

5.10 ParentFont

Applies to

FormHelp component.

Declaration

```
property ParentFont: Boolean;
```

Description

The ParentFont property determines whether the FormHelp should use font of its parent form.

Set ParentFont to true in order to ensure that the text on context-sensitive help window will have the same font as on form and to have a uniform appearance for entire controls on the form. For example, if ParentFont is True, changing the form's Font property to 12-point Courier causes the context-sensitive help windows to to use that font.

 When the value of a FormHelp'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.

5.11 PopupWidth

Applies to

FormHelp component.

Declaration

```
type
```

```
  TFormHelpPopupWidth = 40..1000;

property PopupWidth: TFormHelpPopupWidth; // in pixels
```

Description

The PopupWidth property controls the width of popup window with context sensitive help. PopupWidth specifies the width of popup window for all controls on current form. It can assign values in range from 40 up to 1000 pixels. By default this value is 300 pixels.

See also

AdjustPopupWidth property.

5.12 ShadowColor

Applies to

FormHelp component.

Declaration

```
property ShadowColor: TColor;
```

Description

The ShadowColor property specifies the shadow color (dots that imitate the window shadow) for popup windows with context-sensitive help. ShadowColor used only if ShowShadow property is True.

See also

ShowShadow and Color properties.

5.13 ShowShadow

Applies to

FormHelp component.

Declaration

```
property ShowShadow: Boolean;
```

Description

The ShowShadow property determines whether the FormHelp component should display a shadow (dots that imitate a shadow) behind popup window with context-sensitive help. You may also specify the color for shadow in ShadowColor property.

See also

ShadowColor property.

5.14 SystemMenu

Applies to

FormHelp component.

Declaration

```
type
  TfhMenuSeparators = set of (seBefore, seAfter);
  TfhSystemMenu = class(TPersistent)
  published
    property ApplyToMenu: Boolean;
```

```
    property Caption: String;
    property Position: Word;
    property Separators: TacMenuSeparators;
  end;

property SystemMenu: TfhSystemMenu;
```

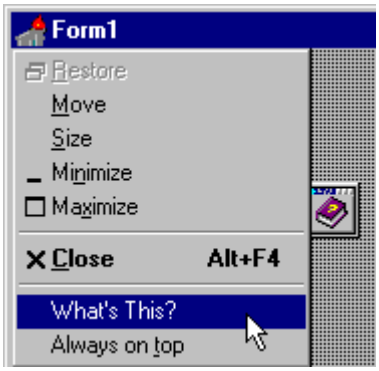
Description

The System Menu is the popup menu that appears when you click on the program icon on the upper-left corner of form's title bar. The SystemMenu property is the list of properties that manages the menu item associated with current button on the form's title bar.

Properties

ApplyToMenu	applies or removes the menu item associated with current button from the system menu;
Caption	text for the title of menu item;
Position	position of current menu item in the system menu;
Separators	specifies the separators for menu item, to separatate it from previous and / or next menu item of the system menu.

Screenshoot



5.14.1 ApplyToMenu

Applies to

FormHelp component as subproperty of SystemMenu structure.

Declaration

```
property ApplyToMenu: Boolean;
```

Description

The ApplyToMenu property controls whether the CaptionButton currently have the menu item assiciated with this button. Set ApplyToMenu property to True to add according menu item (with text specified by Caption property, in position specified by Position property) to the form's system menu and False to remove.

See also

Caption property.

5.14.2 Caption

Applies to

FormHelp component as subproperty of SystemMenu structure.

Declaration

```
property Caption: String;
```

Description

The Caption property specifies the text for menu item in the form's system menu, associated with current caption button on the title bar. If Caption is not specified, text from Hint property will be taken as item title.

See also

Hint property of CaptionButton.

5.14.3 Position

Applies to

FormHelp component as subproperty of SystemMenu structure.

Declaration

```
property Position: Word;
```

Description

The Position property determines the position for menu item in the system menu associated with current caption button on the form's title bar. Change the Position value to move current menu item by system menu.

Maximum value for Position property is the current maximum number of items in the system menu (since orders for menu items starts with 0).

See also

Separators property.

5.14.4 Separators

Applies to

FormHelp component as subproperty of SystemMenu structure.

Declaration

```
type
  TacMenuSeparators = set of (seBefore, seAfter);

property Separators: TacMenuSeparators;
```

Description

The Separators property specifies the separators for current menu item to separate it from previous and / or next menu items of the system menu. To set separator before menu item — set `seBefore` to True. To set separator after menu item — set `seAfter` to True.

Specify Separators to separate current menu item in the system menu from another menu items.

See also

Position property.

5.15 TextStyle

Applies to

FormHelp component.

Declaration

```
type
  TfhTextStyle = class
  published
    property PlainText: Boolean default False;
    property TagOpen: Char default '[';
    property TagClose: Char default ']';
  end;
```

Description

The TextStyle structure used to specify the style of text displayed by this component.

The PlainText property specifies whether you would like to use right-formatted text or just flat, plain text. TagOpen and TagClose properties used to specify prefered signs for brackets which identifies begin and end of tags used for rich formatting. By default every tag beginning with '**[**' sign and ending with '**]**' sign.

For example, when PlainText is False, TagOpen is '[' and TagClose is ']', following text:

```
[i]Hello[]  [red][b]World[def]!
```

will looks like:

Hello **World**!

Otherwise, if PlainText is True, the tags will not be used and users will see just flat text, including all tags.

5.15.1 PlainText

Applies to

FormHelp component as subproperty of TextStyle structure.

Declaration

```
property PlainText: Boolean;
```

Description

The PlainText property controls whether the text should be displayed as rich text (if PlainText is False) or as plain, non-formatted text (if PlainText is True). When PlainText is True, the text will be displayed as usual flat text and nor tags will be used.

See also

TagOpen and TagClose properties;
Using tags in context-sensitive help topic.

5.15.2 TagClose

Applies to

FormHelp component as subproperty of TextStyle structure.

Declaration

```
property TagClose: Char;
```

Description

The TagClose property specifies the sign which marks the ending of formatting area of rich-text. The rich formatting tags of AppControls pack are similar to brackets "<>" of tags in HTML format. Default TagClose value is "**]**" sign.

Examples of standard tags
`[b]` - makes text after this tag **bold**.
`[i]` - makes text after this tag *italic*
`[biu]` - makes text after this tag ***bold, italic and underlined***.
`[]` - clears text formatting.
`[red]` - makes text after this tag red.
`[lime]` - makes text after this tag light green (lime).
`[def]` or `[default]` - returns the text formatting to default state as specified in Font property.

Note
If PlainText property is True, nor rich text will be displayed. All formating will be shown as usual flat text.

See also
TagOpen and PlainText properties and example of rich-text formatting of context help.

5.15.3 TagOpen

Applies to
FormHelp component as subproperty of TextStyle structure.

Declaration
```
property TagOpen: Char;
```

Description
The TagOpen property specifies the sign which marks the beginning of rich-text formatting area. The rich formatting tags of AppControls pack are similar to brackets "<>" of tags in HTML format. Default TagOpen value is "**[**" sign.

Examples of standard tags
`[b]` - makes text after this tag **bold**.
`[i]` - makes text after this tag *italic*
`[biu]` - makes text after this tag ***bold, italic and underlined***.
`[]` - clears text formatting.
`[red]` - makes text after this tag red.
`[lime]` - makes text after this tag light green (lime).
`[def]` or `[default]` - returns the text formatting to default state as specified in Font property.

See also
TagClose and PlainText properties and example of rich-text formatting of context help.

5.16 UseF1

Applies to
FormHelp component.

Declaration
```
property UseF1: Boolean;
```

Description
The UseF1 property specifies whether the FormHelp should invoke the context-sensitive help from the control under mouse pointer when user press F1 key.

Set UseF1 to True to let the FormHelp to process F1 key presses, or set it to False to disable this

feature.

5.17 WhatsThis

Applies to
FormHelp component.

Declaration
```
type
  TfhWhatsThis = class
  published
    property Enabled: Boolean;
    property Caption: String; // Caption for "What's this?" menu item.
Use this property for localization
    property MenuItem: TMenuItem;
    property ToolbarButton: TControl;
  end;

property WhatsThis: TfhWhatsThis;
```

Description
The **WhatsThis** structure used to specify whether the FormHelp component supports the "What's This?" feature (whether every control on the form contains the "What's This?" menu item associated with the context-sensitive help), to specify the Caption for this menu item (i.e. in the multilingual programs), and specify the external menu item and toolbar button which operates with the built-in context-sensitive help.

When the Enabled property is True, the FormHelp will add the "What's This ?" menu item to every control of the form with context-sensitive help in the secondary part of Hint property. When user selects the "What's This ?" menu item, the context-sensitive help associated with the control will appears.

Screenshoot

5.17.1 Caption

Applies to
FormHelp component as subproperty of WhatsThis structure.

Declaration
```
property Caption: String;
```

Description
The Caption property is the caption for "What's This?" popup menu item used to invoke the context-sensitive help. Change the Caption property to translate the "What's This?" question to another language in the international programs.

5.17.2 Enabled

Applies to
FormHelp component as subproperty of WhatsThis structure.

Declaration

```
property Enabled: Boolean;
```

Description

The Enabled property controls whether the FormHelp should apply the "What's This ?" menu item to every control with the context-sensitive help. Set Enabled to True to add the "What's This ?" popup menu to controls with context-sensitive help or False otherwise.

5.17.3 MenuItem

Applies to

FormHelp component as subproperty of WhatsThis structure.

Declaration

```
property MenuItem: TMenuItem;
```

Description

The MenuItem property points to the "What's This?" menu item (in any menu, main or popup). When user clicks this menu item, the cursor will be changed to a question mark with a pointer.

If the user then clicks a control in the form, the control receives a WM_HELP message to show the context-sensitive help taken from secondary part of the Hint property.

 When you point this property to the menu item, you don't need to handle OnClick event of this element of menu. The FormHelp will handle if for you automatically. However, if you need to specify *many* "What's this?" menu items — call the InvokeHelpPointer method in the OnClick event handler of this menu item, without pointing this property to it.

See also

ToolbarButton property and InvokeHelpPointer method.

5.17.4 ToolbarButton

Applies to

FormHelp component as subproperty of WhatsThis structure.

Declaration

```
property ToolbarButton: TControl;
```

Description

The ToolbarButton property points to the "What's This?" button on the toolbar (or any control, in general). When user clicks this button, the cursor will be changed to a question mark with a pointer.

If the user then clicks a control in the form, the control receives a WM_HELP message. The FormHelp component hooks this message to show the context-sensitive help taken from secondary part of the Hint property.

 When you point this property to the control, you don't need to handle OnClick event of this control. The FormHelp will handle if for you automatically. However, if you need to specify *many* "What's this?" buttons — just call the InvokeHelpPointer method in the OnClick event handler of this control, without pointing this property to it.

Snapshot

See also
MenuItem property and InvokeHelpPointer method.

6 Methods

6.1 InvokeFormHelp

Applies to
FormHelp component.

Declaration
```
procedure InvokeHelpPointer;
```

Description
The InvokeHelpPointer method changes the cursor to a question mark with a pointer. If the user then clicks a control in the form, the control receives a message to show the context-sensitive help taken from secondary part of the Hint property.

Example
```
procedure TForm1.Button2Click(Sender: TObject);
begin
  FormHelp1.InvokeHelpPointer;
end;
```

The InvokeHelpPointer procedure is equal to SendMessage(FormHandle, WM_SYSCOMMAND, SC_CONTEXTHELP, 0). So example above is equal to following code:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  SendMessage(Handle, WM_SYSCOMMAND, SC_CONTEXTHELP, 0);
end;
```

See also
MenuItem and ToolbarButton properties of the WhatsThis structure.

6.2 ShowHelp

Applies to
FormHelp component.

Declaration
```
procedure ShowHelp(Help: String);
```

Description
The ShowHelp method displays the popup window with text specified by Help string parameter from current mouse position. Text string can contain either plain or rich text.

Example
```
procedure TForm1.Whatsthis1Click(Sender: TObject);
```

```
begin
  acFormHelp1.ShowHelp('Hello world!');
end;
```

See also
ShowHelpFromControl and ShowHelpFromPoint methods;
Rich formatting of context-sensitive help.

6.3 ShowHelpFromControl

Applies to
FormHelp component.

Declaration
```
procedure ShowHelpFromControl(Control: TControl);
```

Description
The ShowHelpFromControl method displays the popup window with context-sensitive help, from any visible control specified by Control parameter. Text for popup window will be taken from secondary part of Hint property of specified Control.

Example
```
procedure TForm1.Button2Click(Sender: TObject);
begin
  FormHelp.ShowHelpFromControl(Button2);
end;
```

See also
ShowHelp and ShowHelpFromPoint methods.

6.4 ShowHelpFromPoint

Applies to
FormHelp component.

Declaration
```
procedure ShowHelpFromPoint(ShowPoint: TPoint; Help: String);
```

Description
The ShowHelpFromPoint method displays the popup window with context-sensitive help, from point specified by ShowPoint parameter and text specified by Help string.

Example
```
procedure TForm1.Whatsthis1Click(Sender: TObject);
var
  Point: TPoint;
begin
  Point.X := Left + 150;
  Point.Y := Top + 100;
  FormHelp.ShowHelpFromPoint(Point, 'Hello world!');
end;
```

See also
ShowHelp and ShowHelpFromControl methods;
Rich formatting of context-sensitive help.

7 Events

7.1 OnButtonClick

Applies to
FormHelp component.

Declaration
```
property OnButtonClick: TNotifyEvent;
```

Description
The OnButtonClick event occurs when user click additional button  on form's title bar. This event, unfortunately, does not works with regular button.

See also
CaptionButton property, OnShow and OnHide events.

7.2 OnHide

Applies to
FormHelp component.

Declaration
```
type
  TFormHelpOnHideEvent = procedure(Sender: TObject; HelpControl:
TControl) of object;

property OnHide: TFormHelpOnHideEvent;
```

Description
The OnHide event occurs before popup window with context-sensitive help is about hiding. Write the OnHide event handler to make some special processing before the help window disappears.

See also
OnShow event.

7.3 OnShow

Applies to
FormHelp component.

Declaration
```
type
  TFormHelpOnShowEvent = procedure(Sender: TObject; HelpControl:
TControl; var HelpMessage: String) of object;

property OnShow: TFormHelpOnShowEvent;
```

Description
The OnShow event occurs before popup window with context-sensitive help became visible. Write the OnShow event handler to make some special processing before the help window appears on screen.

Example

```
procedure TForm1.FormHelpShow(Sender: TObject; HelpControl: TControl;
   var HelpMessage: String);
begin
  if HelpControl.Name = 'Edit2' then
   begin
    FormHelp.DelayTime := 2000; // Two seconds delay
    FormHelp.Color := $00FFFFEC;
    FormHelp.Font.Name := 'Arial';
    FormHelp.Font.Size := 8;
   end;
  if HelpMessage = '' then HelpMessage := 'Hello World!';
end;
```

See also

OnHide event.

7.4 OnWhatsThis

Applies to

FormHelp component.

Declaration

```
type
  TWhatsThisEvent = procedure(Sender: TObject; HelpControl: TControl;
MousePos: TPoint; var ShowPopup, ShowHelp: Boolean) of object;

property OnWhatsThis: TWhatsThisEvent;
```

Description

The **OnWhatsThis** event occurs when user clicks right mouse button (*right clicks*) on the control with context-sensitive help in the secondary part of Hint property.

Use *HelpControl* parameter to get pointer to control from wich with the context-sensitive help was taken. The coordinates where user clicks the mouse button passed with *MousePos* parameter.

If you don't want to invoke the "What's This ?" popup menu for some certain controls — make *ShowPopup* parameter False in the event handler (see example below). However, even if you don't want to show the popup menu but still would like to show the context-sensitive help, then make *ShowHelp* parameter True.

Example

```
procedure TForm1.FormHelp1WhatsThis(Sender: TObject;
  HelpControl: TControl; MousePos: TPoint; var ShowPopup,
  ShowHelp: Boolean);
begin
  if HelpControl is TLabel then
   begin
    ShowPopup := False; // don't invoke "What's This?" for all labels
    if HelpControl = Label1 then
     ShowHelp := True; // show the context-sensitive help for "Label1"
on right click
   end;
end;
```

See also

WhatsThis structure.

8 Text-formatting tags

If you would like to make rich-text formatting of your context sensitive help you may:
1. Use built-in WYSIWYG context-sensitive help designer (drop acFormHelp onto your form and try to edit secondary part of Hint property of any visible control like TButton, TCheckBox or TGroupBox).
2. Make formatting manually in the secondary part of Hint property of any visible control, using special tags:

Style tags *(for changing the style attributes)*:
`[B]` - Bold font. Example: "[b]**Hello world**[]"
`[I]` - Italic font. Example: "[i]*Hello world*[] "
`[U]` - Underlined font. Example: "[u]Hello world[]"
`[S]` - Striked font. Example: "[s]~~Hello world~~[]"
`[]` - normal, regular font
You can also combine several stiles simultaneously:
`[BI]` - Bold+Italic. Example: "[bi]***Hello world***[]"
`[US]` - Underline+Striked Example: "[us]~~Hello world~~[]"
`[BIUS]` - Bold+Italic+Underline+Striked Example: "[bius]~~***Hello world***~~[]"

Color tags *(for changing the text colors)*:
`[Black]` Example: the [black]FormHelp[def] component
`[Maroon]` Example: the [maroon]FormHelp[def] component
`[Green]` Example: the [green]FormHelp[def] component
`[Olive]` Example: the [olive]FormHelp[def] component
`[Navy]` Example: the [navy]FormHelp[def] component

`[Purple]` Example: the [purple]FormHelp[def] component
`[Teal]` Example: the [teal]FormHelp[def] component
`[Gray]` Example: the [gray]FormHelp[def] component
`[Silver]` Example: the [silver]FormHelp[def] component
`[Red]` Example: the [red]FormHelp[def] component
`[Lime]` Example: the [lime]FormHelp[def] component

`[Yellow]` Example: the [yellow]FormHelp[def] component
`[Blue]` Example: the [blue]FormHelp[def] component
`[Fuchsia]` Example: the [fuchsia]FormHelp[def] component
`[Aqua]` Example: the [aqua]FormHelp[def] component
`[White]` Example: the [white]FormHelp[def] component

`[Def]` or `[Default]` - these tags will return text color and style to default state, as specified in Font property.

Note
If PlainText property is True, nor rich text will be displayed. All formating will be shown as usual flat text.

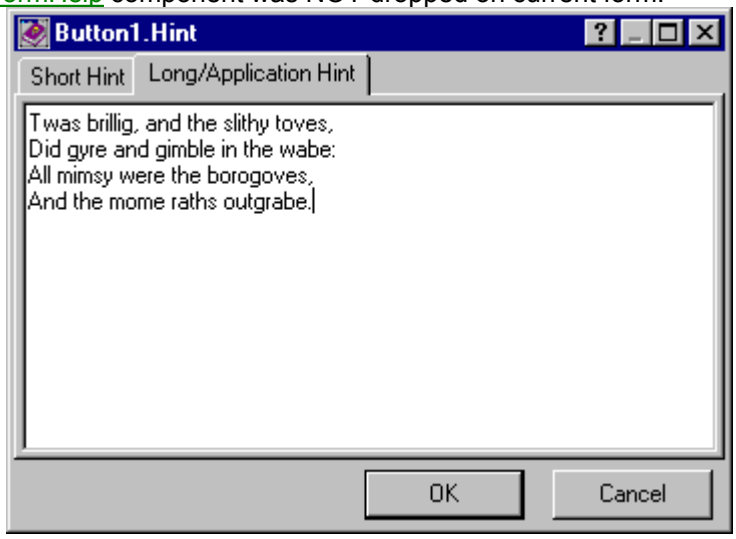
See also
TagOpen, TagClose and PlainText properties.

9 FormHelp designer / Hint property editor

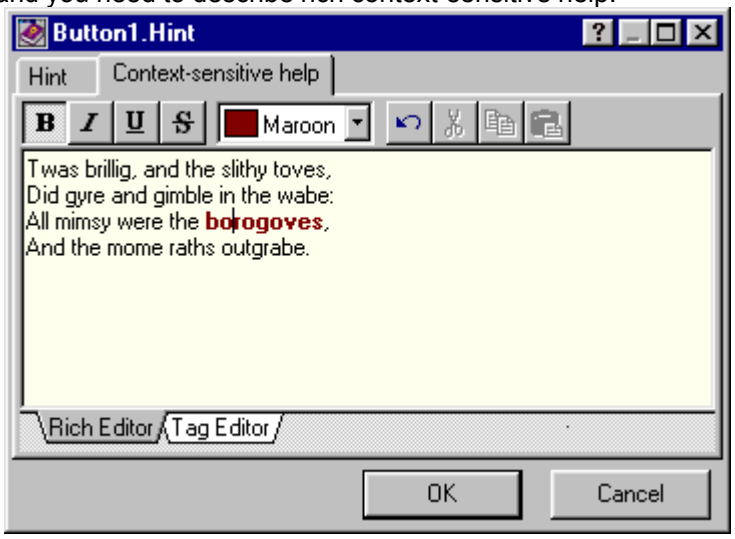
When you installing AppControls / acFormHelp, the FormHelp designer will be installed to the Delphi IDE additionally as new property editor for all "Hint" properties (doubleclick the Hint line in Object Inspector to see the designer).

FormHelp Designer let you to

1. Edit first and secondary part of hint separately, without separating parts by "|" character — if FormHelp component was NOT dropped on current form:



2. Edit Hint and Context-sensitive help — if FormHelp component has been dropped on current form and you need to describe rich context-sensitive help:



The "FormHelp designer" will overwrite all previously installed editors for the Hint property.

See also

Description of the FormHelp component.

10 Application.Hint problem

All previous FormHelp versions (before v3.0), used first part of hint for displaying the context-help. This makes a big problem, because forms with FormHelp could not show normal hints. To avoid this problem, FormHelp v3.0+ uses only secondary part of hint, also known as Application.Hint. However, unfortunately, sometimes this may lead to another problem.

For example, you are using the *Application.OnHint* event to display the secondary part of hint in the status bar. In this case, the status bar will display the context-sensitive help but with all formatting tags:

To avoid new problem you may use following code:

```
// let's say you would like hook the Application hints... ShowHint
procedure described in the public section of TMainForm class
procedure TMainForm.FormCreate(Sender: TObject);
begin
  Application.OnHint := ShowHint;
end;
procedure TMainForm.ShowHint(Sender: TObject);
begin
  { for example, status bar contains several sections to display  some
useful information but we would like to switch it to the  simple panel
mode to show second part of hint for menu items }
  if (Length(Application.Hint) > 0) and // if Application.Hint is not
empty
     (Copy(Application.Hint, 1, 1) <> '[') then { super kludge. '['
character means that second part of hint is the context-sensitive help.
The '[' sign opens the tag for text formatting. In this case we don't
want to display this text in the status bar. }
   begin
    StatusBar.SimplePanel := True;
    StatusBar.SimpleText := Application.Hint;
   end
  else StatusBar.SimplePanel := False;
end;
```

We really think that second part is more convenient for context-sensitive help than first, because it let us to show commonly used regular hints as well. For example, we have a toolbar with some buttons, and we would like to show either normal hints and the context-sensitive help. Third version allows to do this. However, if the Application.Hint problem critical for you, we appreciate some feedback with ideas and suggestions! Email us: info@appcontrols.com