Demystifying Help

Part 2

By Kevin S. Gallagher

WinHelp windows

When creating WinHelp you have several choices of windows for displaying help topics in, standard, secondary and popup. Within the secondary style there are several alterations, which are influenced by color, size and position. Both standard and secondary windows are by default sizeable and also moveable were popup windows close when clicking the left mouse button in the client area of the popup.

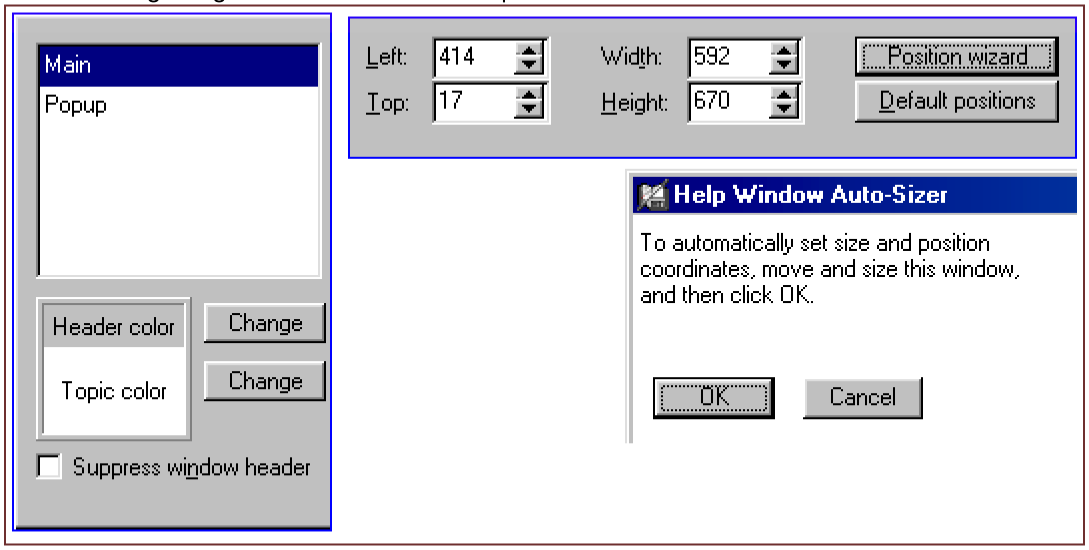
By default the foreground and background colors for the windows in your help project are defined by you, but the user can override the colors via a menu option in WinHelp, they can also change the font size too. With this in mind I would suggest using drab black on white to display help topics in main windows along with at least a 9-point font such as Arial for contents and 10-point or greater for the topic heading. This makes the easy to read and keeps users from changing to some horrible color scheme which many times works against the user. Keeping things simple makes using help (hopefully) a pleasant experience.

Window type	**Description**
Main window	Main Help application window, which contains the menu bar and button bar and that most Help files use to display topics. The main Help window can include a nonscrolling region.
Secondary window	Independent windows similar to the main Help window, except secondary windows lack a menu bar and button bar. Secondary windows can include a nonscrolling region.
Pop-up window	Temporary windows that appear in a fixed size and location over the main Help window or a secondary window.
Embedded window	Rectangular windows that appear in a fixed size and location within another Help window and use a

How to create windows:

Let's begin with the medieval method, using Microsoft's help workshop. The process begins with pressing a button called *Windows*, which presents a dialog box with many options to set. None of the options for the window provide direct feedback other the colors, you must try the window out by compiling the project. Just another reason for purchasing a third party tool to create help. Now for a look at how to change the attributes of a help windows using professional tools.

The following images were taken from Help and Manual toolkit



Main
Popup

Header color
Topic color

Change
Change

☐ Suppress window header

Left: 414
Top: 17
Width: 592
Height: 670

Position wizard
Default positions

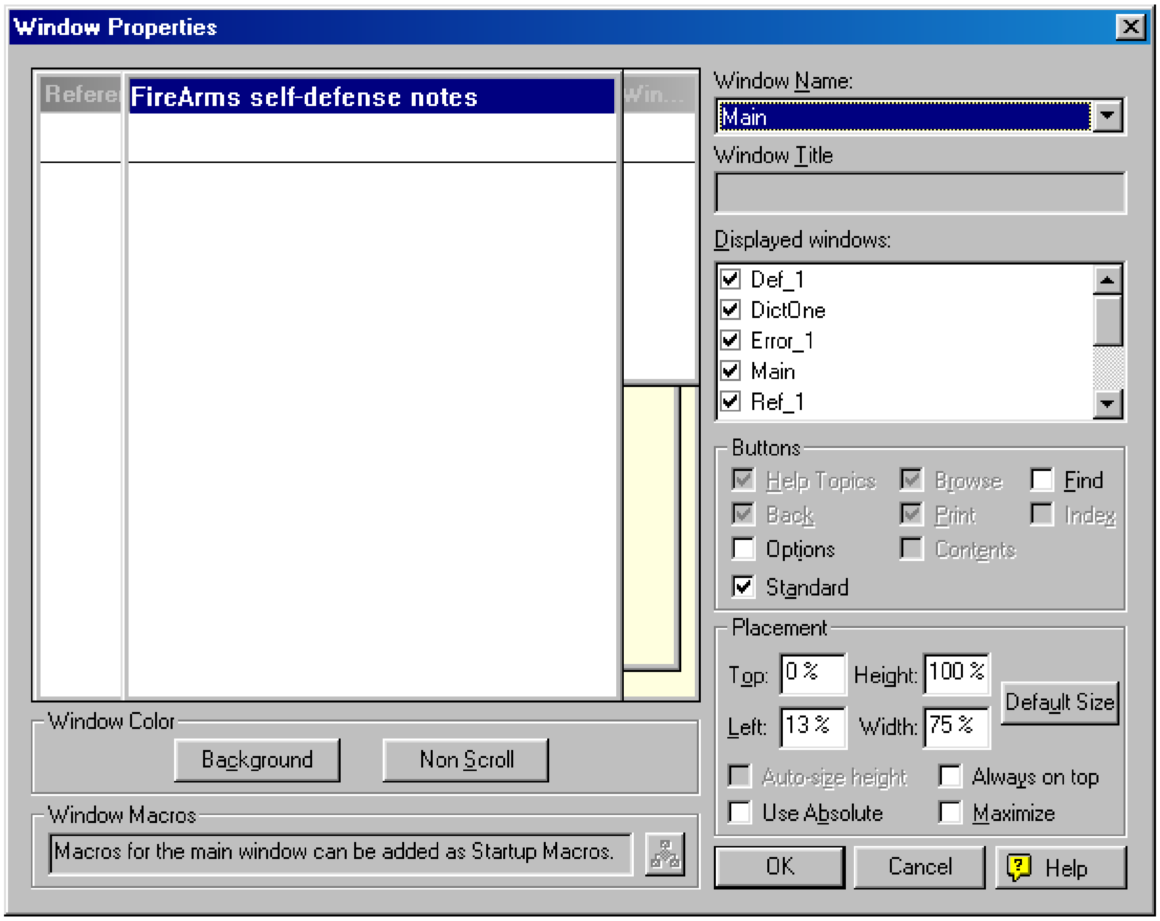
Help Window Auto-Sizer

To automatically set size and position coordinates, move and size this window, and then click OK.

OK
Cancel

The left image shows a small box depicting colors to be used for the selected window, simply press the button to the right which calls up a color dialog for changing the background color for either the header or contents section of the window. The checkbox below if checked will remove the header section from the window which translates into using the same color as the topic area. The top right window visually displays the coordinates for the chosen window and also provides two buttons for setting the width, height and size. Pressing the *Position Wizard* button displays a window, which you can size and position. Once satisfied with the results you press the OK button to save the changes or abort by pressing Cancel button.

RoboHelp provides the same methods as in “Help and Manual” but in a slightly different style as shown below:



Window Name: Main

Window Title:

Displayed windows:
- [✓] Def_1
- [✓] DictOne
- [✓] Error_1
- [✓] Main
- [✓] Ref_1

Buttons
- [✓] Help Topics
- [✓] Browse
- [ ] Find
- [✓] Back
- [✓] Print
- [ ] Index
- [ ] Options
- [ ] Contents
- [✓] Standard

Placement

Top: 0 % Height: 100 %

Left: 13 % Width: 75 %

- [ ] Auto-size height
- [ ] Always on top
- [ ] Use Absolute
- [ ] Maximize

Macros for the main window can be added as Startup Macros.

Note: In the bottom left corner of the RoboHelp windows dialog is a place to embedded help macros into the selected window (other then the main window which is done in project options).

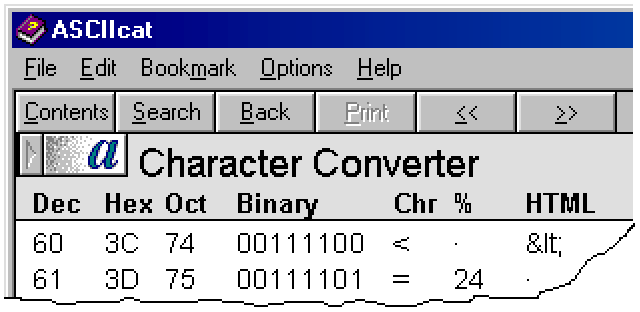
Colors of two most common help windows

Main windows	Black text with white background
Secondary windows	Black text with off-yellow background

There are reasons which the above colors were selected, remember this is not a contest to make things pretty, instead allow the users to easily read the contents of your help topics. When using other colors it would be wise to review them with selected users or testers. Also you might check to see if there are color impaired users which will use the help file and if so have them review the colors.

Non-Scrolling regions

This is the top portion of main and secondary help windows were the text within this area remain stationary, independent of the topic text. Normally only the title of the help topic is placed into the non-scrolling region, other times it makes sense to include more then just the title, for instance in the topic below there is a table, if the column headings were not placed into the non-scrolling region they would not be visible when scrolling through the help topic. Placing the column headings into the non-scrolling region provides a reference point for the data in the columns.

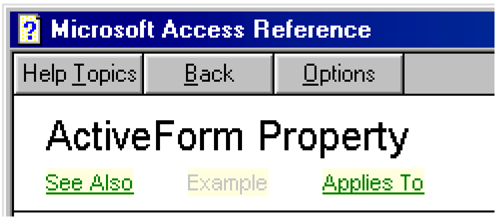

ASCIIcat
File Edit Bookmark Options Help
Contents Search Back Print << >>
Character Converter

Dec	Hex	Oct	Binary	Chr	%	HTML
60	3C	74	00111100	<	·	&lt;
61	3D	75	00111101	=	24	·

An interesting point on the help file used here, the arrow and the letter "A" in the upper left corner of the window have *hot-spots* to other topics, yet another use for non-scrolling regions. If you used older versions of Delphi then you have seen "properties", "methods" and "events" in the non-scrolling regions, which are *hot spots* to other topics.

When placing text into topics in your help files it is best to keep text to a minimum amount, do not overload this area. One reason not to place a large amount of text into a non-scrolling region is that users with small monitors will be restricted to how much of a help topic's text will be visible. Keep text in non-scrolling regions to the title and possibly "See-Also" or "Related-Topics" jumps.

The following example shows a help topic with the title and "See-Also" and "Applies-to" jumps as places visit.



Notice that the jump for "Example" is not enabled, this keeps things consistent through out the help file.

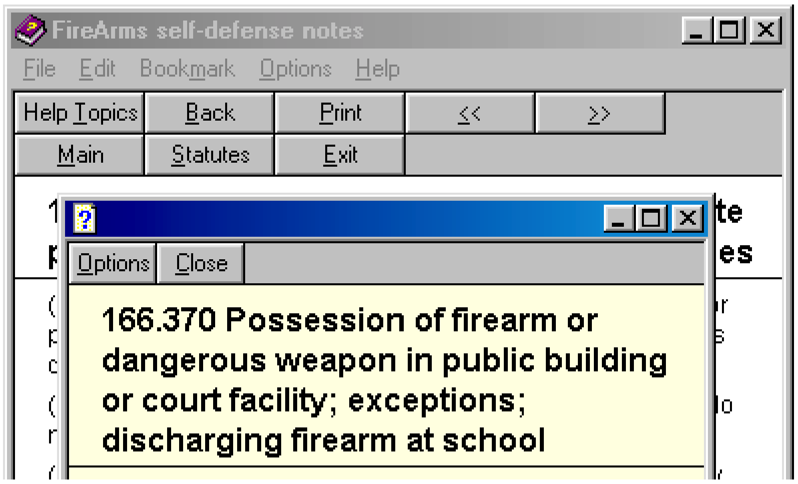
Important note Make sure when applying a non-scrolling region to your help topics that they are not for *popup* style help topics, this includes *What's this* style help also. If a non-scrolling region is applied to either of these types of topics you will not see the topics text, only the title of the topic. This is a good reason to plan out help topics purposes prior to writing them.

Size and position of help windows

For the main window, it should be large enough that you do not force the users to resize the window to view the entire contents of your help topics. If this is not possible then it would be wise to restructure some or all topics (review part one of this article on *layering*). Under Windows95 and higher the main windows size and position are saved upon closing the specific help file, this is not true with secondary windows. If you have some help files with secondary windows which you didn't create and would like to change their size and position, go to http://www.ec-software.com! and search for a utility called "HelpPos". The utility permits you to alter help windows, which the source code is not available to you.

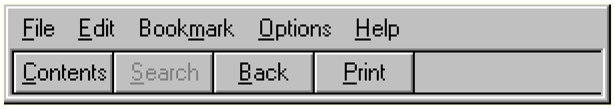
Secondary windows

These types of windows are called from the main window in a help file. They are all essentially the same, what makes them different is, placement, size and color. Other differences from a main window are the menu and buttons. The most common use for secondary windows is to display another topic while keeping the current topic open or for displaying the definition of something within the current topic. Note, if the definition is short then a popup window might be a better chose for a window. Secondary windows can have the same menu items and buttons as defined in the main window but generally are not needed. Below is shows a secondary window over the main window of a help file.


FireArms self-defense notes
File Edit Bookmark Options Help
Help Topics Back Print << >>
Main Statutes Exit
Options Close

166.370 Possession of firearm or dangerous weapon in public building or court facility; exceptions; discharging firearm at school

Notice in the secondary window all, which can be done, is to access *options* and a button to close the current window. This is not a steadfast rule, I have noticed several help files which provide more buttons then needed which end up confusing the users. Speaking of buttons, make sure your testers try them out to insure they do what you designed them for. For instance, suppose you create a button to close the current window, make sure the name provided in the macro is the name of the current window and not another window. So why a close button? Believe it or not many users do not know what the X button will close a window. Secondly what if you decide that you do not want the caption area so what more information can be displayed (as shown below).



This is done using WinHelp macros.

Macro Declarations
RegisterRoutine(`user',`SetWindowLong',`UiU')
RegisterRoutine(`user',`SetWindowPos',`Uuiiiiu')
RegisterRoutine(`user',`GetActiveWindow',`u=')

Macro Syntax
SetWindowLong(GetActiveWindow(),-16,style):SetWindowPos(GetActiveWindow(),0,0,0,0,0,39)

The macro definitions are stored in the projects project file while the macro is stored (in this case) in a window. Each window (except for popup windows) can have what is called an *entry macro*.

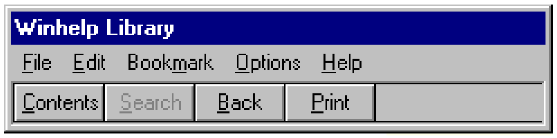
CONSTANTS (STYLE)

WS_OVERLAPPED	&H0&	WS_BORDER	&H800000
WS_POPUP	&H80000000	WS_DLGFRAME	&H400000
WS_CHILD	&H40000000	WS_VSCROLL	&H200000
WS_MINIMIZE	&H20000000	WS_HSCROLL	&H100000
WS_VISIBLE	&H10000000	WS_THICKFRAME	&H40000
WS_DISABLED	&H8000000	WS_GROUP	&H20000
WS_CLIPSIBLINGS	&H4000000	WS_TABSTOP	&H10000
WS_CLIPCHILDREN	&H2000000	WS_SYSMENU	&H80000
WS_MAXIMIZE	&H1000000	WS_MINIMIZEBOX	&H20000
WS_CAPTION	&HC00000	WS_MAXIMIZEBOX	&H10000

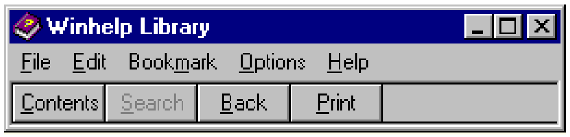
NOTES

1) Winhelp Main Window: these macros works fine as Project HPJ entries in [Config] section, as well as Topic entries in RTF files, both in Windows 3.1 and Windows 95
2) Winhelp Secondary Windows: these macros works fine in Windows 3.1 and higher.
3) In Windows 95 the parameter `user' should be changed to `user32'

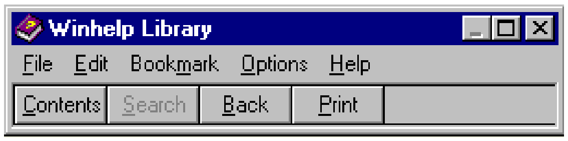
Here are some modified examples of help windows



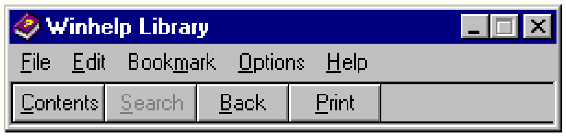
Title bar	Yes
Max button	No
Min button	No
Control box	No
Resize	Yes
Border	Yes
Style	0x16C70000



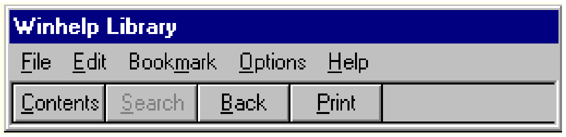
Title bar	Yes
Max button	Yes
Min button	Yes
Control box	Yes
Resize	No
Border	yes [95]/ no [3.1]
Style	0x16CB0000



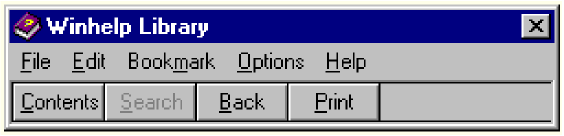
Title bar	Yes
Max button	Yes
Min button	Disabled
Control box	Yes
Resize	Yes
Border	Yes
Style	0x16CD0000



Title bar	Yes
Max button	Disabled
Min button	Yes
Control box	Yes
Resize	No
Border	yes [95] / no [3.1]
Style	0x16CA0000



Title bar	Yes
Max button	No
Min button	No
Control box	No
Resize	No
Border	yes [95] / no [3.1]
Style	0x16C00000



Title bar	Yes
Max button	No
Min button	No
Control box	Yes
Resize	No
Border	yes [95] / no [3.1]
Style	0x16C80000

When thinking about using the above styles, make sure there is a valid reason, don't use them just for showing off.

Before moving on to popup windows let's take a look at other things, which can be done to help windows. You can farther alter both main and secondary windows by removing their menu and or button bar.

Macro Declarations for Button Bar
RegisterRoutine(`user',`FindWindow',`u=SS')
RegisterRoutine(`user',`GetWindow',`u=uu')
RegisterRoutine(`user',`DestroyWindow',`u')
RegisterRoutine(`user',`PostMessage',`uuuU')

Macro Syntax for Button Bar
DestroyWindow(GetWindow(GetWindow(GetWindow(GetWindow(FindWindow(`MS_WINDOC', `TitleOfTheWindow'), 5), 1), 1), 1))') :
PostMessage(FindWindow(`MS_WINDOC',`TitleOfTheWindow'), 0x0100, 91, 1) :
PositionWindow(30, 30, 982, 852, 1, `main')

Macro Declarations for menu
RegisterRoutine(`user',`GetMenu',`U=U')
RegisterRoutine(`user',`GetActiveWindow',`U=')
RegisterRoutine(`user',`DeleteMenu',`U=U')
RegisterRoutine(`user',`SetMenu',`Uu')

Macro Syntax for menu
DeleteMenu(GetMenu(GetActiveWindow())) : SetMenu(GetActiveWindow(),0)

As in earlier examples the macro definitions need to be placed into the project file similar to declaring functions unknown to Delphi in the interface portion of a unit.

Notes:
1. Parameter TitleOfTheWindow must be exactly the same text which appears in the Title Bar of the window, as defined in [Window] section
2. In PositionWindow macro, third / fourth parameters must be slightly different than the original Width / Height defined in [Window] section, to force proper redraw of the window

Going off the deep end, although macros have not be covered yet, below shows code to toggle a windows title/caption bar on/off (not a pleasant sight and not for the faint of heart). Macros will be covered shortly.

FThenElse(IsMark(`T'), DeleteMark(`T'): SetWindowLong(hwndApp, -16, 0x16480000): SetWindowPos(hwndApp, 0, 0, 0, 0, 0, 0x0027), SaveMark(`T'): SetWindowLong(hwndApp, -16, 0x16CB0000): SetWindowPos(hwndApp, 0, 0, 0, 0, 0, 0x0027))

Creating Jumps to Secondary Windows

After you have defined a secondary window for your Help file, you will probably want to create hot spots with the topics that access the secondary window. In most cases, you can create a hot spot that causes a jump:

- From the main Help window to a secondary window.
- From the secondary window back to the main window.
- To a secondary window in a different Help file.

Creating Standard Jumps to Secondary Windows
You create jumps to topics in secondary windows the same way you create standard jumps. The difference is that you must include the name of the secondary window in the hot spot information, as shown here:

context-string>window-name
Context-string is the context string for the topic you are jumping to, and window-name is the name of the secondary window where you want the topic to appear.
Many of the currently available help authoring tools provide wizards to write the code to do jumps to secondary windows it is not a bad idea to know how this works. There have been times when I have ran into wizards thinking on their own, in turn invoking compiler errors.

To create a jump to a secondary window

1. Follow the steps to create a standard jump hot spot.
2. Insert a right angle bracket (>) immediately after the last character of the context string.
3. Type the name of the secondary window where you want the topic to appear.

Note The angle bracket and window name must be formatted as hidden text.

Creating Jumps to the Main Help Window
Generally, if you create one or more secondary windows, you will also need to create hot spots in secondary window topics that jump back to the main window. To do that, you use the same jump format and specify "main" as the window-name, as in this example:

context-string>main
The window name "main" is reserved for the main Help window and must be used when specifying jumps to the main window.

Jump location	**Type of jump**	**Result of the jump**
Main window	standard	The topic is displayed in the main Help window
Main window	>main	The topic is displayed in the main Help window
Main window	>window-name	The topic is displayed in the specified secondary window. If Help opens a new secondary window to display the topic, the previous size and location of the secondary window may change. The main Help window is unaffected by this type of jump.
Secondary window	standard	The topic is displayed in the specified secondary window. The main Help window is unaffected by this type of jump.
Secondary window	>main	The topic is displayed in the main Help window.
Secondary window	>window-name	The topic is displayed in the specified secondary window. If Help opens a new secondary window to display the topic, the previous size and location of the secondary window may change. The main Help window is unaffected by this type of jump.

Popup windows

Within topics, certain terms or concepts often require further explanation. Instead of having Windows Help jump away from the current topic to provide the additional information, you can have Help display the information in a pop-up window on top of the current topic. A pop-up window is a temporary window you can use to display subordinate or complementary information. Pop-up windows have many uses but are very effective when used to display any subordinate information that complements the information in the main topic. Here are a few of the most common uses.

You can use pop-up windows to:

- Display glossary definitions of difficult or special terms in the application software or Help file.
- Display example text or graphics of information that is explained in the main topic.
- Display a list of cross-references for the current topic information.
- Complement a hypergraphic by displaying a brief explanation of the object when the user chooses a hot spot.
- Provide quick context-sensitive Help on dialog box options, commands, and other interface elements in the application.
- Display examples, hints, tips, and notes.

How Pop-Up Windows Work

Pop-up windows are stripped-down windows that do not include any standard window elements, such as title bar, menu bar, Control menu, scroll bars, or even standard-width window borders. They do include a shadow to make them more distinguishable from the main window. Users cannot move or resize pop-up windows.

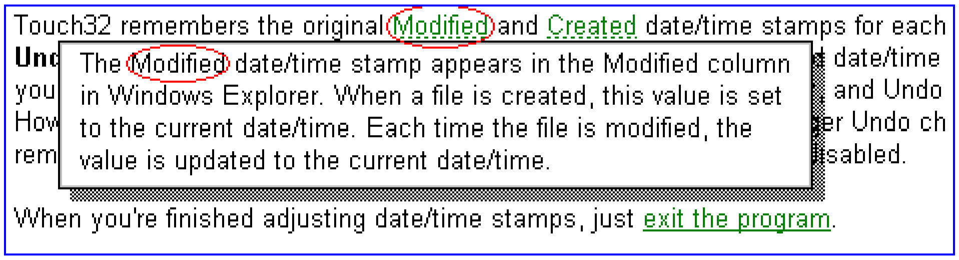
Popup-topics can include most standard Help features, including any mixture of text, graphics, hot spots (jump hot spots, pop-up hot spots, or macro hot spots), and embedded windows. However, if you have a hot spot within a pop-up window, the first pop-up window closes when the users chooses the hot spot. In other words, you can display only one pop-up window at a time on the screen. Topics displayed in pop-up windows cannot use topic-entry macros, but they can contain jumps and macros executed from hot spots.

Pop-up windows appear on demand, usually when the user chooses a menu item, button, or hot spot. A macro in the Help project file or a WinHelp API call can also be used to display a pop-up window. For information about using Help macros to display a pop-up window, see Chapter 15, "Help Macro Reference." For information about how an application can send an API call to Help in order to display a pop-up window, see Chapter 19, "The WinHelp API."
Pop-up windows remain open until the user clicks the mouse button or presses any key (which includes choosing a hot spot within the pop-up window).

When displaying pop-up windows, Help follows these guidelines:

- The pop-up window has approximately the same width as the Help window.
- The pop-up window has only as much vertical height as necessary to display all the text. If the pop-up topic contains more text than will fit in a full-size pop-up window, the user will not be able to see the hidden portion of the topic.
- Help attempts to position the pop-up window immediately below the hot spot so that the user can read both the text in the pop-up and the hot spot text that initiated the pop-up window.
- If the topic to be displayed in the pop-up window includes both a nonscrolling region and a scrolling region, Help displays the region that contains the context-string footnote (# footnote) in the pop-up window.

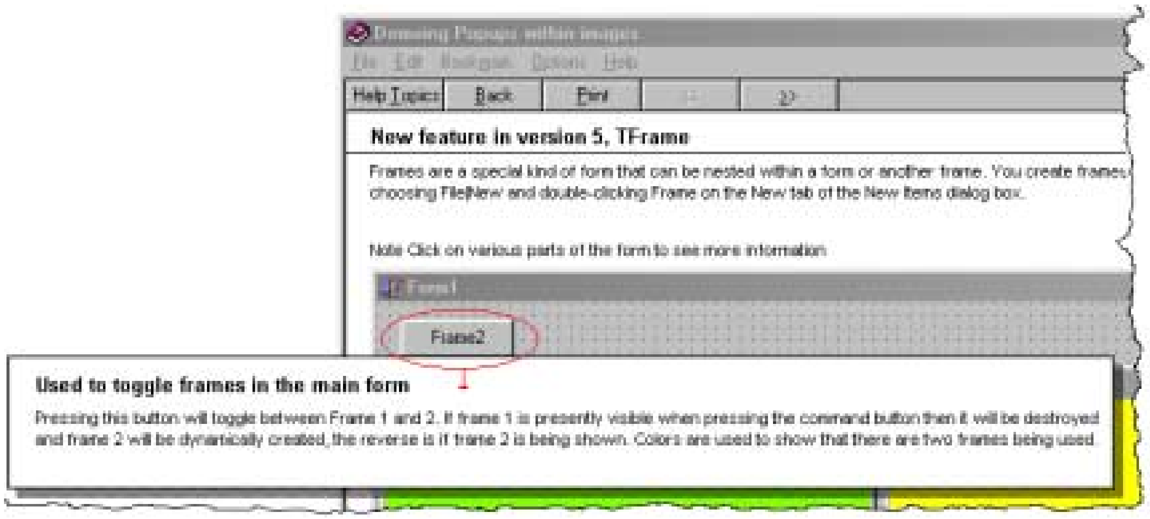
Example one
Here the author uses a popup window to supply information about a word in the context of the help topic.


Touch32 remembers the original Modified and Created date/time stamps for each

The Modified date/time stamp appears in the Modified column in Windows Explorer. When a file is created, this value is set to the current date/time. Each time the file is modified, the value is updated to the current date/time.

When you're finished adjusting date/time stamps, just exit the program.

Example two
Here a popup is used over an image, a framed window (secondary window) could have been used also but a popup is better suited in this situation.



New feature in version 5, TFrame

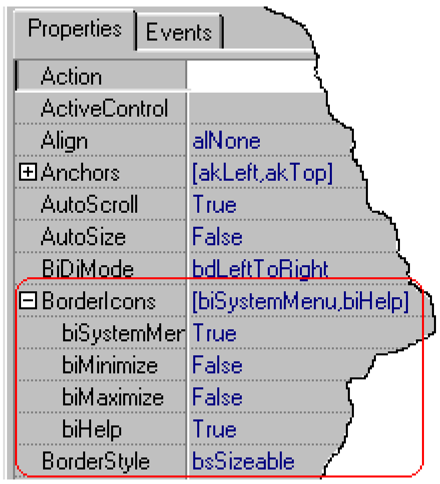
Frames are a special kind of form that can be nested within a form or another frame. You create frames choosing File|New and double-clicking Frame on the New tab of the New Items dialog box.

Note Click on various parts of the form to see more information

Used to toggle frames in the main form

Pressing this button will toggle between Frame 1 and 2. If frame 1 is presently visible when pressing the command button then it will be destroyed and frame 2 will be dynamically created, the reverse is if frame 2 is being shown. Colors are used to show that there are two frames being used

Yet another use for popup windows is for *What's this* style of help in your applications. This style of help can only be used on forms which have a border style of a dialog.

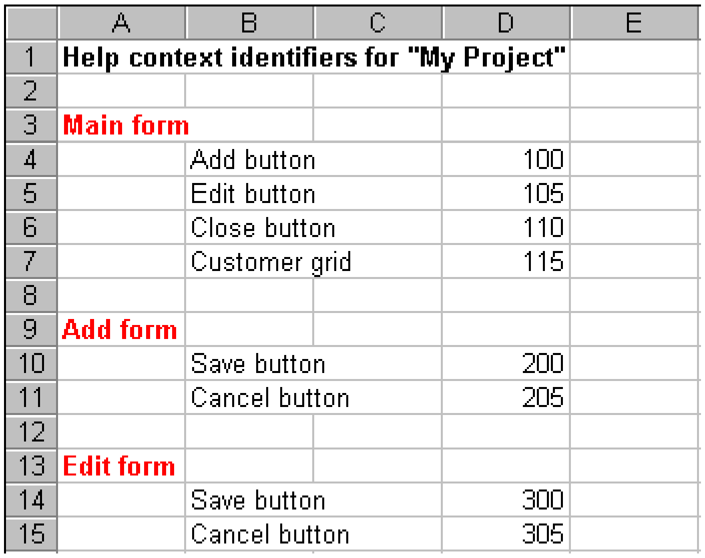


Action	
ActiveControl	
Align	alNone
⊞Anchors	[akLeft,akTop]
AutoScroll	True
AutoSize	False
BiDiMode	bdLeftToRight
⊟BorderIcons	[biSystemMenu,biHelp]
biSystemMer	True
biMinimize	False
biMaximize	False
biHelp	True
BorderStyle	bsSizeable

Note: If you were to add biMinimize and or biMaximize to the form's BorderIcon property no *What's this* button would appear in the form's title area.

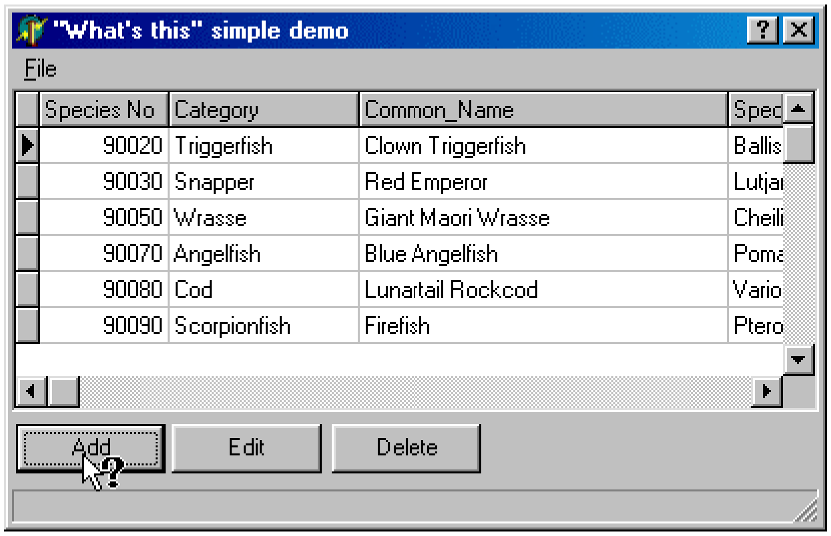
To implement this style of help

1. Assign a *Help Context ID* to each component on a form which will contain a *What's This* topic
   Tip: When assigning the *Help Context ID* create a spreadsheet and record the following; Form name, beneath the name record each control name along with the Help Context ID. Usually I will assign a number separated by hundred between each form. Each control increments by five, this leaves room for other controls, which may be added at a later time period.



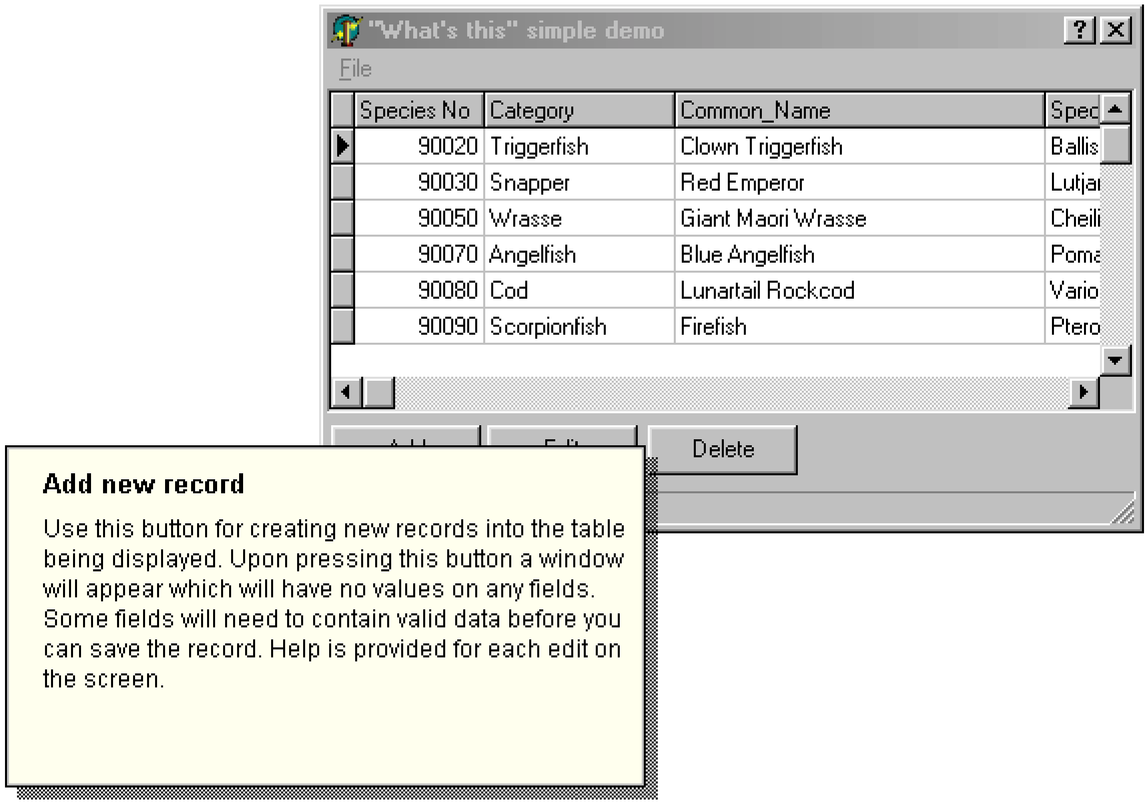
	A	B	C	D	E
1	**Help context identifiers for "My Project"**				
2					
3	**Main form**				
4		Add button		100	
5		Edit button		105	
6		Close button		110	
7		Customer grid		115	
8					
9	**Add form**				
10		Save button		200	
11		Cancel button		205	
12					
13	**Edit form**				
14		Save button		300	
15		Cancel button		305	

2. Write topics and assign the ID in step one to the Map ID in the newly created topic.
3. Change each form's BorderStyle so that they can not be minimize or maximized.
4. Assign the name of the help file to the project (under project options->Application tab).
5. Compile and run the project, single click on the question mark button in the title area of the form.



	Species No	Category	Common_Name	Spec
▶	90020	Triggerfish	Clown Triggerfish	Ballis
	90030	Snapper	Red Emperor	Lutjai
	90050	Wrasse	Giant Maori Wrasse	Cheili
	90070	Angelfish	Blue Angelfish	Poma
	90080	Cod	Lunartail Rockcod	Vario
	90090	Scorpionfish	Firefish	Ptero

6. Place the mouse cursor over a control, which has a *Help Context ID* assigned to it and single click the mouse. If all goes right you will get a PopUp window with the help topic.


"What's this" simple demo

File

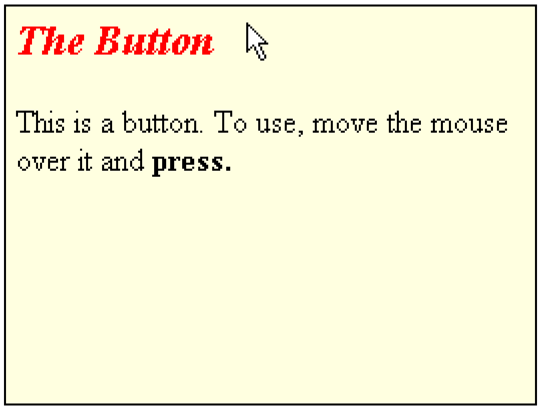
	Species No	Category	Common_Name	Spec
▶	90020	Triggerfish	Clown Triggerfish	Ballis
	90030	Snapper	Red Emperor	Lutjai
	90050	Wrasse	Giant Maori Wrasse	Cheili
	90070	Angelfish	Blue Angelfish	Pom
	90080	Cod	Lunartail Rockcod	Vario
	90090	Scorpionfish	Firefish	Ptero

Add new record

Use this button for creating new records into the table being displayed. Upon pressing this button a window will appear which will have no values on any fields. Some fields will need to contain valid data before you can save the record. Help is provided for each edit on the screen.

Caveat: In the above example there is no non-scrolling region, if there had been a non-scrolling region the help topic would not be shown, only the help topics title would appear. Many times help authors will create help topics that will be used in several different places, as What's this and also within a help file running outside of an application. This can pose problems, one method is to simply not use non-scrolling regions while another is to have duplicate topics.

For something a bit different, Harry Kakoulidis produced a component which is worthy to mention here. Harry came up idea which uses RTF codes placed onto a form with a TRichEdit component on the a form. Here is what it looks like

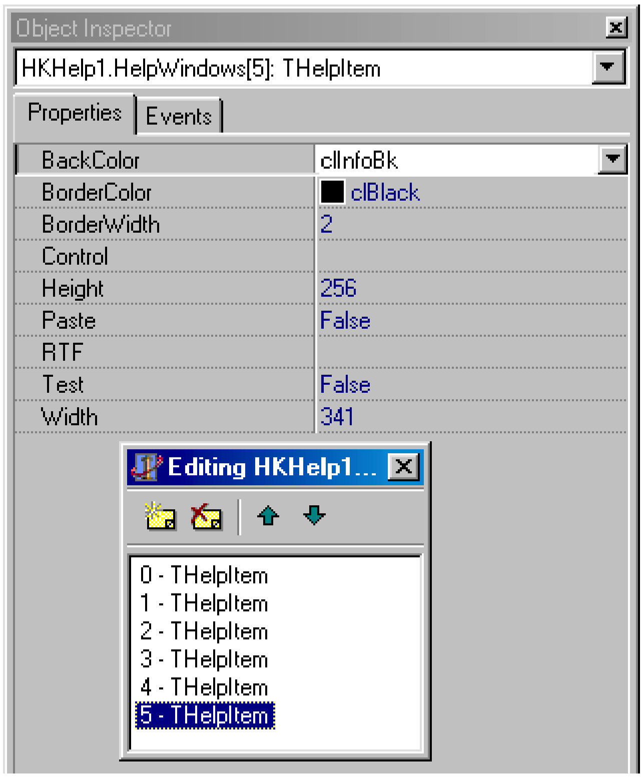

The Button

This is a button. To use, move the mouse over it and **press.**

To implement this style of help

1. At design time drop the component onto a form.

2. Define a window for each control which you want help for, associate a component on your form to this window.


Object Inspector

HKHelp1.HelpWindows[5]: THelpItem

BackColor	clInfoBk
BorderColor	■ clBlack
BorderWidth	2
Control	
Height	256
Paste	False
RTF	
Test	False
Width	341

Editing HKHelp1...

0 - THelpItem
1 - THelpItem
2 - THelpItem
3 - THelpItem
4 - THelpItem
5 - THelpItem

3. Write the help in MS-Word or any word processor, which does RTF.
4. Copy the text from step 3 into a property of the component.
5. Add *biHelp* to the BorderStyle of the form to show help in and make it of style dialog. The question mark in the form's caption area is used to trigger the component rather then then calling WinHelp. If the form is not a modal form then add a button to the form and add the following line of code → HKHelp1.Helpmode := **not** HKHelp1.HelpMode;
6. To test the help at design time simple right click on the component and select *Test*, the mouse cursor changes to the *What's this* style of cursor. Now single click the left mouse cursor over a control with help assigned to it.
7. Most likely you will need to adjust the width and height of the help window.

At run time the help windows may be moved around the screen simply by holding down the left mouse button anywhere on the form. Pressing ESC or single right click a help window will close them.

Comment: This component is a bit unorthodox and may not be accepted by all users. Before using such a component show it to some of the users of your application to get feedback.

For downloading or contacting the author of this freeware component (with full source)
THKHelp v1.1 by Harry Kakoulidis 1/2000
kcm@mailbox.gr
http://kakoulidis.homepage.com

More on *What's this* help can be found at http://www.ec-software.com within a freeware help file which covers many topics on creating help files. This site is the home of **Help and Manual** a help authoring tool.

RoboHelp caveat

RoboHelp provides a way to share windows, this is done be creating your windows then saving them in template. The template is saved as a file on your computer along with a description.

The Windows Help Coordinate System

When defining windows, you use the Help coordinate system to specify window position and size values. Help uses its own device-independent coordinate system because the screen resolution, or number of pixels, on Windows displays can vary. This device-independent system divides the screen into 1024 vertical units and 1024 horizontal units. When Help displays a Help file, it converts the size and position values specified by the Help file into values appropriate to the resolution of the display device. This allows Help to display Help windows using a consistent size and location despite variations in screen resolution.

The Help coordinate system is mapped to the horizontal and vertical resolutions of the video card, so you'll have to do some calculations to determine the exact pixel location. For example, if the resolution of your video card is horiz by vert pixels, and the horizontal and vertical locations you specify are at Windows Help coordinates x and y, then the x-coordinate of the upper-left corner is at the pixel given by the following formula:
x * (horiz/1024)

The y-coordinate of the upper-left corner is at the pixel given by the following formula:
y * (vert/1024)
The window's width and height are also specified in Help's coordinate system. For example, a window may specify a width of 511 and a height of 511. A standard VGA card has a resolution of 640-by-480 pixels, so the width is actually the following number of pixels:
511 * (640/1024) = 319
The height is actually the following number:
511 * (480/1024) = 240

In other words, the window would take up approximately one quarter of the area of the Windows Help screen (half the width multiplied by half the height).
To convert from pixels to Windows Help coordinates, you invert the ratio between Windows Help's resolution and the video resolution. Again, assuming the resolution of your video card is horiz by vert pixels, and the horizontal and vertical locations (or dimensions) you want are x and y pixels, the x-coordinate (or width), in Windows Help coordinates, is as follows:

x * (1024/horiz)
The y-coordinate (or height), in Windows Help coordinates, is:
y * (1024/vert)

Because the Windows Help coordinate system ranges from 0 through 1023 in both directions, the horizontal position plus the width must be less than or equal to 1023. Similarly, the vertical position plus the height must be less than or equal to 1023.