What is Help Workshop?

Help Workshop is a program that you use to create Help (.hlp) files, edit project and contents files, and test and report on help files. Help Workshop takes the information in the project (.hpj) file to combine the topic (.rtf) files, bitmaps, and other sources into one Help file that can be viewed using the Microsoft Windows Help program.

Help Workshop's executable name is hcw.exe.

{button ,AL(`BAS_COMPILE_MSG;HCW;HCRTF;TMPDIR')} Related Topics

Help Workshop components

Help Workshop includes the following files and documents:

- Help Workshop (Hcw.exe and Hcrtf.exe). Enables you to edit project and contents files, and to compile, test, and create reports for Help files.
- Help Author's Guide (Hcw.hlp). Describes how to author and compile Help files by using Help Workshop.
- Hotspot Editor version 2.0 (Shed.exe). Enables you to create a graphic that has multiple hotspots.
- Multi-Resolution Bitmap Compiler version 1.1 (Mrbc.exe). Enables you to create bitmaps that have different resolutions. You can combine these bitmaps into a single graphic to compensate for differences between the aspect ratios of the bitmaps and the aspect ratio supported by a user's display.

{button ,AL(`BAS_MULT_RESOLUTION;BAS_THE_HELP_COMPILER;MRBC;SHED')} Related Topics

Notational Conventions

This Help file uses the following conventions.

Convention	Use
Bold	Indicates options and commands specific to Help authoring (for example, the **CloseWindow** macro) and C language-specific terms and options (for example, the **WinHelp** function).
[brackets]	Encloses optional items in syntax statements. For example, [window-name] indicates a window name might be needed in the syntax statement. Type only the information within the brackets, not the brackets themselves.
{braces**}**	Encloses RTF statements in topic files. Type the bold braces and commands, and substitute your values for parameters that are not displayed in bold.
Semicolon (;)	Separates keywords in topic footnotes and macro strings. Also introduces comments in the project (.hpj) file.
Horizontal ellipsis (...)	Indicates that the item shown might appear more than once.
Vertical ellipsis(. . .)	Indicates that the example omits a portion of the file or program.
User	Indicates the person using the finished Help system.
Author	Indicates the person (or persons) preparing the Help system.
Developer	Indicates the person (or persons) developing the program from which the Help file is accessed, if there is one.

Copyrights

Software requirements

To author Windows Help version 4.0 files, you need the following software on your computer:

- Help Workshop (Hcw.exe and Hcrtf.exe)
- WinHelp version 4.0 (Winhlp32.exe)
- A word processor that saves files in rich-text format (RTF), inserts custom footnotes, supports single and double underlining, and allows hidden text.
- Windows 95 or Windows NT.

Notes

- Although you can author Help files on any version of Windows NT, WinHelp version 4.0 runs only on Windows NT version 3.51 or later. Help Workshop runs on Windows 95 or later, and on any version of Windows NT.
- If you are familiar with RTF coding, you can prepare and save your topic files using any word processor or text editor that can retain the files in that format.
- Do not use Microsoft WordPad to author topic files.

{button ,AL("COMPRESS;FTS;RTFHelpRTFStatements")} Related Topics

To return to the list of topics, click Help Topics.

New Help Topics dialog box

The Help Topics dialog box contains a Contents tab, an Index tab, and a Find tab. This dialog box appears when users:

- Click Help Topics on the Help menu in a program
- Double-click a Help file in Windows Explorer or My Computer or a Help shortcut icon on the desktop. The Help file must be associated with a contents (.cnt) file that specifies a Contents tab.

The Contents tab contains information that is specified by the new contents (.cnt) file. The Index tab contains the keywords for one or more Help files. The Find tab provides full-text search capability for all Help files.

{button ,AL(`PRO_CNT_CREATE')} Related Topics

New context menu

When users click a Help window using their right mouse button, a menu that contains the following commands appears:

- Annotate
- Copy
- Print Topic
- Font
- Keep Help On Top
- Use System Colors

These commands also appear if users click the Options button in a Help window. An Options button appears on a Help window's button bar if you specify it in your project (hpj) file.

Notes

- When you are viewing a Help file in Help Author mode, two additional commands appear on the menu:
- Topic Information
- Ask On Hotspots

{button ,AL(`askonhot')} Related Topics

New Options menu

A new Options menu in the main Help window contains four commands:

- Keep Help On Top
- Display History Window
- Font
- Use System Colors

Most of these commands also appear when users click a Help window using their right mouse button, and when users click the Options button in a Help window. An Options button appears on a Help window's button bar if you specify it in your project (hpj) file.

New annotation capabilities

WinHelp version 4.0 supports read-only annotation files. System administrators can insert annotations in a Help file, and then add a read-only attribute to the annotation (.ann) file. This prevents users from adding or editing annotations, but enables them to view annotations by clicking the paper-clip icon in annotated topics.

System administrators can place annotation files in network locations.

Improved copying

Users can now copy Help text to the clipboard by selecting the text and then clicking the Copy command on the Edit menu of a main window, the Options button (if it is included on the window's button bar), or the context menu. To copy the contents of a pop-up window, users can use the right mouse button to click the window and then click Copy on the context menu.

Users can also drag and drop selected text from a Help window to any program that supports drag-and-drop functions for OLE.

New printing options

Users can now print multiple topics by clicking a book on the Contents tab, and then clicking the Print button. Users can also print topics by clicking an author-defined hotspot in a topic that has been set up to run the InitMPrint and EndMPrint macros.

To print the contents of a pop-up window, users can click the pop-up using their right mouse button, and then click Print Topic.

New ESC key function

The ESC key now closes the current Help window when no dialog box or pop-up window is open. If the current Help window is the only Help window that is open, WinHelp closes.

Background color override

If users set the high contrast flag or change the background or text colors for windows in Display properties, WinHelp ignores any window background colors that you defined in your project (.hpj) file.

Context-sensitive Help access

In addition to using the F1 key to access context-sensitive Help for an item in a dialog box, users can click the ? button on the title bar of a dialog box, and then click the item. Users can also use their right mouse button to click an item, and then click What's This.

Graphical interfaces

Help Workshop has a new graphical interface that enables you to compile your Help files and to create and edit project (.hpj) and contents (.cnt) files. Using the graphical interface, you can easily specify or change settings. If you don't understand what a setting is for, you can use context-sensitive Help to learn more about it.

Note

- Help Workshop can read (.hpj) files that were created for version 3.x of WinHelp. However, if you use Help Workshop to edit these files, they are saved in a slightly different format, and the MS-DOS–based compiler will be unable to process them.

New topic file features

Help Workshop provides two new topic statements and allows longer hotspot definitions.

Authorable buttons

You can create an authorable button by using the {button} command in the topic (.rtf) file. Any WinHelp macro(s) can be assigned to the button. The width of the button is determined by the text defined for it. If no text is specified, WinHelp will create a 12-by-12–pixel button.

Multimedia support

You can include multimedia files (.avi) in Help topics by using the {mci} command in the topic file.

Increased hotspot capacity

The size limitation for a hotspot has been raised to 4095 characters.

New window-type footnote

Topics are usually displayed in the main or current window type unless another window type is specified in jump instructions for a hotspot or for an item on the Contents tab. However, you can override the window type when the topic is accessed from the Index or Find tabs, or by an ALink or KLink macro.

You can use the greater-than sign (>) footnote in your topic (.rtf) files to specify a window type for any given topic. The window type you specify must be defined in the project (.hpj) file.

Secondary-window features

Secondary windows can now contain a button bar, which is completely configurable in the project (.hpj) file. You can use standard buttons and add your own.

You can now specify as many as 255 secondary-window types in your project file. Up to nine of these secondary windows can be displayed at the same time. You can also specify an absolute position for a Help window.

You can add a Back button to a secondary window. Each window type has its own history list and will display only the history of topics that were displayed in that window type. This functionality is designed for Help files that use a secondary window exclusively. You can use either the Back() macro or add the Back button to the secondary window's button bar.

You can set secondary windows to automatically adjust to the length of a topic. In addition, users can more easily specify that secondary windows should stay on top of other windows.

New macros and enhanced macro processing

More than 20 new macros are available for WinHelp version 4.0 Help files, and Help Workshop has fewer limitations on macro processing. In addition, there is no practical limit to the number of macros that can be nested.

Help Workshop now automatically changes macro names to their short form. (Using long names for macros in .rtf files greatly enhances the readability of those files.) For macros requiring numeric arguments, you can use more descriptive string values instead. In most cases, quotation marks are no longer necessary. If the final parameter for a macro is 0 or " ", that parameter can be omitted.

Hotspot macros can now use as many as 4096 characters.

For a list of macros, see the Macro Quick Reference.

Bitmaps supplied by Help Workshop

Help Workshop comes with new bitmaps that you can use. You can specify the filename for any of these bitmaps in your topic (.rtf) files, even though the bitmap is not in your bitmaps folder. If your bitmaps folder contains a bitmap that has the same filename as one specified below, that bitmap is used instead of the one supplied by Help Workshop.

Bitmap	Filename
▪	bullet.bmp
—	emdash.bmp
	shortcut.bmp
▸	onestep.bmp
	open.bmp
	closed.bmp
	document.bmp
▸	do-it.bmp
	chiclet.bmp
▸	prcarrow.bmp
	cshelp.bmp

Greater color support

WinHelp version 4.0 supports any color depth that the operating system supports (256 colors, 24-bit color, and so on). If you use 16-color bitmaps, you are no longer limited to the standard 16 colors.

Help Workshop version enables you to specify more than one color-depth version of a graphic in a single {bmx} bitmap statement in your topic (.rtf) file. WinHelp determines the color-depth capability of a user's monitor, and then uses the most appropriate bitmap file.

Note

- You can now make the white background color in bitmaps transparent, so the colored background of a Help topic shows through.

New default buttons

In addition to the regular default buttons, you can add a Find button to the button bar, and you can use a Help Topics button instead of the Contents and Index buttons. In addition, browse buttons are now available in secondary windows.

Duplicate accelerators

If two or more buttons use the same accelerator, WinHelp now attempts to move the accelerator of the second button so that the letter chosen does not conflict with the accelerator of any other button. If WinHelp cannot create a unique accelerator and the user presses that accelerator key, WinHelp displays a dialog box containing the text of every button that contains the duplicated accelerator and asks the user to choose.

If no accelerator is specified for the text of a button, WinHelp automatically chooses the first letter that does not conflict with any other button. If WinHelp cannot find a unique accelerator, WinHelp uses the first letter of the text string. WinHelp ignores conflicts with menu accelerators.

New topic-linking features

WinHelp version 4.0 provides two new linking macros, KLink and ALink, that enable you to provide jumps to multiple topics based on keywords rather than specific context strings.

These jumps are resolved when the user clicks them instead of when the file is compiled. This enables you to create interfile jumps for Help files that might change after the original jump was created or for Help files that might or might not have been installed at the time the program was released.

KLink macros use K-keywords — the same keywords that you use to create the index. If a user clicks a hotspot that runs the KLink macro, WinHelp displays the Topics Found dialog box, which contains a list of all the topics that have the associated keywords throughout the family of Help files. The macro can be configured so that if only one topic is found, the jump occurs immediately without displaying the Topics Found dialog box. KLink (and ALink) macros can access multiple Help files if they are properly specified in the contents file.

ALink macros function similarly to KLink macros except A-keywords never appear in the index. Jumps using ALink macros are unaffected by any changes to the index.

New WinHelp API commands

The following new commands are available for the WinHelp function.

Command	Purpose
HELP_CONTEXTMENU	Displays context-sensitive Help when the right mouse button is clicked.
HELP_FINDER	Displays the Help Topics dialog box in its last state.
HELP_SETPOPUP_POS	Sets the position for the next pop-up window. Overrides the actual mouse position.
HELP_TCARD	Indicates that a command is for a training-card instance of the WinHelp program. A program combines the HELP_TCARD flag with other commands by using the bitwise OR operator.
HELP_WM_HELP	Displays context-sensitive Help when the F1 key is pressed.

New author-specified font capability

WinHelp version 4.0 enables you to set the default fonts for lists that appear on the Contents and Index tabs and in the Topics Found dialog box. You can set default fonts while editing your project (.hpj) file by clicking the Options button, and then clicking the Fonts tab.

Character set as part of font description

WinHelp version 4.0 expects Help files to include the character set as part of the font description. If you are using a word processor that includes this information as part of the font table description (for example, Microsoft Word for Windows version 6.0), Help Workshop includes this information automatically. If the information is not supplied in the topic (.rtf) file, Help Workshop uses default values for the character set.

Longer font names

Help Workshop increases the limit for font names from 20 to 31 characters.

WinHelp also stores the character set, making it possible to use fonts like Script and various international fonts. The Wingdings font is now fully supported.

Greater RTF character support

Help Workshop now supports the following RTF statements:

- \emdash
- \endash
- \emspace
- \enspace
- \lquote
- \rquote
- \ldblquote
- \rdblquote

Automatic localization of quotation marks

The \ldblquote and \rdblquote RTF statements are used for generating quotation marks. The actual characters that are displayed in the Help file are determined by the language specified in the project (.hpj) file.

WinHelp displays French-style quotation marks for the following character sets:

- Algerian
- Belgian
- French Canadian
- French (Luxembourg)
- French (Standard)
- French (Swiss)
- Moroccan
- Tunisian

WinHelp displays German-style quotation marks for the following character sets:

- German (Austrian)
- German (Liechtenstein)
- German (Luxembourg)
- German (Standard)
- German (Swiss)

All other non – double-byte character sets display the standard curly quotation marks. Curly quotation marks are displayed in Times Roman, using the current font's point size.

Examples

French: «text»
German: „text“
Other: “text”

New context-sensitive help features

In WinHelp 4.0, the procedure for applying context-sensitive pop-up Help topics to interface elements and controls has been greatly simplified. At the same time, it is now possible to apply context-sensitive Help directly to a greater range of individual interface elements. This increase in ease and range encourages Help authors and developers to make fuller use of context-sensitive Help. Where conceptual overviews are needed for dialog boxes, an overview topic can be linked to a Help button in those dialog boxes.

{button ,AL(`cs_help')} Related Topics

Help Author command

A new Help Author command provides additional authoring information and debugging capabilities. When you check the Help Author command on the File menu in Help Workshop:

- The text on the title bar of the Help topic is replaced with the topic number. This number is determined by the order in which topics appear in your topic (.rtf) files, as listed in the project (.hpj) file.
- You can press CTRL+SHIFT+RIGHT ARROW, and CTRL+SHIFT+LEFT ARROW to step through the topics in your Help file and CTRL+SHIFT+HOME and CTRL+SHIFT+END to move to the beginning or end of your Help file.
- WinHelp displays specific information about problems in the Help (.hlp) or contents (.cnt) file.
- Topic Information and Ask On Hotspots commands appear when you click a topic using your right mouse button. The Topic Information command displays information about the current topic, including the text of the title ($) footnote, the macros specified in the entry macro (!) footnote, the name of the current window type, and the Help filename and path. The Ask On Hotspots command changes the effect of clicking a hotspot. If this command is checked, WinHelp displays the topic ID or macro syntax of any hotspot you click and prompts you to specify whether the macro or hotspot jump should be carried out.

Report command

You can generate a report that contains information about a Help file by clicking the File menu in Help Workshop, and then clicking Report. Reports can contain information about a Help file's topic titles, hash numbers, and keywords. You can also request a report containing all the ASCII text in a Help file.

New test commands

The following commands are available on the Help Workshop Test menu:

- Contents File. Checks the contents file for parameter limits, level coherence, nesting depth, duplicate elements, missing parameters, file availability, and other components.
- Send A Macro. Enables you to see how a specific macro will perform in a Help file.
- WinHelp API. Enables you to call a WinHelp API as if it were invoked from another program.

Easy jumping to topics

A new Jump dialog box enables you to jump directly to a topic by specifying its topic ID. You can open the Jump dialog box by pressing CTRL+SHIFT+J in any topic window except pop-up windows.

You can run any macro from the Jump dialog box by specifying a macro string, preceded by an exclamation point (!), instead of a topic ID. You can specify several macros by separating them with colons (:) or semicolons (;).

Missing-bitmaps error

When Help Workshop cannot find a bitmap, it inserts its own bitmap in its place and includes the name of the missing bitmap. In this way, you can readily tell the difference between a missing bitmap and a bitmap that WinHelp cannot display.

Improved error handling

Errors are no longer reported by levels. Accordingly, Help Workshop ignores the WARNINGS= option in the project (.hpj) file. Problems are now divided into Notes, Warnings, and Errors. All message identifiers begin with the letters "HC" and end with a 4-digit number, for example, "HC1000." You can look up a message identifier in this Help file for more information. Help Workshop displays:

- Notes for problems that probably do not affect the Help file adversely.
- Warnings if a Help file contains problems but can still be compiled.
- Errors if the Help file cannot be compiled.

New monitoring of WinHelp messages

You can now monitor the WinHelp messages that are processed while a Help file is running. Help Workshop displays information about WinHelp API calls, context-sensitive calls, hotspot jumps, embedded window commands, which macros are being executed, which topics and pop-up windows are current, and which configuration (.gid) file is being read. To access this feature, click the View menu in Help Workshop, and then click WinHelp Messages.

Help file search path

WinHelp now searches for Help files in the following locations, in the order listed.

- If a path is specified, WinHelp first searches for the file in the specified path. If the file is not found, the path is removed and the following locations are searched.
- The folder of the current Help file.
- The current folder.
- The System subfolder of the Windows folder.
- The Windows folder.
- The folders listed in the PATH environment.
- The location specified in the Winhelp.ini file.
- The registry.

If WinHelp cannot find a Help file, it displays a dialog box that enables the user to specify the location of the Help file. If the users finds the file, WinHelp will add the location information in the registry.

Bitmap search path

Help Workshop now searches in three locations for a bitmap. If a path for a bitmap is specified, it first searches in that location. If it cannot find the bitmap, the path is removed and the following locations are searched:

- The folder(s) specified in the project (.hpj) file for bitmaps
- The folder(s) specified in the project file for topic (.rtf) files
- The folder that contains the topic file in which the bitmap is specified
- The folder that contains the project file

Training cards

Training cards are a new feature that enables WinHelp to determine which Help topic to display by communicating with another program. You can use this feature when the topic that should be displayed next is dependent on the actions a user takes in a program. For example, you could create a topic that contains only the first step of a procedure. When the user carries out that step, the program tells WinHelp to display the topic that contains the next step.

When you use the training cards feature, you must work with a program developer to set it up in the program's code.

Help macros

WinHelp provides a set of macros that enable you to control and customize Help functionality. Macro reference topics describe the macros included with WinHelp and explain how to use them in your Help file.

When creating your Help files, you can make your Help macros run by:

- Placing macros in the project (.hpj) file, so WinHelp runs the macros whenever a user opens the Help file or a particular window type.
- Placing macros in the project file, so WinHelp runs the macros whenever a user chooses a keyword from the Index tab in the Help Topics dialog box.
- Placing macros in a topic footnote, so the macros run when a user displays that topic.
- Placing macros in the contents (.cnt) file, so WinHelp runs the macros whenever a user double-clicks the associated page icon.
- Configuring the menu bar and button bar, so WinHelp runs a macro when a user clicks the menu item or button.
- Adding hotspots to your topic, so the macro runs when a user clicks the hotspot.
- Using an external program to send a function call that prompts WinHelp to run a Help macro.

Help macros are designed to imitate standard C-language format; however, standard Help macros do not support variables or expression evaluation.

Note

- If you find that the standard Help macro set falls short of your ultimate goal, you can also create your own Help macros by using DLL functions.

{button ,AL("macro_syntax;macro_dlls;macro_error_chk;macro_return_vals;macro_from_app")} Related Topics

Macro syntax

Help macro statements have two main components: the macro name and the macro parameters enclosed in parentheses. The macro name must be spelled exactly as it is shown in the syntax and the parameters must be used in the order they are given in the syntax. Parameters provide information for the macro; for example, the JumpId macro, which jumps to a topic with a specific topic ID, has parameters for the name of the Help file and the topic's ID.

All Help macros use the following syntax:

```
MacroName(parameter1, parameter2, ...)
```

The opening parenthesis, closing parenthesis, and commas are required characters when included in the syntax statement for a macro. Macro names are not case sensitive. Help Workshop automatically changes macro names to their short form and removes all unnecessary white space. This enables you to use long names for macros in your topic (.rtf) file to enhance readability.

Parameters can be text strings or numbers, and are separated by commas. If the final parameter for a macro is 0 or "", that parameter can be omitted.

Some macros have no parameters, but the parentheses are still required.

If you create custom macros, the macro name should begin with an alphabetic character, followed by any combination of alphabetic characters, numbers, or the underscore character, as in this example:

```
PlayAudio()
```

You can include more than one macro in a macro string by placing a colon or semicolon between each macro. WinHelp runs the macros sequentially.

Backslashes are used as an escape character, allowing you to include quotes as part of a macro parameter. If you need a literal backslash (for example, in a path specification) you must double the backslash for every level of macro nesting. For path specifications, use a forward slash instead of backslashes.

{button ,AL("syntax_more;conventions")} Related Topics

Macro error checking

Help Workshop checks the validity of each macro included in a Help file as it compiles. If it finds an error, it displays an error message and stops processing that macro. Help Workshop checks whether you:

- Spelled the macro's name correctly
- Specified the correct macro syntax (for example, the number of opening and closing parentheses and quotation marks should match)
- Specified the required number of macro parameters
- Specified types that match the parameter types (for example, you specified numeric parameters with numerals)

Using string parameters

Some macros require that string parameters be enclosed within quotation marks. In most cases, quotation marks are not necessary if the macro is in a topic (.rtf) or project (.hpj) file. Quotation marks can be either double quotation marks (") or matching single quotation marks(`').

Using single quotation marks eliminates ambiguities in situations where strings are nested within other strings.

Note

- When calling a macro from an external program or from a contents (.cnt) file, string parameters must be enclosed within quotation marks, and numeric parameters must use a number.

Using numeric parameters

WinHelp and Help Workshop recognize decimal and hexadecimal numbers for numeric parameters. Use a prefix of 0x to indicate a hexadecimal number. For example, the following numbers both represent decimal 64:

```
64
0x40
```

To specify a negative number, add a minus sign before the number.

Using special characters in macro strings

To use the following characters within a macro as part of a parameter value, you must preface the character with a backslash.

- Double quotation mark (")
- Single opening quotation mark (`)
- Single closing quotation mark (')
- Backslash (\)

Commas or parentheses do not need to be preceded by a backslash within a macro string.

Note

- For every level of nested macros that you use, you must double the backslashes. To specify a filename path, use forward slashes rather then backslashes, as in c:/myapp/myapp.exe.

Making DLL calls in Help macros

You can call functions in a DLL if you inform WinHelp about the call by using the RegisterRoutine macro. For example, you could register the MessageBox function that is part of Windows with the following:

```
[CONFIG]
RegisterRoutine("user", "MessageBox", "USSU")
```

Having registered this function, you could display a message box with a call similar to the following:

```
MessageBox(hwndContext, "This is my message", "Help Message", 0)
```

{button ,AL("DLL_WRITE_OVERVIEW;RegisterRoutine")} Related Topics

Return values in Help macros

Generally, Help Workshop ignores return values in macros. However, the IsMark and IfThenElse macros can be used together to test a condition and run a macro if the condition evaluates to a non-zero, or true, value. The following example shows a typical use of IsMark and IfThenElse:

```
IfThenElse(FileExist(myapp.exe), ExecFile(myapp.exe),    JumpId(how to install myapp))
```

The first parameter of the IfThenElse macro is a boolean value (zero or non-zero number). Since FileExist returns a boolean value, it can be used as the first parameter. Help runs this macro as follows:

1 WinHelp runs the FileExist macro and obtains a boolean result from it (non-zero if the file exists, zero if it doesn't).

2 The IfThenElse macro runs. WinHelp passes the value returned from FileExist to the first parameter and passes the ExecFile and JumpId macro strings to the second and third parameters.

3 If the number passed to the first parameter is not zero, the IfThenElse macro runs the ExecFile macro; otherwise, it runs the JumpId macro.

{button ,AL("FileExist;ExecFile;IfThenElse;JumpId")} Related Topics

Using DLL functions as Help macros

If you want to add a feature to your Help file that WinHelp does not support, and you have programming experience for Windows, you can develop custom extensions to support the functions you need. You can use calls to functions in the DLLs as Help macros by registering the functions in the project (.hpj) file using the RegisterRoutine macro.

{button ,AL("DLL_WRITE_OVERVIEW;registerroutine")} Related Topics

Calling macros from a contents file

When you call a macro from a contents (.cnt) file, you cannot use Help Workshop to turn macros into a format that WinHelp can understand. This means you must use the short name for the macro, use quotation marks around all string parameters, use numeric values for numeric parameters, and supply all the parameters for a macro . You cannot omit most parameters as you can in your topic (.rtf) file. The only parameters you can omit are trailing arguments that would have a value of 0 or "".

{button ,AL("macro_from_app ")} Related Topics

Calling macros from a program

A program can send a HELP_COMMAND parameter in the WinHelp API call that specifies a macro to run. This WinHelp call would look like the following:

```
WinHelp(hwndProgram, "HelpFile[>WindowName]", HELP_COMMAND, "macro");
```

The following example uses the ALink macro in the Myhelp.hlp file:

```
case IDM_HELP_KEYBOARD:
  WinHelp (hwnd, "myhelp.hlp", HELP_COMMAND, (DWORD)   "AL(`HELP_KEYBOARD\')");
  return 0;
```

When you call a macro from a program, you cannot use Help Workshop to turn macros into a format that WinHelp can understand. This means you must use the short name for the macro, use quotation marks around all string parameters, use numeric values for numeric parameters, and supply all the parameters for a macro . You cannot omit most parameters as you can in your topic (.rtf) file. The only parameters you can omit are trailing arguments that would have a value of 0 or "".

{button ,KL("winhelp")} Related Topics

Nested Macros and Nested String Parameters

Help Workshop supports nested macros, a macro that is included in another macro as a parameter value. Because nested macros often have their own string parameters, you must frequently specify a string enclosed within another string.

For example, the following macro creates a button called Time that uses the ExecFile macro as a parameter. When the user clicks the button, the Microsoft Windows Clock program starts. Because the ExecFile macro takes a string as its first parameter, the string is enclosed in single quotation marks:

```
CreateButton("btn_time", "&Time", "ExecFile(`clock', 0)")
```

If the nested macro has any string parameters, they must have quotation marks that are different from the enclosing macro quotation marks. In other words, if double quotation marks enclose a macro, you must enclose any nested strings in single quotation marks.

You can also use single quotation marks for the outermost parameters. You can avoid confusion with nested string parameters by using single quotation marks for all string parameters. Just be sure to match the opening and closing quotation marks correctly.

If a nested macro contains a path, be sure to double each backslash for every level that the macro is nested, starting with double the typical number of backslashes in a macro nested one level. Double the backslashes again for a macro nested two levels, and so on. Alternatively, you might be able to use forward slashes in place of backslashes. You do not need to double forward slashes in paths in nested macros. Forward slashes, however, are not compatible with all networks. Make sure forward slashes are compatible with your network before using them.

{button ,AL("ExecFile")} Related Topics

To create a topic file

1 Open a new file in a word processor or text editor that can handle rich text format.

2 Write your individual topics, separating each topic with a hard page break.

3 Add appropriate footnotes at the beginning of the file.

See Related Topics below.

4 Save the file in rich text format with the .rtf filename extension.

Tip

- Before creating a complete topic file, test the compatibility of your word processor or text editor by compiling a short test topic file.

{button ,AL(`POP_OVERVIEW_TOPIC;BAS_TOPIC_FOOTNOTE;PRO_ADD_TOPIC_ID;PRO_ADD_TITLE;PRO_ADD_INDEX_ENTRIES')} Related Topics

To add a topic ID to a topic

1 Place the insertion point at the beginning of the topic you want to add a topic ID to.

2 Insert a number sign (#) as a footnote mark.

3 Type the topic ID name as the footnote text, observing these restrictions:

- Topic IDs can contain spaces, but avoid leading and trailing spaces.
- Do not use reserved characters (# = + @ * % !).
- Use no more than 255 characters.
- Do not begin a topic ID with a number if the ID will also be used in the [MAP] section.

Notes

- To enable Help Workshop to track errors, start all topics IDs that will be called from a program with IDH_ . Help Workshop will list any undefined topics that begin with IDH_ in the [MAP] section of the project file, and will warn of any topic of any IDH_ topic IDs that do not appear in the [MAP] section.
- If you want users to jump to a specific location in a topic, you can insert a topic ID at that location.

To add a title footnote to a topic

1 Place the insertion point at the beginning of the topic you want to add a title footnote to.

2 Insert a dollar sign ($) as a footnote mark.

3 Type the title as the footnote text

Note

- Titles can have up to 255 characters (127 for DBCS characters). Spaces count as characters.

To mark a topic as an ALink target

1 Place the insertion point at the beginning of the topic.

2 Insert an A as a footnote mark.

3 Type one or more keywords as the footnote text, observing the following restrictions:

- Separate keywords with a semicolon.
- Spaces before and after keywords are removed.
- Do not use carriage returns.
- Use no more than 255 characters per keyword.

Notes

- The ALink macro searches for topics containing an A-keyword.
- Keywords are case-sensitive, but Help Workshop assumes that two keywords that differ only in case are meant to be the same keyword. Help Workshop changes one of the keywords to agree with the other and displays a warning message.

To create index entries for a topic

1 Place the insertion point at the beginning of the topic.

2 Insert a K as a footnote mark.

3 Type a list of keywords as the corresponding footnote text. Observe the following restrictions:

- Separate keywords with a semicolon.
- Spaces before and after keywords are removed.
- Do not use carriage returns.
- Use no more than 255 characters per keyword.

Notes

- The KLink macro searches for topics containing a K-keyword.
- Keywords are case-sensitive, but Help Workshop assumes that two keywords that differ only in case are meant to be the same keyword. Help Workshop changes one of the keywords to agree with the other and displays a warning message.

{button ,AL(`BAS_ADDING_KEYWORDS;PRO_ADD_LEVEL2_KEYWORDS;BAS_WRITE_INDEX')} Related Topics

To create second-level index entries for a topic

1 In the K-footnote of a topic, type the first-level keyword entry followed by a semicolon (;).

2 Immediately after the semicolon, type the first-level entry again, followed by a comma (,) or a colon (:), a space, the second-level entry, and a semicolon.

Example

The following K-footnote text results in the index entries shown below it:

macro; macro, library; macro, automatic; macro, complex; macro, simple

macro
  automatic
  complex
  library
  simple

Notes

- Entries appear alphabetically in the index, regardless of their order in footnote text.
- Spaces before and after keywords are removed.
- You can change the characters you use to identify second-level index entries. For more information, click Related Topics.

{button ,AL(`PRO_SPECIFY_SEP_CHAR;PRO_ADD_INDEX_ENTRIES;BAS_WRITE_INDEX;BAS_ADDING_KEYWORDS')}
Related Topics

To add a topic to a browse sequence

1 At the beginning of the topic that you want to include in a browse sequence, insert a plus sign (+) as a footnote mark.

2 Type a browse code as the footnote text. Observe the following restrictions:

- Do not use reserved characters (# = + @ * % !) or spaces.
- Use no more than 50 characters.

Notes

- To use browse sequences, you must also activate the browse buttons in the project file.
- WinHelp creates an auto-browse sequence for topics to which you have added a browse footnote (+) but for which you have specified no browse code.

Tips

- When using numbers in browse codes, include placeholding zeros (001, 002, 003, and so on).
- To make it easier to add topics later, use the same name, but no numbers for each browse footnote in a sequence. Help Workshop will arrange your browse sequence in which the order that the topics appear. To change the browse sequence, simply change the order in which topics appear in your .rtf file.
- If your Help file has only one browse sequence, use "auto" as the browse code. Help Workshop automatically creates a browse sequence based on the order in which topics appear in your topic files.

{button ,AL(`PRO_ADD_BROWSE_BUTTONS;BrowseButtons;BAS_ADDING_BROWSE_CODE;browse_seq')} Related Topics

To mark a topic so you can exclude or include it in a Help file

1 Place the insertion point at the beginning of the topic, ahead of any other footnote marks.

2 Insert an asterisk (*) as a footnote mark.

3 Type one or more build tags as the corresponding footnote text. Separate build tags with a semicolon.

4 In your project (.hpj) file, make sure you have specified which topics to include in a build. For more information, click Related Topics.

Notes

- Build tags are not case sensitive.
- Topics without build tags are always included in a build, as are all topics that are not expressly excluded from a build in your project file.

{button ,AL(`PRO_HPJ_BUILD_TAGS;BAS_SPECIFY_BUILD;BUILD;BUILDTAGS')} Related Topics

To specify the type of window to display a topic in

1 Place the insertion point at the beginning of the topic.

2 Insert a greater-than sign (>) as a footnote mark.

3 Type the name of the window type, as defined in your project (.hpj) file.

Notes

- This footnote specifies the window type a topic will appear in when the topic is opened from the Index or Find tab, or from an ALink or KLink macro.
- Window types must be defined in your project file. For more information, click Related Topics.

{button ,AL(`BAS_ADD_WINDOW_TYPE;BAS_HPJ_CREATE_WINDOW;PRO_HPJ_CREATE_WINDOW;PRO_HPJ_CUSTOMIZE_WINDOW')} Related Topics

To specify a macro to run when a topic is opened

1 Place the insertion point at the beginning of the topic in which you want the entry macro to run.

2 Insert an exclamation point (!) as a footnote mark.

3 Type the entry macro commands as the corresponding footnote text. Observe the following restrictions:

- Separate macros with a semicolon
- Do not use carriage returns

{button ,AL(`MACRO_REF_OVR')} Related Topics

To add a nonscrolling region to a topic

1 Make sure that the paragraph or paragraphs you want to be in the nonscrolling region are at the top of the topic.

2 Apply the Keep With Next paragraph style (or equivalent) to the paragraphs.

Notes

- Users can select (for copying or printing purposes) either the scrolling region or the nonscrolling region of a topic, but not both. However, both scrolling and nonscrolling regions are copied when users click a topic using their right mouse button, and then click Copy.
- Do not add nonscrolling regions to topics that will be displayed in pop-up windows.
- You can specify different background colors for nonscrolling and scrolling regions by opening your project (.hpj) file in Help Workshop, clicking the Windows button, and then clicking the Color tab.
- If you use a transparent bitmap in both the scrolling region and the non-scrolling region of the same topic, the transparent area of both bitmaps is displayed using the background color of the scrolling region.

To prevent a line of text from wrapping

1 Place the insertion point anywhere in the line that you do not want to wrap.

2 Apply the Keep Lines Together paragraph style (or equivalent).

Note

- If the text of the paragraph is too wide for the window, WinHelp adds a horizontal scroll bar.

The title footnote specifies the title of the topic as it appears in the Topics Found dialog box, the Bookmark dialog box, the History window, and the Find tab.

Topic footnotes

Instructions and commands specific to a topic are inserted as footnote text for that topic. Each footnote is identified by a custom footnote mark, as shown in the following table:

Parameter	Footnote mark	Purpose
Topic ID	pound sign (#)	Defines a unique identifier for a topic. This footnote is required for every topic.
Title	dollar sign ($)	Specifies the title of the topic as it appears in the Topics Found dialog box, the Bookmark dialog box, and the History window.
Keyword	K	Specifies words used in the index and in KLink macros.
A-Keyword	A	Specifies words that identify the topic to ALink macros.
Browse Code	plus sign (+)	Defines topic's place in a browse sequence.
Entry Macro	exclamation point (!)	Specifies a macro that runs when the user opens the topic.
Build tag	asterisk (*)	Identifies topics for conditional builds.
Window type	greater than sign (>)	Specifies a default window type for the topic.

Note

- Only the topic ID footnote (#) is mandatory. Include the others only if you want the functionality they offer.

Topic keywords

Topic keywords are words or phrases associated with a topic, usually for indexing purposes. The same topic keyword may be associated with more than one topic. You assign keywords to a topic in topic footnotes.

There are three kinds of keywords:

- Keywords specified by a K-footnote appear on the Index tab. K-keywords are also used by the KLink macro.
- Keywords specified by an A-footnote do not appear in the Index tab, but are used by the ALink macro.
- In addition to K- and A-keywords, you can assign multi-index keywords, using any of the remaining uppercase and lowercase letters of the alphabet. Each of these is used to create a separate index. This index can be searched only when Help is called from a program.

{button ,AL(`BAS_TOPIC_FOOTNOTE;BAS_WRITE_INDEX;BAS_ADDING_ALINK_KEYWORD;BAS_ADD_MULT_KEYWORD')}
Related Topics

Browse sequences

A browse sequence is the order in which topics appear when users click the browse buttons in a Help window.

A Help file can have a single browse sequence or several smaller browse sequences. Each browse sequence is self-contained, and users cannot move from one browse sequence to another by using the browse buttons.

To specify a browse sequence, you must first add browse buttons to the project (.hpj) file, and then assign plus sign (+) footnotes containing browse sequence codes to the topics that you want to include in the sequence.

Note

- If your Help file has only one browse sequence, use "auto" as the browse footnote. Help Workshop will automatically create a browse sequence based on the order in which topics appear in your topic (.rtf) files.

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_TOPIC_FOOTNOTE')} Related Topics

Creating Help topics

The Help topic is the basic unit of information in a Help file. Topics can range in size from short examples containing a single picture or word to lengthy explanations involving several graphics and hundreds of words.

A topic contains one or more of the following components:

- Informational material. This is usually text, but can be graphics or multimedia.
- Hotspots. These are textual or graphical elements that users can click to jump to other topics or to run macros.
- Footnotes. These provide instructions to Help Workshop regarding the topic's unique topic ID, the title that will appear in the Topics Found dialog box, keywords that will appear in the index, the position of a topic in a browse sequence, the topic's inclusion in or exclusion from a Help file, and the type of window in which you want the topic to appear.

Topics are stored in topic files and saved in rich-text format (.rtf), one topic to a page. You can have any number of topics per topic file, and any number of topic files per Help file.

{button ,AL(`POPUP_TOPICS_DESIGN;3X_CONTENTS_DESIGN;TOPICS_DESIGN;BAS_OVERVIEW;BAS_CREATE_TOPIC_FILE;BAS_TOPIC_FOOTNOTE')} Related Topics

This is an example of a pop-up topic. Note that it appears near the hotspot you clicked, has no title, and is relatively brief. It disappears if you click anywhere on your screen or press any key.

Context-sensitive Help is Help that a user can access within a dialog box of a program. Typically, the user does this by using the right mouse button to click an item in a dialog box, and then clicking What's This.

Observe these restrictions when adding index keywords:

- Separate keywords with a semicolon.
- Spaces before and after keywords are removed.
- Do not use carriage returns.
- Use no more than 255 characters per keyword.

To create a new project file

1 In Help Workshop, click the File menu, and then click New.

2 Double-click Help Project.

3 Type the name of the project (.hpj) file, and then click OK.

Note

- When you create a project file, Help Workshop configures the minimum settings you need to create a Help file. Before you can compile, however, you must click the Files button and specify at least one .rtf file.

{button ,AL(`BAS_ABOUT_HPJ_FILES;NEW_AUTHOR_HPJ')} Related Topics

To suppress compiler messages

1 In Help Workshop, open the project (.hpj) file.

2 Click Options.

3 To hide notes that appear while compiling, make sure the Notes check box is clear.

4 To hide the progress report while compiling, make sure the Progress check box is clear.

{button ,AL(`BAS_THE_HELP_COMPILER')} Related Topics

To specify the default topic for the Help file

1 In Help Workshop, open the project (.hpj) file.

2 Click Options.

3 In the Default Topic box, type the topic ID of the topic you want to be the default topic.

{button ,AL(`BAS_PROJ_FILE_EDITOR')} Related Topics

To specify the title of a Help file

1 In Help Workshop, open the project (.hpj) file.

2 Click Options.

3 In the Help Title box, type the title.

Note

- The Help title can have up to 127 characters.

{button ,AL(`BAS_HELP_TITLE;BAS_PROJ_FILE_EDITOR')} Related Topics

To add copyright information

1 In Help Workshop, open the project (.hpj) file.

2 Click Options.

3 In the Copyright Information area, type the copyright information you want to display in the Version dialog box or when users paste or print Help text.

Notes

- You can specify up to 255 characters of copyright information for the Version dialog box. There is no limit to the number of characters you can specify for the copyright information that appears when users paste or print Help text.
- If you add **%date** at the end of the text you want to display in the Version dialog box, Help Workshop will substitute the current date whenever you compile the Help file.

{button ,AL(`BAS_PROJ_FILE_EDITOR')} Related Topics

To specify compression options

1 In Help Workshop, open the project (.hpj) file.

2 Click Options.

3 Click the Compression tab, and then specify the compression settings you want.

Tips

- For help on an item, click ? at the top of the dialog box, and then click the item.
- Help Workshop typically takes three times longer to compile a help file using maximum compression than it does to compile a file with no compression.
- You can reduce compile time by temporarily turning off compression without affecting the compression setting in the project file. To do this, click the Compile button, and then make sure the Turn Off Compression box is checked.

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_COMPRESSION')} Related Topics

To specify the language of a Help file

1 In Help Workshop, open the project (.hpj) file.

2 Click Options.

3 Click the Sorting tab, and then specify the language you want.

Notes

- By default, Help Workshop uses a language based on the regional setting specified in Control Panel.
- If the version of Windows you are running does not support the language you specify, Help Workshop cannot compile the project file.

{button ,AL(`BAS_WRITE_INDEX;BAS_PROJ_FILE_EDITOR;BAS_TRANSLATING')} Related Topics

To change characters that identify second-level index entries

1 In Help Workshop, open the project (.hpj) file.

2 Click Options.

3 Click the Sorting tab.

4 In the box for specifying characters for separating index entries, type one or more characters.

Example

If you specify "@&%", the line "display; display@configuring; keyboard; keyboard&configuring; mouse;mouse%configuring; network, configuring" would appear this way in the index:

display
  configuring
keyboard
  configuring
mouse
  configuring
network, configuring

Note

- The default characters are a comma (,) and colon (:). Any characters you specify here will override these default characters.

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_TRANSLATING')} Related Topics

To save compiler messages to a file

1 In Help Workshop, open the project (.hpj) file.

2 Click Options.

3 Click the Files tab.

4 In the Log File box, type the name of the file. If you want the file to be saved in a folder other than the project folder, specify a path.

Notes

- Help Workshop can display up to 64K of messages in Windows 95 and 1 MB of messages in Windows NT. You can save more messages than Help Workshop can display by specifying a log file, which you can open using any ASCII text editor.
- You can suppress some of the messages Help Workshop displays. For more information, click Related Topics.

{button ,AL(`PRO_HPJ_CONTROL_ERRORS;BAS_PROJ_FILE_EDITOR;BAS_THE_HELP_COMPILER;PRO_HPJ_CONTROL_ERRORS')} Related Topics

To store DLL-related data files within the Help file

1 In Help Workshop, open the project (.hpj) file.

2 Click Data Files.

3 To select a data file, click Add.

Notes

- DLL data filenames are case sensitive. To retrieve a data file, Help uses the filename without the path.
- WinHelp provides callback functions for DLLs to enable a program or DLL to retrieve the appropriate data file from the Help file.
- To remove a data file, click it, and then click Remove.

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAGGAGE')} Related Topics

To assign a contents file to a Help file

1 In Help Workshop, open the project (.hpj) file.

2 Click Options.

3 Click the Files tab.

4 In the Contents File box, type the name of the contents (.cnt) file.

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_CNT_OVERVIEW')} Related Topics

To specify where to store temporary files

1 In Help Workshop, open the project (.hpj) file.

2 Click Options.

3 Click the Files tab.

4 In the TMP Folder box, type the name of the folder you want Help Workshop to use for temporary files.

Notes

- Help Workshop uses the folder you specify for temporary files instead of the folder specified by the TEMP environment variable.
- Help Workshop rarely uses temporary files unless your Help file exceeds 8 MB in size.

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_THE_HELP_COMPILER')} Related Topics

To change the path to source files if you move your Help project

1 In Help Workshop, open the project (.hpj) file.

2 Click Options.

3 Click the Files tab.

4 To specify a new path, click Edit.

Notes

- Substituting a path is unnecessary if all files have been specified using paths that are relative to the location of the project file. Help Workshop automatically uses relative paths when it adds filenames.
- Substituting a path is useful when you move your entire Help project to a different drive or network location. Instead of manually changing all the file paths in your project file, you can quickly change them all here.

{button ,AL(`BAS_PROJ_FILE_EDITOR')} Related Topics

To generate an index file for full-text search

1 In Help Workshop, open the project (.hpj) file.

2 Click Options.

3 Click the FTS tab.

4 Make sure Generate Full Text Search Index is checked, and then change other settings as needed.

Notes

- For help on an item, click ? at the top of the dialog box, and then click the item.
- To generate a full-text search index file when compiling, Maximum compression or Hall compression must be checked on the Compression tab.
- If you do not generate a full-text search index file, WinHelp prompts users to generate it when they first open your Help file and click the Find tab.

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_FULLTEXT_INDEX')} Related Topics

To specify which topics to include in a build

1 In Help Workshop, open the project (.hpj) file.

2 Click Options.

3 Click the Build Tags tab.

4 Add the tags for the topics you want to be included in your help file.

Notes

- For help on an item, click ? at the top of the dialog box, and then click the item.
- Topics without build tags are always included in builds. Topics with build tags are included only if they contain one or more of the build tags you specify on the Build Tags tab.
- You create build tags by adding footnotes to topic (.rtf) files. For more information, click Related Topics.
- If you specify a build tag in either the Include or Exclude section of the Build Tags tab, then Help Workshop will ignore any BUILD option that might have been specified.

{button ,AL(`PRO_INSERT_BUILD_TAG;BAS_PROJ_FILE_EDITOR;BAS_INSERTING_BUILD_TAG')} Related Topics

To change the font used in WinHelp dialog boxes

1 In Help Workshop, open the project (.hpj) file.

2 Click Options.

3 Click the Fonts tab.

4 To change the font, click Change.

Notes

- If you choose a font that is not available on a user's computer, Windows will use the font that is its closest match.
- Changing the font for WinHelp dialog boxes is useful if your Help file will be shipped to international markets unlocalized. For example, WinHelp is localized to use larger fonts in Far East versions. If you do not want large fonts used in WinHelp dialog boxes, you can specify your own.

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_TRANSLATING')} Related Topics

To run two language versions of a Help file side by side

1 Start the original-language Help file and display any topic that is in a main window.

2 Press CTRL+SHIFT+J.

3 Type the following, and then click OK:

!Compare(“filename.ext**”)**

For filename.ext, specify the name of the translated version of the Help file. Specify the path, if necessary, typing two backlashes in place of each one you would typically type.

Note

- For best results, temporarily rename your contents (.cnt) file for both Help files, and start the original-language Help file by typing the following in the command prompt:

  winhlp32 -g filename.ext

  For filename.ext, type the name of the original-language Help file.

- If the translated version of the Help file was properly prepared, moving through one file (by using CTRL+SHIFT+RIGHT ARROW or CTRL+SHIFT+LEFT ARROW) updates both windows.

{button ,AL(`BAS_TRANSLATING;Compare')} Related Topics

To change every occurrence of a font used in a Help file

1 In Help Workshop, open the project (.hpj) file.

2 Click Options.

3 Click the Fonts tab.

4 Click Add, and then specify the settings you want.

Note

- For help on an item, click ? at the top of the dialog box, and then click the item.
- This procedure enables you to change the fonts used in a Help file without having to reformat the text in every topic (.rtf) file. There is no limit to the number of fonts you can replace.

{button ,AL(`BAS_ABOUT_FONTS;NEW_FONT_HANDLING;BAS_PROJ_FILE_EDITOR')} Related Topics

To specify topic files for a Help file

1 In Help Workshop, open the project (.hpj) file.

2 Click Files.

3 To add a file, click Add.

Tip

- To remove a topic (.rtf) file, click it, and then click Remove.

{button ,AL(`BAS_ABOUT_HPJ_FILES;BAS_PROJ_FILE_EDITOR')} Related Topics

To define a secondary window

1 In Help Workshop, open the project (.hpj) file.

2 Click the Windows button.

3 Click Add.

4 Type the name of the secondary window. Do not type **main** as the name of the window.

5 In the list, click the name of the standard window you want to base the secondary window on.

Notes

- For help on an item, click ? at the top of the dialog box, and then click the item.
- You can define up to 255 secondary windows.
- You can define a default window for an entire Help file or for individual topics. For more information, click Related Topics.

{button ,AL(`PRO_CNT_DEFAULT_HELP_FILE;PRO_ADD_WINDOW_TYPE;BAS_PROJ_FILE_EDITOR;BAS_HPJ_CREATE_WINDOW')} Related Topics

To customize the main Help window

1 In Help Workshop, open the project (.hpj) file.

2 Click the Windows button.

3 Click Add.

4 Type the following, and then click OK:

main

5 Click the various tabs to specify the settings you want.

Note

- For help on an item, click ? at the top of the dialog box, and then click the item.

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_HPJ_CREATE_WINDOW')} Related Topics

To add browse buttons to a window

1 In Help Workshop, open the project (.hpj) file.

2 Click the Windows button.

3 Click the Buttons tab.

4 In the Window Type list, click the window type.

5 In the Buttons area, make sure Browse is checked.

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_HPJ_CREATE_WINDOW;BAS_ADDING_BROWSE_CODE')} Related Topics

To specify a title for a window

1 In Help Workshop, open the project (.hpj) file.

2 Click the Windows button.

3 In the Window Type list, click the window type.

4 In the Title Bar Text box, type the title.

Notes

- The title appears in the window's title bar and on its taskbar button.
- If a secondary window does not have a title, but the contents (.cnt) file for the Help file specifies a title, the title from the contents file is used. If neither the secondary window nor the contents file specify a title, no text appears in the title bar.

{button ,AL(`BAS_HELP_TITLE;BAS_PROJ_FILE_EDITOR;BAS_HPJ_CREATE_WINDOW')} Related Topics

To make a window stay on top by default

1 In Help Workshop, open the project (.hpj) file.

2 Click the Windows button.

3 In the Window Type list, click the window type.

4 In the Window Attributes area, make sure Keep Help Window On Top is checked.

Note

- Users can change this setting by clicking the Help window using their right mouse button, and then pointing to Keep Help On Top.

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_HPJ_CREATE_WINDOW')} Related Topics

To specify the size and position of a window

1 In Help Workshop, open the project (.hpj) file.

2 Click the Windows button.

3 Click the Position tab.

4 In the Window Type list, click the window type.

5 To set the size and position of the window, click Auto-Sizer.

Notes

- If you want WinHelp to determine the position of a window, click Default Positions.
- If you want to set the size and position manually, type the settings in the Left, Top, Width, and Height boxes.
- If you want to automatically size the secondary window to the length of the topic, click the General tab, and then make sure Auto-Size Height is checked. (This feature is not available for the main Help window.)

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_HPJ_CREATE_WINDOW')} Related Topics

To specify background colors for a window

1 In Help Workshop, open the project (.hpj) file.

2 Click the Windows button.

3 Click the Color tab.

4 In the Window Type list, click the window type.

5 To select colors for the nonscrolling and scrolling regions, click the Change buttons.

Note

- The background color you specify can be overridden if users change their window or window text colors in Display properties or if users click a topic using their right mouse button, and then click Use System Colors.
- If you use a transparent bitmap in both the scrolling region and the non-scrolling region of the same topic, the transparent area of both bitmaps is displayed using the background color of the scrolling region.

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_HPJ_CREATE_WINDOW')} Related Topics

To add buttons to a secondary window

1 In Help Workshop, open the project (.hpj) file.

2 Click the Windows button.

3 Click the Buttons tab.

4 In the Window Type list, click the window type.

5 In the Buttons area, make sure the buttons you want to appear in the window are checked.

Tips

- If you do not want a button bar, make sure all the button types in the Buttons area are cleared.
- You can add custom buttons by clicking the Macros tab and adding CreateButton macros. You cannot add custom buttons unless you have used at least one standard WinHelp button.

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_HPJ_CREATE_WINDOW;CreateButton')} Related Topics

To add buttons to the main Help window

1 In Help Workshop, open the project (.hpj) file.

2 Click the Windows button.

3 If "main" appears in the Window Type list, click it.

If "main" does not appear in the list, click Add, type the following, and then click OK:

main

4 Click the Buttons tab.

5 In the Buttons area, make sure the buttons you want to appear in the window are checked.

Note

- If you do not want a button bar, make sure all the button types in the Buttons area are cleared.

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_HPJ_CREATE_WINDOW')} Related Topics

To run a macro when a window is opened

1 In Help Workshop, open the project (.hpj) file.

2 Click the Windows button.

3 Click the Macros tab.

4 In the Window Type list, click the window type.

5 Click Add, and then specify the macro you want to run.

Notes

- To remove a macro, click it, and then click Remove.
- To specify macros that run when the Help file is opened, open your project file, and then click the Config button.

{button ,AL(`MACRO_REF_OVR;BAS_PROJ_FILE_EDITOR;BAS_HPJ_CREATE_WINDOW')} Related Topics

To run a macro when an index entry is clicked

1 In Help Workshop, open the project (.hpj) file.

2 Click Options, and then click the Macros tab.

3 Click Add, and then specify the settings you want.

Notes

- For help on an item, click ? at the top of the dialog box, and then click the item.
- Running a macro when users click an index entry is useful for starting a wizard, a series of training cards, or a DLL.
- The keywords you assign here will appear in the index regardless of whether they are specified as K-footnotes in topics.
- The title you specify will appear only if users double-click a keyword that is linked to a macro and to one or more topics. In this case, clicking the title in the Topics Found dialog box starts the macro.

To specify the location of bitmap files

1 In Help Workshop, open the project (.hpj) file.

2 Click Bitmaps.

3 Click Add, and then specify the folder that contains the bitmaps.

Tips

- By default, Help Workshop looks for bitmaps in the same folder as your project file and the current topic (.rtf) file, and any folders specified for topic files.
- To remove a folder, click it, and then click Remove.

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

To enable a program to display an individual Help topic

1 In Help Workshop, open the project (.hpj) file.

2 Click Map.

3 If you want to add an individual topic, click Add.

If you want to include a C header file, click Include.

Note

- When assigning topic IDs to context-sensitive Help topics, make sure the first four characters in the ID are IDH_. Help Workshop recognizes that topic IDs beginning with these characters are ones that a program will call. While compiling, Help Workshop displays a list of any topic IDs that are in your topic (.rtf) files, but have not been mapped to numeric values. Help Workshop also lists IDs that have been mapped to numeric values, but are not in your topic files. Using the IDH_ naming convention will help you find and resolve problems in your context-sensitive Help. If you want to use a prefix other than IDH_, specify it in the Map dialog box.

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_HPJ_ADD_MAP')} Related Topics

To alias one topic ID to another

1 In Help Workshop, open the project (.hpj) file.

2 Click Alias.

3 Click Add, and then specify the settings you want.

Tips

- For help on an item, click ? at the top of the dialog box, and then click the item.
- Aliases are an easy way to redirect jumps to topics without changing every instance of a jump throughout the Help file.
- Aliases are also useful where you've made a change that causes a mapped numeric value to refer to a topic other than the one you intended. An alias enables you to display the intended topic in the Help file without changing the code in the program.

To run a macro when a Help file is opened

1 In Help Workshop, open the project (.hpj) file.

2 Click Config.

3 Click Add, and then specify the settings you want.

Tips

- To remove a macro, click it, and then click Remove.
- You can also assign a macro to run every time a particular window type opens, or any time a specific topic is displayed. For more information, click Related Topics.

{button ,AL(`BAS_PROJ_FILE_EDITOR;MACRO_REF_OVR;PRO_HPJ_WINDOW_CONFIG_MACROS;PRO_ADD_MACRO_COMMAND')} Related Topics

The Compile button looks like this:

To create a contents file

1 In Help Workshop, click the File menu, and then click New.

2 Double-click Help Contents.

3 In the Default Filename box, type the Help filename that most of your topics are in.

4 In the Default Title box, type the text you want to appear in the title bar of the Help Topics dialog box. This text also appears in title bars that do not have a title specified in the [WINDOWS] section of the project file.

5 Click Add Above or Add below to add the headings and topics that you want to display on the Contents tab.

6 If you want to include the keywords of several files on the Index tab, click Index Files and specify the files you want to include.

{button ,AL(`BAS_CNT_OVERVIEW;PRO_ADD_MULT_KEYWORDS')} Related Topics

To specify a default window in which to display topics

1 In Help Workshop, open your contents (.cnt) file.

2 Click the Edit button in the upper right corner of the Help Workshop window.

3 In the Default Window box, specify the window type.

Notes

- Topics appear in the default window when users open them from the Index and Find tabs, unless a different window is specified for a topic by a window-type footnote (>). Topics also appear in the default window when users open them from the Contents tab, unless the contents entry includes a file or window type specification.
- If a default window is not specified, topics appear in the main Help window.
- The window type you specify must be defined in your project (.hpj) file. For more information, click Related topics.

{button ,AL(`BAS_CNT_OVERVIEW;PRO_HPJ_CREATE_WINDOW')} Related Topics

To include additional contents files

1 Open the main contents file, and then click the place where you want the information in the second contents file to appear.

2 Click Add Above or Add Below, and then click Include.

3 Type the name of the contents file that you want to include.

Notes

- If the included contents file is not available when WinHelp initializes the Help file, the included file will be not appear on the Contents tab.
- A default Help filename specified in the included contents file affects only the included file, not the main contents file.

{button ,AL(`BAS_CNT_COMBINE_HLP_FILES;BAS_CNT_OVERVIEW')} Related Topics

To add headings to the Contents tab

1 In Help Workshop, open the contents (.cnt) file.

2 Click the item above or below which you want to add a heading, and then click Add Above or Add Below.

3 Click Heading, and then type the heading in the Title box.

Note

- You can assign up to nine levels of headings and topics by using the Move Right and Move Left buttons.

{button ,AL(`BAS_CNT_OVERVIEW;PRO_CNT_ADD_TOPIC')} Related Topics

To add topics to the Contents tab

1 In Help Workshop, open the contents (.cnt) file.

2 Click the item above or below which you want to add a topic, and then click Add Above or Add Below.

3 Click Topic, and then type a title in the Title box.

4 In the Topic ID box, type the topic ID.

5 If the topic is not in the default Help file, type the name of the file that contains the topic in the Help File box.

6 If you want to display the topic in a window type other than the default window, specify the window in the Window Type box. The window you specify must be defined in the project (.hpj) file.

Notes

- If you do not specify information in the Help File, WinHelp searches the default Help file for the topic. If you do not specify information in the Window Type box, WinHelp displays the topic in the window type specified in the default Help file.
- You can assign a macro to a topic instead of a topic ID. In the Topic ID box, type an exclamation point (!) followed by the macro. To separate multiple macros, use colons (:).

{button ,AL(`BAS_CNT_OVERVIEW;PRO_CNT_ADD_BOOK')} Related Topics

To change the level of a heading or topic

1 In Help Workshop, open the contents (.cnt) file.

2 Click the heading whose level you want to change.

3 Click the Move Right or Move Left button.

Note

- When a heading is indented to the left or right, its topics and subheadings move with it.

{button ,AL(`BAS_CNT_OVERVIEW')} Related Topics

To combine keywords from several Help files

1 In Help Workshop, open the contents (.cnt) file.

2 Click the Index Files button.

3 Click Add.

4 In the Help Title box, type the title of the Help file. The title will appear if users choose Custom in the Find Setup Wizard. It also appears in the Topics Found dialog box if WinHelp finds identical topic titles in different Help files.

5 In the Help Filename box, type the name of the Help file whose keywords you want to add to your index.

Note

? All Help files included in the index are automatically searched when an ALink or KLink macro is run.

{button ,AL(`BAS_CNT_OVERVIEW;BAS_CNT_COMBINE_HLP_FILES')} Related Topics

To enable ALink and KLink jumps to other Help files

1 In Help Workshop, open the contents (.cnt) file.

2 Click the Link Files button.

3 Click Add, and then type the name of the Help file that you want to search when an ALink or KLink macro is run.

Note

- You do not need add files that have already been included in the Index Files dialog box.

{button ,AL(`BAS_CNT_OVERVIEW;BAS_CNT_COMBINE_HLP_FILES')} Related Topics

To add another tab to the Help Topics dialog box

1 In Help Workshop, open the contents (.cnt) file.

2 Click Tabs, and then click Add.

3 In the Tab Name box, type the name you want to display on the tab.

4 In the DLL Filename box, type the filename of the DLL that provides the dialog procedure to support the tab.

{button ,AL(`new_tab;DLL_WRITE_OVERVIEW;BAS_CNT_OVERVIEW')} Related Topics

To translate the contents file

1 In Help Workshop, click the File menu, and then make sure Translation is checked.

2 Open the contents (.cnt) file.

3 Select a heading or topic you want to translate, and then click Edit.

Note

- When the translation command is checked, you can modify only the visible text that appears on the Contents tab. You cannot modify topic IDs, Help filenames, or window names.

{button ,AL(`BAS_CNT_OVERVIEW;BAS_TRANSLATING')} Related Topics

To compile a Help file

1 If the project file is already open in Help Workshop, save the file.

2 Click the Compile button.

3 In the Project File box, type the name and location of the project (.hpj) file you want to compile, and then click Compile.

Tips

- You do not have to open a project file to compile it.
- If a project file is open, you can click the Save And Compile button to compile the file.
- For information about the other settings in this dialog box, click ? at the top of the dialog box, and then click the setting.

{button ,AL(`BAS_ABOUT_HPJ_FILES;EXEC_HCW')} Related Topics

To compile more than one Help file at a time

1. Using a text editor, create an ASCII text file that lists the full path of each of your project files on a separate line. Specify the project files in the order in which you want Help Workshop to compile them.
2. Save the file using the filename extension .hmk.
3. Open Help Workshop, and then click Compile button.
4. In the Project File box, type the name of your .hmk file, and then click Compile.

{button ,AL(`BAS_ABOUT_HPJ_FILES;EXEC_HCW')} Related Topics

To view a Help file from Help Workshop

1 After you compile a Help file, click the Run WinHelp button.

2 Specify the name of the Help file.

3 If you want to view topics whose IDs are mapped to numeric values in your project (.hpj) file, type the name and location of the project file and click Refresh.

4 Click View Help.

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

To test a contents file

1 In Help Workshop, click the Test menu, and then click Contents File.

2 Type the name of the contents (.cnt) file you want to test.

Help Workshop checks your contents file to make sure that the syntax is correct. It then jumps to every topic specified in the contents file and displays a message if one of the jumps does not work.

Note

- If your contents file includes other contents files, make sure they are available to be tested.

{button ,AL(`BAS_ABOUT_HPJ_FILES;PRO_HCW_CLOSE_ALL_HELP')} Related Topics

To monitor WinHelp messages

1 In Help Workshop, click the View menu, and then click WinHelp Messages.

2 Click the View menu again, and then click Message Options to specify what WinHelp messages you want to view.

{button ,AL(`BAS_TESTING_HELP_FILES')} Related Topics

To close all instances of WinHelp

In Help Workshop, click the Test menu, and then click Close All Help.

Note

- This procedure is useful if you are testing Help files and run into problems because WinHelp is running in the background.

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

To generate a report on a Help file

1 In Help Workshop, click the File menu, and then click Report.

2 In the Help Filename box, type the path and name of the Help (.hlp) file.

3 In the Output Filename box, type the path and name of the file in which you want to save the report.

4 In the Display area, select the type of report you want, and then click Report.

Note

- For help on an item, click ? at the top of the dialog box, and then click the item.

{button ,AL(`BAS_TESTING_HELP_FILES')} Related Topics

To simulate a program call to the WinHelp API

1 In Help Workshop, click the Test menu, and then click WinHelp API.

2 In the Help File box, type the path and name of the Help file you want to test.

3 In the Command list, click the command you want to send in the WinHelp API.

4 In the box at the bottom of the dialog box, type the value you want to send.

5 Click Call.

{button ,AL(`BAS_TESTING_HELP_FILES')} Related Topics

The Run WinHelp button looks like this:

Macro quick reference

The following list organizes Help macros according to function and provides a quick overview of macros that you may want to use when you customize your Help file.

Button macros

Keyboard macros

Linking macros

Menu macros

Program macros

Text-Marker macros

Window macros

{button ,AL(`MACRO_OVERVIEW')} Related Topics

Button macros

You can use the following macros to access standard Help buttons, create new buttons, or modify button functionality. Macros that are new in WinHelp version 4.0 are noted with an asterisk.

Back	Displays the topic that was previously displayed in the current window.
BrowseButtons	Adds browse buttons to the Help button bar.
ChangeButtonBinding	Changes the macro assigned to a button on the button bar.
ChangeEnable*	Assigns a macro to a button on the button bar and enables that button.
Contents	Displays the Contents tab or default topic of the current Help file.
CreateButton	Creates a new button and adds it to the button bar.
DestroyButton	Removes a button from the button bar.
DisableButton	Disables a button on the button bar.
EnableButton	Enables a button on the button bar.
Find*	Displays the Find tab of the Help Topics dialog box.
Finder*	Displays the Help Topics dialog box in its last state.
Menu*	Displays the context menu, which is typically accessed by using the right mouse button.
Next	Displays the next topic in a browse sequence.
Prev	Displays the previous topic in a browse sequence.
Search	Displays the Index tab of the Help Topics dialog box.

{button ,AL(`MACRO_REF_OVR')} Related Topics

Menu macros

You can use the following macros to create or modify menus and menu items.

AppendItem	Appends a menu item to the end of a menu.
ChangeItemBinding	Changes the macro assigned to a menu item.
CheckItem	Displays a check mark next to a menu item.
DeleteItem	Removes a menu item from a menu.
DisableItem	Disables a menu item.
EnableItem	Enables a disabled menu item.
ExtAbleItem	Enables or disables a menu item.
ExtInsertItem	Inserts a menu item at a given position on a menu, in a given state.
ExtInsertMenu	Inserts a submenu in a previously defined menu.
InsertItem	Inserts a menu item at a given position on a menu.
InsertMenu	Adds a new menu to the Help menu bar.
ResetMenu	Resets the Help menu bar and menus to their default states.
UncheckItem	Removes a check mark from a menu item.

{button ,AL(`MACRO_REF_OVR')} Related Topics

Linking macros

You can use the following macros to create hypertext links to specific Help topics. Macros that are new in WinHelp version 4.0 are noted with an asterisk.

ALink*	Jumps to the topics that contain the specified A-keyword.
JumpContents	Jumps to the contents topic of the specified Help file.
JumpContext	Jumps to the topic specified by a context number.
JumpHash	Jumps to the topic specified by a topic hash number.
JumpHelpOn	Jumps to the contents topic of the How To Use Help file.
JumpId	Jumps to the topic specified by a topic ID.
JumpKeyword	Jumps to the topic containing a specified K-keyword.
KLink*	Jumps to the topics that contain the specified K-keywords.
PopupContext	Displays in a pop-up window the topic specified by a context number.
PopupHash	Displays in a pop-up window the topic specified by a topic hash number.
PopupId	Displays in a pop-up window the topic specified by a topic ID.
UpdateWindow*	Jumps to the specified topic in the specified window, and then returns the focus to the window that called the macro.

{button ,AL(`MACRO_REF_OVR')} Related Topics

Window macros

You can use the following macros to control or modify the behavior of various Help windows. Macros that are new in WinHelp version 4.0 are noted with an asterisk.

CloseSecondarys*	Closes all Help windows except the current secondary window.
CloseWindow	Closes the main or secondary Help window.
FocusWindow	Changes the focus to a specific Help window.
HelpOnTop	Places all Help windows on top of other windows.
PositionWindow	Sets the size and position of a Help window.
SetPopupColor*	Sets the background color of pop-up windows.

{button ,AL(`MACRO_REF_OVR')} Related Topics

Keyboard macros

You can use the following macros to add keyboard access to a Help macro.

AddAccelerator	Assigns an accelerator key to a Help macro.
RemoveAccelerator	Removes an accelerator key from a Help macro.

{button ,AL(`MACRO_REF_OVR')} Related Topics

Program macros

You can use the following macros to access programs. These macros are new in WinHelp version 4.0.

ControlPanel	Opens a specific tab in a Control Panel dialog box.
ExecFile	Runs a program or opens a file and runs the program associated with that file.
FileExist	Checks to see whether the specified file exists on a user's computer.
ShellExecute	Opens, prints or runs a file or program.
ShortCut	Runs or activates a program and sends it a WM_COMMAND message.

{button ,AL(`MACRO_REF_OVR;ALink;KLink')} Related Topics

Text-marker macros

You can use the following macros to create and manipulate text markers. Macros that are new in WinHelp version 4.0 are noted with an asterisk.

DeleteMark	Removes a marker added by the SaveMark macro.
GotoMark	Jumps to a marker set by the SaveMark macro.
IfThen	Runs a Help macro if a given marker exists.
IfThenElse	Runs one of two macros if a given marker exists.
IsMark	Tests whether a marker set by the SaveMark macro exists.
IsNotMark*	Tests whether a marker set by the SaveMark macro does not exist.
Not	Reverses the result returned by the IsMark macro.
SaveMark	Saves a marker for the current topic, window and Help file.

{button ,AL(`MACRO_REF_OVR')} Related Topics

About macro

About()

Displays WinHelp's Version dialog box.

Parameters

This macro does not take any parameters.

Comment

You can add your own text to the Version dialog box by putting a COPYRIGHT string in the project file.

{button ,AL("MACRO_REF_MENU;macro_all;COPYRIGHT")} Related Topics

AddAccelerator macro

AddAccelerator(key**,** shift-state**,** macro**)**

Assigns a Help macro to an accelerator key (or key combination). WinHelp runs the macro whenever a user presses the accelerator key.

Parameter	Description
key	Specifies the Windows virtual-key value. This parameter can be a numeric digit, a quoted character (as in `A'), or one of the following strings:

VK_LBUTTON	VK_NUMPAD2
VK_RBUTTON	VK_NUMPAD3
VK_CANCEL	VK_NUMPAD4
VK_MBUTTON	VK_NUMPAD5
VK_BACK	VK_NUMPAD6
VK_TAB	VK_NUMPAD7
VK_CLEAR	VK_NUMPAD8
VK_RETURN	VK_NUMPAD9
VK_SHIFT	VK_MULTIPLY
VK_CONTROL	VK_ADD
VK_MENU	VK_SEPARATOR
VK_PAUSE	VK_SUBTRACT
VK_CAPITAL	VK_DECIMAL
VK_KANA	VK_DIVIDE
VK_KANJI	VK_F1
VK_HANGEUL	VK_F3
VK_JUNJA	VK_F5
VK_HANJA	VK_F6
VK_ESCAPE	VK_F7
VK_SPACE	VK_F8
VK_PRIOR	VK_F9
VK_NEXT	VK_F10
VK_END	VK_F11
VK_HOME	VK_F12
VK_LEFT	VK_F13
VK_UP	VK_F14
VK_RIGHT	VK_F15
VK_DOWN	VK_F16
VK_SELECT	VK_F17
VK_PRINT	VK_F18
VK_EXECUTE	VK_F19
VK_SNAPSHOT	VK_F20
VK_INSERT	VK_F21
VK_DELETE	VK_F22
VK_HELP	VK_F23
VK_SCROLL	VK_F24
VK_NUMPAD0	VK_NUMLOCK
VK_NUMPAD1	

shift-state	Specifies the combination of ALT, SHIFT, and CTRL keys to be used with the accelerator. This parameter can be one of the following values:

NONE (0)

SHIFT (1)

	CTRL (2)
	SHIFT+CTRL (3)
	ALT (4)
	ALT+SHIFT (5)
	ALT+CTRL (6)
	SHIFT+ALT+CTRL (7)
macro	Specifies the Help macro or macro string to run when the user presses the accelerator key(s). Multiple macros in a string must be separated by colons or semicolons.

Comment

Help Workshop automatically converts the AddAccelerator macro to AA.

Example

The following macro runs the Notepad program when a user presses SHIFT+CTRL+F9:

```
AddAccelerator(VK_F9, SHIFT+CTRL, ExecFile(notepad))
```

{button ,AL("MACRO_REF_KEYBOARD;removeaccelerator;macro_all")} Related Topics

ALink macro

ALink(keyword[**;** keyword] [**,** type[**,** topic-ID [**,** window-name]]]**)**

Searches for keywords specified by A-footnotes.

Parameter	Description
keyword	Specifies one or more keywords to search for. Multiple keywords must be separated by a semicolon. If any keyword contains a comma, the entire keyword string must be enclosed in quotation marks.
type	Specifies the action to perform if one or more keywords are found. If this parameter is not specified or is zero, the default action is always to display the Topics Found dialog box containing the topic title. This parameter may specify one or more of the following values, separated by spaces.

Value	Meaning
JUMP (1)	Specifies that if only one topic matches any of the keywords, WinHelp should jump directly to that topic.
TITLE (2)	Specifies that if a keyword is found in more than one Help file, WinHelp should display the title of the Help file (as specified in the contents [.cnt] file) beside the topic title in the Topics Found dialog box.
TEST (4)	Specifies that the macro should return a value indicating whether or not there is at least one match. The TestALink macro is converted by Help Workshop into an ALink macro with this parameter.

topic-ID	Specifies the topic to display in a pop-up window if no matches are found. If this parameter is not specified, WinHelp displays a message box with the text “No additional information is available”. To specify a topic in a different Help file, the topic ID should end with an ‘@’ character and the name of the Help file.
window-name	Specifies the window type in which to display the topic. If this parameter is not specified, the window type that is specified for a topic (if one is defined) is used, or the default or current window is used. If this macro results in an interfile jump, the window type must be defined in the project file for the Help file that is being jumped to.

Comments

The ALink macro searches for A-keyword matches in the current Help file. If the Help file is associated with a contents file, WinHelp searches all Help files specified in the contents file (by the :Index and :Link commands) for matching A-keywords.

The ALink macro is identical to the KLink macro except that it searches for A-keywords instead of K-keywords.

Help Workshop automatically converts the ALink macro to AL, which is the only form of the macro that WinHelp can use.

The ALink macro is new for WinHelp version 4.0.

Examples

The following example generates a list of topics that contain the “network printing” or “local printing” keywords in A-footnotes:

```
ALink(network printing;local printing)
```

The following example generates a list of topics that contain the keyword “how_to” in A-footnotes and displays those topics in the Steps window:

```
ALink(how_to, , , Steps)
```

The following example returns a value of “1” if at least one topic with the keyword “saving” is found:

```
ALink(saving, TEST)
```

{button ,AL("MACRO_REF_LINK;KLink;macro_all")} Related Topics

Annotate macro

Annotate()

Displays the Annotation dialog box.

Parameters

This macro does not take any parameters.

Comment

Do not run the Annotate macro from a pop-up window.

{button ,AL("MACRO_REF_MENU;macro_all")} Related Topics

AppendItem macro

AppendItem(menu-ID**,** item-ID**,** item-name**,** macro**)**

Appends a menu item to the end of an existing menu.

Parameter	Description
menu-ID	Specifies the name used in the InsertMenu macro that creates the menu, or the name of a standard WinHelp menu. The new item is appended to this menu.
item-ID	Specifies the name that WinHelp uses internally to identify the menu item. This name is used by the DisableItem, DeleteItem, EnableItem, and ExtAbleItem macros.
item-name	Specifies the name that WinHelp displays on the menu for the item. Place an ampersand (&) before the character used for the macro's accelerator key.
macro	Specifies one or more macros that WinHelp should run when a user chooses the menu item. Separate multiple macros with colons or semicolons (use only colons if multiple macros are specified in a project file).

Comments

WinHelp ignores this macro if it is run in a secondary window.

You can append as many as 20 items to WinHelp's context menu (mnu_floating).

If the keyboard accelerator conflicts with other menu access keys, WinHelp ignores the macro.

Help Workshop automatically converts the AppendItem macro to AI.

Example

The following macro appends a menu item labeled "Tools." Choosing the menu item causes a jump to a topic with the topic ID "tpc1" in the Tls.hlp file:

```
AppendItem(IDM_BKS, IDM_TLS, &Tools, JumpId(tls.hlp, tpc1))
```

{button ,AL("MACRO_REF_MENU;ResetMenu;ChangeItemBinding;DeleteItem;DisableItem;ExtAbleItem;ExtInsertItem;InsertItem;InsertMenu;macro_all;REF_STD_MENU;FloatingMenu")} Related Topics

Back macro

Back()

Displays the previous topic in the history list of the current window. The history list is a list of the last 30 topics (not including pop-up topics) that the user displayed in the current window. This list is kept separately from the list displayed in the History window.

Parameters

This macro does not take any parameters.

Comments

The Back macro works only for topics that were displayed in the current window type; it does not display topics that were displayed in other window types.

If a user runs the Back macro when the Back list is empty, WinHelp takes no action.

A user can use Back to back up to a topic only once. After Back is used to return to a topic, that topic is removed from the list of previously displayed topics. For example, after jumps to topics A, B, and C, the user can use Back to return to B. After a jump back to C from B, however, Back goes to A, not B.

{button ,AL("macro_all;History;BACKTRACK;BackFlush")} Related Topics

BackFlush macro

BackFlush()

Removes the back history list from the current window. This macro does not affect the history list displayed in the History window.

Parameters

This macro does not take any parameters.

Comment

Help Workshop automatically converts the BackFlush macro to BF, which is the only form of this macro that WinHelp can use.

{button ,AL("MACRO_REF_BUTTON;macro_all;Back")} Related Topics

BookmarkDefine macro

BookmarkDefine()

Displays the Define dialog box, which is available from the Bookmark menu in a main Help window.

Parameters

This macro does not take any parameters.

Comments

Bookmarks do not keep track of the window type the topic was originally displayed in. When you use the Bookmark menu or call the BookmarkMore macro, the bookmark topic is displayed in the current window type.

If the BookmarkDefine macro is run from a pop-up window, the bookmark is attached to the topic that invoked the pop-up window.

{button ,AL("MACRO_REF_MENU;macro_all;BookmarkMore")} Related Topics

BookmarkMore macro

BookmarkMore()

Displays a Bookmark dialog box, that enables users to choose which bookmarked topic to display.

Parameters

This macro does not take any parameters.

Comment

If a bookmark is chosen from the Bookmark dialog box, the topic is displayed in the current window type.

{button ,AL("MACRO_REF_MENU;macro_all;BookmarkDefine")} Related Topics

BrowseButtons macro

BrowseButtons()

Adds browse buttons to the button bar.

Parameters

This macro does not take any parameters.

Comments

If the BrowseButtons macro is used with one or more CreateButton macros in the [CONFIG] section of the project file, the order of the browse buttons on the WinHelp button bar is determined by the order of the BrowseButtons macro in relation to the other macros listed in the [CONFIG] section.

Browse buttons cannot be removed from the button bar after they have been added with the BrowseButtons macro.

For secondary windows, you can add browse buttons as part of the window definition in the project file.

Example

The following macros in the project file cause the Notepad button to appear immediately before the two browse buttons on the button bar:

```
[CONFIG]
CreateButton(btn_clock,&Notepad, ExecFile(notepad))
BrowseButtons()
```

{button ,AL("MACRO_REF_BUTTON;macro_all;CreateButton;Next;Prev;CONFIG")} Related Topics

ChangeButtonBinding macro

ChangeButtonBinding(button-ID**,** button-macro**)**

Assigns a macro to a button bar button.

Parameter	Description
button-ID	Specifies the identifier assigned to the button by the CreateButton macro, or for a standard Help button.
button-macro	Specifies the macro assigned to the button. Separate multiple macros using colons or semicolons. If multiple macros are specified in a project file, use colons.

Comment

Help Workshop automatically converts the ChangeButtonBinding macro to CBB.

Example

In the following macro, an ALink macro is assigned to a button whose ID is SeeAlso:

```
[CONFIG]
ChangeButtonBinding(SeeAlso, ALink(another topic))
```

{button ,AL("MACRO_REF_BUTTON;macro_all;CreateButton;ChangeEnable;DestroyButton;DisableButton;EnableButton;REF_STD_BTN;CONFIG")} Related Topics

ChangeEnable macro

ChangeEnable(button-ID**,** button-macro**)**

Assigns a macro to a button bar button and enables that button.

Parameter	Description
button-ID	Specifies the identifier assigned to the button by the CreateButton macro, or for a standard Help button.
button-macro	Specifies the macro assigned to the button.

Comments

This macro is equivalent to calling both ChangeButtonBinding and EnableButton.

Help Workshop automatically converts the ChangeEnable macro to CE, which is the only form of this macro that WinHelp can use.

The ChangeEnable macro is new for WinHelp version 4.0.

Example

In the following macro, an ALink macro is assigned to a button whose ID is SeeAlso:

```
ChangeEnable(SeeAlso, ALink(another topic))
```

{button ,AL("MACRO_REF_BUTTON;macro_all;CreateButton;REF_STD_BTN")} Related Topics

ChangeItemBinding macro

ChangeItemBinding(item-ID**,** item-macro**)**

Assigns a new macro to a WinHelp menu item.

Parameter	Description
item-ID	Identifies the menu item. The ID must have been created with the AppendItem or InsertItem macros, or must be the ID for a standard WinHelp menu item.
item-macro	Specifies the Help macro or macro string to run when a user chooses the menu item. Separate multiple macros in a string using colons or semicolons.

Comments

This macro is ignored if a user runs it in a secondary window.

Help Workshop automatically converts the ChangeItemBinding macro to CIB.

Example

The following macro changes the menu item identified by "notepad_item" to open the Notepad program:

```
ChangeItemBinding(notepad_item, ExecFile(notepad))
```

{button ,AL("MACRO_REF_MENU;macro_all;AppendItem;InsertItem;DeleteItem;EnableItem;ExtAbleItem;ExtInsertMenu;UncheckItem;ResetMenu;REF_STD_MENU")} Related Topics

CheckItem macro

CheckItem(item-ID**)**

Places a check mark beside a menu item.

Parameter	Description
item-ID	Identifies the menu item. The ID must have been created using the AppendItem or InsertItem macros, or must be a standard ID for a WinHelp menu item.

Comments

This macro is ignored if a user runs it in a secondary window.

Use the UncheckItem macro to clear a check mark.

Help Workshop automatically converts the CheckItem macro to CI.

{button ,AL("MACRO_REF_MENU;macro_all;UncheckItem")} Related Topics

CloseSecondarys macro

CloseSecondarys()

Closes all but the current secondary window.

Parameters

This macro does not take any parameters.

Comments

Help Workshop automatically converts the CloseSecondarys macro to CS, which is the only form of this macro that WinHelp can use.

The CloseSecondarys macro is new for WinHelp version 4.0.

{button ,AL("MACRO_REF_WINDOW;macro_all;CloseWindow;Exit")} Related Topics

CloseWindow macro

CloseWindow([window-name]**)**

Closes either a secondary window or the main Help window.

Parameter	Description
window-name	Specifies the name of the window to close. The name "main" is reserved for the main Help window. For secondary windows, the window name is defined in the [WINDOWS] section of the project file.

Comments

If no name is specified, the main window is closed.

If the main window is closed, and no other windows are open, WinHelp will quit.

If the specified window-name does not exist, WinHelp ignores this macro.

Help Workshop automatically converts the CloseWindow macro to CW.

Example

The following macro closes the secondary window named "keys":

```
CloseWindow(keys)
```

{button ,AL("MACRO_REF_WINDOW;macro_all;CloseSecondarys;Exit;WINDOWS")} Related Topics

Compare macro

Compare(HLP-filename**)**

Displays a Help file in a second instance of WinHelp. The current Help file and the second Help file are displayed side-by-side. Most actions performed in one Help file (for example, clicking jumps or the Back and browse buttons) will be automatically reflected in the other file.

Parameter	Description
HLP-filename	Specifies the name of the Help file to display.

Comments

This macro is useful for comparing original and translated versions of the same Help file. You can run this macro from the Jump dialog box by typing the following in the Enter Topic Identifier text box:

!compare("filename.hlp")

For filename.hlp, substitute the name of the file that you want to compare to the one that is already open.

The Compare macro is new for WinHelp version 4.0.

{button ,AL("macro_all;new_help_author;BAS_TRANSLATING;Test")} Related Topics

Contents macro

Contents()

Displays the Contents tab in the Help Topics dialog box if the Help file is associated with a contents file that specifies Contents tab information. If the Help file is not associated with a contents file, the Contents macro displays the default (contents) topic in the current Help file.

Parameters

This macro does not take any parameters.

Comments

The contents file is specified by the CNT option in the [OPTIONS] section of the project file. If the project file does not have a CNT option, WinHelp searches for a .cnt file with the same base name as the current Help file.

The default topic is defined by the CONTENTS option in the [OPTIONS] section of the project file. If the project file does not have a CONTENTS option, the first topic in the first topic file specified in the [FILES] section in the project file is used as the default topic.

{button ,AL("MACRO_REF_BUTTON;macro_all;JumpContents;SetContents;CNT;CONTENTS_hpj")} Related Topics

ControlPanel macro

ControlPanel(CPL_name[**,** panel_name**,** tabnum]**)**

Opens a control panel applet with a specific tab on top.

Parameter	Description
CPL_name	Specifies the name of the program that contains the control panel applet.
panel_name	Specifies the name of the control panel applet. This must be identical to the text that appears under the control panel applet's icon.
tabnum	Specifies the number of the tab to display on top. The first tab is number 0, the second tab is number 1, and so on.

Comments

Help Workshop converts the ControlPanel macro into the ExecFile macro, which is in turn converted to EF. EF is the only form of this macro that WinHelp can use.

Not all control panel applets recognize the panel_name and tabnum parameters.

The ControlPanel macro is new for WinHelp version 4.0.

{button ,KL("macros, auxiliary;macros, syntax;ShortCut macro;ExecFile macro ")} Related Topics

CopyDialog macro

CopyDialog()

This macro is obsolete. In version 3.x of WinHelp, this macro displayed a dialog box. Later versions of Help simply copied the entire topic to the Clipboard. New Help files should use the CopyTopic macro instead.

Parameters

This macro does not take any parameters.

{button ,AL("CopyTopic")} Related Topics

CopyTopic macro

CopyTopic()

Copies all the text in the currently displayed topic to the Clipboard.

Parameters

This macro does not take any parameters.

Comments

The CopyTopic macro copies only text, not bitmaps or other graphics.

If the CopyTopic macro is run from a pop-up window, only the text of the topic that invoked the pop-up window is placed on the Clipboard.

Help Workshop automatically converts the CheckTopic macro to CT.

{button ,AL(`MACRO_REF_MENU;macro_all')} Related Topics

CreateButton macro

CreateButton(button-ID**,** name**,** macro**)**

Appends a new button to the button bar.

Parameter	Description
button-ID	Specifies the name that WinHelp uses internally to identify the button. Use this name in the EnableButton, DisableButton or DestroyButton macro to enable, disable, or destroy the button. Use this name in the ChangeButtonBinding macro to change the macro that the button runs in certain topics.
name	Specifies the text that appears on the button. To specify which letter in this text should be used as the accelerator key for the button, place an ampersand (&) before that letter (for more information, see the Comments section). You can use as many as 96 characters on a button label.
macro	Specifies the macro or macros WinHelp should run when a user clicks the button. Separate multiple macros using colons or semicolons.

Comments

If no accelerator is specified, WinHelp automatically chooses the first letter that can be used as an accelerator without conflicting with any previous buttons on the button bar. If an accelerator is specified and it conflicts with a previous button (including standard WinHelp buttons), WinHelp attempts to move the accelerator to the first letter that does not conflict. If no such letter can be found, the accelerator will not be changed. If users attempt to use the accelerator in this case, WinHelp displays a dialog box asking them to choose which button they want.

WinHelp allows a maximum of 22 buttons on a button bar.

If the BrowseButtons macro is used with one or more CreateButton macros in the project file, the buttons appear in the same order on the button bar as the macros appear in the project file.

Help Workshop automatically converts the CreateButton macro to CB, which is the only form of this macro that WinHelp can use.

Example

The following macro creates a new button labeled "Ideas" that jumps to a topic located in the Ideas.hlp file whose topic ID is "dir":

```
CreateButton(btn_ideas, Ideas, JumpId(ideas.hlp, dir))
```

{button ,AL("MACRO_REF_BUTTON;macro_all;BrowseButtons;EnableButton;DisableButton;DestroyButton;ChangeButtonBinding;button;CONFIG")} Related Topics

DeleteItem macro

DeleteItem(item-ID**)**

Removes a menu item from a WinHelp menu.

Parameter	Description
item-ID	Specifies the item identifier used in the AppendItem, ExtInsertItem, or InsertItem macro.

Comment

WinHelp ignores this macro if it is run in a secondary window.

{button ,AL("MACRO_REF_MENU;macro_all;AppendItem;ExtInsertItem;InsertItem;ChangeItemBinding;DisableItem;EnableItem;ExtAble Item;ResetMenu;REF_STD_MENU")} Related Topics

DeleteMark macro

DeleteMark(marker-text**)**

Removes a text marker added by using the SaveMark macro.

Parameter	Description
marker-text	Specifies the text marker previously added by the SaveMark macro.

Comment

If the marker does not exist when the DeleteMark macro is run, WinHelp displays an error message.

Example

The following macro removes the marker “Managing Memory” from a Help file if the marker actually exists:

```
IfThen("Managing memory", DeleteMark(Managing Memory))
```

{button ,AL("MACRO_REF_TEXT_MARKER;macro_all;ifthen;SaveMark;GotoMark;IsMark;IsNotMark")} Related Topics

DestroyButton macro

DestroyButton(button-ID**)**

Removes a button added by using the CreateButton macro.

Parameter	Description
button-ID	Identifies a button previously created by the CreateButton macro.

Comment

The button identifier cannot specify a standard Help button.

{button ,AL("MACRO_REF_BUTTON;macro_all;CreateButton;ChangeButtonBinding;REF_STD_BTN;DisableButton;EnableButton")}
Related Topics

DisableButton macro

DisableButton(button-ID**)**

Disables (grays out) a button on the WinHelp button bar.

Parameter	Description
button-ID	Specifies the standard button identifier or the identifier assigned to the button by the CreateButton macro.

Comments

A button that is disabled by the DisableButton macro cannot be used in the topic until it is re-enabled by the EnableButton macro or re-activated by a user. For example, a disabled Contents, Index, or Help Topics button is re-enabled if a user clicks an interfile jump.

Help Workshop automatically converts the DisableButton macro to DB.

{button ,AL("MACRO_REF_BUTTON;macro_all;EnableButton;DestroyButton;REF_STD_BTN")} Related Topics

DisableItem macro

DisableItem(item-ID**)**

Disables (grays out) a menu item.

Parameter	Description
item-ID	Identifies a menu item that was previously appended by the AppendItem, ExtInsertItem, or InsertItem macro.

Comments

The menu item cannot be used in the topic until the EnableItem or ExtAbleItem macro is run.

WinHelp ignores this macro if it is run in a secondary window.

Help Workshop automatically converts the DisableItem macro to DI.

{button ,AL("MACRO_REF_MENU;macro_all;EnableItem;ExtAbleItem;AppendItem;InsertItem;ResetMenu;REF_STD_MENU")} Related Topics

EnableButton macro

EnableButton(button-ID**)**

Re-enables a button that was disabled by the DisableButton macro.

Parameter	Description
button-ID	Specifies the standard button identifier or the identifier assigned to the button by the CreateButton macro.

Comments

Help Workshop automatically converts the EnableButton macro to EB.

To enable a button and change its macro binding at the same time, use the ChangeEnable macro.

{button ,AL("MACRO_REF_BUTTON;macro_all;BrowseButtons;ChangeButtonBinding;ChangeEnable;DisableButton;CreateButton;DestroyButton;REF_STD_BTN")} Related Topics

EnableItem macro

EnableItem(item-ID**)**

Re-enables a menu item that was disabled by using the DisableItem or ExtAbleItem macro.

Parameter	Description
item-ID	Specifies the identifier assigned to the menu item by the AppendItem, ExtInsertItem, or InsertItem macro.

Comments

WinHelp ignores this macro if it is run in a secondary window.

Help Workshop automatically converts the EnableItem macro to EI.

{button ,AL("MACRO_REF_MENU;macro_all;AppendItem;DisableItem;ResetMenu")} Related Topics

EndMPrint macro

EndMPrint()

Dismisses the printing message box and terminates the printing of multiple topics.

Parameters

This macro does not take any parameters.

Comment

The EndMPrint macro is new for WinHelp version 4.0.

{button ,AL("MACRO_REF_BUTTON;macro_all;InitMPrint;MPrintID;MPrintHash")} Related Topics

ExecFile macro

ExecFile(program[**,** arguments[**,** display-state[**,** topic-ID]]]**)**

Runs a program or the program associated with a file.

Parameter	Description
program	Specifies the name of the program to be run or the name of a file. If a file is specified, the program associated with that file type is started.
arguments	Specifies the command-line arguments to send to the program.
display-state	Specifies a value indicating how the program's window is to be shown. If this parameter is not specified, SW_SHOW is used (activates the window and shows it in its default size and position). This optional parameter can be one of the following values.

Value	Meaning
SW_HIDE	Hides the window.
SW_MINIMIZE	Minimizes the window.
SW_RESTORE	Restores the window to its original size and position if the window is minimized or maximized.
SW_SHOW	Activates the window and displays it in its current size and position.
SW_SHOWMAXIMIZED	Activates the window and maximizes it.
SW_SHOWMINIMIZED	Activates the window and displays it minimized.
SW_SHOWMINNOACTIVE	Displays the window minimized, but WinHelp keeps the focus.
SW_SHOWNA	Displays a window in its current state, but WinHelp keeps the focus. (If the window was minimized before this call, it will stay minimized.)
SW_SHOWNOACTIVATE	Displays a window in its most recent size and position, but WinHelp keeps the focus.
SW_SHOWNORMAL	Activates and displays the window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_RESTORE).

topic-ID	Specifies the ID of the topic to display if the specified file or program cannot be started.

Comments

If a path is specified, WinHelp first searches for the file in the specified path. If the file is not found or no path was specified, WinHelp searches the same locations as it does when it searches for Help files:

- The folder of the current Help file.
- The current folder.
- The System subfolder in the Windows folder.
- The Windows folder.
- The folders listed in the PATH environment.
- The location specified in the Winhelp.ini file.
- The Help portion of the registry.

Help Workshop automatically converts the ExecFile macro to EF, which is the only form of this macro that WinHelp can use.

The ExecFile macro is new for WinHelp version 4.0.

Example

The following example opens the WIN.INI file in the program associated with .ini files:

```
ExecFile(win.ini)
```

{button ,KL("macros, auxiliary;macros, syntax;ControlPanel macro;ExecFile macro;ShellExecute")} Related Topics

ExecProgram macro

ExecProgram(command-line[**,** display-state]**)**

This macro is obsolete. Use the ExecFile macro instead.

{button ,AL("ShortCut;ExecFile")} Related Topics

Exit macro

Exit()

Quits the WinHelp program.

Parameters

This macro does not take any parameters.

{button ,AL("MACRO_REF_MENU;macro_all")} Related Topics

ExtAbleItem macro

ExtAbleItem(item-ID**,** display-state**)**

Enables or disables a menu item.

Parameter	Description
item-ID	Specifies the identifier assigned to the menu item by the AppendItem, ExtInsertItem, or InsertItem macro.
display-state	Specifies the state of the menu item. This parameter can be one of the following values:

Value	Meaning
GRAYED	Disables (grays out) the menu item.
CHECKED	Places a check mark next to the menu item.

Comments

You cannot use this macro to change a standard WinHelp menu item.

WinHelp ignores this macro if it is run in a secondary window.

The ExtAbleItem macro is new for WinHelp version 4.0.

{button ,AL("MACRO_REF_MENU;macro_all;AppendItem;ChangeItemBinding;DeleteItem;DisableItem;EnableItem;ExtInsertItem;ExtInsertMenu;InsertItem;ResetMenu;REF_STD_MENU")} Related Topics

ExtInsertItem macro

ExtInsertItem(menu-ID**,** item-ID**,** item-name**,** macro**,** position[**,** display-state]**)**

Inserts a menu item at a given position on an existing menu.

Parameter	Description
menu-ID	Identifies either a standard WinHelp menu or a menu previously created by the ExtInsertMenu or InsertMenu macro. The new item is inserted in this menu.
item-ID	Specifies the name that WinHelp uses internally to identify the menu item.
item-name	Specifies the name WinHelp displays on the menu for the item. This name is case-sensitive. An ampersand (&) before a character in the name identifies it as the item's accelerator key.
macro	Specifies the macro or macro string to run when a user clicks the menu item. Separate multiple macros in a string using colons or semicolons. Colons are required if this macro is specified in a project file.
position	Specifies the numeric position of the menu item in the menu. Position 0 is the first or topmost position in the menu.
display-state	Specifies the state of the menu item. This parameter may be one of the following values:

Value	Meaning
GRAYED	Disables (grays-out) the menu item.
CHECKED	Places a check mark next to the menu item.
SEPARATOR	Adds a separator bar to the menu.

Comments

The item-ID parameter can be used in a subsequent DisableItem, EnableItem, ExtAbleItem, or DeleteItem macro to remove or disable the item or to change the operations that the item performs in certain topics.

WinHelp ignores this macro if it is run in a secondary window.

The specified accelerator keys must be unique. If a key conflicts with other accelerator keys, WinHelp displays the error message "Unable to add item" and ignores the macro.

The ExtInsertItem macro is new for WinHelp version 4.0.

Example

The following macro inserts a menu item labeled “WordPad” on WinHelp’s File menu:

```
ExtInsertItem(mnu_file, itm_wpad, &WordPad, ExecFile(wordpad.exe), 3)
```

{button ,AL("MACRO_REF_MENU;macro_all;AppendItem;ChangeItemBinding;DeleteItem;DisableItem;EnableItem;ExtAbleItem;InsertItem;ResetMenu;REF_STD_MENU;CONFIG")} Related Topics

ExtInsertMenu macro

ExtInsertMenu(parent-ID**,** menu-ID**,** menu-name**,** menu-position[**,** display-state]**)**

Inserts a submenu in a previously defined menu.

Parameter	Description
parent-ID	Specifies the name that WinHelp uses to identify the parent menu to which this submenu is added. For a custom menu, this parameter is the name that was used when the menu was created by the ExtInsertMenu or the InsertMenu macro.
menu-ID	Specifies the name that WinHelp uses internally to identify the menu. This identifier can be used by the AppendItem, ExtInsertItem or ExtInsertMenu macros to access to the menu.
menu-name	Specifies the name that WinHelp displays for this menu. An ampersand (&) before a character in the name identifies it as the menu's accelerator.
menu-position	Specifies the integer position on the menu bar of the new menu name. Position 0 is the first position.
display-state	Specifies the state of the menu item. This parameter may be one of the following values:

Value	Meaning
CHECKED	Places a check mark next to the menu item.
GRAYED	Disables (grays out) the menu item.

Comment

WinHelp ignores this macro if it is run in a secondary window.

Example

The following macro adds a Wizard submenu to the standard Help menu:

```
ExtInsertMenu(mnu_help, mnu_wizard, &Wizard, 2)
```

{button ,AL("MACRO_REF_MENU;macro_all;ExtInsertItem;ExtAbleItem;AppendItem;InsertMenu;ResetMenu;REF_STD_MENU;CONFIG")} Related Topics

FileExist macro

FileExist(filename**)**

Checks to see whether the specified file or program exists.

Parameter	Description
filename	Specifies the name of the file.

Comments

This macro can be used in conjunction with macros such as IfThenElse, which use the result of a Boolean macro to determine what action to take.

If a path is specified, WinHelp first searches for the file in the specified path. If the file is not found, or no path was specified, WinHelp searches the same locations it does to find a Help file:

- The folder of the current Help file.
- The current folder.
- The System subfolder in the Windows folder.
- The Windows folder.
- The folders listed in the PATH environment.
- The location specified in the Winhelp.ini file.
- The Help portion of the registry.

Help Workshop automatically converts the FileExist macro to FE, which is the only form of this macro that WinHelp can use.

The FileExist macro is new for WinHelp version 4.0.

Example

The following macro checks to see if “Myapp.exe” has been installed. If the file is present, WinHelp runs it. If the file is not present, WinHelp displays a topic:

```
IfThenElse(FileExist(myapp.exe), ExecFile(myapp), JumpId(install_my_app))
```

{button ,AL(`macro_all;execfile;ifthenelse;ifthen')} Related Topics

FileOpen macro

FileOpen()

Displays the Open dialog box, which is available from the File menu in the main Help window.

Parameters

This macro does not take any parameters.

Comment

Help Workshop automatically converts the FileOpen macro to FO.

{button ,AL(`MACRO_REF_MENU;macro_all')} Related Topics

Find macro

Find()

Displays the Find tab in the Help Topics dialog box.

Parameters

This macro does not take any parameters.

Comment

The Find macro is new for WinHelp version 4.0.

{button ,AL("macro_all;contents;finder;Search")} Related Topics

Finder macro

Finder()

Displays the Help Topics dialog box.

Parameters

This macro does not take any parameters.

Comments

Help Workshop automatically converts the Finder macro to FD, which is the only form of this macro that WinHelp can use.

The Finder macro is new for WinHelp version 4.0.

{button ,AL("MACRO_REF_BUTTON;macro_all;contents;find;Search")} Related Topics

FloatingMenu macro

FloatingMenu()

Displays the context (floating) menu at the current mouse cursor position. This menu also appears when users click a topic using their right mouse button.

Parameters

This macro does not take any parameters.

Comments

You can add as many as 20 menu items to the context menu.

The FloatingMenu macro is new for WinHelp version 4.0.

{button ,AL("MACRO_REF_MENU;macro_all;ResetMenu;AppendItem;InsertItem;ExtInsertItem;menu")} Related Topics

Flush macro

Flush()

Causes WinHelp to process any pending messages, including previously called macros.

Parameters

This macro does not take any parameters.

Comments

Help Workshop automatically converts the Flush macro to FH, which is the only form of this macro that WinHelp can use.

The Flush macro is new for WinHelp version 4.0.

{button ,AL("macro_all")} Related Topics

FocusWindow macro

FocusWindow(window-name**)**

Changes the focus to the specified window.

Parameter	Description
window-name	Specifies the name of the window that should receive the focus. The name "main" is reserved for the main Help window. For secondary windows, the window name is defined in the [WINDOWS] section of the project file.

Comment

This macro is ignored if the specified window does not exist.

Example

The following macro changes the focus to the secondary window “keys”:

```
FocusWindow(keys)
```

{button ,AL("MACRO_REF_WINDOW;macro_all;CloseSecondarys;CloseWindow;PositionWindow;UpdateWindow;WINDOWS")}
Related Topics

Generate macro

Generate(message**,** wParam**,** lParam**)**

Posts a message to the currently active Help window.

Parameter	Description
message	Specifies a message to send to the currently active Help window.
wParam	Specifies the first argument of the message.
lParam	Specifies the second argument of the message.

Comment

The Generate macro is new for WinHelp version 4.0.

{button ,AL("macro_all;helpontop")} Related Topics

GotoMark macro

GotoMark(marker-text**)**

Jumps to a marker set by the SaveMark macro.

Parameter	Description
marker-text	Specifies a text marker previously defined by the SaveMark macro.

Comment

WinHelp displays an error message if the specified text marker does not exist.

Example

The following macro jumps to the marker “Managing Memory” if it has been previously set.

```
IfThen("Managing Memory", GotoMark(Managing Memory))
```

{button ,AL("MACRO_REF_TEXT_MARKER;macro_all;DeleteMark;IsMark;IsNotMark;SaveMark")} Related Topics

HelpOn macro

HelpOn()

Displays the Help file for the WinHelp program. The macro carries out the same action as pressing F1 in WinHelp.

Parameters

This macro does not take any parameters.

{button ,AL("MACRO_REF_MENU;macro_all;JumpHelpOn ")} Related Topics

HelpOnTop macro

HelpOnTop()

Toggles the authored on-top state of a Help window.

Parameters

This macro does not take any parameters.

Comments

A Help window can have three states:

Default	Authored as part of the window definition in the project file
On Top	Forced on
Not On Top	Forced off

The latter two states, set by the user, affect all Help windows. If the user sets either the On Top or Not On Top options (from the Options menu, the Menu button, or a context menu), the HelpOnTop macro has no effect.

You can use the Generate macro to send a message to WinHelp as if the user had selected a menu command. To specify the Default option, use a value of 1470 as the message parameter in the Generate macro. To set the On Top or Not On Top options, use 1471 or 1472, respectively.

In WinHelp version 4.0, the HelpOnTop macro affects only the current window.

{button ,AL("MACRO_REF_WINDOW;macro_all;WINDOWS")} Related Topics

History macro

History()

Displays the history list, which shows the last 31 topics a user viewed in the main Help window since opening the file in WinHelp.

Parameters

This macro does not take any parameters.

Comments

The number of topics kept in the history list can be changed by using the BACKTRACK switch in the Win.ini file.

WinHelp ignores this macro if it is run in a secondary window.

{button ,AL("MACRO_REF_BUTTON;macro_all;Back;BACKTRACK")} Related Topics

IfThen macro

IfThen("marker"/macro**,** macro**)**

Runs a macro if a given marker exists or a macro returns a non-zero value.

Parameter	Description
marker/macro	Specifies either a text marker previously created by using the SaveMark macro, or a macro that returns a Boolean value. If a marker is used, it must be enclosed in quotation marks. If a macro is used, it must **not** be enclosed in quotation marks (so Help Workshop can distinguish between a marker and a macro).
macro	Specifies one or more macros to run if the marker exists. Separate multiple macros using colons or semicolons.

Comment

Help Workshop automatically converts the IfThen macro to IF.

Example

The following macro jumps to the topic whose topic ID is “man_mem” if a marker named "Managing Memory" has been set by the SaveMark macro:

```
IfThen("Managing Memory", JumpId(man_mem))
```

{button ,AL("MACRO_REF_TEXT_MARKER;macro_all;DeleteMark;GotoMark;IfThenElse;IsMark;IsNotMark;Not;SaveMark")} Related Topics

IfThenElse macro

IfThenElse("marker"/macro**,** macro1**,** macro2**)**

Runs one of two macros depending on whether or not a marker exists.

Parameter	Description
marker/macro	Specifies either a text marker previously created by the SaveMark macro or a macro that returns a Boolean value. If a marker is used, it must be enclosed in quotation marks. If a macro is used, it must **not** be enclosed in quotation marks (so Help Workshop can distinguish between a marker and a macro).
macro1	Specifies one or more macros to run if the marker exists. Separate multiple macros using colons or semicolons.
macro2	Specifies one or more macros to run if the marker does not exist. Separate multiple macros using colons or semicolons.

Comment

Help Workshop automatically converts the IfThenElse macro to IE.

Example

The following macro jumps to the topic whose ID is “mem” if a marker named “Memory” has been set by the SaveMark macro. If the marker does not exist, the macro jumps to the next topic in the browse sequence.

```
IfThenElse("Memory", JumpId(mem), Next())
```

{button ,AL("MACRO_REF_TEXT_MARKER;macro_all;DeleteMark;GotoMark;IfThen;IsMark;IsNotMark;Not;SaveMark")} Related Topics

InitMPrint macro

InitMPrint()

Initializes WinHelp in preparation for printing multiple topics.

Parameters

This macro does not take any parameters.

Comments

This macro displays the Printer Setup dialog box. If a user clicks OK, the InitMPrint macro returns TRUE. Otherwise, it returns FALSE. This macro should be used as the first parameter in an IfThen macro to ensure that printing multiple topics occurs only if a user clicks OK.

The InitMPrint macro is new for WinHelp version 4.0.

Example

The following example prints the two topics whose IDs are “topic_a” and “topic_b”:

```
IfThen(InitMPrint(), MPrintId(topic_a): MPrintId(topic_b): EndMPrint())
```

{button ,AL("MACRO_REF_BUTTON;macro_all;EndMPrint;MPrintID;MPrintHash;Print")} Related Topics

InsertItem macro

InsertItem(menu-ID**,** item-ID**,** item-name**,** macro**,** position**)**

Inserts a menu item at a given position on an existing menu. The menu can either be one you create by using the InsertMenu macro or a standard WinHelp menu.

Parameter	Description
menu-ID	Identifies either a standard WinHelp menu or a menu previously created by the ExtInsertMenu or InsertMenu macro.
item-ID	Specifies the name that WinHelp uses internally to identify the menu item.
item-name	Specifies the name that WinHelp displays on the menu for the item. This name is case-sensitive. An ampersand (&) before a character in the name identifies it as the item's accelerator.
macro	Specifies a macro or macro string to run when a user clicks the menu item. The macro must be enclosed in quotation marks. If multiple macros are specified in the project file, separate the macros by using semicolons or colons.
position	Specifies the integer position of the menu item in the menu. Position 0 is the first or topmost position on the menu.

Comments

The item ID parameter can be used in a subsequent EnableItem, ExtAbleItem, DisableItem or DeleteItem macro to remove or disable the item or to change the operations that the item performs in certain topics.

WinHelp ignores this macro if it is run in a secondary window.

The specified accelerator keys must be unique. If a key conflicts with other accelerator keys, WinHelp displays the error message "Unable to add item" and ignores the macro.

Example

The following macro inserts a menu item labeled “Tools” as the third item on a menu whose identifier is “MNU_BKS”. Selecting the menu item causes a jump to a topic located in the Tls.hlp file whose topic ID is "tls1":

```
InsertItem(mnu_bks, m_tls, &Tools, JI(`tls.hlp', `tls1'), 3)
```

{button ,AL("MACRO_REF_MENU;macro_all;AppendItem;ChangeItemBinding;DeleteItem;DisableItem;EnableItem;ExtAbleItem;ExtInsertItem;InsertMenu;ResetMenu;REF_STD_MENU;CONFIG")} Related Topics

InsertMenu macro

InsertMenu(menu-ID**,** menu-name**,** menu-position**)**

Inserts a new menu in the WinHelp menu bar.

Parameter	Description
menu-ID	Specifies the name that WinHelp uses internally to identify the menu. This identifier can be used by the AppendItem, ExtInsertItem, and ExtInsertMenu macros to add macros to the menu.
menu-name	Specifies the name that WinHelp displays on the menu bar. An ampersand (&) before a character in the name identifies it as the menu's accelerator.
menu-position	Specifies the position on the menu bar of the new menu name. This parameter must be an integer. Positions are numbered from left to right, with position 0 being the left-most menu.

Comments

The specified accelerator keys must be unique. If a key conflicts with other accelerator keys, WinHelp displays the error message "Unable to add menu" and ignores the macro.

WinHelp ignores this macro if it is run in a secondary window.

Example

The following macro adds a menu named “Utilities” to the WinHelp program. The label “Utilities” appears as the fourth item on the WinHelp menu bar. The user can press ALT+U to open the menu.

```
InsertMenu(IDM_UTIL, &Utilities, 3)
```

{button ,AL("MACRO_REF_MENU;macro_all;AppendItem;ExtInsertItem;ExtInsertMenu;ResetMenu")} Related Topics

IsBook macro

IsBook()

Determines whether WinHelp is running as a standalone system (a double-clicked book icon), or if it is being run from a program. This macro can be used as the first parameter of a IfThen or IfThenElse macro to take specific action depending on whether the current Help file is being run as a double-clicked book icon.

Parameters

This macro does not take any parameters.

Comments

The IsBook macro returns a value of TRUE if WinHelp is being run as a book; otherwise, the macro returns FALSE.

The IsBook macro is new for WinHelp version 4.0.

{button ,AL("macro_all;IfThen;IfThenElse ")} Related Topics

IsMark macro

IsMark(marker-text**)**

Tests whether or not a marker that was set by the SaveMark macro exists.

Parameter	Description
marker-text	Specifies a text marker that was previously created by the SaveMark macro.

Comments

The IsMark macro is used as a parameter in the conditional macros IfThen and IfThenElse. The IsMark macro returns TRUE if the mark exists or FALSE if it does not.

To reverse the result of the IsMark macro, use the IsNotMark macro.

{button ,AL("MACRO_REF_TEXT_MARKER;macro_all;IfThen;IfThenElse;IsNotMark;SaveMark")} Related Topics

IsNotMark macro

IsNotMark("marker-text"**)**

Tests whether or not a marker that was set by the SaveMark macro exists.

Parameter	Description
marker-text	Specifies a text marker that was previously created by the SaveMark macro.

Comments

The IsNotMark macro is used as a parameter in the conditional macros IfThen or IfThenElse. The IsNotMark macro returns FALSE if the mark exists or TRUE if it does not. This result is the exact opposite of the implicit IsMark macro that Help Workshop usually uses for the IfThen and IfThenElse macros.

Help Workshop automatically converts the IsNotMacro macro to NM, which is the only form of this macro that WinHelp can use.

The IsNotMark macro is new for WinHelp version 4.0.

{button ,AL("MACRO_REF_TEXT_MARKER;macro_all;IfThen;IfThenElse;IsMark;SaveMark")} Related Topics

JumpContents macro

JumpContents(filename**)**

Jumps to the default (contents) topic of the specified Help file.

Parameter	Description
filename	Specifies the name of the destination file for the jump. If WinHelp cannot find this file, it displays an error message and does not perform the jump.

Example

The following macro jumps to the contents topic of the Progman.hlp file:

```
JumpContents(progman.hlp)
```

{button ,AL("MACRO_REF_LINK;macro_all;Contents;SetContents;HelpOn;CONTENTS_hpj")} Related Topics

JumpContext macro

JumpContext([filename>window-name**,**] context-number**)**

Jumps to a topic identified by a context number.

Parameter	Description
filename	Specifies the name of the destination file for the jump. This parameter is optional.
window-name	Specifies the window type in which to display the topic.
context-number	Specifies the context number of the topic in the destination file. The context number must be defined in the [MAP] section of the project file.

Comments

The context number is identified by an entry in the [MAP] section of the project file.

Help Workshop automatically converts the JumpContext macro to JC, which is the only form of this macro that WinHelp can use.

Example

The following macro jumps to the topic mapped to the context number 801 in the Progman.hlp file:

```
JumpContext(progman.hlp, 801)
```

{button ,AL("MACRO_REF_LINK;macro_all;JumpHash;JumpId;JumpKeyword;PopupContext;PopupHash;PopupId;MAP")} Related Topics

JumpHash macro

JumpHash([filename>window-name**,**] hash-code**)**

Jumps to a topic identified by a hash number.

Parameter	Description
filename	Specifies the name of the Help file containing the hash number. This optional parameter is used when jumping to a topic that is not in the current Help file.
window-name	Specifies the window type in which to display the topic.
hash-code	Specifies the hash number of the topic in the destination file.

Comments

You can use the Report command in Help Workshop to list the hash numbers of all the topics in a Help file. This enables you to display information about a Help file for which you have no topic, project, or contents files.

Help Workshop automatically converts the JumpHash macro to JH.

The JumpHash macro is new for WinHelp version 4.0.

Example

The following macro displays this topic:

```
JumpHash(hcw, 4104506004)
```

{button ,AL("MACRO_REF_LINK;macro_all;JumpId;PopupHash;PopupId;REPORT")} Related Topics

JumpHelpOn macro

JumpHelpOn()

Displays the Help Topics dialog box for the Winhlp32.hlp Help file.

Parameters

This macro does not take any parameters.

Comment

In WinHelp 4.0, the JumpHelpOn macro and the HelpOn macro function identically.

{button ,AL("macro_all;HelpOn ")} Related Topics

JumpId macro

JumpId([filename>window-name**,**] topic-ID**)**

Jumps to the topic whose topic ID is specified.

Parameter	Description
filename	Specifies the name of the Help file that contains the topic ID. This optional parameter is used when jumping to a topic that is not in the current Help file.
window-name	Specifies the window type in which to display the topic.
topic-ID	Specifies the ID of the topic you want to jump to.

Comments

When the window-name parameter is used, it must be preceded by a greater-than sign (>).

Help Workshop automatically converts the JumpId macro to JI.

Example

The following macro jumps to a topic with “another_topic” as its topic ID in the current Help file:

```
JumpId(another_topic)
```

{button ,AL("MACRO_REF_LINK;macro_all;JumpHash;JumpKeyword;JumpContext;PopupContext;PopupHash;PopupId")} Related Topics

JumpKeyword macro

JumpKeyword([filename**,**] keyword**)**

Searches for keywords specified by K-footnotes. If WinHelp finds no matches, it displays the Help Topics dialog box with the Index tab on top. If WinHelp finds one match, it displays the topic. If WinHelp finds more than one match, it displays the Topics Found dialog box and lists the topic titles.

Parameter	Description
filename	Specifies the name of the Help file that contains the keyword. Use this optional parameter when you jump to a topic that is not in the current Help file.
keyword	Specifies one or more keywords to search for. Separate multiple keywords using a semicolon. If any keyword contains a comma, enclose the entire keyword string in quotation marks.

Comments

This macro is functionally equivalent to the KLink macro.

Help Workshop automatically converts the JumpKeyword macro to JK.

{button ,AL("MACRO_REF_LINK;macro_all;KLink")} Related Topics

KLink macro

KLink("keyword[**;** keyword]"[**,** type[**,** "topic-ID"[**,** window-name]]]**)**

Searches for keywords specified by K-footnotes.

Parameter	Description
keyword	Specifies one or more keywords to search for. Separate multiple keywords using a semicolon. If any keyword contains a comma, enclose the entire keyword string in quotation marks.
type	Specifies the action to perform if one or more keywords is found. If this parameter is not specified or is zero, the default action is to display the Topics Found dialog box containing the topic title. This parameter can be one or more of the following values, separated by spaces.

Value	Meaning
JUMP (1)	Specifies that if only one topic matches any of the keywords, WinHelp should jump directly to that topic.
TITLE (2)	Specifies that if a keyword is found in more than one Help file, WinHelp should display the title of the Help file (as specified in the contents [.cnt] file) beside the topic title in the Topics Found dialog box.
TEST (4)	Specifies that the macro should return a value indicating whether or not there is at least one match. The TestKLink macro is converted by Help Workshop into an KLink macro with this parameter.

topic-ID	Specifies the topic to display in a pop-up window if no matches are found. If this parameter is not specified, WinHelp displays a message box with the text “No additional information is available”. To specify a topic in a different Help file, the topic ID should end with an ‘@’ character and the name of the Help file.
window-name	Specifies the window type in which to display the topic. If this parameter is not specified, the window type that is specified for a topic (if one is defined) is used, or the default or current window is used. If this macro results in an interfile jump, the window type must be defined in the project file for the Help file that is being jumped to.

Comments

The KLink macro searches for K-keyword matches in the current Help file. If the Help file is associated with a contents file, WinHelp searches all Help files specified in the contents file (by the :Index and :Link commands) for matching K-keywords.

The KLink macro is identical to the ALink macro except that it searches for K-keywords instead of A-keywords.

Help Workshop automatically converts the KLink macro to KL, which is the only form of this macro that WinHelp can use.

The KLink macro is new for WinHelp version 4.0.

Examples

The following example generates a list of topics that contain the “network printing” or “local printing” keywords in K-footnotes:

```
KLink(network printing; local printing)
```

The following example generates a list of topics that contain the keyword “how_to” in K-footnotes and displays those topics in the Steps window:

```
KLink(how_to, , , Steps)
```

{button ,AL("MACRO_REF_LINK;macro_all;ALink;WINDOWS")} Related Topics

Menu macro

Menu()

Displays the context menu for the current Help window.

Parameters

This macro does not take any parameters.

Comments

Unlike the FloatingMenu macro, the Menu macro is carried out the moment it is encountered. If the Menu macro is used in a macro string, it may be carried out before any macros that precede it.

If the current window has a Menu button, the menu will be displayed relative to that button. Otherwise, it will be displayed relative to the mouse position.

Help Workshop automatically converts the Menu macro to MU, which is the only form of this macro that WinHelp can use.

The Menu macro is new for WinHelp version 4.0.

{button ,AL("MACRO_REF_BUTTON;macro_all")} Related Topics

MPrintHash macro

MPrintHash(hash-code**)**

Prints a topic identified by a hash number. This macro must be used in conjunction with the InitMPrint and EndMPrint macros.

Parameter	Description
hash-code	Specifies the hash number of the topic to be printed.

Comments

The MPrintHash macro is new for WinHelp version 4.0.

You can use the Report command in Help Workshop to list the hash numbers of all the topics in a Help file. This enables you to display information about a Help file for which you have no topic, project, or contents files.

{button ,AL("macro_all;MPrintID;Print;EndMPrint;InitMPrint")} Related Topics

MPrintID macro

MPrintID(topic-ID**)**

Prints a topic. This macro must be used in conjunction with the InitMPrint and EndMPrint macros.

Parameter	Description
topic-ID	Specifies the ID of the topic to print.

Comment

The MPrintID macro is new for WinHelp version 4.0.

{button ,AL("macro_all;MPrintHash;Print;EndMPrint;InitMPrint")} Related Topics

Next macro

Next()

Displays the next topic in the current browse sequence for the Help file.

Parameters

This macro does not take any parameters.

Comment

If the currently displayed topic is the last topic in a browse sequence, this macro does nothing.

{button ,AL("MACRO_REF_BUTTON;macro_all;BrowseButtons;Prev")} Related Topics

NoShow macro

NoShow()

Prevents a Help window from being displayed if it has not already been displayed.

Parameters

This macro does not take any parameters.

Comment

The NoShow macro is new for WinHelp version 4.0.

{button ,AL("MACRO_REF_WINDOW;macro_all")} Related Topics

Not macro

Not("marker"/macro**)**

Returns zero (FALSE) if the marker text specified by the SaveMark macro exists or non-zero (TRUE) if the marker text does not exist. When used with a macro, the Not macro reverses the results of the macro.

Parameter	Description
marker/macro	Specifies either a text marker that was previously created by the SaveMark macro or a macro that returns a Boolean value. If a marker is used, it must be enclosed in quotation marks. If a macro is used, it must **not** be enclosed in quotation marks (so Help Workshop can distinguish between a marker and a macro).

Comment

The Not macro is used with the macros IfThen and IfThenElse.

{button ,AL("MACRO_REF_TEXT_MARKER;macro_all;IsNotMark;ifthen;ifthenelse ")} Related Topics

PopupContext macro

PopupContext([filename**,**] context-number**)**

Displays in a pop-up window the topic identified by a context number.

Parameter	Description
filename	Specifies the name of the destination Help file for the jump. Use this optional parameter if the pop-up is not in the current Help file.
context-number	Specifies the context number of the topic to be displayed. The context number must be specified in the [MAP] section of the project file.

Comment

Help Workshop automatically converts the PopupContext macro to PC.

Example

The following macro displays in a pop-up window the topic mapped to the context number 801 in the Progman.hlp file:

```
PopupContext(801)
```

{button ,AL("MACRO_REF_LINK;macro_all;JumpContext;JumpHash;JumpId;JumpKeyword;PopupHash;PopupId;MAP;CONTENTS_hpj")} Related Topics

PopupHash macro

PopupHash([filename**,**] hash-code**)**

Displays in a pop-up window the topic identified by a hash number.

Parameter	Description
filename	Specifies the name of the destination Help file for the jump. Use this optional parameter if the pop-up is not in the current Help file.
hash-code	Specifies the hash number of the topic to be displayed.

Comments

You can use the Report command in Help Workshop to list the hash numbers of all the topics in a Help file. This enables you to display information about a Help file for which you have no topic, project, or contents files.

The PopupHash macro is new for WinHelp version 4.0.

Example

The following macro displays this topic in a pop-up window:

```
PopupHash(hcw, 803830446)
```

{button ,AL("MACRO_REF_LINK;macro_all;JumpHash;PopupContext;PopupId;REPORT")} Related Topics

PopupId macro

PopupId([filename**,**] topic-ID**)**

Displays a topic in a pop-up window.

Parameter	Description
filename	Specifies the name of the Help file containing the topic ID. Use this optional parameter if the topic is not in the current Help file.
topic-ID	Specifies the ID of the topic to display.

Comment

Help Workshop automatically converts the PopupId macro to PI.

Example

The following macro displays a topic whose ID is "another_topic" in a pop-up window:

```
PopupId(another_topic)
```

{button ,AL("MACRO_REF_LINK;macro_all;JumpContext;JumpHash;JumpId;JumpKeyword;PopupContext;PopupHash")} Related Topics

PositionWindow macro

PositionWindow(x**,** y**,** width**,** height**,** display-state**,** window-name**)**

Sets the size and position of a Help window.

Parameter	Description
x	Specifies the x-coordinate, in Help units, of the upper left corner of the window. WinHelp assumes the screen is 1024 Help units wide (regardless of resolution). For example, if the x-coordinate is 512, the left edge of the Help window is in the middle of the screen.
y	Specifies the y-coordinate, in Help units, of the upper left corner of the window. WinHelp assumes the screen is 1024 Help units high (regardless of resolution). For example, if the y-coordinate is 512, the top edge of the Help window is in the middle of the screen.
width	Specifies the default width, in Help units, of the window.
height	Specifies the default height, in Help units, of the window.
display-state	Specifies how the window is sized. This parameter can be one of the following values:

Value	Meaning
SW_HIDE	Hides the window.
SW_MINIMIZE	Minimizes the window.
SW_RESTORE	Restores the window to its original size and position if the window is minimized or maximized.
SW_SHOW	Activates the window and displays it in its current size and position.
SW_SHOWMAXIMIZED	Activates the window and maximizes it.
SW_SHOWMINIMIZED	Activates the window and displays it minimized.
SW_SHOWMINNOACTIVE	Displays the window minimized, but WinHelp keeps the focus.
SW_SHOWNA	Displays a window in its current state, but WinHelp keeps the focus. (If the window was minimized before this call, it will stay minimized.)
SW_SHOWNOACTIVATE	Displays a window in its most recent size and position, but WinHelp keeps the focus.
SW_SHOWNORMAL	Activates and displays the window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_RESTORE).

window-name	Specifies the name of the window to position. The name "main" is reserved for the main Help window. For secondary windows the window name must be defined in the [WINDOWS] section of the project file.

Comments

If the window to be positioned does not exist, WinHelp ignores the macro.

Help Workshop automatically converts the PositionWindow macro to PW.

Example

The following macro positions the secondary window "Samples" in the upper left corner (100, 100) with a width and height of 500 (in Help units):

```
PositionWindow(100, 100, 500, 500, NORMAL, "Samples")
```

{button ,AL("MACRO_REF_WINDOW;macro_all;WINDOWS",0,`main')} Related Topics

Prev macro

Prev()

Displays the previous topic in the browse sequence of the Help file. If the currently displayed topic is the first in a browse sequence, this macro does nothing.

Parameters

This macro does not take any parameters.

{button ,AL("macro_all;BrowseButtons;Next")} Related Topics

Print macro

Print()

Sends the currently displayed topic to the printer.

Parameters

This macro does not take any parameters.

{button ,AL("macro_all;MPrintID;EndMPrint;InitMPrint")} Related Topics

PrinterSetup macro

PrinterSetup()

This macro is obsolete. WinHelp always displays the Print Setup dialog box before printing.

Parameters

This macro does not take any parameters.

RegisterRoutine macro

RegisterRoutine(DLL-name**,** function-name**,** format-spec**)**

Registers a function within a dynamic-link library. Once registered, the function can be used the same as any WinHelp macro.

Parameter	Description
DLL-name	Specifies the filename of the DLL. If WinHelp cannot find the library, it displays an error message.
function-name	Specifies the name of the function to run in the designated DLL.
format-spec	Specifies a string indicating the formats of parameters passed to the function. The format string must be enclosed in quotation marks. Characters in the string represent C parameter types:

Character	Description
U or u	unsigned number
I or i	signed number
S or s	string
v	type unknown
=	The character previous to this one is the return type of the parameter.

Comments

It is recommended that you include the .dll filename extension when specifying the name of the DLL. Without the extension, WinHelp first looks through all its various folders trying to find the file exactly as specified. Only after this fails does it add the .dll extension and look through all the search paths again.

Functions in 32-bit DLLs must use the _stdcall calling convention. Functions in 16-bit DLLs (which are not available when running Windows NT) must use the _pascal calling convention.

Help Workshop automatically converts the RegisterRoutine macro to RR.

Example

The following macro registers the Windows MessageBox function:

```
[CONFIG]
RegisterRoutine("user32.dll", "MessageBox", "USSU")
```

Having registered this function, you could display a message box with a call similar to the following:

```
MessageBox(hwndContext, “This is my message”, “Help Message”, 0)
```

{button ,KL("macros, syntax;MessageBox")} Related Topics

RemoveAccelerator macro

RemoveAccelerator(key**,** shift-state**)**

Removes the assignment of a macro to an accelerator key or key combination.

Parameter	Description
key	Specifies the Windows virtual-key value. This can be a numeric digit, a quoted character (as in `A'), or one of the strings listed in the AddAccelerator macro.
shift-state	Specifies the combination of ALT, SHIFT, and CTRL keys to be used with the accelerator. This parameter may be one of the following values: NONE (0) SHIFT (1) CTRL (2) SHIFT+CTRL (3) ALT (4) ALT+SHIFT (5) ALT+CTRL (6) SHIFT+ALT+CTRL (7)

Comments

An error does not occur if this macro is used with an accelerator for which no macro was defined.

Help Workshop automatically converts the RemoveAccelerator macro to RA.

Example

The following macro disassociates a macro from the ALT+SHIFT+CTRL+F9 key combination:

```
RemoveAccelerator(VK_F9, SHIFT+ALT+CTRL)
```

{button ,AL("MACRO_REF_KEYBOARD;macro_all;AddAccelerator")} Related Topics

ResetMenu macro

ResetMenu()

Deletes all added menus and menu items, restores and enables all standard menu items, and restores the item bindings of all standard menu items to their defaults.

Parameters

This macro does not take any parameters.

{button ,AL("MACRO_REF_MENU;macro_all ")} Related Topics

SaveMark macro

SaveMark(marker-text**)**

Saves the location of the currently displayed topic and the window it is displayed in and associates a text marker with that information. You can then jump to the topic in that window by using the GoToMark macro.

Parameter	Description
marker-text	Specifies the text marker that identifies the topic location. If the same text is used for more than one marker, the most recently entered marker is used.

Comments

If the marker-text was already used, the SaveMark macro replaces the information associated with the previously saved marker-text.

When the user quits WinHelp, all text markers are deleted.

The SaveMark macro is carried out as soon as it is encountered. If you use it in a macro string, it may be carried out before any macros that precede it.

{button ,AL("MACRO_REF_TEXT_MARKER;macro_all;IfThen;IfThenElse;DeleteMark;GotoMark;IsMark;IsNotMark;Not")} Related Topics

Search macro

Search()

Displays the Help Topics dialog box with the Index tab on top.

Parameters

This macro does not take any parameters.

{button ,AL("macro_all;JumpKeyword;Finder")} Related Topics

SetContents macro

SetContents(filename**,** context-number**)**

Designates a specific topic as the contents (default) topic in the specified Help file.

Parameter	Description
filename	Specifies the name of the Help file that contains the contents (default) topic. If WinHelp cannot find this file, it displays an error message and does not carry out the jump.
context number	Specifies the context number of the topic in the specified file. The context number must be defined in the [MAP] section of the project file. If the context number is not valid, WinHelp displays an error message.

Example

The following example sets the topic mapped to the context number 801 in the Progman.hlp file as the default topic. After running this macro, any unsuccessful jump causes the topic specified by the context-number parameter to appear:

```
SetContents(PROGMAN.HLP, 801)
```

{button ,AL("MACRO_REF_BUTTON;macro_all;Contents;JumpContents;CONTENTS_hpj;MAP")} Related Topics

SetHelpOnFile macro

SetHelpOnFile(filename**)**

This macro is obsolete. WinHelp version 4.0 will ignore this macro.

SetPopupColor macro

SetPopupColor(r, g**,** b**)**

Sets the background color for all subsequent pop-up windows.

Parameter	Description
r	Specifies the red component of the color. This value is an integer in the range 0 to 255.
g	Specifies the green component of the color. This value is an integer in the range 0 to 255.
b	Specifies the blue component of the color. This value is an integer in the range 0 to 255.

Comments

After this macro is run, the set color applies to all pop-up topics.

Help Workshop automatically converts the SetPopupColor macro to SPC, which is the only form of this macro that WinHelp can use.

The SetPopupColor macro is new for WinHelp version 4.0.

Example

The following macro sets the background color for pop-up windows to blue:

```
SetPopupColor(0, 0, 255)
```

{button ,AL("MACRO_REF_WINDOW;macro_all")} Related Topics

ShellExecute macro

ShellExecute(filename**,** [options[**,** show-flag[**,** operation[**,** path[**,** topic-id]]]]**)**

Opens or prints the specified file.

Parameter	Description
filename	Specifies the name of the file to open.
options	Specifies parameters passed to the program when the filename parameter specifies an executable (.exe) file. If the filename parameter specifies a document file, this parameter is empty.
show-flag	Specifies how the program is shown when it is opened. If the filename parameter specifies a document file, this parameter should be zero.

Value	Meaning
HIDE (0)	Hides the window.
MAXIMIZE (3)	Activates the window and maximizes it.
MINIMIZE_ACTIVATE (6)	Activates the window and minimizes it.
MINIMIZE_NOACTIVE (7)	Minimizes the window, but WinHelp keeps the focus.
NOACTIVATE (4)	Displays a window in its current state, but WinHelp keeps the focus. (If the window was minimized before this call, it stays minimized.)
NORMAL (5)	Activates the window and displays it in its current size and position.
RESTORE (9)	Restores the window to its original size and position if the window is minimized or maximized.

operation	Specifies the operation to perform. This parameter can be "open" or "opencpl" or "print". If this parameter is not specified, "open" is the default value.
path	Specifies the default folder. Use “” if you do not need to specify a default folder.
topic-id	Specifies the ID of the topic to display if this macro fails.

Comments

The filename can be a document file or an executable file. If it is a document file, this function opens or prints it, depending on the value of the operation parameter. If it is an executable file, this function opens it, even if the operation parameter specifies print.

If no path is specified with the filename, WinHelp searches the following locations in the order listed:

- The folder in which the Winhelp.exe program is located
- The folder that contains the current Help file
- The System subfolder in the Windows folder
- The Windows folder
- The PATH environment
- The registry

The ShellExecute macro calls the Windows ShellExecute function.

Help Workshop automatically converts the ShellExecute macro to SE, which is the only form of this macro that WinHelp can use.

The ShellExecute macro is new for WinHelp version 4.0.

{button ,KL("ExecFile macro;ControlPanel macro;macros, syntax;ShellExecute")} Related Topics

ShortCut macro

ShortCut(window-class**,** program[**,** wParam[**,** lParam[**,** topic-ID]]]**)**

Runs the specified program if it is not already running. If the specified program is running, WinHelp activates it. If the wParam parameter is specified, a WM_COMMAND message with the specified wParam and lParam values are sent to the program.

Parameter	Description
window-class	Specifies the class name of a top level window of the program. WinHelp uses this class name to determine if the program is already running.
program	Specifies the executable name of the program. This is the name of the program that runs if the window class name cannot be found. The .exe extension does not need to be included.
wParam	Specifies the first argument to the WM_COMMAND message that is sent to the program. If this value is not specified, the program is started, but no message is sent and the program is not activated.
lParam	Specifies the second argument to the WM_COMMAND message.
topic-ID	Specifies the topic ID of the topic to jump to if the program cannot be found. If this parameter is not specified, WinHelp displays a message box indicating that the program could not be found. If it is specified and no filename is included, the filename specified by the :Base command in the contents (.cnt) file is used. If there is no associated contents file, the current Help file is used. To specify a specific filename, the topic ID should end with an '@' character and the name of the Help file.

Comments

If you need to include a topic ID, but do not want to send the program any messages, you can either use -1 for the wParam argument and 0 for the lParam argument, or you can simply include the commas without specifying the values for the arguments as follows:

```
ShortCut(class_name, myapp, , , topic-ID)
```

Some programs, such as the Microsoft Visual C++ compiler (MSVC), cannot receive a message in the same call that runs them. In this case, you must first call the ShortCut macro without specifying wParam or lParam. You can then call this macro again and send the wParam and lParam messages. Note that the second time you call this macro, you can use an empty string ("") for the program name.

Help Workshop automatically converts the ShortCut macro to SH, which is the only form of this macro that WinHelp can use.

The ShortCut macro is new for WinHelp version 4.0.

Example

The following example copies the current topic to the Clipboard, runs or activates Notepad, and pastes the text from the Clipboard into Notepad:

```
CopyTopic():ShortCut(notepad, notepad, 0x0302)
```

{button ,AL("macro_all;MACRO_REF_PROGRAMS;ControlPanel;ExecFile ")} Related Topics

TCard macro

TCard(command**)**

Sends a message to the program that is invoking WinHelp as a training card.

Parameter	Description
command	Specifies one of the following commands, a numeric value, or a topic ID. If one of the following values is specified, its numeric equivalent is sent as the wParam value of the WM_TCARD message:

Value	Meaning
IDABORT (3)	The user clicked an authorable Abort button.
IDCANCEL (2)	The user clicked an authorable Cancel button.
IDCLOSE (8)	The user closed the training card.
IDHELP (9)	The user clicked an authorable Help button.
IDIGNORE (5)	The user clicked an authorable Ignore button.
IDOK (1)	The user clicked an authorable OK button.
IDNO (7)	The user clicked an authorable No button.
IDRETRY (4)	The user clicked an authorable Retry button.
IDYES (6)	The user clicked an authorable Yes button.

If a numeric value is used, the training card program is sent HELP_TCARD_DATA (16) for the wParam parameter, and the numeric value is passed as the lParam value of the WM_TCARD message.

If a topic ID is used, the ID is mapped to the numeric value found in the project file and it is passed to the program.

Comment

The TCard macro is new for WinHelp version 4.0.

{button ,AL("macro_all")} Related Topics

Test macro

Test(test-num**)**

Runs an internal WinHelp test.

Parameter	Description
test-num	Specifies the testing option. This parameter can be one of the following values:

Value	Description
1	Displays all the topics in the Help file, starting with the current topic.
2	Displays all the topics in the Help file, starting with the first topic.
3	Continuously displays all the topics in the Help file, starting with the first topic.
4	Displays all the topics in the Help file, starting with the first topic, and then quits.
5	Jumps to all topics specified in the contents file excluding macros.
6	Jumps to all topics specified in the contents file, excluding macros, and then quits.
7	Turns on Help Mode for the current instance of WinHelp only, and then launches a second instance of WinHelp on the same Help file. Both instances of WinHelp resize their windows so they can appear side-by-side and both display the first topic in the current Help file. Moving through one file (using CTRL+SHIFT commands) updates both windows. The functionality is the same as the Compare macro.

Comment

The Test macro is new for WinHelp version 4.0.

{button ,AL("macro_all;Compare")} Related Topics

TestALink macro

TestALink(keyword[**;** keyword]**)**

Tests whether an ALink macro has an effective link to at least one topic.

Parameter	Description
keyword	Specifies one or more A-keywords to search for. Separate multiple keywords using semicolons. If a keyword contains a comma, enclose the entire keyword string in quotation marks.

Comment

If an effective link is found, TestALink returns a value of "1".

{button ,AL("TestKLink;ALink;macro_all")} Related Topics

TestKLink macro

TestKLink("keyword[**;** keyword]"**)**

Tests whether a KLink macro has an effective link to at least one topic.

Parameter	Description
keyword	Specifies one or more K-keywords to search for. Separate multiple keywords using semicolons. If a keyword contains a comma, enclose the entire keyword string in quotation marks.

Comments

The TestKLink macro is typically used as the first parameter in an IfThen or IfThenElse macro.

Help Workshop converts this macro into a KLink macro with a TEST parameter.

Example

The following macro enables or disables a SeeAlso button, depending on whether at least one topic contains a "print" keyword:

```
IfThenElse(TestKLink(print), ChangeEnable(btn_seealso, KLink(print)), DisableButton(btn_seealso))
```

{button ,AL("macro_all;KLink;TestALink")} Related Topics

UncheckItem macro

UncheckItem(item-ID**)**

Removes from a menu item the check mark that was placed by the CheckItem macro.

Parameter	Description
item-ID	Identifies the menu item to uncheck. The ID must have been created by the AppendItem or InsertItem macros, or must be a standard WinHelp menu item ID.

Comment

Help Workshop automatically converts the UncheckItem macro to UI.

{button ,AL("MACRO_REF_MENU;macro_all;AppendItem;InsertItem;CheckItem")} Related Topics

UpdateWindow macro

UpdateWindow([filename>]window-name**,** topic-ID**)**

Jumps to the topic with the specified topic ID in the specified window, and then returns the focus to the window that called the macro.

Parameter	Description
filename	Specifies the name of the Help file to jump to, if it is not the current Help file. If specified, this parameter must be followed by a greater than sign (>).
window-name	Specifies the window type to display the topic in.
topic-ID	Specifies the ID of the topic to jump to.

Comments

If the secondary window was not already created, the UpdateWindow macro is ignored.

This macro is usually used in an entry macro to update the secondary window whenever the topic is shown.

Help Workshop automatically converts the UpdateWindow macro to JW, which is the only form of this macro that WinHelp can use.

The UpdateWindow macro is new for WinHelp version 4.0.

{button ,AL("MACRO_REF_LINK;macro_all;CloseSecondarys;CloseWindow;PositionWindow;JumpId;FocusWindow")} Related Topics

:Base command

:Base HLP-filename[>window-name]

When you specify a default filename in your contents (.cnt) file , Help Workshop adds a :base command to the file.

WinHelp searches the default Help file for all topics specified in the contents file, unless another filename is specified in the syntax for a topic.

If a :base command is specified in an included contents file, the :base command will apply only to the topic jumps in the included file.

{button ,AL(`BAS_CNT_OVERVIEW;CNT_REF')} Related Topics

:Include command

:Include cnt-filename

When you click the Include radio button in the Edit Contents Tab Entry dialog box, and then specify a filename, Help Workshop adds an :Include command to your contents (.cnt) file.

If WinHelp cannot find the file specified by the :Include command when the user opens the Help file, no error message appears. If WinHelp finds the specified file, the topics in the included contents file are added to the Contents tab at the position where the :Include command exists in the original contents file.

{button ,AL(`BAS_CNT_OVERVIEW;CNT_REF')} Related Topics

:Index command

:Index help_title = HLP-filename

When you click the Index Files button and add a file whose keywords you want to display in your index, Help Workshop adds a :Index command to your contents (.cnt) file. This command specifies the title of the Help file and its filename. The keywords in the file appear on the Index tab and the unique words in the file appear on the Find tab.

{button ,AL(`BAS_CNT_OVERVIEW;CNT_REF')} Related Topics

:Link command

:Link HLP-filename

When you add a file using the Link Files button, Help Workshop adds a :Link command to your contents (.cnt) file. This command specifies which file(s) will be searched if a user clicks a hotspot that launches the ALink or KLink macro. Keywords in Help files specified by the :Link command do not appear on the Index tab and the unique words in the file do not appear on the Find tab.

{button ,AL(`BAS_CNT_OVERVIEW;index_cnt')} Related Topics

:Nofind command

The :Nofind command specifies that no Find tab should appear in the Help Topics dialog box. You cannot add a :Nofind command to your contents (.cnt) file using Help Workshop. To add a :Nofind command, use an ASCII text editor.

{button ,AL(`BAS_CNT_OVERVIEW;CNT_REF')} Related Topics

:Tab command

:Tab tab-name **=** DLL-name

When you add a tab using the Tabs button, Help Workshop adds a :Tab command to your contents (.cnt) file. This adds a custom tab to the Help Topics dialog box.

{button ,AL("BAS_CNT_OVERVIEW;new_tab;cnt_ref")} Related Topics

:Title command

:Title title

When you specify a default title, Help Workshop adds a :Title command to your contents (.cnt) file. This title appears on the title bar of the Help Topics dialog box, and is used for any secondary window that does not specify a title.

{button ,AL(`BAS_CNT_OVERVIEW;CNT_REF')} Related Topics

Heading statements

x visible-text

When you click the Add Above or Add Below button, click the Heading radio button, and then type text, Help Workshop adds a heading statement to your contents (.cnt) file. The heading statement specifies the text you want to display next to a book icon on the Contents tab of the Help Topics dialog box.

Parameter	Description
x	Specifies an integer that denotes the level of the book. This number determines the indent level of the book when it is displayed on the Contents tab. The number 1 is the first level, 2 is the second level, and so on.
visible-text	Specifies the text that the user sees next to the book icon on the Contents tab.

Comments

Heading statements are containers for other headings and for topics. A heading statement cannot jump to a topic or run a macro.

You can use as many as 1023 characters in a book statement.

{button ,AL(`BAS_CNT_OVERVIEW;topic_cnt')} Related Topics

Topic statements

x visible-text = topic-ID[**@**HLP-filename][>window-name]

When you click the Add Above or Add Below button, click the Topic radio button, and then type a topic title and ID, Help Workshop adds a topic statement to your contents (.cnt) file. The topic statement specifies the text you want to display next to a page icon on the Contents tab of the Help Topics dialog box. It also specifies which topic will appear when a user double-clicks the topic icon.

Parameter	Description
x	Specifies an integer that denotes the level of the topic. This number determines the indent level of the topic when it is displayed on the Contents tab. The number 1 is the first level, 2 is the second level, and so on. This number is one higher than the book that contains the topic.
visible-text	Specifies the text that the user sees next to the page icon on the Contents tab.
topic-ID	Specifies the topic ID of the topic that will appear when the user selects this item in the Contents tab.
HLP-filename	Specifies the name of a Help file. This parameter must be specified only if the topic exists in a Help file other than the default Help file, which is specified by the :Base command. If you specify this parameter, it must be preceded by an at (@) sign.
window-name	Specifies the window type in which you want the topic to appear. This parameter must be specified only if you want to display the topic in a window other than the one specified by the :Base command. If you specify this parameter, it must be preceded by a greater-than (>) sign .

Comments

You can use as many as 1023 characters in a topic statement.

You can also specify a macro in a topic statement, using the following syntax:

x visible-text**=!**macro

For macro, substitute the macro syntax. (Use colons to separate multiple macros.) When the user selects the topic from the Contents tab, the specified macro runs. Note that when specifying a macro from a contents file, you must use the short name and specify all quotation marks and parameters.

{button ,AL(`BAS_CNT_OVERVIEW;book_cnt;base_cnt')} Related Topics

project file section reference

The following table describes the sections that can be used in a project file.

Section	Function
[ALIAS]	Aliases one or more topic IDs to a different topic ID.
[BAGGAGE]	Lists files that are to be placed within the Help file. (The Help file contains its own file system.)
[BITMAPS]	Specifies bitmap files. This section is maintained for backwards compatibility only.
[BUILDTAGS]	Specifies build tags. This section is maintained for backwards compatibility only.
[CONFIG]	Specifies author-defined menus and buttons used in the Help file and registers DLLs and DLL functions used as macros within the Help file. This section is required if the Help file uses any of these features.
[CONFIG:name]	Same as the [CONFIG] section, but used for individual window configurations (for name, substitute the name of the window).
[EXCLUDE]	Specifies the build tags that identify the topics that should be excluded from the Help file.
[FILES]	Specifies topic (.rtf) files to be included in the build. This section is required.
[FONTS]	Specifies the fonts in topic files that are to be replaced with different fonts, point sizes, or character sets.
[INCLUDE]	Specifies the build tags that identify the topics that should be included in the Help file.
[MACROS]	Specifies macros to run when a user selects from the index one of the keywords listed in this section.
[MAP]	Associates topic IDs with context numbers (numeric values) that a program can use to invoke the Help file.
[OPTIONS]	Specifies options that control the build process. This section is required.
[WINDOWS]	Defines the characteristics of the main Help window and secondary window types used in the Help file.

Semicolons (;) are used to indicate a comment in the project file, except in the [MACROS] section. In all other sections, Help Workshop ignores all text from the semicolon to the end of the line on which it occurs.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;HPJ_REF')} Related Topics

[ALIAS] section

[ALIAS]
context_string=alias-topic_id

The [ALIAS] section in your project (.hpj) file associates one set of topic IDs with an alternate set of topic IDs. Alias strings correspond to the IDs assigned by the #-footnotes in your topic (.rtf) files. If this section is included, it must precede the [MAP] section in the project file.

Parameter	Description
context_string	Specifies the program ID or other topic ID that you want to reassign.
alias-topic_id	Specifies the topic ID that appears in the #-footnote of the topic you want Help to recognize. An alias topic ID has the same form and follows the same conventions as a standard topic ID.

Comments

Because topic IDs must be unique for each topic and cannot be used for any other topic in the Help project, the [ALIAS] section provides a way to reassign topic IDs that are no longer used or are invalid. For example, suppose the program defines a context string for each field in a dialog box, but your Help file only provides one topic for all the fields. You can use the [ALIAS] section to map all the program context strings to your one Help topic. In this way, no matter which field the user has selected in the dialog box, Help will display your Help topic when the user requests context-sensitive Help.

You can also use the [ALIAS] section to combine Help topics without having to recode your files. For example, if you create a topic that replaces the information in three other topics, you could manually search through your files for invalid cross-references to the deleted topics. The easier approach, however, would be to use the [ALIAS] section to assign the name of the new topic to the deleted topics.

You can use alias names in the [MAP] section of the project file. If you do, however, the [ALIAS] section must precede the [MAP] section.

Example

The following example creates several aliases within an [ALIAS] section:

```
[ALIAS]
sm_key=key_shrtcuts
cc_key=key_shrtcuts
st_key=key_shrtcuts      ;combined into keyboard shortcuts topic
clskey=us_dlog_bxs
maakey=us_dlog_bxs       ;covered in using dialog boxes topic
chk_key=dlogprts
drp_key=dlogprts
lst_key=dlogprts
opt_key=dlogprts
tbx_key=dlogprts         ;combined into parts of dialog box topic
frmtxt=edittxt
wrptxt=edittxt
seltxt=edittxt           ;covered in editing text topic
```

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;map')} Related Topics

[BAGGAGE] section

[BAGGAGE]
filename

The [BAGGAGE] section lists files that Help Workshop stores within the Help file's internal file system. The files may then be read by a Help DLL.

Comments

Help Workshop stores all filenames listed in the [BAGGAGE] section exactly as they are typed. Baggage filenames are case-sensitive. To retrieve a baggage file, Help uses the filename without the path. This means that you must specify the filename exactly as it appears in the [BAGGAGE] section.

To access the data from the Help file's internal file system, WinHelp provides callback functions for DLLs so that the DLL can retrieve the appropriate data file from the Help file's [BAGGAGE] section.

The {mci} command automatically places its files in the baggage section of the Help file.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;ROOT')} Related Topics

[BITMAPS] section

[BITMAPS]
filename.bmp

This section is obsolete and is maintained for backwards compatibility only. New projects should use the Bitmaps button in Help Workshop which defines one or more BMROOT statements.

{button ,AL(`BMROOT')} Related Topics

BMROOT

BMROOT=pathname

You specify the BMROOT option in the [OPTIONS] section of the project (.hpj) file. BMROOT provides Help Workshop with information about where to find the graphics files specified in the source files.

By default, Help Workshop searches for graphics files in the same folder as the project file, then in the same folder as the .rtf file that specified the graphics, and then in all folders specified by the ROOT option. The BMROOT option is necessary only if graphics files are not in one of these locations.

You can specify any number of BMROOT options.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;ROOT;BITMAPS;PRO_HPJ_BITMAP_LOCATION')} Related Topics

BUILD

BUILD=expression

You specify the BUILD option in the [OPTIONS] section of the project (.hpj) file. You can use the BUILD option to include certain topics in and exclude them from the compiled Help file. The topics you include and exclude must contain the specified build tags. Use this option only if your topic (.rtf) files have build tags.

A topic contains a build tag if it includes a build tag footnote (*). Topics without build tags are always compiled, regardless of the expression in the BUILD option.

Parameter	Description
expression	A logical statement that specifies which topics to include in or exclude from the build. This parameter consists of a combination of build tags (which must be specified in the [BUILDTAGS] section) and the following logical operators.

Operator	Description
~	Help Workshop compiles a topic only if the tag is not present. This operator has the highest precedence; Help Workshop applies it before any other operator.
&	Help Workshop compiles a topic only if it contains both build tags used in the expression. Help Workshop applies this operator only after the ~ operator has been applied.
|	Help Workshop compiles a topic if it has at least one of the build tags used in the expression. This operator has the lowest precedence; Help Workshop applies it only after all other operators have been applied.

Parentheses may be used to override operator precedence. Expressions enclosed within parentheses are always evaluated first.

Comments

The Help Workshop interface does not provide a way to specify the BUILD option in your project file. Instead, it provides a Build Tags tab in the Options dialog box that enables you to specify what build tags to include and exclude. These build tags are placed in the [INCLUDE] and [EXCLUDE] sections respectively. If there is an [INCLUDE] or [EXCLUDE] section in your project file when you compile a file, Help Workshop ignores any BUILD option you may have specified.

To use a BUILD option instead of an [INCLUDE] or [EXCLUDE] section, you must load the project file into a text editor and add the option to the [OPTIONS] section. You must also add a [BUILDTAGS] section specifying all the build tags that you will be using.

Only one BUILD option can be specified per project file.

Help Workshop evaluates all build expressions from left to right, using the specified precedence rules.

Examples

The following examples assume that the [BUILDTAGS] section in the Help project defines the build tags DEMO, MASTER, and TEST_BUILD. Although the following examples show several BUILD options on consecutive lines using these build tags, only one BUILD option would be allowed in the project file.

```
BUILD=DEMO          ;Compile all topics that have the DEMO tag
BUILD=DEMO & MASTER          ;Compile all topics with both the DEMO and MASTER tag
BUILD=DEMO | MASTER          ;Compile all topics with either the DEMO or MASTER tag
BUILD=(DEMO | MASTER) & TEST_BUILD          ;Compile all topics that have either the DEMO or MASTER tag and also the
TEST_BUILD tag
BUILD=~ MASTER          ;Compile all topics that do not have the MASTER tag
```

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;BUILDTAGS;BAS_INSERTING_BUILD_TAG')} Related Topics

[BUILDTAGS] section

[BUILDTAGS]
tag

The [BUILDTAGS] section defines the valid build tags for a Help file. Help Workshop uses the build tags to determine which topics to include or exclude when building the Help file.

This section is used in conjunction with the build tag footnote (*) and the BUILD option. The build tag footnote associates a particular build tag with a given topic. If the build tag is included in the [BUILDTAGS] section and defined in the BUILD expression, Help Workshop compiles the topic; otherwise, it excludes the topic from the build.

Parameter	Description
tag	Specifies a build tag consisting of any combination of characters except spaces. Help Workshop strips any spaces it finds between the start of the build tag and the end of the tag. Build tags are case insensitive, so Help Workshop treats uppercase and lowercase characters as the same. Each build tag can have as many as 32 characters.

Comments

The Help Workshop interface does not provide a way to modify the [BUILDTAGS] section in your project file. To create or modify a [BUILDTAGS] section, load the project file into a text editor, and then add or edit the section.

Example

The following example defines four build tags in a sample project file:

```
[BUILDTAGS]
DEMO        ;topics to include in demo build
MASTER      ;topics to include in master Help file
DEBUGBUILD  ;topics to include in debugging build
TESTBUILD   ;topics to include in a mini-build for testing
```

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;BUILD;BAS_INSERTING_BUILD_TAG')} Related Topics

CHARSET

CHARSET=charset-value

You specify the CHARSET option in the [OPTIONS] section of the project (.hpj) file. CHARSET specifies the default character set to use for the fonts in the Help file.

If your RTF writer specifies a character set, it takes precedence over the CHARSET value. Microsoft Word for Windows versions 6.0 and later specify a character set. Earlier versions of Microsoft Word do not.

The CHARSET option is new.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;DEFFONT')} Related Topics

CITATION

CITATION=text

You specify the CITATION option in the [OPTIONS] section of the project (.hpj) file. CITATION appends a citation to the end of any information (except in a context-sensitive pop-up window) that is copied or printed from the Help file. This option is commonly used to attach a copyright notice to any information in your Help file that is copied or printed. You can specify up to 2,000 characters (1,000 for DBCS languages) for the citation text.

Note

- The COPYRIGHT option also displays copyright text in WinHelp's Version dialog box.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;COPYRIGHT')} Related Topics

CNT

CNT=contents-file

You specify the CNT option in the [OPTIONS] section of the project (.hpj) file. CNT identifies the name of the contents (.cnt) file to associate with the Help file.

If the [OPTIONS] section does not include a CNT option, WinHelp looks for a contents file with the same filename as the Help file (and the .cnt filename extension). If neither of these files is found, WinHelp displays the topic specified by the topic ID in the CONTENTS option.

The CNT option is new.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;CONTENTS_hpj')} Related Topics

COMPRESS

COMPRESS=compression-level

You specify the COMPRESS option in the [OPTIONS] section of the project (.hpj) file. COMPRESS specifies the type of compression Help Workshop should use when compiling the Help file.

Parameter	Description
compression-level	Specifies the level of compression. The default value is 0.

Hex Value	Meaning
0x00	Help Workshop does not compress the Help file.
0x01	Help Workshop attempts to determine the best possible compression for the current Help file.
0x02	Help Workshop uses Phrase compression on text.
0x04	Help Workshop uses Hall compression on text.
0x08	Help Workshop uses Zeck compression on text.
0x10	Help Workshop uses RLE compression on bitmaps.
0x20	Help Workshop uses Zeck compression on bitmaps.

Comments

If you specify maximum compression using Help Workshop, it will write 0x12 as the value for the COMPRESS option in your project file.

If you specify phrase compression, Help Workshop creates a phrase-table file that has a .ph filename extension. If Help Workshop finds an existing phrase file, it deletes the file, and then creates a new one. If you select the Use Existing Phrase (.Ph) File check box on the Compression tab in the Options dialog box, Help Workshop uses your existing phrase file. Using an existing phrase file reduces compile time, but compression is less effective.

Hall compression uses the Ftsrch.dll file, which is included with Windows 95 and with Windows NT version 3.51.

If the COMPRESS option is set to 1, HIGH, or YES, Help Workshop chooses between Hall and Phrase compression based on the size of your topic (.rtf) files. If the total size of the topic files is greater than 1 MB, Help Workshop uses Hall compression.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;OLDKEYPHRASE;BAS_COMPRESSION')} Related Topics

[CONFIG] section

[CONFIG]
macro

The [CONFIG] section contains one or more WinHelp macros that are run when WinHelp opens the Help file. This section can contain macros that carry out actions, such as creating buttons or menus, and macros that register routines in external DLLs as WinHelp macros. These routines can then be used the same as WinHelp macros.

Comments

When listing macros in the [CONFIG] section, include only one macro per line.

When a Help file opens, WinHelp runs the macros listed in the [CONFIG] section in the order in which they are specified.

Example

The following example registers two DLLs, creates a button, and sets the name of the How To Use Help file:

```
[CONFIG]
RegisterRoutine(bmp, CopyBmp, S)
CreateButton(btn_copyart, Copy Art, CopyBmp(`myart.bmp'))
```

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;MACRO_OVERVIEW;config_x')} Related Topics

[CONFIG:x] section

[CONFIG:name**]**
macro

The [CONFIG:name] section contains one or more macros that are run when WinHelp opens the corresponding window.

Comments

When listing macros in the [CONFIG:name] section, include only one macro per line.

When a Help file opens, WinHelp runs the macros listed in the [CONFIG:name] section in the order in which they are specified.

The [CONFIG:name] section is new.

Example

The following example closes all open secondary windows when the proc4 window is opened.

```
[CONFIG:proc4]
CloseSecondarys()
```

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;MACRO_OVERVIEW;config_x')} Related Topics

CONTENTS

CONTENTS=topic-id

You specify the CONTENTS option in the [OPTIONS] section of the project (.hpj) file. CONTENTS identifies the topic ID of the default topic.WinHelp displays this topic by default if it encounters a broken jump or the associated contents file is missing.

Note

- If the [OPTIONS] section does not specify a default topic, Help Workshop uses the first topic in the first topic (.rtf) file that is listed in the [FILES] section of the project file.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;CNT')} Related Topics

COPYRIGHT

COPYRIGHT=copyright-notice

You specify the COPYRIGHT option in the [OPTIONS] section of the project (.hpj) file. COPYRIGHT places a custom copyright notice in the Version dialog box of WinHelp. WinHelp displays the notice immediately below the Microsoft copyright notice. You can specify up to 255 characters (127 for DBCS languages) as the copyright text.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;CITATION')} Related Topics

DBCS

DBCS=value

You specify the DBCS option in the [OPTIONS] section of the project (.hpj) file. DBCS specifies whether the files that will be compiled use a double-byte character set (DBCS).

Comments

Help Workshop recognizes certain language IDs (LCID option) and forces the DBCS option to a particular setting, no matter how you specify it in the project file. The following language IDs force DBCS=YES.

Identifier	Locale	Identifier	Locale
0x0411	Japanese	0x1004	Singapore
0x0404	Taiwan	0x0C04	Hong Kong

The following language IDs force DBCS=NO.

Identifier	Locale	Identifier	Locale
0x0409	American	0x042f	Macedonia
0x0C09	Australian	0x080A	Mexican
0x0C07	Austrian	0x0819	Moldavia
0x042D	Basque	0x0818	Moldavia
0x080C	Belgian	0x1801	Morocco
0x0809	British	0x1409	New Zealand
0x0402	Bulgaria	0x0414	Norwegian (Bokmal)
0x1009	Canadian	0x0814	Norwegian (Nynorsk)
0x041A	Croatian	0x2001	Oman
0x0405	Czech	0x0415	Polish
0x0406	Danish	0x0416	Portuguese (Brazilian)
0x0413	Dutch (Standard)	0x0816	Portuguese (Standard)
0x0C01	Egypt	0x0418	Romania
0x040B	Finnish	0x0419	Russian
0x040C	French (Standard)	0x0401	Saudi Arabia
0x0C0C	French Canadian	0x081A	Serbian
0x0407	German (Standard)	0x041B	Slovak
0x042E	Germany	0x0424	Slovenia
0x0408	Greek	0x0C0A	Spanish (Modern Sort)
0x040E	Hungarian	0x040A	Spanish (Traditional Sort)
0x040F	Icelandic	0x0430	Sutu
0x0801	Iraq	0x041D	Swedish
0x1809	Ireland	0x100C	Swiss (French)
0x040D	Israel	0x0807	Swiss (German)
0x0410	Italian (Standard)	0x0810	Swiss (Italian)

0x2C01	Jordan	0x2801	Syria
0x3401	Kuwait	0x041E	Thailand
0x0426	Latvia	0x0431	Tsonga
0x3001	Lebanon	0x041f	Turkish
0x1001	Libya	0x3801	U.A.E.
0x1407	Liechtenstein	0x0422	Ukraine
0x0427	Lithuania	0x0420	Urdu
0x140C	Luxembourg (French)	0x0436	Zulu
0x1007	Luxembourg (German)		

Comment

The DBCS option is new.

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

DEFFONT

DEFFONT=fontname, font-size, charset

You specify the DEFFONT option in the [OPTIONS] section of the project (.hpj) file. DEFFONT specifies the default font for the lists on the Contents, Index and Find tabs and in the Topics Found dialog box.

Parameter	Description
fontname	Specifies the name of the font to use.
font-size	Specifies the point size of the font to use.
charset	Specifies the character set (CHARSET) to use.

Comment

The DEFFONT option is new.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;CHARSET')} Related Topics

ERRORLOG

ERRORLOG=error-filename

You specify the ERRORLOG option in the [OPTIONS] section of the project (.hpj) file. ERRORLOG directs Help Workshop to write all messages generated during the compilation of a Help file to a log file. As it compiles, Help Workshop also displays messages on the screen.

Note

- Help Workshop can display only 64K of information when it runs with Windows 95, and up to 1 MB of information when it runs with Windows NT. However, there is no limit to the amount of information that Help Workshop can write to a log file.

{button ,AL(`BAS_COMPILE_MSG;HPJ_SECTIONS;REPORT;NOTES')} Related Topics

[FILES] section

[FILES]
filename

The [FILES] section lists all topic (.rtf) files used to create the Help file. A Help project (.hpj) file must have a [FILES] section.

Comments

You can create an ASCII text file that contains a list of the topic files you want to include in a build, and then use the #include statement in the [FILES] section to specify the ASCII file.

The #include statement has the following syntax:

#include filename

The filename must reside in the Help project folder, or must specify a complete path. Help Workshop does not use the INCLUDE environment variable to search for files.

Help Workshop searches for files in the project folder, in the path (if any) specified for the file, and in all folders specified by ROOT options in the [OPTIONS] section of the project file.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;ROOT')} Related Topics

FORCEFONT

FORCEFONT=fontname

You specify the FORCEFONT option in the [OPTIONS] section of the project (.hpj) file. FORCEFONT causes Help Workshop to substitute the specified font for any font used in your topic (.rtf) files.

Comments

The fontname can be any of the following fonts:

- Arial
- Athens
- Bookman
- Courier
- Courier New
- Geneva
- Helv
- Helvetica
- London
- Modern
- Monaco
- MS Sans Serif
- MS Serif
- Roman
- Times
- Times New Roman
- Tms Rmn

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;MAPFONTSIZE')} Related Topics

FTS

FTS=fts-value

You specify the FTS option in the [OPTIONS] section of the project (.hpj) file. FTS specifies what type of index file to create for the full-text search feature, which you can access using the Find tab in the Help Topics dialog box.

Parameter	Description
fts-value	Specifies the level of information to include in the index file used by the full-text search engine. This value can be a sum of the following hex values.

Hex Value	Meaning
0x01	Create an index file for full-text searching.
0x02	Include untitled topics (such as context-sensitive Help topics) in the index file.
0x04	Include information to support phrase searching.
0x08	Include information to provide feedback on phrase searches.
0x10	Include information to support similarity searches.

Comment

The FTS option is new.

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

HCW

HCW=value

The HCW option is specified in the [OPTIONS] section of the project (.hpj) file. This option is reserved for use by Help Workshop.

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

HLP

HLP=help-file

You specify the HLP option in the [OPTIONS] section of the project (.hpj) file. HLP identifies the name of the Help (.hlp) file to compile.

Comments

If the [OPTIONS] section does not include an HLP option, Help Workshop creates an .hlp file that has the same filename as the current project file.

The HLP option is new.

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

INDEX_SEPARATORS

INDEX_SEPARATORS="separators**"**

You specify the INDEX_SEPARATORS option in the [OPTIONS] section of the project file. INDEX_SEPARATORS identifies the characters WinHelp uses to delineate separate keyword entries.

Parameter	Description
separators	Specifies the character used to separate keywords in an entry (for example, "keywords, entering") and the character used to separate entries in a list. If this option is not used, the default comma and colon values (",:") are used.

Comments

The INDEX_SEPARATORS option is new.

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

LCID

LCID=language case-insensitive case-sensitive

You specify the LCID option in the [OPTIONS] section of the project file. LCID specifies the language of the Help file. This information is used when sorting keywords in the index, and determines what character to use for curly quotes.

Parameter	Description
language	Specifies the hex value of the language on which to base sorting. This parameter can have only one value.
case-sensitive	Specifies the bit flags for ignoring non-spacing marks and symbols in case-sensitive text.

Hex Value	Meaning
0x2	Ignore non-spacing marks.
0x4	Ignore punctuation.
0x6	Ignore non-spacing marks and symbols.

Parameter	Description
case-insensitive	Specifies the bit flags for ignoring non-spacing marks and symbols in text that is not case-sensitive.

Hex Value	Meaning
0x2	Ignore non-spacing marks.
0x4	Ignore punctuation.
0x6	Ignore non-spacing marks and symbols.

Comment

The **LCID** option is new.

{button ,AL(`BAS_ABOUT_HPJ_FILES;BAS_TRANSLATING')} Related Topics

[MACROS] section

[MACROS]
keyword[;keyword1]
macro[;macro1]
title

The [MACROS] section associates macros with keywords. These macros run when a user selects the specified keyword in the index.

Parameter	Description
keyword	Specifies the keyword or keywords to associate with the macro(s) on the following line. Use semicolons to separate keywords.
macro	Specifies which macro to run when a keyword from the preceding line is selected in the index. Separate multiple macros using colons.
title	Specifies the title to display in the Topics Found dialog box when the selected keyword is linked to two or more Help topics. If no title is to display, leave this line blank. Title text is limited to 255 characters.

Comments

Help Workshop uses semicolons as keyword and macro delimiters in the [MACROS] section, so you cannot use comments in this section.

There is no limit to the number of three-line keyword macro definitions you can specify in this section, however you cannot use the same keyword more than once.

You cannot include a file in the [MACROS] section. If you want to include a list of keyword macro definitions, you must include a file (at the end of the [OPTIONS] section) that also contains the [MACROS] section heading.

It is strongly recommended that you use only Help Workshop to create or modify the [MACROS] section.

The [MACROS] section is new.

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

[MAP] section

[MAP]

topic-id numeric value

The [MAP] section associates topic IDs (or aliases) with context numbers for context-sensitive Help. The context number corresponds to a value the parent program passes to WinHelp to display a particular topic.

Parameter	Description
topic-id	Specifies the topic ID of a topic in the Help file. The ID can be any combination of characters, except spaces, and must also be specified in a topic ID footnote (#) in some topic in the Help file.
numeric value	Specifies the context number to associate with the topic ID. If the number begins with a digit other than zero (0), Help Workshop interprets it as decimal. If it begins with a zero (0) followed by the letter X, it is interpreted as hexadecimal. If it begins with a zero (0) followed by a digit, it is interpreted as octal.

Comments

You must use at least one space or tab between the context number and the topic ID. You can precede the topic ID with #define as in:

```
#define IDH_MYHELP      12
```

You can include C header files in the [MAP] section using the #include statement as in:

```
#include myhelp.h
```

When you include a C header file, observe the following:

- You can use C-style comments (/* and */). The comments can occur anywhere in the line as in:

  ```
  #define topic-id      context-number      /* comment */
  ```

- You can use C++-style comments (//) with the #define statement. Everything from // to the end of the line will be ignored.
- Help Workshop supports 32-bit constants in #define statements. It also accepts (as 32-bit constants) numbers that end with L and are accepted by the C compiler for a long constant:

  ```
  #define vscroll       1234000L
  ```

- Help Workshop does not perform arithmetic on the object of the #define statement. It does not support the following forms of #define:

  ```
  #define A   1
  #define B   (A+1)
  #define C   (A+2)
  ```

- Help Workshop accepts only #define statements; it does not support other forms such as #ifdef and #endif.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;ALIAS')} Related Topics

MAPFONTSIZE

MAPFONTSIZE=m[-n]:p

You specify the MAPFONTSIZE option in the [OPTIONS] section of the project (.hpj) file. MAPFONTSIZE maps the font sizes specified in topic (.rtf) files to different sizes when they are displayed in the Help window. You can use one font size in your topic files and have Help Workshop change the font so that it is a different size when the Help topic is displayed.

This option is especially useful if there is a significant size difference between the authoring display and the intended user display.

Parameter	Description
m[-n]	Specifies the size of the source font. This parameter is either a single point size or a range of point sizes, as indicated by the optional parameter n, which specifies a font range to be mapped. A range of point sizes consists of the low and high point sizes separated by a hyphen (-). If a range is specified, all fonts in the range are changed to the size specified by the p parameter.
p	Specifies the size of the desired font for the Help file.

Comment

Although you can specify as many as five font ranges in the [OPTIONS] section of the project file, you can map only one font size or range with each MAPFONTSIZE statement. If you include more than one MAPFONTSIZE statement, the source font size or range that is specified in subsequent statements cannot overlap previous mappings.

Examples

The following example illustrates a correct use of the MAPFONTSIZE option, using two statements, in a project file:

```
[OPTIONS]
MAPFONTSIZE=8:12          ; display all 8 pt. fonts as 12 pt.
MAPFONTSIZE=12-24:16      ; display fonts from 12 to 24 pts. as 16 pt.
```

The following two statements show an incorrect use of the MAPFONTSIZE option because the second statement contains a point size already mapped in the preceding statement (14 falls in the 12-24 range):

```
[OPTIONS]
MAPFONTSIZE=12-24:16
MAPFONTSIZE=14:20
```

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;FORCEFONT')} Related Topics

MULTIKEY

MULTIKEY=footnote-character

You specify the MULTIKEY option in the [OPTIONS] section of the project file. MULTIKEY specifies the footnote character to use for an alternate keyword table in the index. An alternate index can be accessed only when a program calls the WinHelp API using the HELP_MULTIKEY command.

Parameter	Description
footnote-character	Specifies the case-sensitive letter to be used for the keyword footnote.

Comment

You can use any alphanumeric character for a keyword table except K, k, A, or a; these are reserved for Help's standard and ALink keyword tables. There is an absolute limit of five keyword tables, including the standard K-keyword table and the A-keyword table.

Example

The following example illustrates how to enable the letter L for a keyword-table footnote:

```
MULTIKEY=L
```

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;ALink;KLink')} Related Topics

NOTES

NOTES=[0 | 1 | yes | no]

You specify the NOTES option in the [OPTIONS] section of the project (.hpj) file. NOTES specifies whether Help Workshop should display notes while compiling the Help file. Notes are messages Help Workshop displays when it encounters problems that may or may not adversely affect your Help file.

Comment

The NOTES option is new.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;REPORT;ERRORLOG')} Related Topics

OLDKEYPHRASE

OLDKEYPHRASE=value

You specify the OLDKEYPHRASE option in the [OPTIONS] section of the project (.hpj) file. OLDKEYPHRASE specifies whether an existing key-phrase file should be used when compressing a Help file with phrase compression.

Parameter	Description
value	If this value is YES, the existing phrase (.ph) file is used for phrase compression. If this value is NO, a new .ph file is created during the build. The default value is NO.

Comment

If you specify phrase compression, Help Workshop creates a phrase-table file that has a .ph filename extension. If Help Workshop finds an existing phrase file, it deletes the file, and then creates a new one — unless OLDKEYPHRASE=YES is specified in your project file. If it is, Help Workshop uses your existing phrase file. Using an existing phrase file reduces compile time, but compression is less effective.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;COMPRESS;BAS_COMPRESSION')} Related Topics

[OPTIONS] section

[OPTIONS]
option

The [OPTIONS] section includes options that control how a Help file is compiled and what feedback the compilation process displays. This section, along with the [FILES] section, is required in all project (.hpj) files. This section should be placed first in the project file so that the options apply during the entire compilation process.

Parameter	Description
option	Specifies one or more of the following project-file options.

Option	Description
BMROOT	Specifies the folder that contains the bitmap files named in {bmc}, {bml}, and {bmr} references in the topic (.rtf) files.
BUILD	Determines which topics to include in the build.
CHARSET	Specifies the default character set.
CITATION	Adds a citation to the end of any text copied or printed from the Help file.
COMPRESS	Specifies how the Help file should be compressed.
CONTENTS	Specifies the default topic ID.
CNT	Specifies the name of the Help file's contents (.cnt) file.
COPYRIGHT	Adds a unique copyright message for the Help file to WinHelp's Version dialog box.
DBCS	Specifies whether topic files use a double-byte character set.
DEFFONT	Specifies the default font used in WinHelp text boxes.
ERRORLOG	Saves compilation messages in a file during the build.
FORCEFONT	Forces all authored fonts in topic files to appear in a different font when displayed in the Help file.
FTS	Specifies what level of information to include (if any) in the index file for full-text searching.
HCW	Reserved for use by Help Workshop.
HLP	Specifies the name of the Help (.hlp) file to create.
INDEX_SEPARATORS	Specifies the characters WinHelp uses to identify first- and second-level index entries.
LCID	Specifies the language of the Help file.
MAPFONTSIZE	Maps a font size in the topic file to a different font size in the compiled Help file.
MULTIKEY	Specifies an alternate keyword table to use for mapping topics.
NOTES	Specifies whether Help Workshop should display notes.
OLDKEYPHRASE	Specifies whether Help Workshop should use the existing key phrase table or create a new one during compilation.
REPLACE	Specifies a path prefix to replace and its replacement.
REPORT	Controls the display of messages during the build process.
ROOT	Specifies the folders that contain the topic files listed in the [FILES] section.
TMPDIR	Specifies the folder in which to place the temporary files that are created while compiling the Help file.
TITLE	Specifies the text that appears in the title bar of the Help window when the file is open.

Comment

Most of these options can appear in any order within the [OPTIONS] section. If used, the REPLACE option must come before the ROOT or BMROOT options.

{button ,AL("BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;BMROOT;BUILD;CHARSET;CITATION;COMPRESS;CONTENTS_hpj;CNT;

COPYRIGHT;DBCS;DEFFONT;ERRORLOG;FORCEFONT;FTS;HCW;HLP;ICON;INDEX_SEPARATORS;LANGUAGE;LCID;MAP
FONTSIZE;MULTIKEY;NOTES;OLDKEYPHRASE;OPTCDROM;REPLACE;REPORT;ROOT;TMPDIR;TITLE;VERSION")}
Related Topics

REPLACE

REPLACE=old-prefix **=** new-prefix

You specify the REPLACE option in the [OPTIONS] section of the project (.hpj) file. REPLACE replaces pathname prefixes with a new prefix. If any specified files (for example, topic and bitmap files) move to a different server or folder, you need to update only this parameter. You do not have to edit pathnames in the project file or in included files.

Parameter	Description
old-prefix	Specifies the pathname prefix to replace.
new-prefix	Specifies the new prefix to use in place of the old prefix.

Comments

This command is not necessary if all files have been specified using path relative to the location of the project file.

The pathname in either prefix can be a UNC path, a drive letter, or a relative path.

When used in a project file, the REPLACE option must come before any path specification, such as those specified in a ROOT or BMROOT option.

The REPLACE option is new.

Example

The following example replaces the current local path to topic files with a network path.

```
REPLACE=c:\georges\project=\\server\share\georges\project
```

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

REPORT

REPORT=(yes | no)

You specify the REPORT option in the [OPTIONS] section of the project (.hpj) file. REPORT causes Help Workshop to display messages while it compiles a Help file. These messages indicate when Help Workshop is performing the different phases of the build, including scanning the file for compression, compiling the file, verifying topic IDs, and resolving jumps, keywords, and browse sequences.

Parameter	Description
value	If YES, specifies that Help Workshop display progress messages on the screen during the build. The default value is YES.

Comments

Displaying the REPORT messages slows down Help Workshop. To speed up the compile process, set the REPORT option to NO.

If you have specified a log file, Help Workshop writes REPORT messages to it.

{button ,AL(`BAS_COMPILE_MSG;HPJ_SECTIONS;ERRORLOG;NOTES')} Related Topics

ROOT

ROOT=pathname

You specify the ROOT option in the [OPTIONS] section of the project (.hpj) file. ROOT specifies the folder in which Help Workshop should search for the topic (.rtf) files that make up the contents of the Help file.

Comments

The ROOT option also specifies paths that Help Workshop should search for files that are included (using an #include statement) in the project file.

If the project file does not have a BMROOT option, Help Workshop searches the folders specified in the ROOT option for bitmaps named in {bmc}, {bml}, and {bmr} references in the topic (.rtf) files. If the project file does not have a ROOT option or if none of the ROOT folders contain the bitmaps, the bitmap filenames must be listed in the [BITMAPS] section of the project file.

Example

The following entry specifies that topic files are located on \\server\share\randys\project:

```
[OPTIONS]
ROOT=\\server\share\randys\project
```

Given this ROOT option, topic files can be listed in the [FILES] section without specifying a path.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;BMROOT;FILES')} Related Topics

TMPDIR

TMPDIR=folder

You specify the TMPDIR option in the [OPTIONS] section of the project (.hpj) file. TMPDIR specifies the folder where temporary files that are created while compiling the Help file should be stored. Help Workshop usually does not create temporary files unless your Help file exceeds 8 MB in size.

Comments

If the TMPDIR option is not set, Help Workshop uses the folder specified by the TEMP environment variable.

The TMPDIR option is new.

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

TITLE

TITLE=titlename

You specify the TITLE option in the [OPTIONS] section of the project (.hpj) file. TITLE assigns a title to the Help file. The title can have up to 127 characters (63 for DBCS languages).

Comments

WinHelp displays the text specified in the TITLE command in the:

- Title bar of the Help Topics dialog box if a title was not specified in the contents file.
- Title bar of the main Help window whenever it displays the Help file if a caption has not already been defined for it and no title was specified in the contents file.
- Topics Found dialog box if you have set up your keyword jump to display the title of the Help file with the topic title.
- Version dialog box in WinHelp if the COPYRIGHT parameter was not specified in the project file.

{button ,AL(`BAS_ABOUT_HPJ_FILES;HPJ_SECTIONS;COPYRIGHT;WINDOWS;title_cnt')} Related Topics

[WINDOWS] section

[WINDOWS]
window-name="caption", (x-coord, y-coord, width, height), state&buttons, (scrolling-RGB), (nonscrolling-RGB), state

The [WINDOWS] section defines the size, location, and colors for the primary Help window and any secondary window types used in a Help file.

Help Workshop provides a simple interface for creating or editing the [WINDOWS] section. It is recommended that you use Help Workshop, rather than a text editor, to make changes to this section.

Parameter	Description
window-name	Specifies the name of the window that uses the defined attributes. For the primary Help window, this parameter is main. For a secondary window, this parameter may be any unique name (other than main) with as many as eight characters. Any jumps that display a topic in a secondary window give this window name as part of the jump.
caption	Specifies the text that appears in the title bar of the window. The caption can have as many as 50 characters.
x-coord	Specifies the x-coordinate, in Help units, of the window's upper-left corner. The horizontal position is defined in terms of WinHelp's 0-1023 coordinate system. (Help assumes the screen is 1024 units wide, regardless of resolution. For information about how to determine actual coordinates for different video resolutions, see the Comments section below.) The x-coordinate is relative to the upper-left corner of the screen, which is 0,0.
y-coord	Specifies the y-coordinate, in Help units, of the window's upper-left corner. The vertical position is defined in terms of Help's 0-1023 coordinate system. (Help assumes the screen is 1024 units wide, regardless of resolution. For information about how to determine actual coordinates for different video resolutions, see the Comments section below.) The y-coordinate is relative to the upper-left corner of the screen, which is 0,0.
width	Specifies the window's default width in Help's 0-1023 coordinate system.
height	Specifies the window's default height in Help's 0-1023 coordinate system.
state&buttons	Specifies whether the window is displayed maximized, and defines the buttons to display in the window's button bar, if any. This parameter is a sum of one or more of the following values.

Hex Value	Meaning
0x0001	Maximize the window, and ignore the x-coord, y-coord, width, height, and state parameters given in the type definition.
0x0004	Turn off the default buttons on the button bar (used only for the main Help window).
0x0100	Add the Options button to the button bar.
0x0200	Add the Browse buttons to the button bar.
0x0400	Add the Contents button to the button bar.
0x0800	Add the Index button to the button bar.
0x1000	Add the Help Topics button to the button bar.
0x2000	Add the Print button to the button bar.
0x4000	Add the Back button to the button bar.
0x8000	Add the Find button to the button bar.

scrolling-RGB	Specifies the background color for the window's scrolling region. Colors are given as standard RGB values, where RRR, GGG, and BBB are three-digit numbers in the range 0 to 255 representing the red, green, and blue components of the color. If this parameter is not given, Help uses the default Windows system color specified by the end-user in Control Panel.
nonscrolling-RGB	Specifies the background color for the window's nonscrolling region (if any). Colors are given as standard RGB values, where RRR, GGG, and BBB are three-digit numbers in the range 0 to 255 representing the red, green, and blue components of the color. If this parameter is not given, Help uses the default Windows system color specified by the end-user in Control Panel.
state	Specifies whether a secondary window type stays on top of other windows. For backwards compatibility, a 1 can be used to indicate that the window is to be displayed on top of other windows. Otherwise, the parameter is the letter 'f' followed by a number representing the sum of one or more of the following values:

Value	Meaning
1	Display the window on top of other windows.
2	Automatically size the window vertically to match the length of the topic (not available for **main** Help window).

Comments

You can define as many as 255 secondary windows in a project file.

If no caption is specified, WinHelp displays in the title bar of the Help window the first string it finds:

- The text specified in the :Title option in the contents (.cnt) file.
- The text specified in the TITLE option in the project file.
- "Windows Help" (in a main Help window) or no text (in a secondary window).

Help Workshop automatically combines RGB values to a single number (prefaced by the 'r' character).

A single comma may be substituted for an entry or a group of entries enclosed by parentheses. Preceding commas are required if you want to use the default settings. Trailing commas are optional.

The Help coordinate system ranges from 0 through 1023 in both directions, so the vertical position plus the height must be less than or equal to 1023. Similarly, the horizontal position plus the width must be less than or equal to 1023. This 1024-by-1024 coordinate system is mapped to the horizontal and vertical resolutions of the video card. To convert from pixels to WinHelp coordinates, you invert the ratio between Help's resolution and the video resolution. Assuming the video card resolution is horizontal by vertical pixels, and the horizontal and vertical locations (or dimensions) you want are in pixels, the x-coordinate (or width), in Help coordinates, is as follows:

x-coord = pixel location * (1024/horizontal)

width = number of pixels * (1024/horizontal)

The y-coordinate (or height), in Help coordinates, is:

y-coord = pixel location* (1024/vertical)

height = number of pixels * (1024/horizontal)

For example, if you want the window's upper-left corner to appear at horizontal pixel 320 and at vertical pixel 120, and the Help file is being displayed on a standard VGA monitor with 640 by 480 resolution, the x-coord is (320 * (1024/640)) = 512, and y-coord is (120 * (1024/480)) = 256.

{button ,AL(`BAS_ABOUT_HPJ_FILES;BAS_HPJ_CREATE_WINDOW;PRO_HPJ_CREATE_WINDOW;PRO_HPJ_CUSTOMIZE_WINDOW;HPJ_SECTIONS;TITLE;title_cnt')} Related Topics

{bmc}, {bml}, {bmr} statements

{bmx[**t**] filename1**}**

The {bmx} statement inserts a graphic into a topic when the Help file is compiled.

Parameter	Description
x	Specifies how the graphic appears in the topic. You can specify the following characters:

Value	Description
c	Aligns the bitmap as a text character on the baseline of the type in exactly the same place in the paragraph where the reference occurs. Because the bitmap is treated as text, any paragraph formatting properties assigned to the paragraph also apply to the bitmap.
l	Aligns the bitmap along the left margin. Text is aligned with the upper right corner of the bitmap and wraps along the right edge of the image.
r	Aligns the bitmap along the right margin. Text is aligned with the upper left corner of the bitmap and wraps along the left edge of the image.

t	Specifies that the white background of the graphic should be replaced by the background color of a Help topic. This character can be specified in addition to any of the other values. It can be used only with 16-color bitmaps.
filename	Specifies the name of the graphic file to include. You can include bitmaps, multiple-hotspot bitmaps (.SHG), and multiple-resolution graphic files (.MRB). You can specify multiple files representing different color depths for the same graphic by separating the filenames with a semicolon as in the following example: { bmc dog_16.bmp;dog_256.bmp;dog_mono} When the Help file is run, WinHelp checks the color capability of the user's computer and displays the appropriate graphic file.

Comments

Help Workshop ignores the **wd** parameter, which was used by previous versions of the Help compiler.

Do not include the path with the bitmap filename. Instead, use Help Workshop to specify in your project (.hpj) file which folders contain the bitmaps that are referenced in your topic (.rtf) files.

A transparent bitmap, specified by a {bmct}, {bmlt}, or {bmrt} command, has its white pixels replaced by the background color of the currently active window the first time the bitmap is displayed. It will not change when the same bitmap is displayed in another window with a different color. If the background color is dithered, WinHelp will pick the solid color closest to that dither for the background of the bitmap.

Examples

The following command displays the specified bitmap at the left margin of the topic and causes Help Workshop to replace the white pixels in the graphic image with the background color of the topic:

```
{bmlt  logo_sh.bmp}
```

{button ,AL("BAS_ADDING_GRAPHICS;BMROOT;SHED;MRBC")} Related Topics

{button} statement

{button [label]**,** macro1[**:** macro2**:** ... **:** macroN]**}**

The {button} reference inserts a button into the Help file and runs the specified macros when a user clicks the button.

Parameter	Description
label	Specifies the label, if any, that appears on the button. You cannot define an accelerator key.
macro	Specifies one or more macros to run when a user clicks the button.

Example

The following command creates a See Also button that runs a KLink macro when the user clicks it.

```
{button See Also, KLink(networks)}
```

{button ,AL("BAS_AUTH_BTN;CreateButton")} Related Topics

{ewc}, {ewl}, {ewr} statements

{ewx DLL-name**,** window-name**,** data**}**

The {ewx} reference creates an embedded window in the Help file.

Parameter	Description
x	Specifies how the graphic appears in the topic. The following characters can be specified:

Value	Description
c	Aligns the window as a text character on the baseline of the type in exactly the same place in the paragraph where the reference occurs.
l	Aligns the window along the left margin.
r	Aligns the window along the right margin.

DLL-name	Specifies the name of the DLL that controls the embedded window. The filename should not include an extension or be fully qualified. WinHelp will search for the file in the same locations as it searches for Help files.
window-name	Specifies the name of the embedded window class as defined in the source for the DLL.
data	Specifies an arbitrary string, which WinHelp passes to the embedded window when it creates the window. The DLL is responsible for parsing this string.

Example

The following shows an embedded window reference:

```
{ewl FADE, AmfWnd, clipbrd.amf}
```

{button ,AL("embed_dll")} Related Topics

{mci} statement

{mci[**_left** | **_right**] [options**,**] filename**}**

The **{mci}** command inserts an .avi file into a topic.

Parameter	Description
options	Specifies the options for the multimedia control window that is created. Use spaces (not commas) to specify more than one option.

Value	Description
EXTERNAL	Keeps the file outside of the Help file.
NOPLAYBAR	No playbar is shown (useful for auto-play and repeat).
NOMENU	No menu button is shown if there is a playbar.
REPEAT	The file automatically repeats when play is done.
PLAY	The file automatically plays when shown.

filename	Specifies the name of the multimedia file to include in the topic.

Comments

For WinHelp version 4.0, use this command only with .avi files. Future versions will support other multimedia formats.

If no options are specified, a playbar appears with a menu button, and the file is not played until the user requests it.

Examples

The following command creates a multimedia control window that automatically plays the file demo.avi when the topic is shown:

```
{mci PLAY, demo.avi}
```

{button ,AL("BAS_ADDING_VIDEO;BAS_ADDING_SOUND;MM_DESIGN;TMPDIR")} Related Topics

HCRTF command-line switches

HCRTF [**/B** path] [**/O** HLP-filename] [**/X**[**CHNRT**]] [RTF-filename] [**/NC**] [**/XT**] [HPJ-filename]

The Hcrtf.exe program is used to compile a Help file, test a contents file, or provide information about an existing Help file. It is called automatically by the Hcw.exe program. If you run Hcrtf.exe directly, you can use the following switches:

Parameter	Description
/A	Specifies that additional information is to be added to the Help file. For version 4.0, the additional information is the topic ID of each topic and the source file the topic appears in. If the Help Author command on the File menu in Help Workshop is checked, you can display this information by clicking a topic using your right mouse button, and then clicking Topic Information.
/B path	Specifies the path to search for bitmaps. You can specify multiple paths with multiple /B switches.
/F(C|**D**|**F**|**G**|**H**|**I)**	Specifies a report command. You must use one of the listed parameters with this switch. The switch and its parameter must be followed by a space, the Help filename, another space, and the fully qualified path of the output filename. The /F switch should always be used in conjunction with a /X switch. Specify one of the following parameters with the /F switch.

Value	Description
C	Writes all the topic titles in the Help file to the output file.
D	Writes all the topic text in the Help file to the output file.
F	Writes all the hash numbers in the Help file to the output file.
G	Writes project file information in the Help file to the output file.
H	Writes all K-keywords in the Help file to the output file.
I	Writes all A-keywords in the Help file to the output file.

/O HLP-filename	Specifies the name of the Help file to create. Use this switch only if you are not compiling an .HPJ file.
/N[**CGW**]	Specifies an option to turn off.

Value	Description
C	Turns off compression, no matter what value is specified by the COMPRESS option in the project file.
G	Prevents the animation window from appearing.
W	Prevents the automatic creation of a window definition, if a window specified in a hotspot or macro was not defined in the project file.

/X[**CHNRT**]	Specifies that there is no parent to send messages to. You can include one or more of the following parameters with the **/X** switch, provided they are not separated by spaces.

Value	Description
C	Specifies that Help Workshop should compile the RTF text on the Clipboard. This switch works only if a text editor has registered the "Rich Text Format" Clipboard format. This switch cannot be used if the /XR switch is used.
H	Specifies that, when the compile is complete, the Help file is to be displayed in WinHelp.
N	Specifies that the progress window should not be shown.
R	Specifies that an .rtf file is being used instead of a project file. You cannot use this switch with the /XC switch. After all other possible switches have been specified, type one or more spaces and the name of the .rtf file.
T	Specifies that there are no errors in the .rtf file. Some error checking will be turned off to speed up the compilation process. This switch should be used only when compilation speed is crucial.

RTF-filename	Specifies the name of the .rtf file to process. Used only when /XR has been specified.
HPJ-filename	If neither the /XC or /XR switch is specified, the file is presumed to be a project file.

Example

The following example would compile the Myfile.rtf topic file, producing the Test.hlp Help file for display in WinHelp:

```
hcrtf -o c:\tmp\test.hlp -xhr myfile.rtf
```

{button ,KL("HCW command;compiling, multiple Help files;COMPRESS option")} Related Topics

HCW command

HCW [[**/C**] [**/M**] [**/E**] [**/N**]] [**/T**] [filename]

The Hcw.exe program provides an interface to the Hcrtf.exe program.

Parameter	Description
/A	Specifies that additional information is to be added to the Help file. For version 4.0, the additional information is the topic ID of each topic and the source file the topic appears in. If the Help Author command on the File menu in Help Workshop is checked, you can display this information by clicking a topic using your right mouse button, and then clicking Topic Information.
/C	Starts compiling the specified project (.hpj) file or Help makefile (.hmk).
/E	Quits the Hcw.exe program after compiling the specified file. Use this switch in conjunction with the /C or /M switch.
/M	Minimizes the Hcw.exe program while compiling the specified project file.
/N	Turns off compression when compiling the specified file, no matter what value is specified for the COMPRESS option in the project file(s).
/R	Specifies that the Help file is to be displayed in WinHelp as soon as it is compiled.
/T	Turns on Translation mode. This limits what can be changed in a contents (.cnt) or project file. Only text that should be translated to another language can be changed. This can also be set by clicking the Translation command on Help Workshop's File menu.
filename	Specifies the name of one or more project, Help makefile, or contents files to open or compile.

Example

The following command compiles the New.hpj project file with no compression, minimizing the window while compiling, and adding additional information to the Help file:

```
hcw /c /m /n /a new
```

{button ,KL("HCRTF command;compiling, multiple Help files;COMPRESS option")} Related Topics

MRBC command

MRBC [**/S**] name.vga [name.ega, name.cga, name.851]

The Mrbc.exe program takes your bitmaps or metafiles with different resolutions (CGA, EGA, VGA, or 8514) and combines them into a single graphic that can be compiled into a Help file.

Parameter	Description
/S	Specifies silent mode. If you use the /S switch, the Mrbc.exe program determines the resolution of each bitmap, based on the first character of the bitmap extension. If you do not use the /S switch, the Mrbc.exe program prompts you for the resolution of each bitmap.
name	Specifies the names of the bitmaps to include in the .mrb file. Bitmap paths can be relative or absolute. Relative paths are relative to the user's current folder. The Mrbc.exe program accepts * and ? as wildcards for the filenames. Note that you can specify a Windows bitmap or a SHED hypergraphic file.

The Mrbc.exe program creates an output file that has the same filename as the first bitmap on the command line but with the .mrb extension. Output files are saved in the current folder.

Comments

Files created by the Mrbc.exe program may not compress as well as .bmp files. If you want to combine bitmaps only to provide different color depth, used the {bmx } command in your .rtf file instead.

Help Workshop removes the aspect ratio from .bmp files so that WinHelp will not stretch or shrink them. Because of this, it is rarely necessary to use the Mrbc.exe program.

If you use the /S switch, the Mrbc.exe program prompts you for the monitor type of each bitmap. You supply the appropriate response, and then the Mrbc.exe program creates the output file and saves it to the current folder. The valid case-insensitive responses to the request for monitor type are CGA, EGA, VGA, and 8514.

If you use the /S switch, MRBC uses the first character of the input file extension to determine the bitmap resolution. The following table shows how the Mrbc.exe program maps the file extensions:

First character	MRBC interpretation
C	CGA bitmap
E	EGA bitmap
V	VGA bitmap
8	8514 bitmap
other	VGA bitmap

You can specify an unlimited number of bitmaps on the command line, but the Mrbc.exe program supports only the four monitor types listed above.

In addition to resolution, WinHelp uses color as a selection criterion. Include a color version and a monochrome version of each bitmap in the same .mrb file to ensure compatibility with both monitor types.

The Mrbc.exe program does not check for valid matches between bitmap resolution and the specified option. Check .mrb bitmaps on the target monitors to ensure that they display correctly.

In some cases, the drawing tool that creates bitmaps stores the resolution in the bitmap file itself. The Mrbc.exe program checks all bitmap files to determine whether the resolution is specified. If the Mrbc.exe program finds the resolution specified in the bitmap files, it uses the stored ratio instead of the one you specified and displays an informative warning message.

The .mrb output file cannot be viewed or edited using any other graphics program.

WinHelp considers an 8514 display to be any monitor that has 120 pixels per inch. Because some high-resolution monitors deliver 96 pixels per inch to display more inches on the screen, WinHelp uses the VGA bitmap on these devices.

Bitmap files for high color depths may be too large to be practical. Where high color depth is important, use the {bmx} command or a metafile rather than a .mrb file.

Example

The following example creates a multi-resolution bitmap that accommodates each monitor type:

```
MRBC /S HOUSE.VGA HOUSE.EGA HOUSE.CGA HOUSE.8XX
```

MRBC creates a House.mrb file that consists of VGA, EGA, CGA, and 8514 bitmaps.

{button ,AL("BAS_MULT_RESOLUTION;BAS_MULT_COLOR_GRAPHICS")} Related Topics

Winhlp32.exe command-line switches

winhlp32.exe [[**-H**] [**-G**[n]] [**-W** window-name] [**-K** keyword] [**-N** context-num] [**-I** topic-id] [**-P** pop-up-id] HLP-filename]

You can use the following switches may be used when starting WinHelp:

Parameter	Description
-G[n]	Creates a configuration (.gid) file and quits. If a number is specified, it determines which extensible tab to display by default the first time the Help file is opened. A value of 1 would be the first tab beyond the Find tab.
-H	Displays the Winhlp32.hlp Help file.
-I topic-id	Displays the Help topic with the specified the topic ID.
-K keyword	Displays the topic identified by the specified keyword.
-N context-num	Displays the topic specified by the context number (defined in the [MAP] section of the project file).
-P pop-up-id	Displays the specified pop-up topic. You must use the -P switch in combination with the -I or -N switch, as shown in the following examples: WINHLP32 -P -I EXEC_WINHELP HCW.HLP WINHLP32 -P -N 311 MYFILE.HLP
-W window-name	Displays the topic in the specified window definition.
HLP-filename	Specifies the Help file to display. If a name is not specified, the File Open dialog box appears.

Comments

In Windows 95, the executable file for Winhelp has been renamed from Winhelp.exe to Winhlp32.exe. For backward compatibility reasons, Winhelp.exe still exists, but it now contains only the information necessary to start the Winhlp32.exe program.

The -G switch should be used by Setup programs when they install a newer version of a Help file or a contents file. This switch causes the .gid file to be rebuilt. WinHelp will automatically build the .gid file the first time a Help file is opened — unless the winhelp -g command has already been run on the file.

Example

The following example displays the topic identified by content number 12 in a pop-up. This would be typically be used when registering a What's This verb for a file extension:

```
winhlp32 -p -n12 myhelp
```

{button ,AL("GID")} Related Topics

Standard Help menus

WinHelp version 4.0 has five standard menus in the main window: File, Edit, Bookmark, Options, and Help. Although you cannot modify the standard Help menus or menu items, you can add additional menu items to the menus in the following table.

Menu/Item	ID	Macro	Description
File menu	mnu_file	None	None
Edit	mnu_edit	None	None
Options	mnu_options	None	None
Help	mnu_help	None	None
Floating menu	mnu_floating	None	Appears when you click a topic using your right mouse button.

{button ,AL(`BAS_AUTH_BTN;REF_STD_BTN;MACRO_REF_MENU')} Related Topics

Standard WinHelp buttons

WinHelp version 4.0 offers the following standard buttons in both main and secondary windows:

Button	ID	Macro	Description
Contents	btn_contents	Contents	Displays the Contents tab in the Help Topics dialog box. If no contents (.cnt) file exists, displays the default topic.
Index	btn_search	Search	Displays the Index tab in the Help Topics dialog box.
Find	btn_find	Find	Displays the Find tab in the Help Topics dialog box.
Help Topics	btn_topics	Finder	Displays the Help Topics dialog box, with the most recently used tab on top. If you use this button, you cannot use the Contents or Index buttons.
Back	btn_back	Back	Jumps to the topic users last displayed in the current window type.
Options	btn_menu	Menu	Displays the context-sensitive menu for the current window.
Print	btn_print	Print	Prints the current topic.
<<	btn_previous	Prev	Jumps to the previous topic in the browse sequence.
>>	btn_next	Next	Jumps to the next topic in the browse sequence.

{button ,AL(`BAS_AUTH_BTN;REF_STD_MENU;MACRO_REF_BUTTON')} Related Topics

About configuration (.gid) files

A configuration (.gid) file is a hidden file that WinHelp creates when a Help file is first opened, or when a Help file is updated. A .gid file contains information about the Help file, including:

- Binary representation of the contents (.cnt) file, including jumps and commands, after it has been processed. Only topics that were found during processing are stored.
- The filenames and titles of all Help files included in the contents file.
- Keywords from other Help files (if :Index statements were used in the contents file).
- List of which files have full-text search index (.fts) files.
- The size and location of Help windows and dialog boxes.

WinHelp creates a .gid file for each Help file that does not have a contents file, or for each contents file — even if that contents file is for a family of Help files.

If a user deletes a .gid file, WinHelp creates a new one the next time the user opens the Help file.

Help files on a network or CD-ROM

If WinHelp cannot create the .gid file in the same location as the contents file, it will try to create the .gid file in the \Windows\Help folder or in the Windows folder if no Help folder exists.

If multiple users try to build a .gid file on a network server at the same time, only the first attempt will actually create the .gid file. Other users will see an animation window until the first attempt to create a .gid file succeeds.

Updating a family of Help files

Whenever you release an updated family of Help files, your setup program should run the following command on the master (default) Help file if you are adding, removing, or updating Help or contents files.

winhlp32 -g

This command forces WinHelp to create a new .gid file for the master contents file, which ensures that users can access the topics in the new or updated Help and contents files. If you do not run this command, users who try to access a topic that has been changed or removed may encounter unexpected results.

{button ,AL("winhelp_exe")} Related Topics

Development strategies for Help files

If you are a software developer, consider the following when implementing Help as part of your programs.

Help menu

Provide a Help menu that contains a single command named Help Topics. The Help Topics command should provide access to the program's Help file(s). The API call should use the following syntax:

```
WinHelp(hwnd, "helpfile.hlp", HELP_FINDER, 0)
```

Context-sensitive Help

Assign context-sensitive help to each element in every dialog box in your program.

- For topics that will be called from the program, use the IDH_ prefix in the topic ID.
- Use a separate header (.h) file to map topic IDs and include it in the [MAP] section of the project (.hpj) file.

Uninstall option

When providing an Uninstall option for a program, consider the following Help-related files that might exist on a user's system:

- .hlp (Help file)
- .cnt (Help contents file)
- .gid (hidden configuration file)
- .fts (full-text search index file)
- .ftg (full-text search group list)

{button ,KL("MAP section;WinHelp")} Related Topics

How WinHelp finds a file

WinHelp searches for Help files, contents files, programs and DLLs in the following locations:

- If a path is specified, WinHelp first searches for the file in the path. If not found, the path is removed and the following locations are searched.
- The folder that contains the current Help file.
- The current folder.
- The System subfolder in the Windows folder.
- The Windows folder.
- The folders listed in the PATH environment.
- The location specified in Winhelp.ini.
- The registry.

The following registry location is used:

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Help

The value of the Help key is the name of the Help file, and the corresponding data is the folder where the Help file resides. The following example shows how to list a Help file in the registry:

```
if (RegCreateKeyEx(HKEY_LOCAL_MACHINE,
      "Software\\Microsoft\\Windows\\Help", 0,
      "Folder", REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS,
      NULL, &hkey, &disposition) == ERROR_SUCCESS) {
   RegSetValueEx(hkey, "your filename", 0, REG_SZ, "your folder",
      strlen("your folder") + 1);
   RegCloseKey(hkey);
}
```

If WinHelp cannot find a Help file, it displays a dialog box that enables users to specify the location of the file. If users find the file, WinHelp adds the location information to the registry.

Providing context-sensitive Help in a dialog box

To provide context-sensitive Help in a dialog box, you must create an array consisting of pairs of double word values. The first value in each pair is the identifier of a control in the dialog box, the second is the topic identifier of the Help topic for the control. The array should contain one pair of identifiers for each control in the dialog box.

The dialog box procedure must process the WM_HELP and WM_CONTEXTMENU messages. The dialog procedure receives the WM_HELP message when users press the F1 key or when users click the ? button in the title bar, and then click an object in the dialog box. The dialog procedure receives the WM_CONTEXTMENU message when users click the right mouse button.

The lParam parameter of the WM_HELP message is a pointer to a HELPINFO structure. The hItemHandle member of this structure identifies the control for which the user has requested Help. You must pass this handle to the WinHelp function along with the HELP_WM_HELP command, the name of your Help file, and a pointer to the array of identifiers. The WinHelp function searches the array for the control identifier of the specified control, then retrieves the corresponding Help context identifier. Next, the function passes the Help context identifier to WinHelp, which finds the corresponding topic and displays it in a pop-up window. If the control has an identifier of -1, the system searches for the next control that is a tab stop, and uses its identifier to find the Help context identifier. For this reason, it is important that static text be placed before controls in your resource file.

Processing the WM_CONTEXTMENU message is similar to processing WM_HELP, with two exceptions when calling the WinHelp function:

- You pass the wParam parameter from the WM_CONTEXTMENU message, which is the handle of the control that sent the message.
- You specify the HELP_CONTEXTMENU command instead of HELP_WM_HELP.

The HELP_CONTEXTMENU command causes WinHelp to display a menu, which is system defined. The menu contains a What's This command and allows users to display Help for the control. In Windows 95, a What's This command does not appear when users click an edit field using their right mouse button.

Modifying common dialog boxes

If you modify a common dialog box and want to hook your context-sensitive Help topics to the new controls, you must install a hook procedure that determines if the WM_HELP or WM_CONTEXTMENU message is for your controls. If so, call WinHelp using your Help file. If not, send the message through to the common dialog box.

Turning off context-sensitive Help for a control

If you need to turn off the context-sensitive Help for a control, such as a group, you can map the Help ID for that control to -1, as shown in the following example:

```
ID_GROUP_BOX, ((DWORD) -1)
```

{button ,AL(`cs_ex;PRO_HPJ_ADD_MAP')} Related Topics

Example code (context-sensitive Help)

The following example shows how to implement context-sensitive Help in a dialog box:

```
LRESULT CALLBACK EditDlgProc(HWND hwndDlg, UINT uMsg, WPARAM wParam,    LPARAM lParam)
{

    // Create an array of control identifiers and context identifiers.

    static DWORD aIds[] = {
        ID_SAVE,     IDH_SAVE,
        ID_DELETE, IDH_DELETE,
        ID_COPY,     IDH_COPY,
        ID_PASTE,   IDH_PASTE,
        0,0
    };

    switch (uMsg) {
        case WM_HELP:
            WinHelp(((LPHELPINFO) lParam)->hItemHandle, "helpfile.hlp",
                HELP_WM_HELP, (DWORD) (LPVOID) aIds);
            break;

        case WM_CONTEXTMENU:
            WinHelp((HWND) wParam, "helpfile.hlp", HELP_CONTEXTMENU,
                (DWORD) (LPVOID) aIds);
            break;
        .
        . // process other messages here
        .
    }

    return FALSE;
}
```

{button ,KL("context-sensitive Help;WM_HELP;WM_CONTEXTMENU")} Related Topics

Adding a tab to the Help Topics dialog box

To add a custom tab to the WinHelp Help Topics dialog box, you must write your own DLL. Your DLL must supply the following function:

HWND WINAPI **OpenTabDialog(**HWND, DWORD, DWORD**)**

This function will be called when you are to create your dialog. The first parameter will be the parent's window handle. The second and third parameters are reserved (for future use). Your DLL must then call the CreateDialog function to create your dialog and return the dialog's window handle.

The dialog template should be for a dialog without a border and non-visible, and it should use the DS_CONTROL style so that the dialog will get the focus when the tab is activated. WinHelp will resize the Help Topics dialog box as necessary to accommodate your dialog.

Use the MSG_TAB_CONTEXT and MSG_TAB_MACRO messages in your DLL to communicate with WinHelp.

Add the tab name and DLL name to the contents (.cnt) file by opening the contents file in Help Workshop, and then clicking the Tabs button.

The following is an example of the style flags to use for the dialog:

STYLE WS_CHILD | DS_3DLOOK | DS_CONTROL | WS_TABSTOP

{button ,KL("DLLs, writing for WinHelp;adding, tabs to Help Topics dialog box;OpenTabDialog;CreateDialog;DS_CONTROL;MSG_TAB_CONTEXT;MSG_TAB_MACRO;DLGTEMPLATE")} Related Topics

Windows Help settings in the Win.ini file

The Win.ini file, which is located in the Windows folder, contains initialization information about WinHelp in the [Windows Help] section. The [Windows Help] section can contain the following entries:

Backtrack=num
Help Author=1
JumpColor=(r,g,b)
IFJumpColor=(r,g,b)
PopupColor=(r,g,b)
IFPopupColor=(r,g,b)
MacroColor=(r,g,b)

{button ,AL("WININI_SETTINGS")} Related Topics

Backtrack=num

This line enables users to set the number of topics WinHelp saves in the History list. You can specify any positive integer between 1 and 500, for example:

```
Backtrack=25
```

Help Author=1

This setting is automatically set or cleared when you check or clear the Help Author command on the File menu in Help Workshop. For information about Help Author mode, click Related topics.

{button ,AL("BAS_HELP_AUTHOR_MODE")} Related Topics

JumpColor=(r,g,b), IFJumpColor=(r,g,b), PopupColor=(r,g,b), IFPopupColor=(r,g,b), and MacroColor=(r,g,b)

You can use any of the following statements in the [WINDOWS HELP] section of your Win.ini file to change hotspot colors from the default green.

JumpColor=(r,g,b)
IFJumpColor=(r,g,b)
PopupColor=(r,g,b)
IFPopupColor=(r,g,b)
and MacroColor=(r,g,b)

The r, g, and b values represent the red, green, and blue (RGB) components of the hotspot color. These color values can be a number from 0 to 255. The meaning of each statement is explained in the following table.

Command	Defines the color for	Default Color
JumpColor	Jump hotspots	Green
IFJumpColor	Interfile jump hotspots	Same as JumpColor
PopUpColor	Pop-up hotspots	Same as JumpColor
IFPopupColor	Interfile pop-up hotspots	Same as JumpColor
MacroColor	Macro hotspots	Same as JumpColor

Creating a Winhelp.ini file for search paths

If you are distributing your program and Help files on removable media such as a CD-ROM, you can have your setup program define the drives and directories where WinHelp should search for Help files. The paths you define will be added to Help's regular search paths for interfile jumps.

The syntax for the Winhelp.ini entry is:

```
[FILES]
HLP-filename=path, message
```

Parameter	Description
HLP-filename	Specifies the name of the Help or contents file for which you want to define a search path.
path	Specifies the fully qualified path to the folder where the file is located.
message	The message that WinHelp should display in a message box if the path cannot be found. This parameter is optional.

Comments

On Windows NT, you must use the Winhlp32.ini file instead of the Winhelp.ini file.

When creating entries in the Winhelp.ini file, make sure that the first program to create the Winhelp.ini file also creates the [FILES] section heading. You can use the Windows WritePrivateProfileString function to create entries in the Winhelp.ini file.

Example

The following entry instructs Help to look in the \Product\Files folder on drive P for the Notepad Help file. If necessary, Help will prompt the user to insert a CD into the drive:

```
[FILES]
Notepad.hlp=P:\Product\Files, Please insert your Windows 95 CD into drive P.
```

{button ,KL("WritePrivateProfileString;cs_find")} Related Topics

About Full-Text Search

The full-text-search (FTS) engine gives programs the same full-text search capabilities as available with Windows Help. The engine provides a set of functions that you use to create full-text-search indexes and use existing indexes to search for words and phrases in your source documents. The engine also provides functions for compressing text and highlighting search hits in your source text.

Sources, Topics, Words, and Phrases

The full-text-search engine works in conjunction with source documents (source files) that contain text organized in words and phrases, separated by wordbreak and other formatting characters. To use the engine, you must also organize the source into topics, where a topic typically corresponds a section in a book or information in a help file that is to be display as a unit. You must assign a unique value or handle to each topic; the engine uses that handle value to identify the topic during searches. Although not required, it is recommended that you assign a title to each topic.

Once you have identified the topics, you construct the index. The index contains the word and phrases from the source document (in compressed format) and data that identifies which words and phrases belong in which topics. Once you have an index, you use the built-in search dialog box provided by the engine to let the user search for words and phrases in the source documents.

Index Construction

You create an full-text-search index by using the **NewIndex** function. You specify the name of the source for the index, the character set and locale for which the index is created, the type of searching methods the index must support, and the kinds of handle values you'll used to identify topics. If successful, the function initializes the index and returns an index handle (a value having HINDEX type) identifying it. You use the index handle in subsequent function calls to add entries to the index.

When you create an index you can specify a combination of search types. If no other type is given, you must specify TOPIC_SEARCH. This ensures that the index supports searches for topics that contain a given word or collection of words. You can use the PHRASE_SEARCH value to create an index that supports searches for topics that contain a given phrase, that is, a specific sequence of words. Use the PHRASE_FEEDBACK value for an index that supports validation of phrases, that is, provides feedback about the set of valid words that can appear next in a given phrase. Use the VECTOR_SEARCH for an index that supports similarity searches, that is, searches that present topics that are statistically similar in terms of type and frequency of words used.

The search types are cumulative capabilities. That is, you can't ask for PHRASE_FEEDBACK without PHRASE_SEARCH, and you can't ask for VECTOR_SEARCH without PHRASE_FEEDBACK.

You add topics to the index by using first calling the **ScanTopicTitle** function to define a title and a handle value for the topic. The handle value may be either a 32-bit value or a 64-bit value. Then you supply the text for the topic by one or more calls to the **ScanTopicText** function. You repeat this procedure, calling **ScanTopicTitle** followed by calls to **ScanTopicText**, until you have indexed all topics.

Once you have added entries to the index, you can save the index to disk by using the **SaveIndex** function. The index remains in memory, even if you have saved it to disk. You must use the **DeleteIndex** function to finally remove the index.

Because indexing large sources may be a lengthy operation, you should give users some indication that progress is being made. To ensure that you can indicate progress while entries are being added to an index, you use the **RegisterAnimator** function to register an **AnimatorProc** callback function. That function will be called frequently during the indexing operation. When your **AnimatorProc** is called you may take appropriate graphical actions to show that your program is "alive" and is making progress.

You should be careful that your **AnimatorProc** doesn't take so much processor time that it slows down the indexing process. One good way to keep your processing under control is to call **GetTickCount** at the start of your **AnimatorProc** and use a global variable to keep track of how recently you updated your progress animation or your progress indicator.

{button ,AL("FTS_Index_Construct")} Related Topics

Where Files are Placed

When the indexer constructs an index file, it first attempts to create the index in the directory defined by the **SaveIndex** function. If that is not possible, it tries to create the index file in the \Windows\HELP folder. If that attempt also fails, it makes one last attempt to create the file in the Windows folder before giving up.

When temporary files are necessary, the indexer first interprets the pbSourceName value passed to **NewIndex** as an initial location and attempts to create the temporary file there. If that fails, it tries the \Windows\HELP folder and finally the \Windows folder before giving up.

{button ,AL("NewIndex;SaveIndex")} Related Topics

Searcher

You use the searcher to carry out a search for the topic or topics in a given collection that contain a particular set of words or phrases. You create a searcher by using the **NewSearcher** function. If successful, the function returns a searcher handle (a value having HSEARCHER type) identifying the searcher. You use the handle in subsequent function calls to open an index and search for words and phrases.

Before you can search for words, you must open an index file and associate it with the searcher. You do this by using the **OpenIndex** function. If the function is successful, it returns an integer value that identifies the index. You can use the index identifier with the **QueryOptions** function to determine what type of searches the index supports. You can also check which type of searches are valid for an index before opening the index by using the **IsValidIndex** function. The index identifier will also be used later when a topic must be displayed from that index.

When you call **OpenIndex**, it will return to you the source name and time stamp values that were originally passed when it was created with the **NewIndex** function. If those don't match what you're expecting, you can call **DiscardIndex** to remove it from the search set. Alternatively you can supply the source name and time stamp values to OpenIndex and have it do the comparisons for you.

You can associate more than one index with a searcher by using the **OpenIndex** function multiple times. If you do associate a group of indexes with a searcher, you can save information about that group to a file on disk by using the **SaveGroup** function. If you save the group in this way, at a later time you can quickly open and associate the group of index files with a new searcher by using the **LoadGroup** function. That is, you don't have to explicitly call the **OpenIndex** function for each index in the group.

Once you have an index or indexes associated with the searcher, you can create a search dialog by calling the **OpenDialog** function. That function returns the window handle for the dialog box, so you can access and manage the dialog box just as you would any other window. Because the dialog box is created invisible and disabled, you must explicitly enable and show it before the user can use it. In general, you should move the dialog box to the location in its parent window, then show it and enable it.

Later when the user double-clicks one of the topic titles, the dialog will send a message to its parent window. That message will tell you the topic handle and index identifier for that title. Then your program must display the corresponding topic text. Usually that message will be MSG_FTS_JUMP_HASH, but it may also be MSG_FTS_JUMP_VA or MSG_FTS_JUMP_QWORD, depending on indexing options which were given when the index was created. In all cases the topic handle will be exactly the value you passed to the **ScanTopicTitle** function when you indexed that topic.

You can supply a "Display" button in the parent window and get a topic displayed by sending a WM_COMMAND message of type ID_OK to the dialog box. It will then send back a message with the handle and index identifier for the topic whose title is highlighted.

The dialog box has several other message it may send to its parent window. You'll get a MSG_FTS_GET_TITLE message when the dialog needs to show the name you've given to the text set that goes with a particular index. You'll be given the identifying number for the index, and you'll supply a null-terminated string.

When you call the **LoadGroup** function, it may send you one or more MSG_FTS_WHERE_IS_IT messages if the FTG file or one of the files in the group doesn't exist or isn't valid. That message will point to the file name that can't be found, and you can change the file name string to point to an alternate directory. If you want to support this message, you must call **SetDirectoryLocator** to identify the window which will process the MSG_FTS_WHERE_IS_IT message.

Finally you may receive a MSG_REINDEX_REQUEST message if the user clicks on the "Rebuild" button. That's your signal to rebuild the index (or indices) perhaps with new indexing options.

If you'd like to temporarily hide the search dialog, simply send it a WM_CLOSE message. The dialog will make itself invisible and will set the desktop to be its parent. Later when you want to make the search dialog visible again, simply send a UM_CONNECT message to it. If you really want to destroy the search dialog, send it a UM_CLOSE message. Alternatively you can just call **DeleteSearcher**.

{button ,AL("FTS_Search")} Related Topics

Highlighting Search Hits

When you're displaying a topic you've found with a searcher you can highlight the target words and phrases using the Hiliting interfaces. After you've opened a Searcher (with **NewSearcher**) you can attach a hiliter to it (with **NewHiliter**). You only need to create one Hiliter for a sequence of search operations.

When you want to get highlighting information for a search topic you must first extract the text to be examined and pass it to the hiliter (using **ScanDisplayText**). This may be done in sections, either for convenience, or because different sections of the text have different Charset and Locale designations. If you later want to begin fresh on a new topic, you must first call **ClearDisplayText** to discard the old display text.

Once all the text has been scanned, you retrieve the hilite count (using **CountHilites**) for all part of the text to determine how much hilite storage space is required. Then you use **QueryHilites** to get the hilites themselves as offsets into the text you passed to ScanDisplayText.

The final stage is to apply the hilite offsets to your original text -- which is something you do entirely in your own code.

{button ,AL("FTS_topic_hiliting")} Related Topics

Phrase Compression

You can take advantage of the built-in compression capabilities to compress your own text. You can create a compressor by using the **NewCompressor** function. If successful, the function returns a compressor handle (a value having HCOMPRESSOR type which identifies the compressor). You use that handle in subsequent function calls to create a phrase table, and compress or decompress text. When you no longer need the compressor, you can delete it by using the **DeleteCompressor** function.

You carry out the compression in two steps. First, you use the **ScanText** function to let the compressor develop compression statistics (a phrase table) for the text you want to compress. Then you use the **CompressText** function to carry out the actual compression. The function returns the compressed text in a buffer and specifies the size in bytes of that text.

Typically, a program saves the compressed text so that it can be expanded later. To expand to the compressed text later, you must always save the compression statistics, called the phrase table, with the compressed text. You can retrieve the phrase table data by using the **GetPhraseTable** function. You can call **GetPhraseTable** after finished scanning all the text in the first pass.

To expand (decompress) the compressed text, you use the **DecompressText** function. Before expanding, however, you must set the phrase table for the text by using the **SetPhraseTable** function.

{button ,AL("FTS_phrase_compress")} Related Topics

Language Support

In most cases the full-text engine automatically adjusts to accommodate the active language based on information from the National Language Support (NLS) APIs. For the cases where that support is not sufficient there are two types of external DLL interfaces — a Stemmer interface and a Word Breaker interface.

{button ,AL("FTS_NLS")} Related Topics

The Stemmer Interface

A stemmer is a language dependent piece of code that reduces related words to a common form. For example an English stemmer might map “publish”, “published”, “publishes”, and “publishing” to the form “publish”. The key idea is that words which denote a common idea map to the same stem.

The full-text engine automatically uses language specific stemmers to build index data to support similarity searches between topics. Similarity search information can still be built when no stemmer is available, but it is more accurate when a stemmer is present. Windows 95 includes a stemmer specific to U. S. English (STEM0409.DLL).

If you want to construct a stemmer for some other language, you must construct a DLL whose name follows the pattern STEMxxxx.DLL, where xxxx is the hexadecimal representation of the local identifier (lcid) for the language. Within that DLL you must export a function named Stemmer.

{button ,AL("Stemmer")} Related Topics

The Word Breaker Interface

A word breaker scans a sequence of characters and divides it into words and punctuation. This is a fundamental operation during index construction.

For most languages word breaking can be done automatically building a table that defines whether a character is a word character or a punctuation character. Then contiguous sequences of word characters are presumed to be words.

For some languages (Japanese, for example) the rules are considerably more complicated. To address those languages the full-text search engine (FTSRCH.DLL) will look for language-specific word breaker DLLs and use them when they are available.

When constructing a language-specific word breaker, you must provide a DLL whose name follows the pattern FTLXxxxx.DLL, where xxxx is the four-digit hexadecimal representation of the language identifier. Thus a word breaker DLL for Japanese would be named FTLX0411.DLL.

Your DLL must export two interfaces — **FTSWordBreakW** and **FTSWordBreakA**.

{button ,AL("WordBreaker")} Related Topics

AnimatorProc

void AnimatorProc(void);

Periodically called by the FTS engine while processing requests to add entries to an index. This program-supplied function should carry out an appropriate action to notify the user that the construction of the index is progressing. Uses standard C calling conventions.

Return Value

None.

Comments

This function should avoid doing a lot of processing during each call. One good approach is to use a variable together with calls to **GetTickCount** to keep track of the amount of time that has elapsed since the last animation action.

{button ,AL("RegisterAnimator")} Related Topics

ClearDisplayText

ERRORCODE ClearDisplayText(HHILITER hHiliter);

Removes any text previously passed to the hiliter.

Parameter	Description
hcmp	Handle of the hiliter. Must have been previously created using the **NewHiliter** function.

Return Value

Returns either zero or a negative error code value.

{button ,AL("ScanDisplayText;CountHilites;QueryHilites;DeleteHiliter;NewHiliter")} Related Topics

CompressText

INT CompressText(HCOMPRESSOR hcmp, PBYTE pbText, UINT cbText, PBYTE *ppbCompressed, UINT iCharset);

Compresses the given text using the phrase table associated with the given compressor.

Parameter	Description
hcmp	Handle of the compressor. Must have been previously created using the **NewCompressor** function.
pbText	Address of the buffer containing the text to compress. That text will be overwritten whenever the compressed text is smaller than the uncompressed text.
cbText	Size of uncompressed text in bytes.
ppbCompressed	Address of a variable that will receive the address of a malloc buffer containing compressed text. That variable will not be set when the compressed text is smaller than the uncompressed text. This parameter may be NULL.
iCharset	Character set of the text to be compressed. If this parameter is USE_DEFAULT, the function uses the default character set specified when the compressor was created.

Return Value

Returns either a negative error code value or the size in bytes of the compressed form of the input text.

If this value is positive and less than cbText, the compressed form will be written into the pbText area. Otherwise if the result value is greater than or equal to cbText and ppbCompressed is non-NULL, the compressed form will be stored in a malloc'd buffer and the address of that buffer will be stored in *ppbCompressed.

{button ,AL("DecompressText;NewCompressor")} Related Topics

CountHilites

INT CountHilites(HHILITER hHiliter, INT base, INT limit);

Used to determine the number of hilites in the specified region of the scanned text.

Parameter	Description
hHiliter	Handle of the hiliter. Must have been previously created using the **NewHiliter** function.
base	Byte offset to the beginning of the region to be queried.
limit	Byte offset just beyond the end of the region to be queried. Use -1 to indicate the end of all the display text.

Return Value

Returns the number of hilites which overlap the designated region. That is, the count may include hilited words and phrases which fall partially outside the region.

{button ,AL("QueryHilites")} Related Topics

DecompressText

INT DecompressText(HCOMPRESSOR hcmp, PBYTE pbCompressed, UINT cbCompressed, PBYTE pbText);

Expands the given compressed text using the phrase table associated with the given compressor.

Parameter	Description
hcmp	Handle of the compressor. Must have been previously created using the **NewCompressor** function.
pbCompressed	Address of the buffer containing the compressed text.
cbCompressed	Size in bytes of the compressed text.
pbText	Address of the buffer that receives the expanded text. This buffer must large enough to hold the uncompressed form of the text.

Return Value

Returns the size in bytes of the expanded text if successful and a negative error code value otherwise.

{button ,AL("CompressText")} Related Topics

DeleteCompressor

ERRORCODE DeleteCompressor(HCOMPRESSOR hcmp);

Deletes the given compressor.

Parameter	Description
hcmp	Handle of the compressor. Must have been previously created using the **NewCompressor** function.

Return Value

Returns 0 if successful; an error value otherwise.

{button ,AL("NewCompressor;fts_errorcode")} Related Topics

DeleteHiliter

ErrorCode DeleteHiliter(HHILITER hHiliter);

Deletes the given hiliter.

Parameter	Description
hcmp	Handle of the hiliter. Must have been previously created using the **NewHiliter** function.

Return Value

Returns 0 if successful; an error value otherwise.

{button ,AL("NewHiliter;fts_errorcode")} Related Topics

DeleteIndex

ERRORCODE DeleteIndex(HINDEX hinx);

Deletes the given index, freeing any resources associated with it.

Parameter	Description
hinx	Handle of the index to delete. Must have been previously created using the **NewIndex** function.

Return Value

Returns 0 if successful; an error value otherwise.

Comments

If the given index is to be used in subsequent searches, save it using the **SaveIndex** function before deleting it.

{button ,AL("NewIndex;SaveIndex;fts_errorcode")} Related Topics

DeleteSearcher

ERRORCODE DeleteSearcher(HSEARCHER hsrch);

Deletes the given searcher, freeing any resources associated with it.

Parameter	Description
hsrch	Handle of the searcher to delete. Must have been previously created using the **NewSearcher** function.

Return Value

Returns 0 if successful; an error value otherwise.

{button ,AL("NewSearcher;fts_errorcode")} Related Topics

DiscardIndex

ERRORCODE DiscardIndex(HSEARCHER hsrch, INT iIndex);

Discards the given index, freeing any resources associated with it.

Parameter	Description
hsrch	Handle of the searcher to delete. Must have been previously created using the **NewSearcher** function.
iIndex	Index identifier. Must have been previously returned by the **OpenIndex** function.

Return Value

Returns 0 if successful; an error value otherwise.

{button ,AL("NewSearcher;OpenIndex;fts_errorcode")} Related Topics

FTSWordBreakA

int APIENTRY FTSWordBreakA (CP wCP, LPSTR *ppbText, LPINT pcbText, LPSTR *papbToken, LPSTR *papbTokenEnd, LPBYTE paType, PUINT paHash, int cwTokens, UINT fTokenizeSpaces);

Scans a sequence of non-Unicode characters and constructs information about where words and punctuation streams occur. It exactly parallels **FTSWordBreakW** except that its input text is composed of 8-bit values with characters being represented by either one or two 8-bit values. This kind of text is called multibyte text.

Parameter	Description
wCP	Code page for the text.
ppbText	Address of a LPWSTR variable which in turn points to the multibyte text which is to be scanned. On exit that variable will be adjusted to point just beyond the text for which token information has been constructed.
pcbText	Address of a variable which defines the number characters remaining in the multibyte text. On exit that variable will be adjusted to count the number of characters not yet processed.
papbToken	Address of an array of LPWSTR items. That array contains slots for cwTokens items. On exit it will contain pointers to the character positions in which tokens start. This pointer may be NULL.
papbTokenEnd	Address of an array of LPWSTR items. That array contains slots for cwTokens items. On exit it will contain pointers to the character positions just beyond the ends of tokens. This pointer may be NULL.
paType	Address of an array of BYTE items. That array contains slots for cwTokens items. On exit each byte value will indicate whether the corresponding token is a word, a number, or punctuation. A zero value denotes punctuation while words and numbers are denoted by 0x02 and 0x01 respectively. This pointer may be NULL.

paHash	Address of an array of UINT items. That array contains slots for cwTokens items. On exit each element will contain a hash value for the corresponding token. This pointer may be NULL.
	Hash values are to be constructed as follows:
	for (pw= pwToken, cw= cwToken, uHash= 0; cw--; ++pw)
	uHash = _rotl(uHash, 5) - *pw;

cwTokens	Defines the number of array slots in papwToken, papwTokenEnd, paType, and paHash.
fTokenizeSpaces	Set of bit flags which determine how tokens are constructed. Can be a combination of these values:

Value	Meaning
TOKENIZE_SPACES (0x01)	When this bit flag is on, a single space character imbedded between two words will be shown as a token. When the flag is off, it will be ignored.
REMOVE_SPACE_CHARS (0x02)	When this bit flag is on, all white space characters will be discarded when punctuation tokens are constructed. This means that the *ppwText buffer will be altered. If a punctuation sequence consists of nothing but white space characters, no token will be constructed.
STARTING_IMBEDS (0x04)	When this flag is on, a punctuation character which may be imbedded in a word is allowed to start a word or a number. An example of such a character is the English decimal point (“.”).

Return Value

Returns the number of tokens for which information has been constructed.

{button ,AL("FTSWordBreakW")} Related Topics

FTSWordBreakW

int APIENTRY FTSWordBreakW (LPWSTR *ppwText, LPINT pcwText, LPWSTR *papwToken, LPWSTR *papwTokenEnd, LPBYTE paType, PUINT paHash, int cTokens, UINT fTokenizeSpaces);

This function scans a sequence of Unicode characters and constructs information about where words and punctuation tokens occur. That information will be copied into a collection of arrays passed to **FTWWordBreakW**.

Parameter	Description
ppwText	Address of a LPWSTR variable which in turn points to the Unicode text which is to be scanned. On exit that variable will be adjusted to point just beyond the text for which token information has been constructed.
pcwText	Address of a variable which defines the number characters remaining in the Unicode text. On exit that variable will be adjusted to count the number of characters not yet processed.
papwToken	Address of an array of LPWSTR items. That array contains slots for cwTokens items. On exit it will contain pointers to the character positions in which tokens start. This pointer may be NULL.
papwTokenEnd	Address of an array of LPWSTR items. That array contains slots for cwTokens items. On exit it will contain pointers to the character positions just beyond the ends of tokens. This pointer may be NULL.
paType	Address of an array of BYTE items. That array contains slots for cwTokens items. On exit each byte value will indicate whether the corresponding token is a word, a number, or punctuation. A zero value denotes punctuation while words and numbers are denoted by 0x02 and 0x01 respectively. This pointer may be NULL.

paHash	Points to an array of UINT items. That array contains slots for cwTokens items. on exit each element will contain a hash value for the corresponding token. This pointer may be NULL.
	Hash values are to be constructed as follows:
	for (pw= pwToken, cw= cwToken, uHash= 0; cw--; ++pw)
	uHash = _rotl(uHash, 5) - *pw;

cwTokens	Defines the number of array slots in papwToken, papwTokenEnd, paType, and paHash.
fTokenizeSpaces	Set of bit flags which determine how tokens are constructed. Can be a combination of these values:

Value	Meaning
TOKENIZE_SPACES (0x01)	When this bit flag is on, a single space character imbedded between two words will be shown as a token. When the flag is off, it will be ignored.
REMOVE_SPACE_CHARS (0x02)	When this bit flag is on, all white space characters will be discarded when punctuation tokens are constructed. This means that the *ppwText buffer will be altered. If a punctuation sequence consists of nothing but white space characters, no token will be constructed.
STARTING_IMBEDS (0x04)	When this flag is on, a punctuation character which may be imbedded in a word is allowed to start a word or a number. An example of such a character is the English decimal point (“.”).

Return Value

Returns the number of tokens for which information has been constructed.

{button ,AL("FTSWordBreakA")} Related Topics

GetPhraseTable

ERRORCODE GetPhraseTable(HCOMPRESSOR hcmp, PUINT pcPhrases, PBYTE *ppbImages, PUINT pcbImages, PBYTE *ppacbImageCompressed, PUINT pcbCompressed);

Retrieves the phrase table for the given compressor.

Parameter	Description
hcmp	Handle of the compressor. Must have been previously created using the **NewCompressor** function.
pcPhrases	Address of the variable that receives the count of phrases in the table.
ppbImages	Address of a variable which will be set to the address of a buffer containing the phrase images. That buffer is allocated with malloc. Assume that the caller will free the buffer.
pcbImages	Address of the variable that receives the size in bytes of the *ppbImages data.
ppacbImageCompressed	Address of a variable which will be set to the address of a buffer containing the phrase size information. That buffer is allocated with malloc. Assume that the caller will free the buffer.
pcbCompressed	Address of the variable that receives the size in bytes of the *ppacbImageCompressed data.

Return Value

Returns 0 if successful; an error value otherwise.

{button ,AL("NewCompressor;fts_errorcode")} Related Topics

IsValidIndex

BOOL IsValidIndex(PSZ pszFileName, UINT dwOptions);

Verifies that the given index file exists and supports searches of the given type.

Parameter	Description
pszFileName	Address of a null-terminated name specifying the index file.
dwOptions	Search type. Can be one of these values:

Value	Meaning
TOPIC_SEARCH	Can search for topics containing a given word or combination of words. If no other option is given, this option is required.
PHRASE_SEARCH	Can search for topics containing a given phrase or combination of phrases.
PHRASE_FEEDBACK	Can provide feedback about the validity of a phrase before carrying out a search. As the user constructs a phrase, the full-text engine can list the words that the user can type next to create a valid phrase. This option is valid only if PHRASE_SEARCH is also given.
VECTOR_SEARCH	Can search for topics that are similar. Similarity is based on the use and frequency of words in the topics. This option is valid only if both PHRASE_SEARCH and PHRASE_FEEDBACK are also given.

Return Value

Returns TRUE if the given search type is supported; FALSE otherwise.

{button ,AL("NewIndex")} Related Topics

LoadGroup

ERRORCODE LoadGroup(HSEARCHER hsrch, PSZ pszFileName);

Loads the given group and associates it with the given searcher.

Parameter	Description
hsrch	Handle of the searcher. Must have been previously created using the **NewSearcher** function.
pszFileName	Address of a null-terminated string specifying the name of the file containing the group. The file is assumed to have been previously saved using the **SaveGroup** function.

Return Value

Returns 0 if successful; an error value otherwise.

{button ,AL("NewSearcher;SaveGroup;fts_errorcode")} Related Topics

NewCompressor

HCOMPRESSOR NewCompressor(UINT iCharsetDefault);

Creates a phrase compressor. The compressor can subsequently be used to compress text.

Parameter	Description
iCharsetDefault	Default character set index for compression. Can be one of these values:

Value
ANSI_CHARSET

Return Value

Returns the handle of the new compressor if successful; NULL otherwise.

{button ,AL("DeleteCompressor")} Related Topics

NewHiliter

HHILITER NewHiliter(HSEARCHER hSearcher);

Creates a hiliter. The hiliter can subsequently be used to find search target words and phrases within a topic.

Parameter	Description
hSearcher	Handle to a previously created searcher.:

Return Value

Returns the handle of the new compressor if successful; NULL otherwise. For NULL results you can call GetLastError to retrieve an error code value.

{button ,AL("DeleteHiliter")} Related Topics

NewIndex

HINDEX NewIndex(const PBYTE pbSourceName, UINT uiTime1, UINT uiTime2, UINT iCharsetDefault, UINT lcidDefault, UINT fdwOptions);

Creates a full-text-search index that supports the types of searches specified by the fdwOptions parameters. Entries can be added to the index by using the **ScanTopicTitle** and **ScanTopicText** functions.

Parameter	Description
pbSourceName	Address of a null-terminated string specifying the name of the file (or text source) for which the index file is being created.
uiTime1	Low 32-bits of the 64-bit value identifying the creation date and time of the index.
uiTime2	High 32-bits of the 64-bit value identifying the creation date and time of the index.
iCharsetDefault	Default character set index for compression. Can be one of these values:

Value
ANSI_CHARSET

lcidDefault	Default locale for the index. Can be any valid locale identifier.
fdwOptions	Type of index to create or action to take. Can be a combination of these values:

Value	Meaning
TOPIC_SEARCH	Can search for topics containing a given word or combination of words. If no other option is given, this option is required.
PHRASE_SEARCH	Can search for topics containing a given phrase or combination of phrases.
PHRASE_FEEDBACK	Can provide feedback about the validity of a phrase before carrying out a search. As the user constructs a phrase, the full-text engine can list the words that the user can type next to create a valid phrase. This option is valid only if PHRASE_SEARCH is also given.
VECTOR_SEARCH	Can search for topics that are similar. Similarity is based on the use and frequency of words in the topics. This option is valid only if both PHRASE_SEARCH and PHRASE_FEEDBACK are also given.

Comments

The following values are define how topics are displayed:

Parameter	Description
WINHELP_INDEX	Not used.
USE_VA_ADDR	Send MSG_FTS_JUMP_VA message to get a topic displayed.
USE_QWORD_JUMP	Send MSG_FTS_JUMP_QWORD message to get a topic displayed.

If neither USE_VA_ADDR nor USE_QWORD_JUMP is part of fdwOptions, the MSG_FTS_JUMP_HASH message will be used to display topics.

Return Value

Returns the handle of the index if successful; NULL otherwise.

{button ,AL("ScanTopicTitle;ScanTopicText;MSG_FTS_JUMP;MSG_FTS_JUMP_QWORD")} Related Topics

NewSearcher

HSEARCHER NewSearcher();

Creates a new searcher. The searcher can be associated with one or more indexes and used to search for words and phrases in the source file associated with the indexes.

Return Value

Returns the handle of the searcher if successful; NULL otherwise.

{button ,AL("OpenIndex;OpenDialog")} Related Topics

OpenDialog

HWND OpenDialog(HSEARCHER hsrch, HWND hwndParent);

Opens a dialog box for the given searcher. The dialog box prompts users for words and phrases to search for. Before you call **OpenDialog** you must load indices by calling either **LoadGroup** or **OpenIndex**.

The new dialog box will be created invisible and disabled.

Parameter	Description
hsrch	Handle of the searcher to associate with the dialog box. Must have been previously created using the **NewSearcher** function.
hwndParent	Handle of the parent window for the dialog box. If NULL, the desktop is assumed to be the parent.

Return Value

Returns the handle of the dialog box if successful; NULL otherwise.

{button ,AL("NewSearcher")} Related Topics

OpenIndex

INT OpenIndex(HSEARCHER hsrch, PSZ pszIndexFileName, PBYTE pbSourceName, PUINT pcbSourceNameLimit, PUINT pTime1, PUINT pTime2);

Opens the given index file for use by the given searcher. It will optionally verify that the supplied *pbSourceName, *pTime1, and *pTime2 values match those stored in the index file.

Parameter	Description
hsrch	Handle of the searcher to associate with the dialog box. Must have been previously created using the **NewSearcher** function.
pszIndexFileName	Address of a null-terminated string that specifies the name of the index file to open. The index file is assumed to have been previously created using the **SaveIndex** function.
pbSourceName	Address of the buffer that receives the name of the source files associated with the index. This is the same as the source name associated with the index when created. If on entry this pointer refers to a non-null string, that string will be compared against the source name stored in the index file. The OpenIndex operation will fail if the strings do not compare equal. The comparison, when performed, will also examine *pcbSourceNameLimit.
pcbSourceNameLimit	Address of the variable that receives the size in bytes of the source name. On entry that variable is assumed to contain the size of the pbSourceName buffer. If that size is smaller than the original source name, the value copied to pbSourceName will be truncated accordingly.
pTime1	Address of the variable that receives the low 32 bits of the 64-bit time value associated with the index. If on entry this points to a non-zero value, that value will be compared against the corresponding value in the index file, and the OpenIndex operation will fail if the two values are unequal.
pTime2	Address of the variable that receives the high 32 bits of the 64-bit time value associated with the index. If on entry this points to a non-zero value, that value will be compared against the corresponding value in the index file, and the OpenIndex operation will fail if the two values are unequal.

Return Value

Returns a non-negative integer identifying the index file if successful and a negative error value otherwise.

{button ,AL("NewSearcher;SaveIndex;fts_errorcode")} Related Topics

QueryHilites

INT QueryHilites(HHILITER hHiliter, INT base, INT limit cHilites, HILITE* paHilites);

Retrieves hilite information for the specified region of the scanned text. The buffer pointed to should be preallocated with the appropriate number of slots of type **HILITE.**

The result includes hilited words/phrases which fall partially inside the designated region.

If you want to call hilites sequentially, you can use the following loop:

```
while (QueryHilites(hHiliter, base, limit, 1, pHilite))
{
// TODO -- your processing code
base = pHilite->limit;
}
```

Parameter	Description
hHiliter	Handle of the hiliter to query. Must have been previously created using the **NewHiliter** function.
base	Byte offset to the beginning of a display text region to query.
limit	Byte offset to just beyond the end of a display text reqion. You may use -1 to indicate the end of the entire display text.
paHilites	Address of a vector of **HILITE** items. The hilite results will be copied to this vector. It is presumed to be large enough to contain all the hilites for the region. You can find out how many hilites will be returned by calling **CountHiltes** first.

Return Value

Returns the number of hilites if successful; an error value otherwise.

{button ,AL("QueryHilites;CountHilites;ScanDisplayText;ClearDisplayText;fts_errorcode")} Related Topics

QueryOptions

ERRORCODE QueryOptions(HSEARCHER hsrch, INT iIndex, PUINT pfdwOptions);

Indicates the type of searching supported by the given index.

Parameter	Description
hsrch	Handle of the searcher to associate with the dialog box. Must have been previously created using the **NewSearcher** function.
iIndex	Integer identifying the index to query. Must have been previously returned by the **OpenIndex** function.
pfdwOptions	Address of the variable that receives one or more of these search type values:

Value	Meaning
TOPIC_SEARCH	Can search for topics containing a given word or combination of words. If no other option is given, this option is required.
PHRASE_SEARCH	Can search for topics containing a given phrase or combination of phrases.
PHRASE_FEEDBACK	Can provide feedback about the validity of a phrase before carrying out a search. As the user constructs a phrase, the full-text engine can list the words that the user can type next to create a valid phrase. This option is valid only if PHRASE_SEARCH is also given.
VECTOR_SEARCH	Can search for topics that are similar. Similarity is based on the use and frequency of words in the topics. This option is valid only if both PHRASE_SEARCH and PHRASE_FEEDBACK are also given.

Return Value

Returns 0 if successful; an error value otherwise.

{button ,AL("NewSearcher;OpenIndex;fts_errorcode")} Related Topics

RegisterAnimator

ERRORCODE RegisterAnimator(ANIMATOR pAnimator, HWND hwndAnimator);

Registers the given animator procedure and associates it with the given window.

Parameter	Description
pAnimator	Address of an **AnimatorProc**, a program-supplied callback function that carries out the animation.
hwndAnimator	Handle of the window associated with the **AnimatorProc**.

Comments

When an **AnimatorProc** is registered, the index construction code will call it frequently. The purpose of an **AnimatorProc** is to drive an animation or a progress indicator while an index is being constructed. Your **AnimatorProc** should be designed to avoid consuming large amounts of processor time. One good strategy is to use a global variable together with calls to **GetTickCount** to keep track of how much time has elapsed since the last animation action.

Return Value

Returns 0 if successful; an error value otherwise.

{button ,AL("AnimatorProc;GetTickCount;fts_errorcode")} Related Topics

SaveGroup

ERRORCODE SaveGroup(HSEARCHER hsrch, PSZ pszFileName);

Saves information about the indexes which have been opened by the given searcher in the given file. That group of indices can be loaded and associated with a searcher later by using the **LoadGroup** function.

Parameter	Description
hsrch	Handle of the searcher to associate with the dialog box. Must have been previously created using the **NewSearcher** function.
pszFileName	Address of a null-terminated string specifying the name of the file to receive the group information.

Return Value

Returns 0 if successful; an error value otherwise.

{button ,AL("LoadGroup;NewSearcher;fts_errorcode")} Related Topics

SaveIndex

ERRORCODE SaveIndex(HINDEX hinx, PSZ pszFileName);

Saves a new index by copying it to the given file.

Parameter	Description
hinx	Handle of an index. Must have been previously created using the **NewIndex** function.
pszFileName	Address of a null-terminated string specifying the name of the index file to create.

Return Value

Returns 0 if successful; an error value otherwise.

{button ,AL("NewIndex;fts_errorcode")} Related Topics

ScanDisplayText

ERRORCODE ScanDisplayText(HHILITER hHiliter, BYTE* pbText, int cbText, UINT iCharset, LCID lcid);

This function supplies the display text for which hilite information is to be computed. It may be called repeatedly to pass across the relevant display text. This is necessary when the text is composed of segments with different Charsets and/or Locales.

Parameter	Description
hHiliter	Handle of the hiliter. Must have been previously created using the **NewHiliter** function.
pbText	Address of the buffer containing the text to be scanned.
cbText	Size in bytes of the text.
iCharset	Character set of the text to be compressed. If this parameter is USE_DEFAULT, the function uses the default character set specified when the compressor was created.
lcid	Locale identifier of the text to scan. If this parameter is USE_DEFAULT, the function uses the default locale specified when the index was created.

Return Value

Returns 0 if successful; an error value otherwise.

{button ,AL("fts_errorcode;ClearDisplayText;CountHilites;QueryHilites ")} Related Topics

ScanText

ERRORCODE ScanText(HCOMPRESSOR hcmp, PBYTE pbText, UINT cbText, UINT iCharset);

Scans the given text to gather statistics about word usage. Those statistics are used to generate the phrase table returned by **GetPhraseTable** and used by **CompressText** and **DecompressText**.

Parameter	Description
hcmp	Handle of the compressor. Must have been previously created using the **NewCompressor** function.
pbText	Address of the buffer containing the text to be scanned.
cbText	Size in bytes of the text.
iCharset	Character set of the text to be compressed. If this parameter is USE_DEFAULT, the function uses the default character set specified when the compressor was created.

Return Value

Returns 0 if successful; an error value otherwise.

{button ,AL("fts_errorcode;GetPhraseTable;CompressText;DecompressText;NewCompressor")} Related Topics

ScanTopicText

ERRORCODE ScanTopicText(HINDEX hinx, PBYTE pbText, UINT cbText, UINT iCharset, UINT lcid);

Adds information about the words or phrases in *pbText to the given index, associating them with the current topic.

Parameter	Description
hinx	Handle of an index. Must have been previously created using the **NewIndex** function.
pbText	Address of the buffer containing the text to scan. The text can be any combination of words and punctuation.
cbText	Size in bytes of the text to scan.
iCharset	Character set of the text to scan. If this parameter is USE_DEFAULT, the function uses the default character set specified when the index was created.
lcid	Locale identifier of the text to scan. If this parameter is USE_DEFAULT, the function uses the default locale specified when the index was created.

Return Value

Returns 0 if successful; an error value otherwise.

{button ,AL("ScanTopicTitle;NewIndex;fts_errorcode")} Related Topics

ScanTopicTitle

ERRORCODE ScanTopicTitle(HINDEX hinx, PBYTE pbTitle, UINT cbTitle, UINT iTopic, HANDLE hTopic, UINT iCharset, UINT lcid);

Begins a new topic and defines a title and handle values for the topic. The title text is internally passed to **ScanTopicText**, and is considered part of the topic text during search transactions.

Subsequent calls to **ScanTopicText** are associated with this topic until another call to **ScanTopicTitle** or a call to **SaveIndex**.

Parameter	Description
hinx	Handle of an index to add the topic to. Must have been previously created using the **NewIndex** function.
pbTitle	Address of the buffer containing the title of the topic. The text can be any combination of words and punctuation.
cbTitle	Size in bytes of the title.
iTopic	The high 32 bits of a 64-bit value which will identify this topic. This value is relevant when the USE_QWORD_JUMP option has been set at index creation time.
hTopic	The low 32 bits of a 64-bit value which will identify this topic.
iCharset	Character set of the title text. If this parameter is USE_DEFAULT, the function uses the default character set specified when the index was created.
lcid	Locale identifier of the title text. If this parameter is USE_DEFAULT, the function uses the default locale specified when the index was created.

Return Value

Returns 0 if successful; an error value otherwise.

{button ,AL("ScanTopicText;SaveIndex;NewIndex;fts_errorcode")} Related Topics

SetPhraseTable

ERRORCODE SetPhraseTable(HCOMPRESSOR hcmp, PBYTE pbImages, UINT cbImages, BYTE pacbImageCompressed, UINT cbCompressed);

Sets the phrase table for the given compressor. Use this function to set a phrase table that you previously retrieved with the **GetPhraseTable** function.

Parameter	Description
hcmp	Handle of the compressor to add the table to. Must have been previously created using the **NewCompressor** function.
pbImages	Address of the buffer containing the phrase images.
cbImages	Count of phrase images.
pacbImageCompressed	Address of the buffer containing information about the phrase sizes.
cbCompressed	Size in bytes of the compressed image data.

Return Value

Returns 0 if successful; an error value otherwise.

{button ,AL("GetPhraseTable;NewCompressor;fts_errorcode")} Related Topics

SetDirectoryLocator

void SetDirectoryLocator(HWND hwndLocator);

Defines the window which will respond to MSG_FTS_WHERE_IS_IT messages.

Parameter	Description
hwndLocator	Handle of an index to add the topic to. Must have been previously created using the **NewIndex** function.

Return Value

None.

{button ,KL("NewIndex;MSG_FTS_WHERE_IS_IT")} Related Topics

Stemmer

void APIENTRY Stemmer(LPTSTR pWordToStem, WORD cCharsInWordToStem);

Maps a word to the smallest related concept name.

Parameter	Description
pWordToStem	Address of the null-terminated word string which is to be stemmed. The string is a Unicode string where each character is represented by a 16-bit value, and the terminating null is a 16-bit zero. The stemmed result will be written into this memory area.
cCharsInWordToStem	Count in bytes of the length of the word to be stemmed. This count does not include the trailing null character.

Return Value

None.

{button ,AL("FTS_NLS")} Related Topics

MSG_FTS_GET_TITLE

MSG_FTS_GET_TITLE
wParam = (WPARAM)(UINT)index;
lParam = (LPARAM)(const PBYTE *)ppszTitle;

Sent to the program when the title for the given topic is needed.

Parameter	Description
index	Value of wParam. Integer value uniquely identifying an index. This value is returned by **OpenIndex**.
ppszTitle	Value of lParam. Address of the variable which will be set to point to the title string for the designated index. If the index is invalid or not title is available, *ppszTitle should be set to NULL.

Return Value

Return zero if the program processes the message.

{button ,AL("ScanTopicTitle;OpenIndex")} Related Topics

MSG_FTS_JUMP_HASH

MSG_FTS_JUMP_HASH
wParam = (WPARAM)(UINT)index;
lParam = (WPARAM)(UINT)HashValue;

Sent to the program when the topic associated with the given lParam value needs to be displayed. (The Windows Help engine interprets this value as a hash value. Other programs should interpret the value as appropriate.)

Parameter	Description
index	Value of wParam. Integer value uniquely identifying an index. This value is returned by **OpenIndex** when the index was added to the searcher. If have loaded a collection of indices by calling **LoadGroup**, you must keep track of the index numbers in place when the group was saved by the **SaveGroup** function.
HashValue	Value of lParam. Integer value associated with the topic when the topic was added to the index. (Corresponds to the hTopic parameter in the **ScanTopicTitle** function.)

Return Value

Return zero if the program processes the message.

{button ,AL("ScanTopicTitle;OpenIndex;LoadGroup;SaveGroup")} Related Topics

MSG_FTS_JUMP_QWORD

MSG_FTS_JUMP_QWORD
wParam = (WPARAM)(UINT)index;
lParam = (LPARAM)(PQWordAddress)pQWA;

Sent to the program when the topic associated with a given quadword address needs to be displayed. This message is sent if and only if the index was created using the USE_QWORD_JUMP option.

Parameter	Description
index	Value of wParam. Integer value uniquely identifying an index. This value is returned by **OpenIndex** when the index was added to the searcher. If have loaded a collection of indices by calling **LoadGroup**, you must keep track of the index numbers in place when the group was saved by the **SaveGroup** function.
pQWA	Value of lParam. Address of a **QWordAddress** structure associated with the topic when the topic was added to the index. (Corresponds to the hTopic parameter in the **ScanTopicTitle** function.)

Return Value

Return zero if the program processes the message.

{button ,AL("ScanTopicTitle;OpenIndex;LoadGroup;SaveGroup;QWordAddress")} Related Topics

MSG_FTS_JUMP_VA

MSG_FTS_JUMP_VA
wParam = (WPARAM)(UINT)index;
lParam = (LPARAM)(UINT)VirtualAddress;

Sent to the program when the topic associated with a given virtual address needs to be displayed. This message is sent if and only if the index was created using the USE_VA_ADDR value.

Parameter	Description
index	Value of wParam. Integer value uniquely identifying the index which sent this message. This is the value is returned by **OpenIndex** when the index was added to the searcher. If you have loaded a collection of indices by calling **LoadGroup**, you must keep track of the index numbers in place when the group was saved by the **SaveGroup** function.
pQWA	Value of lParam. Address of a **QWordAddress** structure. This is a 64-bit value associated with the topic when the topic was added to the index. (Corresponds to the iTopic and hTopic parameters in the **ScanTopicTitle** function.)

Return Value

Return zero if the program processes the message.

{button ,AL("ScanTopicTitle;OpenIndex;LoadGroup;SaveGroup;QWordAddress")} Related Topics

MSG_FTS_WHERE_IS_IT

MSG_FTS_WHERE_IS_IT
wParam = (WPARAM)(UINT)fStartEnumeration;
lParam = (LPARAM)(LPTSTR)pszFile;

Sent to the locator window when a file cannot be opened or created, or when the file content is invalid. This message is a request for an alternate directory location for that file.

Parameter	Description
fStartEnumeration	Value of wParam. Indicates whether the location enumeration is starting or continuing. A value of TRUE indicates that the first location in the enumeration sequence should be returned. A value of FALSE indicates that the next location in the enumeration sequence should be returned.
pszFile	Value of lParam. Address of a file name buffer. On entry this buffer will contain the filename which was attempted. On exit the path portion of the filename should be adjusted to refer to the next candidate location for opening or creating a file.

Return Value

Return zero if the program processes the message.

{button ,AL("ScanTopicTitle")} Related Topics

MSG_REINDEX_REQUEST

MSG_REINDEX_REQUEST

Sent to the program when the “Rebuild” button is pressed. The program should use the **NewIndex** and related functions to create a new indices.

Return Value

Return zero if the program processes the message.

{button ,AL("NewIndex")} Related Topics

HILITE

```
typedef struct _HILITE {
    INT base;
    INT limit;
} HILITE, *PHILITE;
```

This structure is used with the QueryHilites function. It defines byte span within a display text region that contain a word or phrase targeted by the searcher associated with a particular hiliter.

{button ,AL("CountHiltes;QueryHilites")} Related Topics

QWordAddress

```
typedef struct _QWordAddress {
    UINT    iSerial;
    HANDLE hTopic;
} QWordAddress, *PQWordAddress;
```

This structure is used with the MSG_FTS_JUMP_QWORD message. It contains two 32-bit values associated with a topic by the **ScanTopicTitle** function.

{button ,AL("ScanTopicTitle;MSG_FTS_JUMP_QWORD")} Related Topics

ERRORCODE Values

ALIGNMENT_ERROR	Invalid phrase table passed to **SetPhraseTable**
ALREADY_WEIGHED	Cannot call **ScanString** after **GetPhraseTable**, **CompressString**, or **DecompressString**.
CANNOT_LOAD	An index file could not be opened or had a structure error, or was wrong version.
CANNOT_OPEN	Could not open an index file or a group file.
CANNOT_SAVE	Invalid file name given to **SaveIndex** or **SaveGroup**.
DIALOG_ALREADY_ACTIVE	Cannot call **OpenIndex** after **OpenDialog**.
DISK_READ_ERROR	A read from a disk file failed.
DISK_WRITE_ERROR	A write to a disk file failed.
EMPTY_PHRASE_TABLE	Cannot call **OpenIndex** after **LoadGroup**.
GROUP_LOADED_ALREADY	Cannot load more than one group at a time.
INDEX_LOADED_ALREADY	Cannot **LoadGroup** again after an index or group has been loaded..
INVALID_INDEX	Index identifier is not valid in call to **DiscardIndex** or **QueryOptions**.
INVALID_LCID	Locale identifier is not valid.
INVALID_PHRASE_TABLE	Format of phrase table is not valid.
NO_INDICES_LOADED	**SaveGroup** called with no indices loaded.
NO_TEXT_SCANNED	No calls to **ScanText** or **SetPhraseTable** before a call to **GetPhraseTable**, **CompressString**, or **DecompressString**.
NO_TITLE	**ScanTopicText** called before **ScanTopicTitle**, or no topics scanned at **SaveIndex** time.
NOT_COMPRESSOR	Compressor handle is not valid.
NOT_INDEXER	Index handle is not valid.
NOT_SEARCHER	Searcher handle is not valid.
OUT_OF_DISK	A write operation failed for lack of disk space.
OUT_OF_MEMORY	A virtual memory allocation failed.
SEARCH_ABORTED	A search has been canceled.
SYSTEM_ERROR	A programming error has been detected.
UNKNOWN_EXCEPTION	An unanticipated exceptional condition has occurred. This probably indicates a programming error.

Writing DLLs for WinHelp

A dynamic-link library (DLL) is an executable module containing functions that Windows-based programs (like WinHelp) can call to perform useful tasks. WinHelp accesses DLLs in two ways:

- Through DLL functions registered as Help macros in a project (.hpj) file. These functions can then be used in hotspots and macro footnotes in topic files.
- Through embedded window (**{ew**x**}**) references in topic files.

If WinHelp's internal macro set doesn't provide all the functionality you need for a Help file, you can write your own DLLs to add extensions to WinHelp. You can extend WinHelp by providing custom DLLs containing author-defined WinHelp macros and by providing DLL access through embedded-window references.

When creating DLLs for embedded windows, you must follow all the standard design requirements for Windows DLLs. These requirements are described in the Microsoft Win32 Software Development Kit (SDK).

To create DLLs for WinHelp, you need the Microsoft Win32 SDK and the Microsoft C/C++ development system or greater, or the Microsoft QuickC graphical development environment.

Note

- If your version 4.0 Help file calls a 16-bit DLL, you will not be able to use WinHelp version 4.0 on Windows NT (WINHLP32.EXE) to display the Help file.

{button ,AL(`ewx;HELPDLL')} Related Topics

Registering DLL Functions as Help Macros

You can use the **RegisterRoutine** macro to register any DLL function as a WinHelp macro. When registering a DLL function, you provide Help with the following information:

- DLL filename.
- Function name.
- Data type returned by the function.
- Number and type of function parameters.

To register the DLL function, you enter a **RegisterRoutine** macro in the **[CONFIG]** section of the Help project file. (If you don't register the DLL function in the **[CONFIG]** section, you must register it another way before using the function.) The **RegisterRoutine** macros are run when the Help file is opened, so the registered functions are available during the entire Help session. You must register a DLL routine before using it, or the Help compiler will report an error when it encounters the unregistered macro in the RTF source files, and the macro will not work when run in the built Help file.

To determine the data type of the function's parameters, consult the application programming interface (API) documentation for the DLL, or ask the person who developed the DLL. For information on parameter types of Windows functions, see the Win32s SDK.

{button ,AL(`DLL_WRITE_OVERVIEW;RegisterRoutine')} Related Topics

How WinHelp Locates DLLs

When running custom DLLs using the **RegisterRoutine** macro, WinHelp loads the DLL only when it is needed by the Help file. To load a DLL , Help must be able to find it on the user's system. When preparing to use a .DLL, Help looks in the following locations, in the following order:

- The folder of the current Help file (used in inter-file jumps and when loading Help dynamic link libraries).
- The Windows SYSTEM folder.
- The HELP folder in the Windows folder.
- The Windows folder.
- The folder specified in WINHELP.INI, if any.
- The PATH environment.

If Help cannot find the DLL after searching in all these locations, it displays an error message.

To increase the likelihood that Help will locate the DLL quickly, you should also observe the following guidelines:

- Use unique names for all DLLs accessed by the Help file.
- When installing your program on a user's hard disk drive, your setup program should copy all custom DLLs to the folder where Help is located.
- If your product is distributed on CD-ROM, copy WINHELP.EXE and any custom DLLs to the user's hard disk drive.
- Define a WINHELP.INI entry for each custom DLL that your Help file is using so that Help knows where to locate them.

{button ,AL(`DLL_WRITE_OVERVIEW')} Related Topics

WinHelp Internal Variables

In general, after you register a DLL function as a Help macro, you can specify WinHelp internal variables as parameters to that function when the function appears in hotspots or macro footnotes.

In this section, you'll learn about other WinHelp internal variables that you can pass to DLL functions registered as WinHelp macros. This section also examines in detail one variable that controls how the DLL handles WinHelp errors.

List of Variables

You can use any of the following WinHelp internal variables in DLL functions.

Variable	Format Spec	Description
hwndApp	U	32-bit handle to the main Help window. This variable is guaranteed to be valid only while the function is running.
hwndContext	U	Handle to the current active window (either the main Help window or a secondary window).
qchPath	S	Fully qualified path of the currently open Help file.
qError	S	Long pointer to a structure containing information about the most recent WinHelp error.
lTopicNo	U	Topic number. This number is relative to the order of topics in the topic files used to build the Help file.
hfs	U	Handle to the file system for the currently open Help file.
coForeground	U	Current foreground color.
coBackground	U	Current background color.

{button ,AL(`DLL_WRITE_OVERVIEW')} Related Topics

Error Handling

The qError internal variable points to a structure containing information about the most recent WinHelp error. The error structure is defined as follows:

```
struct
{     WORD   fwFlags;
      WORD   wError;
      char   rgchError[wMACRO_ERROR];
} QME;
```

fwFlags

The fwFlags field contains flags indicating how WinHelp responds to errors. The following are possible error flags and their values.

Flag	Value	Description
fwMERR_ABORT	0x0001	Allows the Abort option. This flag is set by default.
fwMERR_CONTINUE	0x0002	Allows the Continue option.
fwMERR_RETRY	0x0004	Allows the Retry option.

wError

The wError field is a number indicating the type of error that occurred. The following are possible error numbers and their values.

Error	Number	Description
wMERR_NONE	0	No error (initial value)
wMERR_MEMORY	1	Out of memory (local)
wMERR_PARAM	2	Invalid parameter passed
wMERR_FILE	3	Invalid file parameter
wMERR_ERROR	4	General Help macro error
wMERR_MESSAGE	5	Help macro error with message

rgchError

If the wError field is wMERR_MESSAGE, the rgchError field contains the error message that WinHelp displays.

{button ,AL(`DLL_WRITE_OVERVIEW')} Related Topics

Notifying DLLs of WinHelp Events

You might want your DLL to receive notification of WinHelp events (for example, when the user selects a jump or changes input focus to a program other than WinHelp). To do this, you add a **LDLLHandler** function to the DLL. The **LDLLHandler** function has the job of processing messages sent from WinHelp to the DLL.

When WinHelp loads a custom DLL, it looks for a function named **LDLLHandler**. If WinHelp finds **LDLLHandler**, it calls **LDLLHandler**.

The **LDLLHandler** function is illustrated in the following example:

```
PUBLIC      LONG PASCAL EXPORT LDLLHandler(
    WORD       wMsg,
    LONG       lParam1,
    LONG       lParam2)
    {
        switch(wMsg) {
            case DW_WHATMSG:
                return DC_INITTERM | DC_JUMP;
            case DW_INIT:
                return TRUE;
            case DW_TERM:
                return TRUE;
            case DW_ACTIVATE:
                return TRUE;
            case DW_CHGFILE:
                return TRUE;
    }
    return FALSE;}
```

In this definition, wMsg identifies the message WinHelp has sent to the DLL. The two LONG parameters are passed with the message and depend on the message type.

When the user first performs an action that requires the custom DLL, WinHelp calls **LDLLHandler** with the wMsg parameter set to DW_WHATMSG. This message asks the DLL what types of WinHelp messages it wants to receive. **LDLLHandler** should return one or more of the flags in the following table with this information. WinHelp uses the flags returned by **LDLLHandler** to determine which messages the DLL wants to receive in the **LDLLHandler** function.

LDLLHandler

returns	**WinHelp sends**	**Description**
DC_MINMAX	DW_MINMAX, DW_SIZE	Minimize, maximize, or resize WinHelp
DC_INITTERM	DW_INIT, DW_TERM	Initialize or terminate DLL
DC_JUMP	DW_STARTJUMP, DW_ENDJUMP, DW_CHGFILE	Jump or change .HLP file
DC_ACTIVATE	DW_ACTIVATE	Give WinHelp or another program input focus
DC_CALLBACKS	DW_CALLBACKS	Give DLL access to WinHelp entry points

These flags are combined using the standard C bitwise-OR (|) operator. For example, many DLLs request notification when WinHelp is about to initialize or terminate them. The sample **LDLLHandler** function above requests this notification and asks to be notified when the user has moved the input focus to or from WinHelp. It processes the DW_WHATMSG message as follows:

```
case DW_WHATMSG:
    return DC_INITTERM | DC_ACTIVATE;
```

{button ,AL(`DLL_WRITE_OVERVIEW')} Related Topics

DC_INITTERM Flag

The DC_INITTERM flag tells WinHelp that the DLL should receive initialization and termination messages.

DW_INIT

After **LDLLHandler** returns a DC_INITTERM flag, WinHelp sends a DW_INIT message. This message tells **LDLLHandler** to perform any required initialization operations (such as initializing variables or loading strings). The lParam1 and lParam2 parameters are not used.

If **LDLLHandler** returns FALSE, WinHelp unloads the DLL and stops sending messages. If **LDLLHandler** returns TRUE, the DLL remains loaded.

DW_TERM

When the user ends a WinHelp session, WinHelp sends a DW_TERM message. This message tells **LDLLHandler** to perform any required cleanup operations before the DLL is unloaded from memory. The lParam1 and lParam2 parameters are not used, and the return value has no meaning.

DC_MINMAX Flag

The DC_MINMAX flag tells WinHelp that the DLL should receive messages when the user minimizes, maximizes, or resizes the WinHelp window.

DW_MINMAX

WinHelp sends the DLL a DW_MINMAX message if the user minimizes or maximizes the context window.

The lParam1 parameter to DW_MINMAX indicates which operation was performed: 1L for minimized or 2L for maximized.

The lParam2 parameter is not used.

If **LDLLHandler** returns TRUE for this message, WinHelp continues sending this message throughout the remainder of the session; if **LDLLHandler** returns FALSE, WinHelp stops sending it.

DW_SIZE

WinHelp sends the DLL a DW_SIZE message if the user resizes the context window.

The lParam1 message specifies the horizontal size of the window in the low-order word and the vertical size in the high-order word.

The lParam2 parameter specifies the handle of the current window.

If **LDLLHandler** returns TRUE for this message, WinHelp continues sending this message throughout the remainder of the session; if **LDLLHandler** returns FALSE, WinHelp stops sending it.

DC_JUMP Flag

The DC_JUMP flag tells WinHelp that the DLL should receive messages when the user runs a jump in the Help file.

DW_STARTJUMP

WinHelp sends the DLL a DW_STARTJUMP message after the user runs a jump.

The lParam1 parameter to DW_STARTJUMP is not used. The lParam2 parameter specifies the handle of the current parent window.

If **LDLLHandler** returns TRUE for this message, WinHelp continues sending this message throughout the remainder of the session; if **LDLLHandler** returns FALSE, WinHelp stops sending it.

DW_ENDJUMP

WinHelp sends the DLL a DW_ENDJUMP message after it displays the jump-destination topic.

The lParam1 parameter specifies the byte offset of that topic within the Help file's file system. This value can be passed to the **LSeekHf** function (defined in the DLL.H file) to perform a seek to that topic within the file.

The lParam2 parameter specifies the position of the scroll box in the topic. This value can be used in any of the standard Windows scrolling functions that accept scroll-position parameters.

If **LDLLHandler** returns TRUE for this message, WinHelp continues sending this message throughout the remainder of the session; if **LDLLHandler** returns FALSE, WinHelp stops sending it.

DW_CHGFILE

WinHelp sends the DLL a DW_CHGFILE message if the user selects Open from the File menu to change .HLP files or jumps to a topic in a different .HLP file. If the message is the result of a jump to a new file, WinHelp sends the DW_CHGFILE message between the DW_STARTJUMP and DW_ENDJUMP messages.

The lParam1 parameter specifies a long pointer to the .HLP filename. The lParam2 parameter is not used.

If **LDLLHandler** returns TRUE for this message, WinHelp continues sending this message throughout the remainder of the session; if **LDLLHandler** returns FALSE, WinHelp stops sending it.

DC_ACTIVATE Flag

The DC_ACTIVATE flag tells WinHelp that the DLL should receive a DW_ACTIVATE message when the user gives either WinHelp or another program the input focus.

DW_ACTIVATE

The lParam1 parameter to the DW_ACTIVATE message specifies whether WinHelp received (nonzero LONG value) or lost (0L) the input focus. The lParam2 parameter is not used.

If **LDLLHandler** returns TRUE for this message, WinHelp continues sending this message throughout the remainder of the session; if **LDLLHandler** returns FALSE, WinHelp stops sending it.

Calling WinHelp Internal Functions

WinHelp provides DLL authors with access to 16 of its internal functions. These functions perform operations in the internal .HLP file system, get global information about the currently open Help file, or display information in a standard WinHelp dialog box.

In this section, you'll learn how to access six of these internal functions. An overview of the mechanism is as follows:

? WinHelp sends a DW_WHATMSG message to the DLL.
? The **LDLLHandler** function in the DLL processes the DW_WHATMSG message by returning a DC_CALLBACKS flag.
? WinHelp sends a DW_CALLBACKS message specifying a pointer to an array of WinHelp internal functions.
? The **LDLLHandler** function uses the pointer passed in the DW_CALLBACKS message to obtain pointers to the WinHelp functions it will use.

{button ,AL(`DLL_WRITE_OVERVIEW')} Related Topics

Accessing WinHelp Functions (DW_CALLBACKS)

If a DLL wants access to WinHelp internal functions, its **LDLLHandler** function handles the DW_WHATMSG message by returning a DC_CALLBACKS flag. When it gets this flag, WinHelp sends a DW_CALLBACKS message to the DLL. The lParam1 parameter is a long pointer to an array containing pointers to each of the 16 internal WinHelp functions. The DLL.H file defines symbolic names for indexing each function in the array.

In processing the DW_CALLBACKS message, the **LDLLHandler** function calls a function named **GetCallBacks** to specify which functions it wants to access. **GetCallBacks** is defined as follows in the example:

```
PUBLIC      BOOL PASCAL EXPORT GetCallBacks(
VPTR        VPtr,
LONG        lVersion)
{

    // hfs level:
    lpfn_HfsOpenSz          = VPtr[HE_HfsOpenSz];
    lpfn_RcCloseHfs         = VPtr[HE_RcCloseHfs];
    lpfn_RcLLInfoFromHfs = VPtr[HE_RcLLInfoFromHfs];

    // bag level routines
    lpfn_FAccessHfs =   VPtr[HE_FAccessHfs];
    lpfn_HfOpenHfs   =   VPtr[HE_HfOpenHfs];
    lpfn_LcbReadHf   =   (LPFN_LCBREADHF) VPtr[HE_LcbReadHf];
    lpfn_RcCloseHf   =   VPtr[HE_RcCloseHf];
    return TRUE;
}
```

The VPtr parameter in **GetCallBacks** is the lParam1 pointer passed by the DW_CALLBACKS message from WinHelp. The **GetCallBacks** function gets pointers to the following WinHelp internal functions, which it uses later in the example.

Function	What it does
HfsOpenSz	Opens the .HLP file system (the compound file containing the files internal to the Help file).
RcCloseHfs	Closes the open .HLP file system.
RcLLInfoFromHfs	Maps a handle to the open .HLP file system (returned by HfsOpenSz) to low-level file information.
FAccessHfs	Determines whether a file within the .HLP file system is accessible.
HfOpenHfs	Opens a file within the .HLP file system.
LcbReadHf	Reads bytes from a file within the .HLP file system.

Note

This sample code does not take multiple instances of WinHelp into account. If more than one instance is started, the DLL receives different pointers to these callback functions—one pointer for each instance. You must keep track of which pointer is associated with each WinHelp instance. One way to do this is to create an array associating the task with the callback pointer.

{button ,AL(`DLL_WRITE_OVERVIEW')} Related Topics

Writing DLLs for Embedded Windows

In WinHelp version 4.0 you can create embedded windows to extend the functionality of WinHelp by placing objects in a fixed-size window under the control of a DLL. For example, you can create a DLL to display animation sequences or to play audio segments in an embedded window. However, in version 4.0 you cannot create an embedded window that functions as a hotspot. This section explains how to write custom DLLs for WinHelp.

To create an embedded window, authors insert an embedded window reference, **{ew**x**}**, in the RTF source file.

WinHelp displays embedded windows within topic windows. To WinHelp, an embedded window is simply a child window of that topic window. Users cannot minimize, maximize, or resize an embedded window. Embedded windows cannot be used as hotspots. However, you can include hotspots in an embedded window if they are controlled by the DLL, but users will not be able to use the keyboard equivalents to access the hotspots. That is because embedded windows cannot receive the input focus, which means they cannot process keystrokes. So, you should not place anything in an embedded window that requires keyboard input from users.

WinHelp positions the embedded window using the justification character (left, right, or character) specified by the author in the embedded window reference. The embedded window DLL is expected to display the information in the window and resize the window appropriately. Help expects the window size to remain fixed as long as the topic is displayed. The window element and DLL determine the size and content of the embedded window, and Help arranges the other elements of the topic around the embedded window.

WinHelp displays embedded windows only when necessary — while the topic containing the embedded window is being displayed. However, an embedded window may exist while it is not being displayed (if the user scrolls the topic past the embedded window, for example). Because it is part of a specific topic, an embedded window goes away when the user displays a different topic.

{button ,AL(`DLL_WRITE_OVERVIEW;ewx')} Related Topics

Initialization of Embedded Windows

WinHelp creates embedded windows with window style WS_CHILD and a default size. It initializes the DLL for an embedded window before it displays the window.

When WinHelp creates the embedded window, it passes two strings in the WM_CREATE message. The lparam parameter points to a CREATESTRUCT structure, and the lpCreateParams field of the CREATESTRUCT structure points to a second structure defined as follows:

```
typedef  struct     tagCreateInfo
         {
         short                  idMajVersion;
         short                  idMinVersion;
         LPSTR                  lpstrFileName;
         LPSTR                  lpstrAuthorData;
         HANDLE                 hfs;
         DWORD                  coForeground;
         DWORD                  coBackground;
         }    CREATEINFO;
```

The fields in the structure are defined in the following table.

Field	Use
idMajVersion, idMinVersion	Identifies the version of WinHelp (and the version of the CREATESTRUCT structure used for embedded windows in that version of WinHelp). Currently, these fields are 0. If the idMinVersion field is a value other than 0, additional interfaces may be available to DLLs, but all the interfaces described in this chapter are still supported. The DLL should always verify that the value in the idMajVersion and idMinVersion fields of the CREATESTRUCT structure are 0. If they are not, the DLL might not work with that version of WinHelp.
lpstrFileName	Points to the fully qualified name of the .HLP file containing the embedded window. This field will be NULL if the data is not available. If the DLL uses the string contained in this field, it should make a copy of the string.
lpstrAuthorData	Points to the author-data parameter in the **{ewl}**, **{ewc}**, or **{ewr}** reference from the source RTF file. This field will be NULL if the data is not available. If the DLL uses the string contained in this field, it should make a copy of the string.
hfs	Specifies a handle to the file system for the .HLP file.
coForeground,coBackground	Specify the foreground and background colors of the main Help window. If the embedded window uses the same colors as the main Help window, the DLL can use the color values in these fields to set those colors in the embedded window.

{button ,AL(`DLL_WRITE_OVERVIEW')} Related Topics

Embedded Window Behavior

While a topic is being displayed, WinHelp creates embedded windows when it needs to lay them out and destroys them when they are no longer needed. The window can be created and destroyed, then created again if necessary, while WinHelp tries to lay out the embedded window object. If you load information, you might want to do so during WM_SHOWWINDOW processing so that you don't have to load this information twice.

An embedded window may exist while it is not being displayed (for example, if the topic requires scrolling and the embedded window resides in the portion not currently displayed in the Help window). Do not set the window style to WS_VISIBLE or call the **ShowWindow** function during WM_CREATE processing. WinHelp displays the window when necessary.

As an embedded window DLL processes a WM_CREATE message, it is expected to set up any window styles the window uses and resize the window appropriately.

WinHelp positions the window using the justification character (left, right, or character) authored into the RTF file. WinHelp expects the window size to remain fixed during the window's lifetime.

Because it is a child window of the main Help window. an embedded window does not need to destroy itself; it will be destroyed when the parent window is destroyed.

{button ,AL(`DLL_WRITE_OVERVIEW')} Related Topics

Message Processing for Embedded Windows

Embedded windows receive most standard Windows messages, including mouse and WM_PAINT messages. Embedded windows should not take the input focus; therefore, they do not need to process keystrokes.

Embedded window DLLs must also process the following messages, which are specifically defined for use with embedded windows:

- EWM_RENDER (0x706A)
- EWM_QUERYSIZE (0x706B)
- EWM_ASKPALETTE (0x706C)

{button ,AL(`DLL_WRITE_OVERVIEW')} Related Topics

EWM_RENDER

WinHelp sends the EWM_RENDER message to an embedded window to get information about the image in the window. It uses this information when printing the image or placing it in the Clipboard. The wParam parameter to EWM_RENDER indicates the type of information returned by the message, and the LOWORD of the return value contains a handle to this information. Possible wParam values and return values are shown in the following table.

wParam	Return Value
CF_TEXT	Sharable global handle to a null-terminated ASCII string. WinHelp frees this handle.
CF_BITMAP	Handle to a bitmap. WinHelp removes this handle.

EWM_QUERYSIZE

WinHelp sends the EWM_QUERYSIZE message to an embedded window to obtain the size of the window. The wParam parameter is a handle to the device context for the window. The lParam parameter points to a POINT structure. The embedded window DLL should fill in the **x** field of the structure with the window's height and the **y** field of the structure with the window's width. WinHelp uses this information when laying out the topic in the topic window, when printing the topic, or when placing it in the Clipboard.

EWM_ASKPALETTE

WinHelp sends the EWM_ASKPALETTE message to all embedded windows displayed within a topic to get information about the palettes used in each window. The wParam and lParam parameters are not used.

When an embedded window receives an EWM_ASKPALETTE message, it should return a handle to the palette that it uses. WinHelp then selects the most appropriate palette for use in that topic. Usually, this palette is the one used for the first embedded window in the currently displayed part of the topic.

Before the DLL repaints an embedded window, it should send an EWM_ASKPALETTE message to the parent window to determine which palette to use. The following code fragment illustrates how to do this:

```
hpal = (HPALETTE)SendMessage(GetParent(hwnd), EWM_ASKPALETTE, 0, 0L)
```

Help RTF tokens

Help RTF tokens are an extended subset of tokens defined by the Microsoft Rich Text Format (RTF) standard. The RTF tokens specify character and paragraph properties, such as font, color, spacing, and alignment, for text and graphics in the Help file. The following topics describe the subset of RTF tokens recognized by Help Workshop. Use this information if you plan to edit RTF files directly or create an RTF reader/writer that converts files to RTF format.

{button ,AL("RTF1")} Related Topics

Rich-Text Format

The rich-text format (RTF) standard is a method of encoding formatted text and graphics for easy transfer between different programs and different operations. Generally, it is used by all Microsoft Word programs — Word for Windows, Word for the Macintosh, and Word for MS-DOS — for moving word-processing documents between different platforms without having to rely on special translation software or conversion utilities. Because the RTF standard provides a format for text and graphics interchange that can be used with different output devices and operating systems, Help Workshop also supports this standard. That means you can use almost any text editor that generates RTF output, including your own custom RTF editor, to create the source files that are built into Help files.

Software that takes a formatted file and turns it into an RTF file is referred to as an RTF "writer." Software that translates an RTF file into a formatted file is referred to as an RTF "reader." An RTF writer separates the program's control information from the actual text and writes a new file containing the text and RTF groups associated with that text. An RTF reader does the converse of this operation.

Elements of RTF

Help RTF tokens are presented to Help Workshop in topic files, which are specified in the [FILES] section of a Help project (.HPJ) file. To Help Workshop a topic file consists of the following elements:

- RTF tokens
- Control symbols
- Groups
- Unformatted text

RTF tokens, control symbols, and braces constitute control information. Text grouping is used to define the format and placement of text and graphics in the Help file. All other characters in RTF text constitute plain text.

{button ,AL("element1;FILES")} Related Topics

RTF token Syntax

An RTFtoken is a formatted tag that specifies a particular kind of information in the RTF file. An RTF token consists of a backslash (\) followed by an RTF token name and a delimiter:

\statement-name<delimiter>

Statments

RTF tokens always begin with a backslash (\). The statement-name identifies the RTF token and specifies what it does. No spaces or other characters may separate the backslash and the statement-name.

An RTF token may optionally have a number parameter immediately after the name. This number must be in the range of a signed integer (-32767 through 32767). No spaces or other characters may separate the name and the numeric parameter (if any).

Certain statements control properties (such as bold and italic) that have only two states. When the RTF token has no parameter or has a non-zero parameter, the statement is used to turn the property on. When the RTF token has a 0 (zero) parameter, the statement is used to turn the property off. For example, **\b** turns on bold, whereas **\b0** turns off bold.

Delimiters

Delimiters are mandatory characters that separate the statement text of a parameter from subsequent text. A delimiter can be one of the following:

- A space. The space is considered part of the statement.
- A digit or a minus sign (-), which indicates that a numeric parameter follows. If a numeric parameter immediately follows the statement name, this parameter becomes part of the statement.
- Any character other than a letter or digit. The delimiting character terminates the statement but is not considered part of the statement. A "letter" is an upper- or lowercase ASCII letter.

When a space is used as a delimiter, Help Workshop discards it. If any other character is used, Help Workshop processes it as text or as the start of another RTF token. For example, if a backslash is used as a delimiter, Help Workshop interprets it as the beginning of the next RTF token.

For example, the following line demonstrates usage of the **\tab** statement and a space delimiter:

text in paragraph\tab more text

Control Symbols

A control symbol consists of a backslash (\) followed by a single non-letter. Control symbols require no further delimiting. The following control symbols are supported:

Control Symbol	Meaning
|	Formula character
\~	Nonbreaking space
\-	Optional hyphen
_	Nonbreaking hyphen
\:	Subentry in an index entry
\'hh	Hexadecimal value of a character in the current character set, where h is a hexadecimal digit in the range 0 through F.

Groups

A group consists of Help RTF tokens and text enclosed in braces (**{ }**). The opening brace **{** indicates the start of the group, and the closing brace **}** indicates the end of the group, as shown in this example:

```
{       group start, and
}       group end.
```

Different groups perform different functions in the RTF file. Some groups specify information such as the fonts, styles, colors, summary information, and document-format attributes used in the file. Other groups simply include text and the RTF tokens that define attributes for that text. Still other groups specify picture data, footnotes, bookmarks, annotations, and section- and paragraph-formatting attributes for the file.

Formatting specified within a group affects only the text within that group. Text within a group inherits any formatting of the text preceding the group.

Destinations

Some statements, referred to as destinations, mark the beginning of a collection of related text. A destination is marked by a control word that signals the beginning of the group. An example of a destination is the group where the Help-specific text follows the statement. Destination statements and their following text must be enclosed in braces, as in this example:

```
        {\footnote main_contents}
```

Unformatted Text

Unformatted text consists of any combination of 7-bit ASCII characters. Although characters whose values are greater than 127 are not permitted in topic files, the \' statement can be used to insert them in the final Help file. Help Workshop treats spaces as part of the text, but it discards carriage return and linefeed characters.

Because the backslash character (\) and braces ({ }) have specific meanings in RTF, they must be preceded with a backslash if you want to use them as plain text, as shown by the control symbols \\, **\{**, and **\}**.

RTF Semantics

When reading a stream of RTF text, an RTF reader must accomplish the following tasks:

1 Separate RTF control information from plain text.

2 Act on control information.

This is designed to be a relatively simple process, as described in the next section.

Some control information just contributes special characters to the plain text stream. Other information changes the program state, which includes document properties as a whole and a stack of group states that apply to parts of the document. The group state is saved by the opening **{** brace and is restored by the closing **}** brace.

The current group state specifies:

- The destination or part of the document that the plain text is building up.
- The section formatting properties.
- The paragraph formatting properties.
- The character formatting properties.

3 Collect and properly dispose of any remaining plain text as directed by the current group state.

{button ,AL("semantics")} Related Topics

Acting on Control Information

When parsing the RTF stream, the RTF reader follows this process:

1 Reads the next character.

2 If the character is an opening brace (char == '{'), stores the current state on the stack.

Or if the current state does not change, it then continues.

3 If the character is a closing brace (char == '}'), retrieves the current state from the stack.

Generally, this will change the state.

4 If the character is a backslash (char == '\'), collects the RTF token or control symbol and its parameter.

a. Looks up the statement or symbol in the symbol table (a constant table) and acts according to the description found there. The parameter is left available for use by the action.

b. Leaves a read pointer before or after the delimiter, as appropriate.

c. After completing the action, continues.

5 If the character is anything other than **{**, **}**, or \, writes the plain text character to the current destination using the current formatting properties.

6 Reads the next character.

Acting on Symbol Table Entries

When looking up RTF tokens and symbols in the symbol table, the RTF reader can take any of the following actions:

- Change the destination to the destination described in the table entry.

  Most destination changes are legal only immediately after an opening brace. Other restrictions may also apply (for example, footnotes may not be nested).

- Change formatting property.

  The symbol table entry will describe the property and whether the parameter is required.

- Insert a special character.

  The symbol table entry will describe the character code.

- End of paragraph.

  This can also be viewed as just a special character.

- End of section.

  This can also be viewed as just a special character.

- Ignore the character.

The RTF File

This section describes the different groups that make up an RTF file. It first presents a sample RTF file, which is used in examples throughout the section. It then describes the required and optional components of an RTF file.

{button ,AL("groups1")} Related Topics

Sample RTF File

The following is an example of a simple but complete, single-topic RTF file that can be compiled by Help Workshop:

```
{\rtf1\ansi \deff0\deflang1024

{\fonttbl
{\f0\froman Times New Roman;}
{\f1\froman Symbol;}
{\f2\fswiss Arial;}
{\f3\froman MS Serif;}
{\f4\fswiss MS Sans Serif;}
}

{\colortbl;
\red0\green0\blue0;
\red0\green0\blue255;
\red0\green255\blue0;
\red255\green0\blue0;
\red255\green255\blue255;

#{\footnote graphics_cont}
${\footnote Graphics}

\pard\plain \li120\sb340\sa120\sl-320 \f3\fs28 Graphics

\par \pard\plain \li120 \f4\fs20 Use graphic images to illustrate concepts and present
information visually. Graphics include line art, icons, screen shots of the interface,
and graphics with hotspots ("hypergraphics").
\par
}
```

As noted earlier, the entire RTF file is one group, so the file begins and ends with braces (**{}**). Other groups nested within the group containing the file define the header and text information.

Required Entries

The following entries must appear in the order shown immediately after the opening brace of the RTF file:

- The RTF token **\rtf**n. This RTF token identifies the version (given by the number n) of the RTF standard used in the file. For Help Workshop, this RTF token must be **\rtf1**.
- An RTF token identifying the character set used in the file. The RTF character set described in this appendix corresponds to the **\ansi** character set statement.

System default values, such as font number (**\deff**n statement), follow the character set statement. The **\deff**n statement specifies which font defined in the font table (see the next section) is the default font for the file.

The required entries appear as follows in the sample RTF file:

```
{\rtf1\ansi\deff0
```

Font Table

The first group listed after the opening statements is the font table group. All the fonts that the RTF file uses must be declared in the font table. These fonts should correspond to the fonts that are present on the user's computer.

The font table group has the following form:

{\fonttbl
{\ffont-number**\f**font-family font-name**;}**
{\ffont-number**\f**font-family font-name**;}**
.
.
.
}

The actual font table is the group beginning with the RTF token **\fonttbl**. Each font is defined as a separate group within the font-table group. Each font definition has three components: the font-number, the font-family, and the font-name.

The font-number is an integer that identifies the font.

The font-family is one of the following standard font families defined by the Windows operating system:

Control Word	Font Family	Examples
fnil	Default or unknown	
froman	Roman	MS Serif and Palatino
fswiss	Swiss	Helvetica, MS Sans Serif
fmodern	Modern	Courier, Elite, Pica
fscript	Script	Cursive, Script, Zapf Chancery
fdecor	Decorative	Old English, Zapf Dingbats
ftech	Technical	Symbol

Each font definition must end with a semicolon (;) immediately before the closing brace.

If the RTF file uses a default font, the RTF token **\deff**font-number appears before the font-table group. The font-number is an integer indicating which font in the table is used for the default.

Example

The following group defines nine fonts:

```
{\fonttbl
{\f0\froman Times New Roman;}
{\f1\froman Symbol;}
{\f2\fswiss Arial;}
{\f3\froman MS Serif;}
{\f4\fswiss MS Sans Serif;}
{\f5\fmodern Courier;}
{\f6\fdecor Zapf Dingbats;}
{\f7\fswiss Helvetica Condensed;}
{\f8\fswiss Helvetica;}
}
```

This example defines the following fonts:

Font #	Font Family	Font
0	Roman	Times New Roman
1	Technical	Symbol
2	Swiss	Arial
3	Roman	MS Serif
4	Swiss	MS Sans Serif
5	Modern	Courier
6	Decorative	Zapf Dingbats

7	Swiss	Helvetica Condensed
8	Swiss	Helvetica

Color Table

If the Help file uses color, the RTF file must also define a color table group to specify the individual colors. The color table group has the following form:

```
{\colortbl;
\rednumber\greennumber\bluenumber;
\rednumber\greennumber\bluenumber;
.
.
.
}
```

Each definition specifies a value in the range 0 through 255 for the red, green, and blue components of the color. Unlike font entries in the font table, color entries in the color table do not specify a color number. Instead, Help Workshop assumes that the first color defined is color zero, the second color is color one, and so on.

Example

The following example defines the standard 16 colors in the Windows palette:

```
{\colortbl;
\red0\green0\blue0;
\red0\green0\blue255;
\red0\green255\blue255;
\red0\green255\blue0;
\red255\green0\blue255;
\red255\green0\blue0;
\red255\green255\blue0;
\red255\green255\blue255;
\red0\green0\blue127;
\red0\green127\blue127;
\red0\green127\blue0;
\red127\green0\blue127;
\red127\green0\blue0;
\red127\green127\blue0;
\red127\green127\blue127;
\red192\green192\blue192;
}
```

Style Sheet

In the standard RTF specification, the style sheet group defines the different styles defined for a document. In Word for Windows, a style is a named combination of formats (including those for characters, paragraphs, tabs, borders, and frames) that authors can apply to text using a defined sequence of keystrokes. Authors can add styles to a document template.

Help Workshop ignores the style sheet group in an RTF file. If you plan to use the RTF file only to create a Help file, you can omit the style sheet group from the file to save space. However, if you also plan to print documents from the RTF file, you can include a style sheet group to define styles for the document. Then, when you open the RTF file in Word for Windows, you can apply the styles that you have defined to text.

A style sheet group has the following format:

{\stylesheet
{\sstyle-num\style-specs style-name**;}**
{\sstyle-num\style-specs style-name**;}**
.
.
.
}

Each style definition appears in a group within the style sheet group. The style definition is identified by a unique style number.

The exact style specifications follow the style number. The style specifications can include RTF tokens for any type of character, text, tab, border, or frame formatting.

The last part of the style definition is the style name. This is the name that appears in the Word for Windows drop-down list boxes that list the document's styles. A single space must separate the style specifications from the style name.

Each style definition must end with a semicolon (**;**) immediately before the closing brace.

In Word for Windows, styles may be based on previously defined styles. In this case, the style inherits the attributes of the previously defined style. A style may also define a style to be applied to the next paragraph in the document. The following RTF tokens may appear in the style sheet to activate these features:

Control Word	Meaning
\sbasedonstyle-num	The style defined in this group inherits the attributes of the style with the given style number. If this style does not inherit attributes from any style, style-num is 0.
\snextstyle-num	The style with the given style number is applied to the next paragraph in the document. If no style is to be applied to the next paragraph in the document, style-num is 0.

Example

Here is a portion of a sample style sheet:

```
{\stylesheet
{\s229\li2160\ri720\tldot\tx8280\tqr\tx8640 \f5\fs20\lang1033 \sbasedon0\snext0 toc 4;}
.
.
.
{\s17\qc\sb240\sl240\keepn \b\f5\fs28\lang1033 \sbasedon0\snext17 head;}
{\s18\qj\sb240\sl240 \b\f5\fs20\lang1033 \sbasedon0\snext18 Company;}
{\s19\qj\sb120\sl240\keepn \i\f5\fs20\lang1033 \sbasedon0\snext19 Product;}
}
```

This portion of the style sheet defines four styles numbered 229 (named "toc 4"), 17 (named "head"), 18 (named "Company"), and 19 (named "Product"). Style sheets 17 through 19 assume that the same style will be applied to the next paragraph as well as the paragraph to which the style is currently assigned. None of these styles inherits the attributes of a previously defined style.

Topic Information

After setting up the required RTF entries for the Help file, you define the topic information. Topics consist of plain text and graphics that the user sees, Help Workshop-specific information that is defined in topic footnotes, character and paragraph formatting information, and page break delimiters.

Font and Formatting Information

Before any text is placed in the RTF file, the font name and font size must be specified. The **\f**n statement specifies the font name (n matches the font number defined in the font table). The **\fs**n statement specifies the font size (n is given in half-points). For example, using the sample font table just described, the following example defines the text as 10-point MS Sans Serif:

```
\f4\fs20
```

If you want the text to have any special formatting characteristics, you must also define those before you write out the plain text. Help Workshop supports a number of character and paragraph formatting attributes that you can use to change the appearance and placement of text and graphics.

The following example defines the topic title text as 14-point MS Serif with 17 points of space before, 6 points of space after, and 16 points of leading. The topic title paragraph is also indented 6 points from the left margin:

```
\pard\plain \li120\sb340\sa120\sl-320 \f3\fs28
```

Help Features in Footnote Groups

The following types of Help information are specified in **\footnote** groups:

- Topic IDs
- Topic titles
- Browse sequences
- Keyword index entries
- Help macros run on topic entry
- Build tags
- Comments
- Window type

Help Features Embedded in Text

Some Help features are implemented using common RTF commands or special codes embedded within text. The following types of Help features can be embedded in text:

- Layout features like nonscrolling regions and nonwrapping text
- Hotspots for activating jumps, displaying pop-up windows, and running Help macros
- Bitmaps
- Embedded panes for custom DLL objects

Topic End

When there is more than one topic in an RTF file, each topic must end with a **\page** statement.

Ending the last topic in a file with a **\page** statement is optional.

{button ,AL("RTF_SAMPLE_TOPIC;topic_info",0,"",`tables')} Related Topics

Topic IDs

Topics within a Help file are usually identified by a unique topic ID. The following group specifies a topic ID for a topic:

#{\footnote topic ID**}**

A topic ID is any string of up to 255 characters. You can use any character except **#**, **=**, **>**, **@**, **!**, or **%**. The first character of a topic ID cannot be a numeric character.

For example, this RTF entry defines the topic ID "main contents":

#{\footnote main contents}

{button ,AL("TopicInfo",0,"",`main')} Related Topics

Topic Titles

Typically, a topic will also contain a topic title, which identifies the topic in Help dialog boxes. The following group specifies a topic title:

${\footnote title-text**}**

A title string is any string of up to 50 characters. Any printable ASCII character may appear in a topic title, except the following reserved characters: # = + > @ * % !

Topic titles must not begin with a number. ASCII characters such as braces ({ }), brackets ([ ]), and backslashes (\) that are used as special characters in RTF must be prefixed by backslashes.

This example defines the title "Saving a Document":

```
${\footnote Saving a Document}
```

{button ,AL("TopicInfo",0,"",`main')} Related Topics

Browse Sequences

A topic may belong to a single browse sequence, which is a group of topics the user can view in forward or backward sequence.

The following group assigns a topic to a browse sequence and to one or more topic groups:

+{\footnote [sequence-name][**:**sequence-number]**}**

The sequence-name is the name of the browse sequence to which the topic is assigned. It may be omitted if the Help file has only one browse sequence.

The sequence-name may be followed by a sequence-number, which indicates where in the sequence the topic appears. In browse sequences, topics appear in order of sequence numbers. If no sequence number appears, topics in the sequence appear in their physical order within the file. If the [FILES] section of the Help project (.HPJ) file lists more than one RTF file, topics from the first RTF file appear first in the browse sequence, then topics from the second RTF file, and so on.

A colon must immediately precede the sequence number, if it appears. If a sequence name also appears, a colon must separate the sequence name and sequence number.

{button ,AL("TopicInfo",0,"",`main')} Related Topics

Keyword Entries

There are three kinds of keyword entries: K-keywords, A-keywords, and multi-index keywords. Each type of entry has its own topic footnote.

K-keywords

K-keywords, denoted by the custom footnote mark 'K,' are used for searches with the Index button. The Index tab in the Help Topics dialog box displays a list of the keywords defined for topics in the Help file. From the Index, the user can jump to any topic in which a keyword has been defined. K-keywords are also used by the KLink macro to provide inter-topic jumps to the topics having the keywords specified as the macro's parameters.

Multi-index keywords

Multi-index keywords are denoted by any alpha character except 'A' and 'K.' They have identical index capabilities to K-keywords, but are intended for alternate indexes.

A-keywords

A-keywords, denoted by the custom footnote mark 'A,' are used for **ALink** macro jumps only. They do not appear in any index.

Examples

The following group assigns one or more K-keywords to a topic:

K{\footnote keyword**;**[keyword**;**]...**}**

The following group assigns one or more A-keywords to a topic:

A{\footnote keyword**;**[keyword**;**]...**}**

The following group assigns one or more multi-index keywords to a topic for an index referenced to R-keywords:

R{\footnote keyword**;**[keyword**;**]...**}**

Keywords must be separated by semicolons (**;**).

{button ,AL("TopicInfo",0,"",`main')} Related Topics

Help Macros Run on Topic Entry

Help can run commands automatically when the user jumps to a topic (from a hotspot, browse button, or search list).

The following group specifies macros to be run on topic entry:

!{\footnote macro-string**}**

The macro-string can be any Help macro documented in this authoring guide or registered in the **[CONFIG]** section of the Help project (.HPJ) file.

{button ,AL("TopicInfo",0,"",`main')} Related Topics

Build Tags

Build tags are labels used to exclude certain topics from a build. Topics without build tags are always included in the build. Topics with build tags are included or excluded using a build expression in the Help project (.HPJ) file.

The following group assigns one or more build tags to a topic:

***{\footnote** tag-name**;**[tag-name**;**]...**}**

Multiple build tags must be separated by semicolons (**;**).

Help Workshop supports two very different ways of processing build tags. If you use Help Workshop to specify the build tags in topics to include or exclude, then by default, any topic with a build tag will be excluded from the help file. If, however, you use another editor to add/edit the BUILD option and [BUILDTAGS] section, then all topics are included by default, whether or not they have a build tag.

{button ,AL("TopicInfo",0,"",`main')} Related Topics

Comments

Comments are any text that you want to include with a topic for your own purposes. Comments are ignored by Help Workshop.

The following group assigns a comment to a topic:

@{\footnote comment-text**}**

Because comments are ignored, the comment-text can have as many standard characters as you want, including accented characters and spaces.

{button ,AL("TopicInfo",0,"",`main')} Related Topics

Window Type

Window types are used when you want to display the topic in a window other than the default window when it is accessed from the Index.

The following group assigns a window type to a topic:

>{\footnote windowname**}**

{button ,AL("TopicInfo",0,"",`main')} Related Topics

Layout Features

The following standard RTF commands are used to specify layout features in Help files:

Control Word	Meaning
\keepn	Makes affected text and pictures part of the nonscrolling region
\keep	Makes affected text and pictures nonwrapping so they are truncated instead of wrapped if the window is resized

{button ,AL("TopicInfo",0,"",`main')} Related Topics

Hotspots

A hotspot is a region within a topic that performs an action when the user selects the region with the mouse. This action may be jumping to another topic, displaying a topic in a pop-up window, or running a Help macro.

The following group is used to create pop-up hotspots in a topic:

{\ul hotspot text**}{\v** [**%**|*****]topic-ID**}**

The hotspot text is the topic text that performs the action when the user selects it. This text may be one of the standard **{bm**x**\}** RTF commands if the hotspot is a bitmap.

The percent sign (%) or asterisk (*) is used if you want to make the hotspot invisible (no green color, no underline) or just underlined (no green color).

The topic-ID identifies the topic to be displayed in the pop-up window.

The following groups are used to create jump or macro hotspots in a topic:

{\uldb hotspot text**}{\v** [**%**|*****]hotspot action**}**

Or, you can use this format:

{\strike hotspot text**}{\v** [**%**|*****]hotspot action**}**

The hotspot text is the topic text that performs the action when the user selects it. This text may be one of the standard **{bm**x**\}** RTF commands if the hotspot is a bitmap.

The percent sign (%) or asterisk (*) is used if you want to make the hotspot invisible (no green color, no underline) or just underlined (no green color).

The hotspot action may be one of the following:

- A jump destination. This action has the following form:

  topic-ID[**@**HLP-filename][>window-name]

  The topic-ID identifies the topic the hotspot jumps to. When present, the second parameter (preceded by an at sign) identifies a different Help file in which the jump destination appears. If the jump destination is to appear in a secondary window, >window-type identifies the name of window in which the topic is to appear. The window-type must be defined in the [WINDOWS] section of the Help project (.HPJ) file.

- A Help macro. This action has the following form:

  !macro-string

  The macro-string identifies the action Help performs when the user selects the hotspot. It can be any standard Help macro or any external DLL function registered in the **[CONFIG]** section of the project file.

If the hotspot text is invisible in the title, a percent sign (**%**) immediately precedes the hotspot action.

{button ,AL("TopicInfo",0,"",`main')} Related Topics

Bitmaps

Bitmaps can be inserted into a topic with a command that tells Help the name of the bitmap file and how to position it. This command has the following form:

\{bm[**c** | **l** | **r**] bitmapfile[**;** bitmapfile[**;**...]]]**\}**

The **\{bm**x**\}** commands specify how the bitmap is aligned in the topic, as shown in the following table:

Control Word	Meaning
\{bmc\}	Positions the bitmap using character alignment. The bitmap is handled as if it were another character and it appears at the position in the line indicated by the reference. Text follows the bitmap, positioned at the base of the bitmap.
\{bml\}	Positions the bitmap along the left margin, with text wrapping automatically along the right edge of the image.
\{bmr\}	Positions the bitmap along the right margin, with text wrapping automatically along the left edge of the image.

The bitmap data is stored separately from the topic text, and a single copy of the bitmap is used for all the \{bmc\}, \{bml\}, and \{bmr\} commands displaying that bitmap.

{button ,AL("TopicInfo",0,"",`main')} Related Topics

Embedded Windows

Embedded windows display text, pictures, or other objects in a window embedded within a Help topic. Authors can write their own custom DLLs to render elements in an embedded window.

An RTF entry for an embedded window has the following syntax:

\{ew[**c**, **l**, **r**] DLL-name, window-class, author-data**\}**

The **\{ew**x**\}** commands specify how the window is aligned in the topic, as shown in the following table:

Control Word	Meaning
\{ewc\}	Positions the window using character alignment. The window is handled as if it were another character and it appears at the position in the line indicated by the reference. Text follows the window, positioned at the base of the window.
\{ewl\}	Positions the window along the left margin, with text wrapping automatically along the right edge of the window.
\{ewr\}	Positions the window along the right margin, with text wrapping automatically along the left edge of the window.

The DLL-name is the filename of the DLL that renders the object in the window. This name should not have an extension, but it can have a relative path.

The window-class is the window-class name for the embedded pane as defined in the C source file for the DLL.

The author-data is a programmer-defined string passed to the DLL (in the WM_CREATE message) when Help creates the window.

{button ,AL("TopicInfo",0,"",`main')} Related Topics

Sample Topic

In the sample RTF file, the following entries make up the single topic the file defines:

```
#{\footnote graphics_cont}
${\footnote Graphics}

\pard\plain \li120\sb340\sa120\sl-320 \f3\fs28 Graphics

\par \pard\plain \li120 \f4\fs20 Use graphic images to illustrate concepts and present
information visually. Graphics include line art, icons, screen shots of the interface,
and graphics with hotspots ("hypergraphics").
\par
```

{button ,AL("TopicInfo",0,"",`main')} Related Topics

Overview of Help RTF token Support

Although Help Workshop supports many RTF tokens, it does not support them all. The following tables present the RTF tokens by group and briefly describe the statements that Help supports in this release of Help Workshop. The following tables are not intended to provide detailed information about the RTF tokens. For complete information, see the "Help RTF token Reference" topic.

Statement descriptions follow these conventions.

Convention	Meaning
Parentheses	Indicates the default value for the statement.
Overloaded	Indicates that the RTF token has a specific meaning in Help that is different from its print-based usage.

{button ,AL("overview")} Related Topics

Overloaded Statements

Help Workshop interprets some RTF tokens differently from their normal print-based usage. For example, standard RTF specifies that the **\uldb** statement indicates a double underline, but Help Workshop uses this statement to indicate a hotspot. The following table lists the RTF tokens that are overloaded for Help Workshop.

Statement	Print-based usage	Help Workshop usage
\footnote	Footnote	Special topic commands
\keep	Keeps paragraph intact	Makes text nonwrapping
\keepn	Keeps paragraph with next	Creates a nonscrolling region at the top of the topic
\page	Creates page break	Ends the current topic
\strike	Creates strikethrough	Indicates a hotspot
\trqc	Centers table row	Uses relative column widths
\ul	Creates continuous underline	Indicates a link to a pop-up topic
\uldb	Creates double underline	Indicates a hotspot
\v	Creates hidden text	Indicates the topic ID to jump to

Character Set

Help Workshop 4.0 supports all RTF character sets.

Statement	Character set
\ansi	ANSI character set
\windows	Windows (default)
\mac	Apple Macintosh
\pc	OEM code page 437
\pca	International English code page 850

Special Characters

If Help Workshop does not recognize a character, it ignores it.

For simplicity, ASCII 9 is treated the same as **\tab** and ASCII 10 is treated the same as **\par**. ASCII 13 is ignored. The control code \<10> is also ignored, even though it may be used to indicate a soft carriage return.

Statement	Meaning
\'hh	Hexadecimal value of specified character
\cell	End of table cell
\emdash	Em dash character
\emspace	Em space character
\endash	En dash character
\enspace	En space character
\ldblquote	Left double quotation mark
\line	Required line break; same as SHIFT+ENTER
\lquote	Left quotation mark
\page	End of current topic. OVERLOADED
\par	End of paragraph
\rdblquote	Right double quotation mark
\row	End of table row
\rquote	Right quotation mark
\sect	End of section and paragraph
\tab	Tab character

Destinations

A destination change resets all properties to their default values. Changes are legal only at the beginning of a group (statement and text enclosed in braces).

Statement	Meaning
\colortbl	Color table
\fonttbl	Font table
\footnote	Topic information. OVERLOADED
\pict	Picture
\rtf	Version of RTF standard used

Paragraph Formatting

The following statements specify paragraph formatting properties:

Statement	Meaning
\box	Boxed paragraph
\brdrb	Bottom border
\brdrbar	Outside border
\brdrdb	Double border
\brdrdot	Dotted border
\brdrl	Left border
\brdrr	Right border
\brdrs	Single-thickness border
\brdrsh	Shadow border
\brdrt	Top border
\brdrth	Thick border
\fin	First-line indent (0)
\intbl	Table paragraph
\keep	Nonwrapping text. OVERLOADED
\keepn	Nonscrolling region. OVERLOADED
\lin	Left indent (0)
\pard	Default paragraph properties
\qc	Centered
\qj	Justified
\ql	Left-aligned (default)
\qr	Right-aligned
\rin	Right indent (0)
\san	Space after (0)
\sbn	Space before (0)
\sln	Line spacing or leading
\tbn	Bar tab
\tqc	Centered tab
\tqr	Flush-right tab
\txn	Custom tab position

Character Formatting

The following statements specify character formatting properties:

Statement	Meaning
\b	Bold
\fn	Font number
\fsn	Font size (24 pt.)
\i	Italic
\plain	Resets program's default character formatting properties
\scaps	Small capitals
\strike	Jump or macro hotspot; identical to \uldb OVERLOADED
\ul	Pop-up hotspot OVERLOADED
\uldb	Jump or macro hotspot; identical to \strike OVERLOADED
\v	Topic ID or macro OVERLOADED

Tables

The following statements specify table formatting properties:

Statement	Meaning
\cellxn	Set absolute position of a table cell's right edge
\clmgf	Mark first cell in a range of cells to be merged
\clmrg	Merge current cell with preceding cell
\trgaphn	Space between text in adjacent cells
\trleftn	Set the position of left margin for the first cell in table row
\trowd	Set table row defaults
\trqc	Relative column widths. OVERLOADED
\trql	Left align text in each cell of table row (default)

Pictures

The following statements specify picture formatting:

Statement	Meaning
\binn	Binary picture data
\box	Boxed picture
\brdrb	Bottom picture border
\brdrbar	Outside picture border
\brdrdb	Double picture border
\brdrdot	Dotted picture border
\brdrl	Left picture border
\brdrr	Right picture border
\brdrs	Single-thickness picture border
\brdrsh	Shadow picture border
\brdrt	Top picture border
\brdrth	Thick picture border
\pichgoaln	Desired picture height
\pichn	Picture height
\picscalexn	Horizontal scaling value (100)
\picscaleyn	Vertical scaling value (100)
\picwgoaln	Desired picture width
\picwn	Picture width
\wbitmapn	Picture type (Default is Windows bitmap)
\wbmbitspixeln	Bits per pixel (1)
\wbmplanesn	Number of bitmap color planes (1)
\wbmwidthbytesn	Bitmap width, in bytes
\wmetafilen	Windows metafile

Fields

Help Workshop supports only one field statement:

Statement	Meaning
\fldrslt	Most recent calculated result of the field

Unsupported Formats

Help Workshop does not support statements of the following types:

- Document Formatting
- Section Formatting
- Absolute-Positioned Objects
- Annotations
- Headers and Footers
- Document Information
- Index Entries
- Table of Contents Entries
- Bookmarks

Help RTF token Reference

The following is an alphabetical list of RTF tokens recognized by Help Workshop. To locate a particular statement, scroll down the topic or use the letter buttons above.

\ansi

The **\ansi** statement sets the American National Standards Institute (ANSI) character set. The Windows character set is essentially equivalent to the ANSI character set.

See Also

\windows

\b

The **\b** statement starts bold text. The statement applies to all subsequent text up to the next **\plain** or **\b0** statement.

Comments

No **\plain** or **\b0** statement is required if the **\b** statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to the enclosed text only.

Example

The following example sets Note to bold:

```
{\b Note}  Setting the Auto option frees novice users from determining their system configurations.
```

See Also

\i, \plain, \scaps

\bin

\binn

The **\bin** statement indicates the start of binary picture data. Help Workshop interprets subsequent bytes in the file as binary data. This statement is used in conjunction with the **\pict** statement.

Parameter	Description
n	Specifies the number of bytes of binary data following the statement.

Comments

A single space character must separate the **\bin** statement from subsequent bytes. Help Workshop assumes that all subsequent bytes, including linefeed and carriage return characters, are binary data. These bytes can have any value in the range 0 through 255. For this reason, the **\bin** statement is typically used in program-generated files only.

If the **\bin** statement is not given with a **\pict** statement, the default picture data format is hexadecimal.

See Also

\pict

\bmc

\{bmc filename**\}**

The **bmc** statement displays a specified bitmap or metafile in the current line of text. The statement positions the bitmap or metafile as if it were the next character in the line, aligning it on the base line and applying the current paragraph properties.

Parameter	Description
filename	Specifies the name of a file containing either a Windows bitmap, a placeable Windows metafile, a multi-resolution bitmap, or a segmented graphics bitmap.

Comments

Since the **bmc** statement is not a standard RTF token, Help Workshop relies on the opening and closing braces, including the backslashes (\), to distinguish the statement from regular text.

If a file containing a metafile is specified, the file must contain a placeable Windows metafile; Help Workshop will not accept standard Windows metafiles. Furthermore, Help Workshop sets the MM_ANISOTROPIC mode prior to displaying the metafile, so the placeable Windows metafile must either set the window origin and extents or set some other mapping mode.

Example

The following example inserts a bitmap representing a keyboard key in a paragraph:

```
?           \par
Press the \{bmc escape.bmp\} key to return to the main window.
\par
```

See Also

\bml, \bmr, \wbitmap

\bml

\{bml filename**\}**

The **bml** statement displays a specified bitmap or metafile at the left margin of the Help window. The first line of subsequent text aligns with the upper-right corner of the image and subsequent lines wrap along the right edge of the image.

Parameter	Description
filename	Specifies the name of a file containing either a Windows bitmap, a placeable Windows metafile, a multi-resolution bitmap, or a segmented graphics bitmap.

Comments

Since the **bml** statement is not a standard RTF token, Help Workshop relies on the opening and closing braces, including the backslashes (\), to distinguish the statement from regular text.

If a file containing a metafile is specified, the file must contain a placeable Windows metafile; Help Workshop will not accept standard Windows metafiles. Furthermore, Help Workshop sets the MM_ANISOTROPIC mode prior to displaying the metafile, so the placeable Windows metafile must either set the window origin and extents or set some other mapping mode.

Example

The following example places a bitmap at the left margin. The subsequent paragraph wraps around the bitmap:

```
?           \par
\{bml roadmap.bmp\}
The map at the left shows the easiest route to the school.
Although many people use Highway 125, there are fewer stops
and less traffic if you use Ames Road.
```

See Also

\bmc, \bmr, \wbitmap

\bmr

\{bmr filename**\}**

The **bmr** statement displays a specified bitmap or metafile at the right margin of the Help window. The first line of subsequent text aligns with the upper-left corner of the image and subsequent lines wrap along the left edge of the image.

Parameter	Description
filename	Specifies the name of a file containing either a Windows bitmap, a placeable Windows metafile, a multi-resolution bitmap, or a segmented graphics bitmap.

Comments

Since the **bmr** statement is not a standard RTF token, Help Workshop relies on the opening and closing braces, including the backslashes (\), to distinguish the statement from regular text.

If a file containing a metafile is specified, the file must contain a placeable Windows metafile; Help Workshop will not accept standard Windows metafiles. Furthermore, Help Workshop sets the MM_ANISOTROPIC mode prior to displaying the metafile, so the placeable Windows metafile must either set the window origin and extents or set some other mapping mode.

Example

The following example places a bitmap at the right margin. The subsequent paragraph wraps around the bitmap:

```
? \par
\{bmr roadmap.bmp\}
The map at the right shows the easiest route to the school.
Although many people use Highway 125, there are fewer stops
and less traffic if you use Ames Road.
```

See Also

\bmc, \bml, \wbitmap

\box

The **\box** statement draws a box around the current paragraph or picture. The statement applies to all subsequent paragraphs or pictures up to the next **\pard** statement.

Comments

For paragraphs, Help Workshop uses the height of the paragraph, excluding space before or after the paragraph, as the height of the box. For pictures (as defined by **\pict** statements), Help Workshop uses the specified height of the picture as the height of the box. For both paragraphs and pictures, the width of the box is equal to the space between the left and right indents.

WinHelp draws the box using the current border style.

Example

The following example draws a box around the paragraph:

```
? \par \box
{\b Note}  Setting the Auto option frees novice users from
determining their system configurations.
\par \pard
```

See Also

\brdrb, \brdrl, \brdrr, \brdrt, \pard

\brdrb

The **\brdrb** statement draws a border below the current paragraph or picture. The statement applies to all subsequent paragraphs or pictures up to the next **\pard** statement.

Comments

WinHelp draws the border using the current border style.

See Also

\box, \brdrbar, \brdrl, \brdrr, \brdrt, \pard

\brdrbar

The **\brdrbar** statement draws a vertical bar to the left of the current paragraph or picture. The statement applies to all subsequent paragraphs or pictures up to the next **\pard** statement.

Comments

WinHelp draws the border using the current border style.

In a print-based document, the **\brdrbar** statement draws the bar on the right side of paragraphs on odd-numbered pages, but on the left side of paragraphs on even-numbered pages.

See Also

\box, \brdrb, \brdrl, \brdrr, \brdrt, \pard

\brdrdb

The **\brdrdb** statement selects a double line for drawing borders. The selection applies to all subsequent paragraphs or pictures up to the next **\pard** statement.

See Also

\brdrdot, \brdrs, \brdrsh, \brdrth, \pard

\brdrdot

The **\brdrdot** statement selects a dotted line for drawing borders. The selection applies to all subsequent paragraphs or pictures up to the next **\pard** statement.

See Also

\brdrdb, \brdrs, \brdrsh, \brdrth, \pard

\brdrl

The **\brdrl** statement draws a border to the left of the current paragraph or picture. The statement applies to all subsequent paragraphs or pictures up to the next **\pard** statement.

Comments

WinHelp draws the border using the current border style.

See Also

\box, \brdrb, \brdrbar, \brdrr, \brdrt, \pard

\brdrr

The **\brdrr** statement draws a border to the right of the current paragraph or picture. The statement applies to all subsequent paragraphs or pictures up to the next **\pard** statement.

Comments

WinHelp draws the border using the current border style.

See Also

\box, \brdrb, \brdrbar, \brdrl, \brdrt, \pard

\brdrs

The **\brdrs** statement selects a standard-width line for drawing borders. The selection applies to all subsequent paragraphs or pictures up to the next **\pard** statement.

See Also

\brdrdb, \brdrdot, \brdrsh, \brdrth, \pard

\brdrsh

The **\brdrsh** statement selects a shadow outline for drawing borders. The selection applies to all subsequent paragraphs or pictures up to the next **\pard** statement.

See Also

\brdrdb, \brdrdot, \brdrs, \brdrth, \pard

\brdrt

The **\brdrt** statement draws a border above the current paragraph or picture. The statement applies to all subsequent paragraphs or pictures up to the next **\pard** statement.

Comments

WinHelp draws the border using the current border style.

See Also

\box, \brdrb, \brdrbar, \brdrl, \brdrr, \pard

\brdrth

The **\brdrth** statement selects a thick line for drawing borders. The selection applies to all subsequent paragraphs or pictures up to the next **\pard** statement.

See Also

\brdrdb, \brdrdot, \brdrs, \brdrsh, \pard

\cell

The **\cell** statement marks the end of a cell in a table. A cell consists of all paragraphs from a preceding **\intbl** or **\cell** statement to the ending **\cell** statement. WinHelp formats and displays these paragraphs using the left and right margins of the cell and any current paragraph properties.

Example

The following example creates a two-column table. The second column contains three separate paragraphs, each having different paragraph properties:

```
?           \cellx2880\cellx5760
\intbl
Alignment\cell
\ql
Left-aligned
\par
\qc
Centered
\par
\qr
Right-aligned\cell
\row \pard
```

See Also

\cellx, \intbl, \row, \trgaph, \trleft, \trowd

\cellx

\cellxn

The **\cellx** statement sets the absolute position of the right edge of a table cell. One **\cellx** statement must be given for each cell in the table. The first **\cellx** statement applies to the left-most cell, the last to the rightmost cell. For each **\cellx** statement, the specified position applies to the corresponding cell in each subsequent row of the table up to the next **\trowd** statement.

Parameter	Description
n	Specifies the position of the cell's right edge, in twips. The position is relative to the left edge of the Help window. It is not affected by the current indents.

Comments

A table consists of a grid of cells in columns and rows. Each cell has an explicitly defined right edge; the position of a cell's left edge is the same as the position of the right edge of the adjacent cell. For the left-most cell in a row, the left edge position is equal to the Help window's left margin position. Each cell has a left and right margin between which WinHelp aligns and wraps text. By default, the margin positions are equal to the left and right edges. The **\trgaph** and **\trleft** statements can be used to set different margins for all cells in a row.

Example

The following example creates a three-column table having two rows. The positions of the right edges of the three cells are 2, 4, and 6 inches, respectively:

```
? \cellx2880\cellx5760\cellx8640
\intbl
Row 1 Cell 1\cell
Row 1 Cell 2\cell
Row 1 Cell 3\cell
\row
\intbl
Row 2 Cell 1\cell
Row 2 Cell 2\cell
Row 2 Cell 3\cell
\row \pard
```

See Also

\cell, \intbl, \row, \trgaph, \trleft, \trowd

\cb

\cbn

The **\cb** statement sets the background color. The new color applies to all subsequent text up to the next **\plain** or **\cb** statement.

Parameter	Description
n	Specifies the color number to set as background. The number must be an integer number in the range 1 to the maximum number of colors specified in the color table for the Help file. If an invalid color number is specified, WinHelp uses the default background color.

Comments

No **\plain** or **\cb** statement is required if the **\cb** statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to the enclosed text only.

If the **\cb** statement is not given, the default background color is the text color set by Control Panel.

Example

The following example creates a light gray background color:

```
? {\colortbl;\red128\green128\blue128;}
{\cb1 This background is light gray.}
```

See Also

\cf, \colortbl

\cf

\cfn

The **\cf** statement sets the foreground color. The new color applies to all subsequent text up to the next **\plain** or **\cf** statement.

Parameter	Description
n	Specifies the color number to set as foreground. The number must be an integer number in the range 1 to the maximum number of colors specified in the color table for the Help file. If an invalid color number is specified, WinHelp uses the default foreground color.

Comments

No **\plain** or **\cf** statement is required if the **\cf** statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to the enclosed text only.

If the **\cf** statement is not given, the default foreground color is the text color set by Control Panel.

Example

The following example displays green text:

```
{\colortbl;\red0\green255\blue0;}
{\cf1 This text is green.}
```

See Also

\cb, \colortbl

\chftn

\chftnn

The **\chftn** statement sets the footnote reference character for subsequent **\footnote** statements.

Help Workshop ignores this statement.

Parameter	Description
n	Specifies the footnote reference character.

See Also

\footnote

\clmgf

The **\clmgf** statement specifies the first cell in a range of cells to be merged.

Help Workshop ignores this statement.

Comments

All cells between the **\clmgf** statement and a subsequent **\clmrg** statement are combined into a single cell. The left edge of the new cell is the same as that of the left-most cell to be merged; the right edge is the same as that of the rightmost cell.

See Also

\clmrg

\clmrg

The **\clmrg** statement merges the current cell with the preceding cell.

Help Workshop ignores this statement.

Comments

All cells between the **\clmgf** statement and a subsequent **\clmrg** statement are combined into a single cell. The left edge of the new cell is the same as that of the left-most cell to be merged; the right edge is the same as that of the rightmost cell.

See Also

\clmgf

\colortbl

{\colortbl\redr**green**g**blue**b**;}**

The **\colortbl** statement creates a color table for the Help file. The color table consists of one or more color definitions. Each color definition consists of one **\red**, **\green**, and **\blue** statement specifying the amount of primary color to use to generate the final color. Each color definition must end with a semicolon (;).

Parameter	Description
r	Specifies the intensity of red in the color. It must be an integer in the range 0 through 255.
g	Specifies the intensity of green in the color. It must be an integer in the range 0 through 255.
b	Specifies the intensity of blue in the color. It must be an integer in the range 0 through 255.

Comments

Color definitions are implicitly numbered starting at zero. A color definition's implicit number can be used in the **\cf** statement to set the foreground color.

The default colors are the window text and window background colors set by Control Panel. To override the default colors, a **\colortbl** statement and **\cf** and **\cb** statements must be given.

Example

The following example creates a color table containing two color definitions. The first color definition is empty (only the semicolon is given), so color number 0 always represents the default color. The second definition specifies green; color number 1 can be used to display green text:

```
{\colortbl;\red0\green255\blue0;}
```

See Also

\cb, \cf

\deff

\deffn

The **\deff** statement sets the default font number. Help Workshop uses the number to set the default font whenever a **\plain** statement is given or an invalid font number is given in a **\f** statement.

Parameter	Description
n	Specifies the number of the font to be used as the default font. This parameter must be a valid font number as specified by the **\fonttbl** statement for the Help file.

Comments

If the **\deff** statement is not given, the default font number is zero.

See Also

\f, \fonttbl, \plain

\emdash

\emdash

The **\emdash** statement inserts an em dash character if either Times New Roman or Arial has been specified in the \fonttable of the current rtf

file. If neither font has been specified, Help Workshop substitutes a hyphen character.

See Also

\endash

\emspace

\emspace

The **\emspace** statement inserts two spaces in the current font.

See Also

\enspace

\endash

\endash

The **\endash** statement inserts an en dash character if either Times New Roman or Arial has been specified in the \fonttable of the current rtf file. If neither font has been specified, Help Workshop substitutes a hyphen character.

See Also

\emdash

\enspace

\enspace

The **\enspace** statement inserts an a space character in Courier font using the current point size if Courier has been specified in the \fonttable of the current rtf file. If Coureir has been specified, Help Workshop substitutes a space in the current font.

See Also

\emspace

\ewc

\{ewc DLL-name, window-class, author-data**\}**

The **ewc** statement displays information (bitmap, multimedia element, and so on) under the control of a dynamic-link library in the current line of text. The statement positions the object as if it were the next character in the line, aligning it on the base line and applying the current paragraph properties.

Parameter	Description
DLL-name	Specifies the name of the DLL that controls the embedded window. The filename must not include an extension or be fully qualified, but it can include a relative path. WinHelp assumes .DLL or .EXE to be the default extension.
window-class	Specifies the name of the embedded window class as defined in the source code for the DLL.
author-data	Specifies an arbitrary string, which WinHelp passes to the embedded window when it creates the window. This string can be one or more substrings separated by any punctuation mark except a comma. The DLL is responsible for parsing this string.

Comments

Since the **ewc** statement is not a standard RTF token, Help Workshop relies on the opening and closing braces, including the backslashes (\), to distinguish the statement from regular text.

Because the embedded window is treated as text, paragraph formatting properties assigned to the paragraph also apply to the window. Text coming before or after the window does not wrap around the window. Help adjusts the height of the line containing the embedded window to allow enough space for the embedded window.

If you use the **\sl** statement in conjunction with an **ewc** statement, don't specify a negative value. If you do, the embedded window might appear on top of the succeeding paragraph when WinHelp displays the topic.

Example

The following example displays a 256-color bitmap in an embedded window within a paragraph:

```
\par
preceding text \{ewc MVBMP, ViewerBMP, ..\art\ocean.bmp\} following text...
\par
```

See Also

\ewl, \ewr

\ewl

\{ewl DLL-name, window-class, author-data**\}**

The **ewl** statement displays information (bitmap, multimedia element, and so on) under the control of a dynamic-link library at the left margin of the Help window. The first line of subsequent text aligns with the upper-right corner of the embedded window and subsequent lines wrap along the right edge of the window.

Parameter	Description
DLL-name	Specifies the name of the DLL that controls the embedded window. The filename must not include an extension or be fully qualified, but it can include a relative path. WinHelp assumes .DLL or .EXE to be the default extension.
window-class	Specifies the name of the embedded window class as defined in the source code for the DLL.
author-data	Specifies an arbitrary string, which WinHelp passes to the embedded window when it creates the window. This string can be one or more substrings separated by any punctuation mark except a comma. The DLL is responsible for parsing this string.

Comments

Since the **ewl** statement is not a standard RTF token, Help Workshop relies on the opening and closing braces, including the backslashes (\), to distinguish the statement from regular text.

Do not put any space characters between the **ewl** statement and the paragraph text unless you want the first line of text indented from the rest of the text that wraps along the right edge of the window. To be sure that text wraps correctly around an embedded window, insert a soft carriage return at the end of each line of text.

Example

The following example places an embedded window at the left margin. The subsequent paragraph wraps around the embedded window:

```
\par
\{ewl FADE, AmfWnd, clipbrd.amf\}
The map at the left shows the easiest route to the school.
Although many people use Highway 125, there are fewer stops
and less traffic if you use Ames Road.
```

See Also

\ewc, \ewr

\ewr

\{ewr DLL-name, window-class, author-data**\}**

The **ewr** statement displays information (bitmap, multimedia element, and so on.) under the control of a dynamic-link library at the right margin of the Help window. The first line of subsequent text aligns with the upper-left corner of the embedded window and subsequent lines wrap along the left edge of the window.

Parameter	Description
DLL-name	Specifies the name of the DLL that controls the embedded window. The filename must not include an extension or be fully qualified, but it can include a relative path. WinHelp assumes .DLL or .EXE to be the default extension.

window-class	Specifies the name of the embedded window class as defined in the source code for the DLL.
author-data	Specifies an arbitrary string, which WinHelp passes to the embedded window when it creates the window. This string can be one or more substrings separated by any punctuation mark except a comma. The DLL is responsible for parsing this string.

Comments

Since the **ewr** statement is not a standard RTF token, Help Workshop relies on the opening and closing braces, including the backslashes (\), to distinguish the statement from regular text.

Do not put any space characters between the **ewr** statement and the paragraph text unless you want the first line of text indented from the rest of the text that wraps from the left topic margin. To be sure that text wraps correctly around an embedded window, insert a soft carriage return at the end of each line of text.

Example

The following example places an embedded window at the right margin. The subsequent paragraph wraps around the embedded window:

```
?           \par
\{ewl FADE, AmfWnd, clipbrd.amf\}
The map at the right shows the easiest route to the school.
Although many people use Highway 125, there are fewer stops
and less traffic if you use Ames Road.
```

See Also

\ewc, \ewl

\f

\fn

The **\f** statement sets the font. The new font applies to all subsequent text up to the next **\plain** or **\f** statement.

Parameter	Description
n	Specifies the font number. This parameter must be one of the integer font numbers defined in the font table for the Help file.

Comments

The **\f** statement does not set the point size of the font; use the **\fs** statement instead.

No **\plain** or **\f** statement is required if the **\f** statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to just the enclosed text.

If the **\f** statement is not given, the default font is defined by the **\deff** token (or is zero if no **\deff** token is supplied).

Example

The following example uses the Arial font to display text:

```
?           {\fonttbl {\f0\fswiss Arial;}}
\par
{\f0
This text illustrates the Arial font.}
\par
```

See Also

\deff, \fonttbl, \fs, \plain

\fi

\fin

The **\fi** statement sets the first-line indent for the paragraph. The new indent applies to the first line of each subsequent paragraph up to the next **\pard** statement or **\fi** statement. The first-line indent is always relative to the current left indent.

Parameter	Description
n	Specifies the indent in twips. This parameter can be either a positive or negative number.

Comments

If the **\fi** statement is not given, the first-line indent is zero by default.

Example

The following example uses the first-line indent and a tab stop to make a numbered list:

```
?           \tx360\li360\fi-360
1
\tab
Insert the disk in drive A.
\par
2
\tab
Type a:setup and press the ENTER key.
\par
3
\tab
Follow the instructions on the screen.
\par \pard
```

See Also

\li, \pard

\field

The **\field** statement defines a field.

Comments

Help Workshop ignores this statement and all related field statements except the **\fldrslt** statement.

See Also

\fldrslt

\fldrslt

The **\fldrslt** statement specifies the most recently calculated result of a field. Help Workshop interprets the result as text and formats it using the current character and paragraph properties.

Comments

Help Workshop ignores all field statements except the **\fldrslt** statement. Any text associated with other field statements is ignored.

See Also

\field

\fonttbl

\fonttbl {\fn\family font-name**; .}**

The **\fonttbl** statement creates a font table for the Help file. The font table consists of one or more font definitions. Each definition consists of a font number, a font family, and a font name.

Parameter	Description
n	Specifies the font number. This parameter must be an integer. This number can be used in subsequent **\f** statements to set the current font to the specified font. In the font table, font numbers should start at zero and increase by one for each new font definition.
family	Specifies the font family. This parameter must be one of the following.

Value	Meaning
fnil	Unknown or default fonts (default)

	froman	Roman, proportionally spaced serif fonts (for example, MS Serif and Palatino)
	fswiss	Swiss, proportionally spaced sans serif fonts (for example, Swiss)
	fmodern	Fixed-pitch serif and sans serif fonts (for example, Courier, Elite, and Pica)
	fscript	Script fonts (for example, Cursive)
	fdecor	Decorative fonts (for example, Old English and ITC Zapf Chancery)
	ftech	Technical, symbol, and mathematical fonts (for example, Symbol)
font-name	Specifies the name of the font. This parameter should specify an available Windows font.	

Comments

If a font with the specified name is not available, WinHelp chooses a font from the specified family. If no font from the given family exists, WinHelp chooses a font having the same character set as specified for the Help file.

The **\deff** statement sets the default font number for the Help file. The default font is set whenever the **\pard** statement is given.

See Also

\deff, \f, \fs, \pard

\footnote

n**{\footnote** text**}**

The **\footnote** statement defines topic-specific information, such as the topic's build tags, topic ID, title, browse code, keywords, and entry macros. Every topic must have at least a topic ID footnote in order to give the user access to the topic through links.

Parameter	Description	
n	Specifies the footnote character. It can be one of the following.	
	Value	**Meaning**
	*****	Specifies a build tag. Help Workshop uses build tags to determine whether to include the topic in the Help file. The text parameter can be any combination of characters but must not contain spaces. Uppercase and lowercase characters are treated as equivalent characters (case insensitive). If a topic has build-tag statements, they must be the first statements in the topic. Help Workshop checks a topic for build tags if the Help project (.HPJ) file specifies a build expression using the **BUILD** option.
	#	Specifies a topic ID. The text parameter can be any combination of letters and digits but must not contain spaces. Uppercase and lowercase characters are treated as equivalent characters (case insensitive). The topic ID can be used with the **\v** statement in other topics to create links to this topic.
	$	Specifies a topic title. WinHelp uses the topic title to identify the topic in the Topics Found and History dialog boxes. The text parameter can be any combination of characters including spaces.
	>	Specifies a default window type for individual topics. WinHelp uses this window definition to display the topic when it is accessed from the index.
	+	Specifies the browse-sequence identifier. WinHelp adds topics having an identifier to the browse sequence and allows users to view the topics by using the browse buttons. The text parameter can be a combination of letters and digits. WinHelp determines the order of topics in the browse sequence by sorting the identifier alphabetically. If two topics have the same identifier, WinHelp assumes that the topic that was compiled first is to be displayed first. WinHelp uses the browse sequence identifier only if the browse buttons have been enabled by using the **BrowseButtons** macro.
	K	Specifies a keyword. WinHelp displays all keywords in the Help file in the Index property sheet and allows a user to choose a topic to view by choosing a keyword. The text parameter can be any combination of characters including spaces. If the first character is the letter K; it must be preceded with an extra space or a semicolon. More than one keyword can be given by separating the keywords with semicolons (;). A topic cannot contain keywords unless it also has a topic title.
	A	Specifies a keyword used for an **ALink** macro. 'A' keywords do not appear in the index.
	!	Specifies a Help macro. WinHelp runs the macro when the topic is displayed. The text parameter can be any

	Help macro.
@	Specifies an author-defined comment about the Help topic. This footnote is provided for authoring purposes only. Help Workshop ignores this footnote when building the Help file. The text parameter can be any combination of letters and digits, including accented characters and spaces.
	If n is any letter (other than K or A), the footnote specifies an alternative keyword. Windows programs can search for topics having alternative keywords by using the **HELP_MULTIKEY** command with the **WinHelp** function.
text	Specifies the build tag, topic ID, topic title, browse-sequence number, keyword, or macro associated with the footnote. This parameter depends on the footnote type as specified by the n parameter.

Comments

A topic can have more than one build tag, topic ID, keyword, and Help-macro statement, but must not have more than one topic-title or browse-sequence-number statement.

In print-based documents, the **\footnote** statement creates a footnote and the footnote is anchored to the character immediately preceding the **\footnote** statement.

Example

The following example defines a topic titled "Short Topic." The topic ID "topic1" can be used to create links to the topic. The keywords "example topic" and "short topic" appear in the Search dialog box and can be used to choose the topic for viewing. Or the user can find the topic by browsing since the topic is included in the short browse sequence. The topic also includes an entry macro that starts the Microsoft Clock program when Help displays the topic:

```
${\footnote Short Topic}
#{\footnote topic1}
K{\footnote example topic;short topic}
+{\footnote short:015}
!{\footnote ExecProgram(`clock.exe', 0)}
This topic has a title, topic ID, and two keywords.
\par
\page
```

See Also

\chftn, \v

\fs

\fsn

The **\fs** statement sets the size of the font. The new font size applies to all subsequent text up to the next **\plain** or **\fs** statement.

Parameter	Description
n	Specifies the size of the font, in half points.

Comments

The **\fs** statement does not set the font face; use the **\f** statement instead.

No **\plain** or **\fs** statement is required if the **\fs** statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to just the enclosed text.

If the **\fs** statement is not given, the default font size is 24.

Example

The following example sets the size of the font to 10 points:

```
{\fs20 This line is in 10 point type.}
\par
```

See Also

\f, \plain

\'

\'hh

The \' statement converts the specified hexadecimal number into a character value and inserts the value into the Help file. The appearance of the character when displayed depends on the character set specified for the Help file.

Parameter	Description
hh	Specifies a two-digit hexadecimal value.

Comments

Since Help Workshop does not accept character values greater than 127, the \' statement is the only method to insert such character values into the Help file.

Example

The following example inserts a trademark in a Help file that uses the **\windows** statement to set the character set:

```
ABC\'99 is a trademark of the ABC Product Corporation.
```

See Also

\ansi, \pc, \pca, \windows

\i

The **\i** statement starts italic text. The statement applies to all subsequent text up to the next **\plain** or **\i0** statement.

Comments

No **\plain** or **\i0** statement is required if the **\i** statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to just the enclosed text.

Example

The following example sets "not" to italic:

```
You must {\i not} save the file without first setting the
Auto option.
```

See Also

\b, \plain, \scaps

\intbl

The **\intbl** statement marks subsequent paragraphs as part of a table. The statement applies to all subsequent paragraphs up to the next **\row** statement.

Example

The following example creates a three-column table having two rows:

```
\cellx1440\cellx2880\cellx4320
\intbl
Row 1 Column 1\cell
Row 1 Column 2\cell
Row 1 Column 3\cell \row
\intbl
Row 2 Column 1\cell
Row 2 Column 2\cell
Row 2 Column 3\cell \row \pard
```

See Also

\cell, \cellx, \row, \trgaph, \trleft, \trowd

\keep

The **\keep** statement prevents WinHelp from wrapping text to fit the Help window. The statement applies to all subsequent paragraphs up to the next **\pard** statement.

Comments

If the text in a paragraph exceeds the width of the Help window, WinHelp displays a horizontal scroll bar.

In print-based documents, the **\keep** statement keeps paragraphs intact.

Example

The following example makes the paragraph beginning "Cardfile has two views — Card and List" nonwrapping text:

```
\par\pard\keep
Cardfile has two views--Card and List.
```

See Also

\line

\keepn

The **\keepn** statement creates a nonscrolling region at the top of the Help window for the given topic. The **\keepn** statement applies to all subsequent paragraphs up to the next **\pard** statement. All paragraphs at the beginning of a topic that meet this criteria are placed in the nonscrolling region.

Comments

If a **\keepn** statement is used in a topic, it must be applied to the first paragraph in the topic (and subsequent paragraphs as needed). Help Workshop displays an error message and does not create a nonscrolling region if paragraphs are given before the **\keepn** statement. Only one nonscrolling region per topic is allowed.

WinHelp formats, aligns, and wraps text in the nonscrolling region just as it does in the rest of the topic. It separates the nonscrolling region from the rest of the Help window with a horizontal bar. WinHelp sets the height of the nonscrolling region so that all paragraphs in the region can be viewed if the Help window is large enough. If the window is smaller than the nonscrolling region, the user will be unable to view the rest of the topic. For this reason, the nonscrolling region is typically reserved for a single line of text specifying the name or title of the topic.

In print-based documents, the **\keepn** statement keeps the subsequent paragraph with the paragraph that follows it.

Example

The following example places the first paragraph of the topic beginning "Cardfile has two views — Card and List" into a nonscrolling region:

```
\par\pard\keepn
Cardfile has two views--Card and List.
\par\pard
```

See Also

\page

\ldblquote

\ldblquote

The **\ldblquote** statement inserts a left double quotation mark if either Times New Roman or Arial has been specified in the \fonttable of the current rtf file. If neither font has been specified, Help Workshop substitutes a straight double-quote character. The actual quote character used depends on the language of the help file.

See Also

\lquote, \rdblquote, \rquote

\li

\lin

The **\li** statement sets the left indent for the paragraph. The indent applies to all subsequent paragraphs up to the next **\pard** or **\li** statement.

Parameter	Description
n	Specifies the indent in twips. The value can be either positive or negative.

Comments

If the **\li** statement is not given, the left indent is zero by default. WinHelp automatically provides a small left margin; if no indent is specified the text does not start immediately at the left edge of the Help window.

Specifying a negative left indent moves the starting point for a line of text to the left of the default left margin. If the negative indent is large enough, the start of the text may be clipped by the left edge of the Help window.

Example

The following example uses the left indent and a tab stop to make a bulleted list. In this example, font number 0 is assumed to be the Symbol font:

? Use the Auto command to:

```
\par
\tx360\li360\fi-360
{\f0\'B7}
\tab
Save files automatically
\par
{\f0\'B7}
\tab
Prevent overwriting existing files
\par
{\f0\'B7}
\tab
Create automatic backup files
\par \pard
```

See Also

\fi, \pard, \ri

\line

The **\line** statement breaks the current line without ending the paragraph. Subsequent text starts on the next line and is aligned and indented according to the current paragraph properties.

See Also

\par

\lquote

\lquote

The **\lquote** statement inserts a left single quotation mark if either Times New Roman or Arial has been specified in the \fonttable of the current rtf file. If neither font has been specified, Help Workshop substitutes a left single quotation mark in the current font.

See Also

\ldblquote, \rdblquote, \rquote

\mac

The **\mac** statement sets the Apple Macintosh character set.

See Also

\windows

\page

The **\page** statement marks the end of a topic.

Comments

In a print-based document, the **\page** statement creates a page break.

Example

The following example shows a complete topic:

```
${\footnote Short Topic}
#{\footnote short_topic}
Most topics in a topic file consist of topic-title and
topic-ID statements followed by the topic text. Every
topic ends with a {\b \\page} statement.
\par
\page
```

See Also

\par

\par

The **\par** statement marks the end of a paragraph. The statement ends the current line of text and moves the current position to the left margin and down by the current line-spacing and space-after-paragraph values.

Comments

The first line of text after a **\par**, **\page**, or **\sect** statement marks the start of a paragraph. When a paragraph starts, the current position is moved down by the current space-before-paragraph value. Subsequent text is formatted using the current text alignment, line spacing, and the left, right, and first-line indents.

Example

The following example has three paragraphs:

```
\ql
This paragraph is left-aligned.
\par \pard
\qc
This paragraph is centered.
\par \pard
\qr
This paragraph is right-aligned.
\par
```

See Also

\line, \pard, \sect

\pard

The **\pard** statement restores all paragraph properties to default values.

Comments

If the **\pard** statement appears anywhere before the end of a paragraph (that is, before the **\par** statement), the default properties apply to the entire paragraph.

The default paragraph properties are as follows.

Property	Default
Alignment	Left-aligned
First-line indent	0
Left indent	0
Right indent	0
Space before	0
Space after	0
Line spacing	Tallest character

Tab stops	None
Borders	None
Border style	Single-width

See Also

\par

\pc

The **\pc** statement sets the OEM character set (also known as code page 437).

See Also

\windows

\pca

The **\pca** statement sets the International English character set (also known as code page 850).

See Also

\windows

\pich

\pichn

The **\pich** statement specifies the height of the picture. This statement must be used in conjunction with a **\pict** statement.

Parameter	Description
n	Specifies the height of the picture, in twips or pixels, depending on the picture type. If the picture is a metafile, the width is in twips; otherwise, the width is in pixels.

See Also

\pict, \picw

\pichgoal

\pichgoaln

The **\pichgoal** statement specifies the desired height of a picture. If necessary, WinHelp stretches or compresses the picture to match the requested height. This statement must be used in conjunction with a **\pict** statement.

Parameter	Description
n	Specifies the desired height, in twips.

See Also

\pict, \picwgoal

\picscalex

\picscalexn

The **\picscalex** statement specifies the horizontal scaling value. This statement must be used in conjunction with a **\pict** statement.

Parameter	Description
n	Specifies the scaling value. The parameter must be a value in the range 0 through 100, representing a percentage.

Comments

If the **\picscalex** statement is not given, the default scaling value is 100.

See Also

\picscaley, \pict

\picscaley

\picscaleyn

The **\picscaley** statement specifies the vertical scaling value. This statement must be used in conjunction with a **\pict** statement.

Parameter	Description
n	Specifies the scaling value. The parameter must be a value in the range 0 through 100, representing a percentage.

Comments

If the **\picscaley** statement is not given, the default scaling value is 100.

See Also

\picscalex, \pict

\pict

\pict picture-statements picture-data

The **\pict** statement creates a picture. A picture consists of hexadecimal or binary data representing a bitmap or metafile.

Parameter	Description
picture-statements	Specifies one or more statements defining the type of picture, the dimensions of the picture, and the format of the picture data. It can be a combination of the following statements:

Statement	Description
\wbitmap	Specifies a Windows bitmap.
\wmetafile	Specifies a Windows metafile.
\picw	Specifies the picture width.
\pich	Specifies the picture height.
\picwgoal	Specifies the desired picture width.
\pichgoal	Specifies the desired picture height.
\picscalex	Specifies the horizontal scaling value.
\picscaley	Specifies the vertical scaling value.
\wbmbitspixel	Specifies the number of bits per pixel.
\wbmplanes	Specifies the number of planes.
\wbmwidthbytes	Specifies the bitmap width, in bytes.
\bin	Specifies binary picture data.

picture-data	Specifies hexadecimal or binary data representing the picture. The picture data follows the last picture statement.

Comments

If a data format is not specified, the default format is hexadecimal.

See Also

\bin, \pich, \pichgoal, \picscalex, \picscaley, \picw, \picwgoal, \wbitmap, \wbmbitspixel, \wbmplanes, \wbmwidthbytes, \wmetafile

---

\picw

\picwn

The **\picw** statement specifies the width of the picture. This statement must be used in conjunction with a **\pict** statement.

Parameter	Description
n	Specifies the width of the picture, in twips or pixels, depending on the picture type. If the picture is a metafile, the width is in twips; otherwise, the width is in pixels.

See Also

\pich, \pict

\picwgoal

\picwgoaln

The **\picwgoal** statement specifies the desired width of the picture, in twips. If necessary, WinHelp stretches or compresses the picture to match the requested height. This statement must be used in conjunction with a **\pict** statement.

Parameter	Description
n	Specifies the desired width, in twips.

See Also

\pichgoal, \pict

\plain

The **\plain** statement restores the character properties to default values.

Comments

The default character properties are as follows.

Property	Default
Bold	Off
Italic	Off
Small caps	Off
Font	0
Font size	24

See Also

\b, \f, \fs, \i, \scaps

\qc

The **\qc** statement centers text between the current left and right indents. The statement applies to subsequent paragraphs up to the next **\pard** statement or text-alignment statement.

Comments

If a **\ql**, **\qr**, **\qc**, or **\qj** statement is not given, the text is left-aligned by default.

See Also

\qj, \ql, \qr, \pard

\qj

The **\qj** statement justifies text between the current left and right indents. The statement applies to subsequent paragraphs up to the next **\pard**

statement or text-alignment statement.

Comments

If a **\ql**, **\qr**, **\qc**, or **\qj** statement is not given, the text is left-aligned by default.

See Also

\qc, \ql, \qr, \pard

\ql

The **\ql** statement aligns text along the left indent. The statement applies to subsequent paragraphs up to the next **\pard** statement or text-alignment statement.

Comments

If a **\ql**, **\qr**, **\qc**, or **\qj** statement is not given, the text is left-aligned by default.

See Also

\qc, \qj, \qr, \pard

\qr

The **\qr** statement aligns text along the right indent. The statement applies to subsequent paragraphs up to the next **\pard** statement or text-alignment statement.

Comments

If a **\ql**, **\qr**, **\qc**, or **\qj** statement is not given, the text is left-aligned by default.

See Also

\qc, \qj, \ql, \pard

\rdblquote

\rdblquote

The **\rdblquote** statement inserts a right double quotation mark if either Times New Roman or Arial has been specified in the \fonttable of the current rtf file. If neither font has been specified, Help Workshop substitutes a straight double-quote character. The actual quote character used depends on the language of the help file.

See Also

\ldblquote, \lquote, \rquote

\ri

\rin

The **\ri** statement sets the right indent for the paragraph. The indent applies to all subsequent paragraphs up to the next **\pard** or **\ri** statement.

Parameter	Description
n	Specifies the right indent, in twips. It can be a positive or negative value.

Comments

If the **\ri** statement is not given, the right indent is zero by default. WinHelp automatically provides a small right margin so that when no right indent is specified, the text does not end abruptly at the right edge of the Help window.

WinHelp never displays less than one word for each line in a paragraph even if the right indent is greater than the width of the window.

Example

In the following example, the right and left indents are set to one inch and the subsequent text is centered between the indents:

```
\li1440\ri1440\qc
Microsoft WinHelp\line
Sample File\line
```

See Also

\li, \pard

\row

The **\row** statement marks the end of a table row. The statement ends the current row and begins a new row by moving down pass the end of the longest cell in the row. The next **\cell** statement specifies the text of the left-most cell in the next row.

Example

The following example creates a table having four rows and two columns:

```
\cellx2880\cellx5760
\intbl
Row 1, Column 1\cell
Row 1, Column 2\cell \row
\intbl
Row 2, Column 1\cell
Row 2, Column 2\cell \row
\intbl
Row 3, Column 1\cell
Row 3, Column 2\cell \row
\intbl
Row 4, Column 1\cell
Row 4, Column 2\cell \row
\par \pard
```

See Also

\cell, \cellx, \intbl

\rquote

\rquote

The **\rquote** statement inserts a right single quotation mark if either Times New Roman or Arial has been specified in the \fonttable of the current rtf file. If neither font has been specified, Help Workshop substitutes a straight single-quote character in the current font.

See Also

\ldblquote, \lquote, \rdblquote

\rtf

\rtfn

The **\RTF token** identifies the file as a rich-text format (RTF) file and specifies the version of the RTF standard used.

Parameter	Description
n	Specifies the version of the RTF standard used.

Comments

The **\RTF token** must follow the first open brace in the Help file. A statement specifying the character set for the file must also follow the **\RTF token**.

See Also

\windows

\sa

\san

The **\sa** statement sets the amount of vertical spacing after a paragraph. The vertical space applies to all subsequent paragraphs up to the next **\pard** or **\sa** statement.

Parameter	Description
n	Specifies the amount of vertical spacing, in twips.

Comments

If the **\sa** statement is not given, the vertical spacing after a paragraph is zero by default.

See Also

\pard, \sb

\sb

\sbn

The **\sb** statement sets the amount of vertical spacing before the paragraph. The vertical space applies to all subsequent paragraphs up to the next **\pard** or **\sb** statement.

Parameter	Description
n	Specifies the amount of vertical spacing, in twips.

Comments

If the **\sb** statement is not given, the vertical spacing before the paragraph is zero by default.

See Also

\pard, \sa

\scaps

The **\scaps** statement starts small-capital text. The statement converts all subsequent lowercase letters to uppercase before displaying the text. This statement applies to all subsequent text up to the next **\plain** or **\scaps0** statement.

Comments

No **\plain** or **\scaps0** statement is required if the **\scaps** statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to just the enclosed text.

The **\scaps** statement does not reduce the point size of the text. To reduce point size, the **\fs** statement must be used.

Example

The following example displays the key name ENTER in small capitals:

```
Press the {\scaps ENTER} key to complete the action.
```

See Also

\plain

\sect

The **\sect** statement marks the end of a section and paragraph.

See Also

\par

\sl

\sln

The **\sl** statement sets the amount of vertical space between lines in a paragraph. The vertical space applies to all subsequent paragraphs up to the next **\pard** or **\sl** statement.

Parameter	Description
n	Specifies the amount of vertical spacing, in twips. If this parameter is a positive value, WinHelp uses this value if it is greater than the tallest character. Otherwise, WinHelp uses the height of the tallest character as the line spacing. If this parameter is a negative value, WinHelp uses the absolute value of the number even if the tallest character is taller.

Comments

If the **\sl** statement is not given or the specified amount of spacing is 1000, WinHelp automatically sets the line spacing by using the tallest character in the line.

See Also

\pard

\strike

The **\strike** statement creates a hotspot. The statement is used in conjunction with a **\v** statement to create a link to another topic. When the user chooses a hotspot, WinHelp displays the associated topic in the Help window.

The **\strike** statement applies to all subsequent text up to the next **\plain** or **\strike0** statement.

Comments

No **\plain** or **\strike0** statement is required if the **\strike** statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to just the enclosed text.

In print-based documents, or whenever it is not followed by **\v**, the **\strike** statement creates strikeout text.

Example

The following example creates a hotspot for a topic. When displayed, the hotspot text, "Hotspot," is green and has a solid line under it:

```
{\strike Hotspot}{\v Topic}
```

See Also

\ul, \uldb, \v

\tab

The **\tab** statement inserts a tab character (ASCII character code 9).

Comments

The tab character (ASCII character 9) has the same effect as the **\tab** statement.

See Also

\tb, \tqc, \tqc, \tx

\tb

The **\tb** statement advances to the next tab stop.

See Also

\tab, \tqc, \tqr, \tx

\tqc

The **\tqc** statement advances to the next tab stop and centers text.

See Also

\tab, \tb, \tqr, \tx

\tqr

The **\tqr** statement advances to the next tab stop and aligns text to the right.

See Also

\tab, \tb, \tqc, \tx

\trgaph

\trgaphn

The **\trgaph** statement specifies the amount of space between text in adjacent cells in a table. For each cell in the table, WinHelp uses the space to calculate the cell's left and right margins. It then uses the margins to align and wrap the text in the cell. WinHelp applies the same margin widths to each cell ensuring that paragraphs in adjacent cells have the specified space between them.

The **\trgaph** statement applies to cells in all subsequent rows of a table up to the next **\trowd** statement.

Parameter	Description
n	Specifies the space, in twips, between text in adjacent cells. If this parameter exceeds the actual width of the cell, the left and right margins are assumed to be at the same position in the cell.

Comments

The width of the left margin in the first cell is always equal to the space specified by this statement. The **\trleft** statement is typically used to move the left margin to a position similar to the left margins in all other cells.

Example

The following example creates a three-column table with one-quarter inch space between the text in the columns:

```
\trgaph360 \cellx1440\cellx2880\cellx4320
\intbl
Row 1 Column 1\cell
Row 1 Column 2\cell
Row 1 Column 3\cell \row
\intbl
Row 2 Column 1\cell
Row 2 Column 2\cell
Row 2 Column 3\cell \row \pard
```

See Also

\cell, \cellx, \intbl, \row, \trleft, \trowd

\trleft

\trleftn

The **\trleft** statement sets the position of the left margin for the first (left-most) cell in a row of a table. This statement applies to the first cell in all subsequent rows of the table up to the next **\trowd** statement.

Parameter	Description
n	Specifies the relative position, in twips, of the left margin. This parameter can be a positive or negative number. The final position of the left margin is the sum of the current position and this value.

Example

The following example creates a three-column table with one-quarter inch space between the text in the columns. The left margin in the first cell is flush with the left margin of the Help window:

? \trgaph360\trleft-360 \cellx1440\cellx2880\cellx4320
\intbl
Row 1 Column 1\cell
Row 1 Column 2\cell
Row 1 Column 3\cell \row
\intbl
Row 2 Column 1\cell
Row 2 Column 2\cell
Row 2 Column 3\cell \row \pard

See Also

\cell, \cellx, \intbl, \row, \trgaph, \trowd

\trowd

The **\trowd** statement sets default margins and cell positions for subsequent rows in a table.

See Also

\cell, \cellx, \intbl, \row, \trgaph, \trleft

\trqc

The **\trqc** statement directs WinHelp to dynamically adjust the width of table columns to fit in the current window.

Comments

In a print-based document, the **\trqc** statement centers a table row with respect to its containing column.

See Also

\trowd, \trql

\trql

The **\trql** statement aligns the text in each cell of a table row to the left.

See Also

\trowd, \trqc

\tx

\txn

The **\tx** statement sets the position of a tab stop. The position is relative to the left margin of the Help window. A tab stop applies to all subsequent paragraphs up to the next **\pard** statement.

Parameter	Description
n	Specifies the tab stop position, in twips.

Comments

If the **\tx** statement is not given, tab stops are set at every one-half inch by default.

See Also

\tab, \tb, \tqc, \tqr

\ul

The **\ul** statement creates a link to a pop-up topic. The statement is used in conjunction with a **\v** statement to create a link to another topic. When the user chooses the link, WinHelp displays the associated topic in a pop-up window.

The **\ul** statement applies to all subsequent text up to the next **\plain** or **\ul0** statement.

Comments

No **\plain** or **\ul0** statement is required if the **\ul** statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to just the enclosed text.

In print-based documents, or whenever it is not followed by **\v**, the **\ul** statement creates continuous underline.

Example

The following example creates a pop-up link for a topic. When displayed, the link text, "Pop-up Link," is green and has a dotted line under it:

```
{\ul Pop-up Link}{\v PopupTopic}
```

See Also

\strike, \uldb, \v

\uldb

The **\uldb** statement creates a hotspot. This statement is used in conjunction with a **\v** statement to create a link to another topic. When the user chooses a hotspot, WinHelp displays the associated topic in the Help window.

The **\uldb** statement applies to all subsequent text up to the next **\plain** or **\uldb0** statement.

Comments

No **\plain** or **\uldb0** statement is required if the **\uldb** statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to just the enclosed text.

In print-based documents, or whenever it is not followed by **\v**, the **\uldb** statement creates double underline.

Example

The following example creates a hotspot for a topic. When displayed, the hotspot text, "Hotspot," is green and has a solid line under it:

```
{\uldb Hotspot}{\v Topic}
```

See Also

\strike, \ul, \v

\v

{\v topic-ID**}**

The **\v** statement creates a link to the topic having the specified topic ID. The **\v** statement is used in conjunction with the **\strike**, **\ul**, and **\uldb** statements to create hotspots and links to topics.

Parameter	Description
topic-ID	Specifies the topic ID of a topic in the Help file. The string can be any combination of characters, except spaces, and must also be specified in a topic ID **\footnote** statement in some topic in the Help file.

Comments

If the topic ID is preceded by a percent sign (%), WinHelp displays the associated hotspot or link without applying the standard underline and color.

If the topic ID is preceded by an asterisk (*), WinHelp displays the associated hotspot or link with an underline but without applying the standard color.

In print-based documents, the **\v** statement creates hidden text.

For links or hotspots, the syntax of the **\v** statement is as follows:

[**%**|*****]context[>secondary-window][**@**filename]

In this syntax, secondary-window is the name of the secondary window to jump to. When the secondary window is not specified, the jump is to the same window as the current Help topic is using. To jump to the main Help window, specify "main" for this parameter. This parameter may not be used with pop-up window.

The filename parameter specifies a jump to a topic in a different Help file.

For a macro hotspot, the syntax of the **\v** statement is as follows:

[**%**|*****]**!**macro[**;**macro][**;...**]

Example

The following example creates a hotspot for the topic having the topic ID "Topic." WinHelp applies an underline and the color green to the text "Hotspot" when it displays the topic:

```
{\uldb Hotspot}{\v Topic}
```

See Also

\footnote, \strike, \ul, \uldb

\wbitmap

\wbitmapn

The **\wbitmap** statement sets the picture type to Windows bitmap. This statement must be used in conjunction with a **\pict** statement.

Parameter	Description
n	Specifies the bitmap type. This parameter is zero for a logical bitmap.

Comments

The **\wbitmap** statement is optional; if a **\wmetafile** statement is not specified, the picture is assumed to be a Windows bitmap.

Example

The following example creates a 32-by-8-pixel monochrome bitmap:

```
{
\pict \wbitmap0\wbmbitspixel1\wbmplanes1\wbmwidthbytes4\picw32\pich8
3FFFFFFC
F3FFFFCF
FF3FFCFF
FFF3CFFF
FFFC3FFF
FFCFF3FF
FCFFFF3F
CFFFFFF3
}
```

See Also

\bmc, \bml, \bmr, \pict, \wmetafile

\wbmbitspixel

\wbmbitspixeln

The **\wbmbitspixel** statement specifies the number of consecutive bits in the bitmap data that represent a single pixel. This statement must be used in conjunction with the **\pict** statement.

Parameter	Description
n	Specifies the number of bits per pixel.

Comments

If the **\wbmbitspixel** statement is not given, the default bits per pixel value is 1.

See Also

\pict, \wbitmap, \wbmplanes

\wbmplanes

\wbmplanesn

The **\wbmplanes** statement specifies the number of color planes in the bitmap data. This statement must be used in conjunction with a **\pict** statement.

Parameter	Description
n	Specifies the number of bitmap planes.

Comments

If the **\wbmplanes** statement is not given, the default number of planes is 1.

See Also

\pict, \wbitmap, \wbmbitspixel

\wbmwidthbytes

\wbmwidthbytesn

The **\wbmwidthbytes** statement specifies the number of bytes in each scan line of the bitmap data. This statement must be used in conjunction with the **\pict** statement.

Parameter	Description
n	Specifies the width of the bitmap, in bytes.

See Also

\pict, \wbitmap

\windows

The **\windows** statement sets the Windows character set.

Comments

If no **\windows**, **\pc**, or **\pca** statement is given in the Help file, the Windows character set is used by default.

See Also

\ansi, \pc, \pca

\wmetafile

\wmetafilen

The **\wmetafile** statement sets the picture type to a Windows metafile. This statement must be used in conjunction with the **\pict** statement.

Parameter	Description
n	Specifies the metafile type. This parameter must be 8.

Comments

WinHelp expects the hexadecimal data associated with the picture to represent a valid Windows metafile. By default, WinHelp sets the MM_ANISOTROPIC mapping mode prior to displaying the metafile. To ensure that the picture is displayed correctly, the metafile data must either set the window origin and extents by using the **SetWindowOrg** and **SetWindowExt** records or set another mapping mode by using the **SetMapMode** record.

Example

The following example creates a picture using a metafile:

```
{{\pict\wmetafile8\picw2880\pich2880
0100090000034f0000000200090000000000
050000000b0200000000050000000c026400
6400090000001d066200ff00640064000000
000008000000fa020000020000000000000
```

040000002d0100000500000140200000000
0500000013026400640005000000140200000
64000500000013026400000008000000fa02
000000000000000000000040000002d010100
04000000f00100000300000000004e0dff00
870020000050000020000000000000000000}
\par }

See Also

\bmc, \bml, \bmr, \pict, \wbitmap

No RTF codes begin with this letter.

Converting from Viewer 2.0 to WinHelp 4.0

Perform the following procedures to convert a Microsoft Viewer 2.0 title to a Microsoft Windows Help version 4.0 file:

- Update macro references.
- Update multimedia references.
- Convert the Viewer project file (.MVP) to a Help Workshop project file.
- Create a contents file.
- Build the new Help (.HLP) file.

Notes

- WinHelp does not support panes, groups, or fields
- WinHelp has its own (optional) full-text search indexer and user interface.
- WinHelp supports the **LDLLHandler** callback function, but does not support other Viewer APIs.

{button ,AL("from_V2_overview")} Related Topics

Update Macro References

Help Workshop ignores unsupported macros in the Help project file, but alerts you to their presence with notes and warning messages. One or more of the following procedures may be necessary:

- Remove Viewer-specific macros.
- Update macros that have a slightly different form in WinHelp.

{button ,AL("from_v2_macros")} Related Topics

Unsupported Viewer Macros

The following Viewer macros are not supported in WinHelp version 4.0:

Viewer Macro	Reason
ChangeCitation	WinHelp supports only a single citation (specified in the project (.HPJ)file.)
ClosePane	Panes are not supported in WinHelp 4.0.
CopyBmp	WinHelp cannot copy images to the Clipboard.
DeleteMenu	Menus can be added or reset in WinHelp 4.0, but not deleted.
DeleteSeparator	Use the DeleteItem macro.
FTSearch	Use the Find macro to provide access to the full-text search user interface.
HideButtonBar	The WinHelp main window button bar can be turned off in the window definition in the project file.
HideMenuBar	WinHelp secondary windows do not display menus.
Highlight	WinHelp version 4.0 does not highlight the words found in full-text searches.
InsertButton	WinHelp does not support specific button positioning.
InsertSeparator	Use the ExtInsertItem macro.
JumpAddr	WinHelp provides its own full-text search.
JumpCookie	WinHelp uses hash codes rather than addresses.
KeyIndex	WinHelp uses the Search macro to display the keyword Index tab in the Help Topics dialog box.
MasterAspect	WinHelp does not support panes.
MasterNSRColor	Colors for nonscrolling regions are part of window definitions, which are specified in the project file.
MasterSRColor	Color for scrolling regions are part of window definitions, which are specified in the project file.
MCICommand	In WinHelp, MCI support is provided by the **{mci}** topic file reference.
PaneId	WinHelp does not support panes.
PositionMaster	WinHelp does not support panes.
PositionTopic	WinHelp provides macro support only for device-independent window positioning (using the PositionWindow macro).
SearchDialog	WinHelp provides a Search macro to display the Index tab in the Help Topics dialog box.
ShowButtonBar	The WinHelp main window button bar is configurable in the window definition in the project file.
ShowMenuBar	Menus are not configurable in WinHelp
Std20Buttons	WinHelp provides its own standard buttons for the main window.
Std20Menus	WinHelp provides its own standard menus for the main window.
SwitchToTopicsFound	WinHelp provides its own full-text search user interface, which does not include a separate Topics Found dialog box.
WindowAspect	WinHelp does not support panes.

Help Workshop ignores these unsupported macros in the Help project file, but alerts you to their presence with notes and warning messages.

{button ,AL("CITATION;Search;Finder;PositionWindow;mci")} Related Topics

WinHelp-Specific Macros

The following macros are not provided in Viewer 2.0, but are supported in WinHelp version 4.0.

WinHelp Macro	Purpose
ALink	Searches for matching 'A' keywords in the current (and associated) Help (.HLP) files. New for WinHelp 4.0.
ChangeEnable	Assigns a macro to a navigation bar button and enables that button. New for WinHelp 4.0.
CloseSecondarys	Closes all secondary windows. New for WinHelp 4.0.
Compare	Displays a Help file in a second instance of WinHelp. New for WinHelp 4.0.
ControlPanel	Displays the specified control panel program. New for WinHelp 4.0.
EndMPrint	Dismisses the printing message box and terminates the printing of multiple topics. New for WinHelp 4.0.
ExecFile	Runs a Windows-based program with a specified file. New for WinHelp 4.0.
ExtAbleItem	Enables or disables a menu item.
ExtInsertItem	Allows the initial state of a menu item to be set as enabled or disabled.
ExtInsertMenu	Allows submenus (cascading menus) to be created.
FileExist	Checks to see whether the specified file exists on the user's computer. New for WinHelp 4.0.
Finder	Displays the Help Topics dialog box. New for WinHelp 4.0.
FloatingMenu	Displays the floating menu at the current mouse position.
Flush	Causes WinHelp to process any pending messages.
Generate	Sends a message to the currently active Help window.
HelpOn	Displays the Help file for the WinHelp program.
HelpOnTop	Toggles the on-top state of WinHelp.
InitMPrint	Begins printing multiple topics. New for WinHelp 4.0.
IsBook	Determines whether WinHelp is being run standalone or called from a program. New for WinHelp 4.0.
IsNotMark	Tests whether or not a marker set by SaveMark exists. Same as Not(IsMark(text)). New for WinHelp 4.0.
JumpContext	Jumps to a specific context (specified in the [MAP] section of the project (.HPJ)file) in a Help file.
JumpHash	Jumps to the topic specified by the hash code.
JumpHelpOn	Jumps to the contents topic of the How to Use Help file.
KLink	Searches for matching keywords in the current (and associated) Help file(s). New for WinHelp 4.0.
Menu	Displays the context (right-click) menu for the current Help window. New for WinHelp 4.0.
NoShow	Causes the current topic (and the Help window) to NOT display. New for WinHelp 4.0.
PopupContext	Displays a specified topic in a pop-up window.
PopupHash	Displays the topic, specified by the hash code, in a pop-up window.
RemoveAccelerator	Removes the assignment made with AddAccelerator.
ResetMenu	Restores the WinHelp menus to their default states.
Search	Displays the Help Topics dialog box with the Index tab selected. New for WinHelp 4.0.
SetContents	Designates a specific topic as the contents topic in the Help file.
SetHelpOnFile	Designates the name of the "How To Use Help" file, if different from the standard file.
SetPopupColor	Sets the background color for pop-up windows. New for WinHelp 4.0.
ShellExecute	Opens or prints the specified file. New for WinHelp 4.0
ShortCut	Runs a program and sends it a WM_COMMAND message. New for WinHelp 4.0.
TCard	Sends a message to the program that is invoking WinHelp as a training card. New for WinHelp 4.0.
Test	Tests the current Help file. New for WinHelp 4.0.
UpdateWindow	Jumps to the topic with the specified context string in the specified window and returns the focus to the window that called the macro. New for WinHelp 4.0.

{button ,AL("SaveMark;AddAccelerator")} Related Topics

Differences in Macros

The following macros are provided in both Viewer 2.0 and WinHelp version 4.0, but in slightly different formats.

Macro	Change
AddAccelerator	Uses slightly different virtual key (VK_*) codes.
PopupId	Does not support the pop-up window specification.

{button ,AL("REF_STD_BTN;REF_STD_MENU")} Related Topics

Update Multimedia References

Help authors include multimedia files (AVI, WAV, MIDI, and so on) in Help topics by using the **{mci}** statement in the topic (.RTF) files.

WinHelp version 4.0 supports all the multimedia APIs used by Microsoft Video for Windows version 1.1.

{button ,AL("BAS_ADDING_SOUND;BAS_ADDING_VIDEO;mci")} Related Topics

Convert the Project File

To convert your Viewer project file into a WinHelp version 4.0 project (.HPJ) file:

1 Rename your Viewer project file, replacing the .MVP filename extension with .HPJ filename extension.

2 Run Help Workshop and open the renamed project file.

3 Edit the project file as necessary.

4 Compile the project file. Help Workshop will display messages if there are problems.

Notes

- Some Viewer options and sections are not supported by the version 4.0 Help Workshop.
- You can add WinHelp-specific options and sections by using Help Workshop.
- Use Help Workshop to make changes and additions to the project file. Editing the project file as a text file can introduce serious malfunctions.

{button ,AL("from_v2_update_hpj")} Related Topics

Unsupported Viewer Project File Options

The following Viewer project file sections are not supported by Help Workshop version 4.0. If present in your project (.HPJ) file, Help Workshop ignores unsupported sections in the Help project file, but alerts you to their presence with notes and warning messages.

- [FTINDEX]
- [GROUPS]
- [KEYINDEX]
- [PANEASSOC]
- [PANES]
- [POPUPS]
- [SEARCHDLG]
- [WWHEEL]

The following options in the **[OPTIONS]** section of the project file are not supported by Help Workshop. The compiler ignores these options in the Help project file, but alerts you to their presence with notes and warning messages.

- ALIAS
- BATCH
- ICON
- IGNORESECTION
- MAKE
- OPERATOR
- SYSTEM
- WARNING

{button ,AL("HPJ_Sections;OVERVIEW;V2_WINHELP_PROJECT_FILE;V2_CHANGED_PROJECT_FILE;V2_UPDATE_PROJECT_FILE;V2_NEW_PROJECT_FILE")} Related Topics

WinHelp-specific Project File Options

These options are supported by Help Workshop but not the Viewer compiler. Use Help Workshop to add or edit these options.

Option	Description
CHARSET	Specifies the character set to use for all fonts.
CNT	Specifies the name of the contents (.CNT) file. to associate with the Help (.HLP) file.
DBCS	Specifies that the topic (.RTF) files use a double-byte character set
DEFFONT	Specifies the default font to use in WinHelp.
ERRORLOG	Specifies the name of the log file to create when compiling the Help file.
FORCEFONT	Forces WinHelp to use the specified font when displaying the Help file, in place of the fonts used in the topic files.
FTS	Specifies the level of information to include in the full-text search index file.
HCW	Reserved for use by Help Workshop.
HLP	Specifies the name of the Help file to create.
INDEX_SEPARATORS	Identifies the characters used to denote separate keyword entries in topics.
LCID	Specifies the locale ID.
MAPFONTSIZE	Maps font sizes specified in the topic files to different sizes when displayed in the WinHelp window.
MULTIKEY	Customizes the keyword search so that it uses more than one keyword table.
NOTES	Specifies whether or not the compiler will display any notes during compilation.
OLDKEYPHRASE	Specifies whether or not to create a new keyword phrase (.PH) file.
REPLACE	Specifies a path prefix to replace and its replacement.
REPORT	Specifies whether or not to display messages when compiling the Help file.
TMPDIR	Specifies the folder to place the temporary files created while compiling the Help file.

{button ,AL("V2_UPDATE_PROJECT_FILE;V2_CHANGED_PROJECT_FILE;V2_UNSUPPORTED_PROJECT_FILE;V2_NEW_PROJECT_FILE")} Related Topics

Differences in Project File Options

The following parameters have the same name in both Viewer and WinHelp, but are slightly different:

The COMPRESS option in the [OPTIONS] section has slightly different values for Help Workshop version 4.0. Help Workshop will automatically change this value to 0, so you must specifically set the compression yourself.

The window definitions in the **[WINDOWS] section** of the Help project file use a different format.

{button ,AL("WINDOWS;COMPRESS;HPJ_Sections;V2_UPDATE_PROJECT_FILE;V2_WINHELP_PROJECT_FILE;V2_UNSUPPORTED_PROJECT_FILE;V2_NEW_PROJECT_FILE")} Related Topics

WinHelp-specific Project File Sections

The following sections are supported by Help Workshop version 4.0, but not by the Viewer compiler. Use Help Workshop to add or edit these options.

[MACROS] Section

The **optional [MACROS] section associates** macros with keywords. The specified macros are run when a user selects an associated keyword from the index.

[CONFIG:name] Section

The optional [CONFIG:name] section is similar to the [CONFIG] section, except any specified macros are run when a particular window name is opened. The x in the section name is a number that associates the section with a particular window definition. This number is assigned by the Project File Editor.

{button ,AL("MACROS;CONFIG_x;V2_UPDATE_PROJECT_FILE;V2_CHANGED_PROJECT_FILE;V2_UNSUPPORTED_PROJECT_FILE;V2_WINHELP_PROJECT_FILE")} Related Topics

Create a Contents File

WinHelp version 4.0 supports the use of a contents (.CNT) file, a text file that specifies what topics display in the Contents tab in the Help Topics dialog box. The Contents tab replaces the "contents topic" used in both former versions of WinHelp and Viewer version 2.0. The contents file also allows authors to create a combined index using the keywords in other Help (.HLP) files, and to specify which Help files to search when the user clicks a hotspot that runs the ALink or KLink macros.

Help Workshop can be used to create or edit contents files.

{button ,AL("BAS_CNT_OVERVIEW;BAS_CONTENTS_FILE_EDITOR;ALink;KLink")} Related Topics

Build the New Help File

Once you have updated your topic (.RTF) files and project (.HPJ)files, use Help Workshop version 4.0 to compile the project file and build the Help (.HLP) file. Just click Compile on the File menu in HCW and specify the name of the project file.

{button ,AL("BAS_THE_HELP_COMPILER;BAS_TESTING_HELP_FILES")} Related Topics

HC1000: Note: A keyword footnote has been specified without a keyword.

Problem: There are no characters in the keyword footnote in the topic (.rtf) files.

Result: Help Workshop ignores the keyword footnote. If there are no additional keyword footnotes, the topic will be inaccessible from the Index tab in the Help Topics dialog box.

Solution: Add one or more keywords to the keyword footnote, and then recompile.

{button ,AL(`BAS_ADDING_KEYWORDS')} Related Topics

HC1001: Note: The keyword [ ] is already defined in this topic.

Problem: The same keyword has be defined more than once in a topic.

Result: Help Workshop ignores the duplicate keyword.

Solution: Open the topic and remove the duplicate.

{button ,AL(`BAS_ADDING_KEYWORDS')} Related Topics

HC1002: Note: Using existing phrase table: [ ].

Problem: Help Workshop found an existing phrase (.ph) table for the Help file you are building, and the project (.hpj) file does not specify not to use the table.

Result: Help Workshop uses the phrase table that was generated from an earlier build instead of creating a new table.

Solution: To achieve the best phrase compression when compiling your Help file, you must delete the old key-phrase table before each recompilation, or clear the "Use old phrase file" option in the Compression tab in the Options dialog box of Help Workshop.

{button ,AL(`OLDKEYPHRASE;BAS_COMPRESSION')} Related Topics

HC1003: Note: A paragraph mark is formatted as a hidden character.

Problem: A paragraph mark has been formatted as hidden text. This often occurs when a hotspot is the last word in a paragraph.

Result: Help Workshop ignores the paragraph mark, so the two paragraphs will run together.

Solution: Reformat the paragraph marker as plain text in the topic file, and then recompile.

HC1004: Note: A previous instance of [ ] does not contain [ ].

Problem: A {bmx} command specifying only one resolution version of a bitmap is followed later in the topic files by a {bmx} command specifying that version plus one or more additional versions.

Result: WinHelp will treat the earlier single-resolution command as if it were the same as the command specifying multiple resolutions.

Solution: Replace the single resolution command with a copy of the multiple-resolution command.

{button ,AL(`BAS_MULT_COLOR_GRAPHICS;bmx')} Related Topics

HC1005: Note: [ ] is not an unsigned number for the macro [ ]. The sign will be ignored.

Problem: The parameter for this macro should be unsigned, but the number has been prefixed with a minus character.

Result: Help Workshop will remove the minus sign when it creates the Help file.

Solution: Remove the minus sign from the macro definition, and then recompile.

{button ,AL(`syntax_more')} Related Topics

HC1006: Note: The ExecProgram macro has been used instead of ExecFile.

Problem: The **ExecProgram** macro has been used.

Result: The ExecProgram macro ignores the restricted programs that can be set in Windows 95. It will not search the registry from programs, so some programs cannot be launched with this command.

Solution: Replace instances of ExecProgram macro commands with ExecFile macro commands, and then recompile.

{button ,AL("ExecProgram;ExecFile")} Related Topics

HC1007: : The include tag [ ] has been specified in the [EXCLUDE] section.

Problem: You have specified the same tag in the [EXCLUDE] section as you have in the [INCLUDE] section.

Result: Help Workshop ignores the tag in the [EXCLUDE] section.

Solution: Remove the tag from either the [EXCLUDE] or the [INCLUDE] section.

{button ,AL(`bmx;PRO_ADD_PICTURE;BAS_ADDING_GRAPHICS')} Related Topics

HC1008: Note: Ignoring the transparent flag on the monochrome bitmap [ ].

Problem: The transparent flag has been specified for a monochrome bitmap. It may only be used with a 16-color bitmap.

Result: Help Workshop ignores the flag.

Solution: Remove the transparent flag.

{button ,AL(`bmx;PRO_ADD_PICTURE;BAS_ADDING_GRAPHICS')} Related Topics

HC1009: Note: The map entry for [ ] has text after the number: [ ].

Problem: Text occurs after the numeric value specified in a map entry.

Result: Help Workshop ignores the text following the number in the entry.

Solution: Use Help Workshop to make sure the map entry is correctly entered in the project file. (Comments must be preceded by a semicolon, which Help Workshop adds automatically.) Recompile after making any changes.

{button ,AL(`MAP;PRO_HPJ_ADD_MAP')} Related Topics

HC1010: Note: The following mapped topic IDs are not used in any topic: [ ].

Problem: Entries appear in the [MAP] section for which you have no topics.

Result: If a program tries to use these mapped IDs, WinHelp will display an error message.

Solution: Check the listed entries to make sure they were not mistyped. Remove any IDs that are not actually used, and add topics for those that are.

{button ,AL(`MAP;PRO_HPJ_ADD_MAP')} Related Topics

HC1011: Note: The right-justified table row style is not supported.

Problem: A table style has been specified that WinHelp does not support.

Result: Help Workshop ignores the style.

Solution: Remove the right-justified style.

HC1012: Note: Table cell borders are not supported.

Problem: A table format has been specified that WinHelp does not support.

Result: Help Workshop ignores table cell borders.

Solution: Remove table cell borders in the topic file(s), and then recompile.

HC1013: Note: The "[ ]" section is missing a right bracket.

Problem: A section in your project file begins with a left bracket, but does not end with a right bracket.

Result: Help Workshop ignores the section heading on this line. The following lines will be processed as if they are part of the previous section which may result in errors.

Solution: Insert the right section heading bracket or remove the line entirely, and then recompile.

{button ,AL(`HPJ_Sections')} Related Topics

HC1014: Note: The multikey value [ ] has already been defined.

Problem: A character used for indicating an alternate keyword table has been used already.

Result: Help Workshop ignores the duplicate keyword table assignment on this line in the project file.

Solution: Remove the duplicate character, and then recompile.

{button ,AL(`MULTIKEY')} Related Topics

HC1015: Note: A page break is formatted as a hidden character.

Problem: A page break has been formatted as hidden text.

Result: Help Workshop ignores the page break. If the page break was used to separate two topics, the topics will appear as one.

Solution: Reformat the page break in the topic file as plain text, and then recompile.

HC1016: Note: A carriage return is formatted as a hidden character.

Problem: A carriage return has been formatted as hidden text.

Result: Help Workshop ignores the carriage return and does not create a new line.

Solution: Reformat the carriage return in the topic file as plain text, and then recompile.

HC1017: Note: A nonscrolling region crosses a page break.

Problem: The Keep With Next paragraph formatting used for specifying a non-scrolling region crosses a page break boundary.

Result: The topic beginning with the page break will have a nonscrolling region, whether or not one is intended.

Solution: Remove the Keep With Next paragraph formatting from the page break, and then recompile.

HC1019: Note: The Language option is obsolete. It should be replaced with the LCID option.

Problem: The LANGUAGE option has been used. This option has been made obsolete by the LCID option which should be used instead.

Result: Help Workshop will attempt to convert the LANGUAGE option into an appropriate LCID option.

Solution: In Help Workshop, open the project file. Help Workshop will automatically remove the LANGUAGE option and replace it with an LCID option.

{button ,AL(`BAS_PROJ_FILE_EDITOR;LCID;LANGUAGE')} Related Topics

HC1024: Note: The [ ] option has been specified more than once.

Problem: The same option has been specified more than once.

Result: Help Workshop uses the last definition for the option and ignores the others.

Solution: Remove duplicate option entries from the project file, and then recompile.

{button ,AL(`HPJ_Overview;OPTIONS')} Related Topics

HC2002: Note: The keywords [ ] and [ ] differ only by case. Help Workshop has modified one keyword to match the other.

Problem: Two keywords are identical except for case.

Result: Help Workshop automatically changes the case of one of the keywords to agree with that of another otherwise identical keyword. For example: The uppercase letters in the keyword "MyKeyword" may be changed to lowercase letters in order to agree with the keyword, "mykeyword."

Solution: Change the keywords so that they all use the same case, and then recompile.

{button ,AL(`BAS_ADDING_KEYWORDS')} Related Topics

HC3000: Warning: The keyword [ ] is longer than [ ] characters.

Problem: A single keyword exceeds 255 characters.

Result: Help Workshop ignores the keyword.

Solution: Check for a missing semicolon or shorten the length of the keyword string, and then recompile.

{button ,AL(`BAS_ADDING_KEYWORDS')} Related Topics

HC3001: The file [ ] does not contain a valid icon.

Problem: You have specified an icon using the ICON option in the [OPTIONS] section of your project file, however, the file does not contain a valid icon image.

Result: Help Workshop ignores icon.

Solution: Since WinHelp 4.0 doesn't use the icon anyway, simply remove the ICON option from your project file and recompile..

{button ,AL(`syntax_more')} Related Topics

HC3002: Warning: [ ] is a number but should be a string for the macro [ ].

Problem: The parameter for the should be a string, but it starts with a numeric character.

Result: Help Workshop ignores the rest of the macro. WinHelp will not be able to run the macro.

Solution: If the parameter really is a string that begins with a number, then you must place the string in quotations marks (`').

{button ,AL(`syntax_more')} Related Topics

HC3003: Warning: The macro name [ ] is invalid.

Problem: Help Workshop does not recognize the macro you have entered.

Result: WinHelp may not be able to run the macro.

Solution: If you meant to use a WinHelp macro, confirm that you have spelled it correctly. Note that macro names are always entered in English no matter what language the text of the topic file is in. If the macro is supplied by a dynamic-link library (DLL), make certain you have registered the macro name using the RegisterRoutine macro in the [CONFIG] section of your project file.

{button ,AL(`syntax_more')} Related Topics

HC3004: Warning: The variable used in the macro [ ] does not match the macro's argument type.

Problem: You have used a string for a numeric parameter or a number for a string parameter.

Result: WinHelp will not be able to run the macro.

Solution: Replace the incorrect parameter types, and then recompile. If the macro is defined in a custom dynamic-link library (DLL), check the RegisterRoutine macro in the project file for the correct parameter types.

{button ,AL(`syntax_more;RegisterRoutine')} Related Topics

HC3005: Warning: The macro variable [ ] is undefined.

Problem: The specified macro contains a variable that is not recognized by Help Workshop.

Result: Help Workshop ignores the rest of the macro. WinHelp will not be able to run the macro.

Solution: Replace incorrect variables, and then recompile.

The following is a list of valid macro variables:

- hwndContext
- hwndApp
- qchPath
- qError
- lTopicNo
- hfs
- coForeground
- coBackground

{button ,AL(`syntax_more;RegisterRoutine')} Related Topics

HC3006: Warning: Missing comma in authorable button command.

Problem: You have specified a {button } command without specifying a comma followed by the macro to run when the button is clicked.

Result: Help Workshop ignores the button command.

Solution: Look up the {button } command in Related Topics and correct the syntax.

{button ,AL(`syntax_more;button')} Related Topics

HC3007: Warning: [ ] is an invalid display-state parameter for the macro [ ].

Problem: A non-supported display state has been specified for a macro that launches a program.

Result: Help Workshop ignores the rest of the macro. WinHelp may not be able to run the macro.

Solution: Look up the macro in the Related Topics, and use one of the documented display states.

{button ,AL(`syntax_more;ExecProgram;ExecFile')} Related Topics

HC3008: Warning: Missing double quotation mark in macro [ ].

Problem: There is an unmatched double quotation mark in the macro.

Result: Help Workshop ignores the rest of the macro. WinHelp cannot run the macro.

Solution: Add the missing double quotation mark and then recompile. For most macros, it is not necessary to supply quotations marks as Help Workshop will supply them for you.

{button ,AL(`syntax_more')} Related Topics

HC3009: Warning: Missing end quotation mark (') in macro [ ].

Problem: A start quotation mark has been specified without an end quotation mark.

Result: Help Workshop ignores the rest of the macro. WinHelp cannot run the macro.

Solution: Add the missing single quotation mark to the macro definition and then recompile. For most macros, it is not necessary to supply quotations marks as Help Workshop will supply them for you.

{button ,AL(`syntax_more')} Related Topics

HC3010: Warning: The macro name [ ] is undefined.

Problem: The macro name you have used is not a valid WinHelp macro and has not been defined by a RegisterRoutine macro.

Result: Help Workshop ignores the rest of the macro. WinHelp cannot run the macro.

Solution: If this is a macro that calls a dynamic-link library (DLL) function, add the RegisterRoutine macro to the project file that defines this function name. Otherwise, you have probably misspelled the macro name.

{button ,AL(`syntax_more;RegisterRoutine')} Related Topics

HC3011: Warning: The accelerator [ ] has already been added.

Problem: The accelerator you have used in the AddAccelerator macro has already been used.

Result: Help Workshop ignores the rest of the macro. You may get a syntax error if WinHelp tries to run the macro.

Solution: Use another accelerator key. Using the same format for accelerators will help avoid duplication. For example, ‘A’, 65, and 0x41 are all the same accelerator.

{button ,AL(`addaccelerator')} Related Topics

HC3012: Warning: The macro [ ] does not include a window name.

Problem: The macro requires a window name, and you have not specified one.

Result: Help Workshop ignores the rest of the macro. WinHelp cannot run the macro.

Solution: Supply the window name in the macro.

{button ,AL(`UpdateWindow')} Related Topics

HC3013: Error: [ ] is not a valid numeric parameter for the macro [ ].

Problem: The parameter for the specified macro can only be a numeric value or a macro that returns a numeric value, and you have specified a string. Some macros accept a limited type of string parameter in place of a number, but if you do not use one of those values, you get an error.

Result: Help Workshop ignores the rest of the macro. WinHelp cannot run the macro.

Solution: Change the macro parameter in the topic file to a numeric value, and then recompile.

{button ,AL(`syntax_more')} Related Topics

HC3014: Warning: Window name [ ] has already been used.

Problem: The same window type has been specified more than once in the [WINDOWS] section of the project file.

Result: Only the first definition of the window is used.

Solution: Run Help Workshop and open the project file. You are prompted for a correction to the duplicate entry. Make the correction, and then recompile.

{button ,AL(`WINDOWS;PRO_HPJ_CREATE_WINDOW;PRO_HPJ_CUSTOMIZE_WINDOW')} Related Topics

HC3015: Warning: This window position is invalid: [ ].

Problem: An invalid window position has been used. The window position in a window definition consists of four numbers that define the window's location on the screen and its width and height. The four numbers must be given in WinHelp's 1024-by-1024 coordinate system, and they must be enclosed in parentheses and separated by commas.

Result: Help Workshop ignores the window definition.

Solution: In Help Workshop, open the project file. Click the Windows button and then click the Position tab. Adjust the position as necessary.

{button ,AL(`WINDOWS;PRO_HPJ_CREATE_WINDOW;PRO_HPJ_CUSTOMIZE_WINDOW')} Related Topics

HC3016: Warning: Invalid syntax for authorable button: [ ].

Problem: The syntax you have used for an authorable button is invalid.

Result: Help Workshop ignores the {button} statement.

Solution: Correct the syntax, and then recompile. Typically the problem occurs by leaving out a comma before the macro definition. The correct syntax for an authorable button is:

{button button name**,** macro()}

{button ,AL(`button')} Related Topics

HC3017: Warning: Topic ID for [ ] macro contains invalid characters: [ ].

Problem: There is an invalid character in the topic ID.

Result: Help Workshop ignores the rest of the macro. WinHelp cannot run the macro.

Solution: Replace the invalid characters in the topic ID, and then recompile.

{button ,AL(`BAS_ADDING_TOPIC_ID')} Related Topics

HC3018: Warning: In the project file, an #include statement is specified without a filename.

Problem: An #include line has been specified in the project file, but the line did not include a filename.

Result: Help Workshop ignores the #include line.

Solution: Either delete the line or add a correct filename to the statement, and then recompile.

{button ,AL(`HPJ_Overview')} Related Topics

HC3019: Warning: Missing quotation mark or parenthesis in the macro [ ].

Problem: There is an unmatched quotation mark or parenthesis.

Result: Help Workshop ignores the rest of the macro. WinHelp cannot run the macro.

Solution: Add the missing quotation or right parenthesis to the macro definition in the topic file. Help Workshop will usually supply the quotes for you — deleting all the quotes in the macro can be an effective method for fixing this problem.

{button ,AL(`syntax_more')} Related Topics

HC3021: Warning: Cannot find the bitmap [ ].

Problem: Help Workshop is unable to find the bitmap you have specified.

Result: Help Workshop creates a temporary bitmap containing the filename of the missing bitmap.

Solution: If the bitmap name is valid, add the folder it is contained in to the list of bitmap folders (click the Bitmaps button in Help Workshop).

{button ,AL(`BMROOT;ROOT')} Related Topics

HC3022: Warning: The bitmap [ ] has already been used as a non-transparent bitmap.

Problem: A bitmap cannot be transparent in one instance and non-transparent in another.

Result: The bitmap will always be displayed as non-transparent.

Solution: Change either bitmap command so that they are both transparent or both non-transparent. Alternatively, you can create a new file for a bitmap.

{button ,AL(`bmx;BAS_ADDING_GRAPHICS')} Related Topics

HC3023: Warning: The bitmap [ ] has already been used as a transparent bitmap.

Problem: A bitmap cannot be transparent in one instance and non-transparent in another.

Result: The bitmap will always be displayed as transparent.

Solution: Change either bitmap command so that they are both transparent or both non-transparent. Alternatively, you can create a new file for a bitmap.

{button ,AL(`BAS_ADDING_GRAPHICS;bmx')} Related Topics

HC3024: Warning: The window name [ ] contains more than eight characters and has been truncated to [ ].

Problem: The window name is too long.

Result: Help Workshop truncates the name to 8 characters. If this results in a duplicate window name, you will get another warning message.

Solution: Run Help Workshop and open the project file. You will be prompted for a correction to the window name. Make the correction, and then recompile the project file.

{button ,AL(`WINDOWS;PRO_HPJ_CREATE_WINDOW;PRO_HPJ_CUSTOMIZE_WINDOW')} Related Topics

HC3025: Warning: Jump to undefined topic ID: [ ].

Problem: No topic has been defined with the topic ID specified in the hotspot.

Result: WinHelp displays the "Topic not found" error message when the user chooses a hotspot with the unresolved topic ID.

Solution: Check for spelling errors in the hotspot topic ID.

{button ,AL(`BAS_INSERTING_JUMPS;BAS_INSERTING_POPUP')} Related Topics

HC3026: Warning: The topic ID [ ] has already been defined in topic [ ] in file [ ].

Problem: Another topic exists with an identical topic ID.

Result: Help Workshop ignores the duplicate topic ID, so the duplicate topic has no identifier.

Solution: Change the duplicate topic ID so that it is unique, and then recompile.

{button ,AL(`BAS_ADDING_TOPIC_ID')} Related Topics

HC3027: Warning: The following alias line does not contain an '=' character separating the topic IDs: [ ].

Problem: An invalid line has been entered in the [ALIAS] section of the project file.

Result: Help Workshop ignores the alias.

Solution: Run Help Workshop and open the project file. You will be prompted for a correction to the specified alias line. Make the correction and then recompile.

{button ,AL(`ALIAS;PRO_HPJ_ADD_ALIAS')} Related Topics

HC3028: Warning: The bitmap file [ ] is corrupted.

Problem: Either the file is corrupt, or it has not been saved as an uncompressed bitmap (.bmp) file.

Result: Help Workshop ignores this bitmap, and WinHelp displays the “Unable to display picture” message in the topic instead of the bitmap.

Solution: Replace the bitmap file with one that is not corrupt, or remove the bitmap reference in the topic file.

{button ,AL(`BAS_ADDING_GRAPHICS')} Related Topics

HC3029: Warning: The file format of the bitmap file [ ] is unrecognized or unsupported.

Problem: Either the file is corrupt, or it has not been saved as an uncompressed bitmap (.bmp) file.

Result: Help Workshop ignores this bitmap, and WinHelp displays the "Unable to display picture" message in the topic instead of the bitmap.

Solution: Convert the bitmap to a supported format, and then recompile.

{button ,AL(`BAS_ADDING_GRAPHICS')} Related Topics

HC3031: Warning: Help Workshop does not support compressed BMP files: [ ].

Problem: The bitmap was saved with RLE compression.

Result: Help Workshop ignores this bitmap, and WinHelp displays the “Unable to display picture” message in the topic instead of the bitmap.

Solution: Save the bitmap without compression.

{button ,AL(`BAS_ADDING_GRAPHICS')} Related Topics

HC3032: Warning: The bitmap file [ ] is truncated.

Problem: The size of the bitmap does not match the size of the file. The bitmap file is corrupt.

Result: Help Workshop ignores this bitmap, and WinHelp displays the “Unable to display picture” message in the topic instead of the bitmap.

Solution: Resave or recreate the specified bitmap file.

{button ,AL(`bmx;BAS_ADDING_GRAPHICS')} Related Topics

HC3033: Warning: The topic ID [ ] contains invalid characters.

Problem: There is an invalid character in the topic ID footnote.

Result: Help Workshop does not assign an identifier to the topic.

Solution: In the topic file, change the characters in the topic ID so that it is valid, and then recompile.

{button ,AL(`BAS_ADDING_TOPIC_ID')} Related Topics

HC3034: Warning: Invalid parameter for TCard macro: [ ].

Problem: The parameter specified for the TCard macro is invalid.

Result: Help Workshop ignores the macro, and WinHelp returns an error when it tries to run the macro.

Solution: Correct the invalid TCard macro definition in the topic file, and then recompile.

{button ,AL(`TCard;syntax_more')} Related Topics

HC3035: Warning: Map entry does not specify a numeric value: [ ].

Problem: Help Workshop expected a numeric value, and encountered text instead.

Result: Help Workshop ignores the entry.

Solution: Change the entry to a numeric value.

{button ,AL(`MAP;PRO_HPJ_ADD_MAP')} Related Topics

HC3037: Warning: The map value for [ ] is the same as the map value for [ ].

Problem: A context number in the [MAP] section of the project (.hpj) file was previously mapped to a different topic ID.

Result: Help Workshop ignores the line with the duplicate context number.

Solution: Verify that all context numbers in the project file are unique. Remove or reassign any duplicate context numbers, and then recompile.

{button ,AL(`MAP;PRO_HPJ_ADD_MAP')} Related Topics

HC3038: Warning: The following topic IDs were not defined in the [MAP] section of the project file.

Problem: Help Workshop cannot find the specified topic IDs listed in the [MAP] section of the project (.hpj) file.

Result: Help Workshop ignores the lines in the project file. If the program uses that number in the WinHelp function, WinHelp displays the "Help topic not found" error message.

Solution: Verify that all topic IDs listed in the [MAP] section of the project file are assigned to topics included in the build and that the topic IDs are spelled correctly. Then recompile the project file.

{button ,AL(`MAP;PRO_HPJ_ADD_MAP')} Related Topics

HC3041: Warning: The phrase file [ ] exceeds 64K. Phrase compression turned off.

Problem: The phrase file is invalid and cannot be used.

Result: Help Workshop cannot perform phrase compression on the Help file.

Solution: Delete the phrase file, and then recompile. If the problem still occurs, switch to Hall or Maximum compression.

{button ,AL(`COMPRESS;BAS_COMPRESSION')} Related Topics

HC3042: Warning: The copyright string exceeds 255 characters. It has been truncated to the following: [ ].

Problem: The copyright string is too long.

Result: Help Workshop truncates the copyright string at 255 characters.

Solution: In Help Workshop, edit the text in the Copyright information box on the General tab in the Options dialog box, and then recompile.

{button ,AL(`COPYRIGHT;PRO_HPJ_COPYRIGHT_INFO')} Related Topics

HC3043: Warning: The title string exceeds 127 characters. It has been truncated to the following: [ ].

Problem: The title string is too long.

Result: Help Workshop truncates the title string.

Solution: In Help Workshop, edit the text in the Help Title box on the General tab in the Options dialog box, and then recompile.

{button ,AL(`TITLE;PRO_HPJ_TITLE')} Related Topics

HC3044: Warning: Hotspot exceeds [ ] characters: [ ].

Problem: The hidden text portion of the hotspot is too large.

Result: Help Workshop does not make the selected text or graphic a hotspot.

Solution: Shorten the hotspot in the topic file, and then recompile.

{button ,AL(`BAS_INSERTING_JUMP')} Related Topics

HC3045: Warning: Footnote exceeds [ ] characters: [ ].

Problem: The footnote is too large.

Result: Help Workshop ignores the footnote.

Solution: Reduce the length of the footnote text in the topic file, and then recompile. For some footnotes, such as the K-footnote, you can use several smaller footnotes instead of one larger footnote.

{button ,AL(`BAS_TOPIC_FOOTNOTE;footnote_rtf')} Related Topics

HC3046: Warning: The [ ] file is not an RTF (Rich Text Format) file. It appears to have been saved as a Microsoft Word document.

Problem: You have saved the topic file in Word format instead of RTF format.

Result: Help Workshop ignores the file.

Solution: Save the document as a Rich-Text Format file, and then recompile.

{button ,AL(`BAS_CREATE_TOPIC_FILE')} Related Topics

HC3047: Warning: No name was specified for the bitmap command.

Problem: The bitmap command ({bmc}, {bml}, or {bmr}) was specified in a topic without a bitmap filename.

Result: Help Workshop ignores the bitmap command.

Solution: In the topic file, specify a bitmap file to include, or remove the bitmap command. Then recompile.

{button ,AL(`bmx;BAS_ADDING_GRAPHICS')} Related Topics

HC3048: Warning: Nonscrolling region is defined after scrolling region.

Problem: The Keep With Next attribute was applied to a paragraph after a paragraph that did not have this attribute.

Result: Help Workshop ignores the Keep With Next attribute; the paragraph is treated as plain text and is displayed as part of the topic text displayed in the scrolling region of the topic.

Solution: Set the Keep With Next attribute for the first paragraph in the topic if you want a nonscrolling region, and remove the Keep With Next attribute from following paragraphs. Then recompile the Help file.

{button ,AL(`keepn_rtf;PRO_NONSCROLL')} Related Topics

HC3049: Warning: Invalid topic IDs in hypergraphic: [ ].

Problem: A topic ID in the shed file contains invalid characters.

Result: The hotspot is not active in the Help file.

Solution: Edit the hypergraphic and ensure that the topic IDs for the hotspots are valid and spelled correctly, and then recompile.

{button ,AL(`SHED')} Related Topics

HC3050: Warning: Missing topic IDs in hypergraphic: [ ].

Problem: A hotspot was specified in the shed file, but no actual topic ID was specified.

Result: The hotspot is not active in the compiled Help file.

Solution: Edit the hypergraphic and ensure that each hotspot calls a valid topic ID, and then recompile.

{button ,AL(`SHED')} Related Topics

HC3051: Warning: Cannot jump to window [ ]. No windows have been defined in the project file.

Problem: There is no [WINDOWS] section in the project file, so you cannot jump to any window type other than main.

Result: The hotspot is not active in the Help file.

Solution: Use Help Workshop to define the specified secondary window, and then recompile.

{button ,AL(`BAS_INSERTING_JUMP;PRO_HPJ_CREATE_WINDOW')} Related Topics

HC3052: Warning: The window name [ ] has not been defined in the project file.

Problem: You have attempted to display a topic in a window type that has not been defined in your project file.

Result: The hotspot is not active in the Help file.

Solution: Use Help Workshop to define the specified secondary window, and then recompile.

{button ,AL(`WINDOWS;PRO_HPJ_CREATE_WINDOW')} Related Topics

HC3053: Warning: The project file contains more than 255 window definitions.

Problem: You have defined more than 255 window types in your project file.

Result: Help Workshop ignores the additional window definitions. If you attempt to jump to a topic in one of the window definitions beyond the first 255, you will get an error message about the window type not being defined.

Solution: Remove the extra window definitions from the project file, and then recompile.

{button ,AL(`WINDOWS;PRO_HPJ_CUSTOMIZE_WINDOW')} Related Topics

HC3054: Warning: Build tag contains invalid characters. The following is invalid: [ ].

Problem: A build tag in the [BUILDTAGS] section contains something other than a alphanumeric character or the underscore (_) character.

Result: Help Workshop ignores the line containing invalid characters.

Solution: Edit the build tag so that it contains only valid characters. Make any necessary changes in topics containing the specified build tag, and then recompile. Alternatively, specify the build tags to include/exclude in the Build Tags tab of the Options dialog box. This form of including/excluding topics places no restrictions on the characters, length, or number of build tags, and does not use the [BUILDTAGS] section.

{button ,AL(`BUILDTAGS;BAS_INSERTING_BUILD_TAG')} Related Topics

HC3055: Warning: The build tag [ ] has already been used.

Problem: The build tag in the [BUILDTAGS] section of your project file has already been declared previously in the same section.

Result: Help Workshop uses the first build tag and ignores the duplicate build tag.

Solution: Remove (or rename) the duplicate build tags in the project file, and then recompile.

{button ,AL(`BUILDTAGS')} Related Topics

HC3056: Warning: The window name [ ] in the macro [ ] contains more than eight characters and has been truncated to [ ].

Problem: The window name you have specified in the macro is invalid due to its length.

Result: Help Workshop ignores the window definition in this macro.

Solution: Specify a valid window name in the macro, and then recompile.

{button ,AL(`WINDOWS;syntax_more')} Related Topics

HC3057: Warning: A hotspot is defined with a macro (using !), but the macro is not specified.

Problem: You started a hotspot with a ‘!’ character, but did not follow it with a macro.

Result: Help Workshop ignores the macro call from the hotspot.

Solution: Specify a valid macro in the hotspot, or remove the '!' character from the hotspot definition. Make certain that all text from the ‘!’ to the end of the macro is marked as hidden.

{button ,AL(`BAS_MACRO_JUMP')} Related Topics

HC3058: Warning: Topic ID for hotspot contains invalid characters: [ ].

Problem: An invalid topic ID has been used in a hotspot.

Result: The hotspot is not active in the Help file.

Solution: Correct the topic ID and recompile.

{button ,AL(`BAS_ADDING_TOPIC_ID;BAS_INSERTING_JUMP;BAS_INSERTING_POPUP')} Related Topics

HC3059: Warning: A hotspot is specified without a macro or topic ID.

Problem: A hotspot was specified without specifying either a topic ID or a macro.

Result: The hotspot is not active in the Help file.

Solution: Edit the hotspot in the topic file and ensure that the topic ID or macro for the hotspot is valid, spelled correctly, and styled as hidden text. Then recompile.

{button ,AL(`BAS_INSERTING_JUMP;BAS_MACRO_JUMP')} Related Topics

HC3060: Warning: Window name is specified for a pop-up jump.

Problem: You have specified a window name in a pop-up hotspot.

Result: The hotspot is not active in the Help file.

Solution: Remove the right angle bracket and secondary window name from the pop-up hotspot, or change the single underline to double underline (for a topic jump). Then recompile the project file.

{button ,AL(`BAS_INSERTING_POPUP;BAS_INSERTING_JUMP')} Related Topics

HC3062: Warning: Window definition is missing an '=' character: [ ].

Problem: The window definition in the [WINDOWS] section is invalid.

Result: Help Workshop ignores the line.

Solution: Use Help Workshop to edit the window definition, and then recompile.

{button ,AL(`WINDOWS;PRO_HPJ_CREATE_WINDOW;PRO_HPJ_CUSTOMIZE_WINDOW')} Related Topics

HC3063: Warning: Window definition does not contain anything after the '=' character: [ ].

Problem: The window definition in the [WINDOWS] section is invalid.

Result: Help Workshop ignores the line.

Solution: Use Help Workshop to edit the window definition, and then recompile.

{button ,AL(`WINDOWS;PRO_HPJ_CREATE_WINDOW;PRO_HPJ_CUSTOMIZE_WINDOW')} Related Topics

HC3064: Warning: Window name is not defined before the '=' character: [ ].

Problem: The window definition in the [WINDOWS] section is invalid.

Result: Help Workshop ignores the line.

Solution: Use Help Workshop to edit the window definition, and then recompile.

{button ,AL(`WINDOWS;PRO_HPJ_CREATE_WINDOW;PRO_HPJ_CUSTOMIZE_WINDOW')} Related Topics

HC3065: Warning: A closing quotation mark () is missing in window caption: [ ].

Problem: The window definition in the [WINDOWS] section is invalid.

Result: Help Workshop ignores the window definition.

Solution: Use Help Workshop to edit the window definition, and then recompile.

{button ,AL(`WINDOWS;PRO_HPJ_CREATE_WINDOW;PRO_HPJ_CUSTOMIZE_WINDOW')} Related Topics

HC3066: Warning: Window caption contains more than 50 characters: [ ].

Problem: The window definition in the [WINDOWS] section contains a title that is too long.

Result: Help Workshop ignores the window definition.

Solution: Use Help Workshop to edit the window definition, and then recompile.

{button ,AL(`WINDOWS;PRO_HPJ_CREATE_WINDOW;PRO_HPJ_CUSTOMIZE_WINDOW')} Related Topics

HC3067: Warning: The [ ] command in the [OPTIONS] section of the project file does not specify a value after the '=' character.

Problem: You have specified an invalid command in the [OPTIONS] section of your project file.

Result: Help Workshop ignores the line.

Solution: Use Help Workshop to edit the option, and then recompile.

{button ,AL(`OPTIONS;PRO_HPJ_OPTIONS')} Related Topics

HC3068: Warning: The following line in the [OPTIONS] section of the project file does not contain an '=' character: [ ].

Problem: You have specified an invalid command in the [OPTIONS] section of your project file.

Result: Help Workshop ignores the line.

Solution: Use Help Workshop to edit the option, and then recompile.

{button ,AL(`OPTIONS;PRO_HPJ_OPTIONS')} Related Topics

HC3069: Warning: The [ ] option in the [OPTIONS] section of the project file is not recognized.

Problem: You have specified an invalid command in the [OPTIONS] section of your project file.

Result: Help Workshop ignores the line.

Solution: Use Help Workshop to edit the option, and then recompile.

{button ,AL(`OPTIONS;PRO_HPJ_OPTIONS')} Related Topics

HC3070: Warning: the major browse string exceeds 50 characters: [ ].

Problem: The name to the left of the colon in a browse footnote contains too many characters.

Result: Help Workshop truncates the string to 50 characters. The browse sequence may be affected.

Solution: Shorten the browse string.

{button ,AL(`BAS_ADDING_BROWSE_CODE')} Related Topics

HC3071: Warning: One or more browse sequences are set, but browse buttons are not enabled in any window.

Problem: You have specified a browse footnote, but none of the windows defined for the Help file contains browse buttons.

Result: The browse information is stored in the Help file (taking up space) but the information can never be accessed by the user.

Solution: Either add browse buttons or remove all browse footnotes.

{button ,AL(`BAS_ADDING_BROWSE_CODE;BrowseButtons;PRO_HPJ_ADD_BUTTON_BAR;PRO_HPJ_MAIN_BUTTON_BAR')}
Related Topics

HC3072: Warning: The macro [ ] does not include a control panel name.

Problem: The macro expected a control panel name, but you didn't specify one.

Result: Help Workshop ignores the ControlPanel macro.

Solution: Specify a control panel name in the topic file macro, and then recompile.

{button ,AL(`ControlPanel')} Related Topics

HC3073: Warning: No section is defined for the line: [ ].

Problem: The section heading on the specified line is not complete, or the first entry in the project (.hpj) file is not a section heading.

Result: Help Workshop ignores the section heading on this line and all succeeding lines until it encounters a valid section heading.

Solution: Edit the specified section in the project file. Then recompile the project file.

{button ,AL(`HPJ_Sections')} Related Topics

HC3074: Warning: The [ ] section is not recognized by this version of Help Workshop.

Problem: Your project file contains a section that Help Workshop does not recognize.

Result: Help Workshop ignores the section heading.

Solution: Verify that the section headings in the project file are valid and spelled correctly, and then recompile.

{button ,AL(`HPJ_Sections')} Related Topics

HC3075: Warning: The [ ] section does not follow the [OPTIONS] section in the project file.

Problem: You have specified a section that requires the [OPTIONS] section before you have specified the [OPTIONS] section.

Result: Help Workshop may not be able to find some files, or the files may be compiled incorrectly.

Solution: Load the project file into Help Workshop and then save it. Help Workshop will automatically save the sections in the correct order.

{button ,AL(`HPJ_Sections')} Related Topics

HC3079: Warning: The alias string [ ] has already been aliased [ ].

Problem: A topic ID can have only one alias.

Result: Help Workshop ignores this line.

Solution: Use Help Workshop to correct the alias string mapping, and then recompile.

{button ,AL(`ALIAS;PRO_HPJ_ADD_ALIAS')} Related Topics

HC3080: Warning: The string [ ] has already been aliased [ ].

Problem: An alias string cannot be assigned another alias.

Result: Help Workshop ignores this line.

Solution: Use Help Workshop to correct the alias string mapping, and then recompile.

{button ,AL(`ALIAS;PRO_HPJ_ADD_ALIAS')} Related Topics

HC3081: Warning: Both alias and topic ID are identical: [ ].

Problem: The alias is the same as the topic ID.

Result: Help Workshop ignores this line.

Solution: Use Help Workshop to correct the alias string mapping, and then recompile.

{button ,AL(`ALIAS;PRO_HPJ_ADD_ALIAS')} Related Topics

HC3082: Warning: The map number [ ] is not a valid number: [ ].

Problem: A context number in the [MAP] section of the project (.hpj) file contains invalid characters.

Result: Help Workshop ignores this line.

Solution: Use Help Workshop to correct the mapping, and then recompile.

{button ,AL(`MAP;PRO_HPJ_ADD_MAP')} Related Topics

HC3083: Warning: Invalid syntax for window color: [ ].

Problem: The syntax specified for the windows color in the [WINDOWS] section of your project file is invalid.

Result: Help Workshop ignores the window definition.

Solution: Use Help Workshop to edit the window definition, and then recompile.

{button ,AL(`WINDOWS;PRO_HPJ_CREATE_WINDOW;PRO_HPJ_CUSTOMIZE_WINDOW')} Related Topics

HC3084: Warning: Invalid window syntax: [ ].

Problem: The syntax you have specified for the window in the [WINDOWS] section of your project file is invalid.

Result: Help Workshop ignores the invalid window definition.

Solution: Use Help Workshop to edit the window definition, and then recompile.

{button ,AL(`WINDOWS;PRO_HPJ_CREATE_WINDOW;PRO_HPJ_CUSTOMIZE_WINDOW')} Related Topics

HC3086: Warning: Window position is out of range: [ ].

Problem: One or more of the window position coordinates exceed the limit of 1024.

Result: Help Workshop ignores the invalid window definition.

Solution: Use Help Workshop to edit the window definition, and then recompile.

{button ,AL(`WINDOWS;PRO_HPJ_CREATE_WINDOW;PRO_HPJ_CUSTOMIZE_WINDOW')} Related Topics

HC3087: Warning: The following filename exceeds 259 characters: [ ].

Problem: You have specified a filename that contains too many characters.

Result: Help Workshop ignores the specified file.

Solution: Shorten the filename. Then recompile the project file.

{button ,AL(`FILES;PRO_HPJ_RTF_FILES')} Related Topics

HC3088: Warning: More than 20 font ranges are mapped.

Problem: You have specified too many font ranges.

Result: Help Workshop ignores additional font ranges.

Solution: Remove the excess ranges, and then recompile. Alternatively, remove the font ranges and instead use the font substitution in the Fonts tab of the Options dialog box in Help Workshop.

{button ,AL(`MAPFONTSIZE;PRO_HPJ_FONT_MAPPING')} Related Topics

HC3089: Warning: Invalid font range: [ ].

Problem: The font range is invalid.

Result: Help Workshop ignores the option.

Solution: Edit the font range, and then recompile. Alternatively, remove the font ranges and instead use the font substitution in the Fonts tab of the Options dialog box in Help Workshop.

{button ,AL(`MAPFONTSIZE;PRO_HPJ_FONT_MAPPING')} Related Topics

HC3090: Warning: Current font range overlaps previously defined range: [ ].

Problem: Two or more font ranges overlap each other.

Result: Help Workshop uses the first range and ignores the second mapping.

Solution: Edit the font range, and then recompile. Alternatively, remove the font ranges and instead use the font substitution in the Fonts tab of the Options dialog box in Help Workshop.

{button ,AL(`MAPFONTSIZE;PRO_HPJ_FONT_MAPPING')} Related Topics

HC3091: Warning: Unrecognized forced font name: [ ].

Problem: You have tried to force all font names to an invalid font.

Result: Help Workshop ignores the font name and uses the default MS Sans Serif font.

Solution: Change the font name in the project file to a supported font, and then recompile. Alternatively, remove the font ranges and instead use the font substitution in the Fonts tab of the Options dialog box in Help Workshop.

{button ,AL(`FORCEFONT')} Related Topics

HC3092: Warning: This version of the project file is not supported by this Help compiler.

Problem: The project file was created with a more recent version of Help Workshop than you are currently using.

Result: Help Workshop may remove options or sections.

Solution: Get a more recent version of Help Workshop.

{button ,AL(`VERSION')} Related Topics

HC3093: Warning: Keyword type is not a letter or number. [ ] is invalid.

Problem: An invalid character has been used for the MULTIKEY option.

Result: Help Workshop ignores the attempted keyword table assignment on this line.

Solution: Edit the MULTIKEY option in the project file, and then recompile.

{button ,AL(`MULTIKEY')} Related Topics

HC3094: Warning: More than three keyword types (besides 'K' and 'A') defined.

Problem: More than three keywords have been defined using the MULTIKEY option.

Result: Help Workshop ignores the additional keyword types.

Solution: Remove the extra MULTIKEY options from the project file, and then recompile.

{button ,AL(`MULTIKEY')} Related Topics

HC3095: Warning: Multikey values of 'K' or 'A' cannot be defined.

Problem: The MULTIKEY option has been used to specify either ‘K’ or ‘A’ as a multikey value. These characters are reserved by Help Workshop and WinHelp.

Result: Help Workshop ignores the attempted keyword table assignment on this line.

Solution: Edit the MULTIKEY option in the project file, and then recompile.

{button ,AL(`MULTIKEY')} Related Topics

HC3096: Warning: The font name [ ] is longer than 31 characters.

Problem: The font name is invalid because it is too long.

Result: Help Workshop ignores the option on this line.

Solution: Use Help Workshop to shorten the font name, and then recompile.

{button ,AL(`BAS_ABOUT_FONTS;DEFFONT;MAPFONTSIZE;FORCEFONT')} Related Topics

HC3097: Warning: The [BUILDTAGS] section is missing from the project file.

Problem: A BUILD option was specified in the project file, but no [BUILDTAGS] section was specified.

Result: Help Workshop includes all topics in the build.

Solution: Add a build section to the project file, and then recompile. Alternatively, specify the topics to include or exclude using the Build Tags tab in the Options dialog box of Help Workshop.

{button ,AL(`BUILDTAGS;PRO_HPJ_BUILD_TAGS')} Related Topics

HC3098: Warning: Invalid build tag expression.

Problem: The BUILD option specifies an invalid expression.

Result: Help Workshop ignores the line with the invalid expression.

Solution: Edit the build tags in the project file, and then recompile. Alternatively, specify the topics to include or exclude using the Build Tags tab in the Options dialog box of Help Workshop.

{button ,AL(`BUILD;PRO_HPJ_BUILD_TAGS')} Related Topics

HC3099: Warning: Build expression too complex.

Problem: The build expression in the project (.hpj) file has too many expressions or is nested too deeply.

Result: Help Workshop includes all topics in the build.

Solution: Edit the build expression in the project file, and then recompile. Alternatively, specify the topics to include or exclude using the Build Tags tab in the Options dialog box of Help Workshop.

{button ,AL(`BUILD;PRO_HPJ_BUILD_TAGS')} Related Topics

HC3101: Warning: Unknown build error.

Problem: Help Workshop was not able to interpret the BUILD option correctly.

Result: Help Workshop includes all topics in the build.

Solution: Edit the build expression in the project file, and then recompile. Alternatively, specify the topics to include or exclude using the Build Tags tab in the Options dialog box of Help Workshop.

{button ,AL(`BUILD;PRO_HPJ_BUILD_TAGS')} Related Topics

HC3102: Warning: No macro is specified for the entry macro footnote.

Problem: You have specified an entry macro footnote, but the footnote did not contain any macros.

Result: Help Workshop ignores the entry macro footnote.

Solution: Add a macro to the entry macro footnote or remove the footnote marker. Then recompile the Help file.

{button ,AL(`BAS_ADD_ENTRY_MACRO')} Related Topics

HC3103: Warning: The entry macro footnote does not precede text.

Problem: The entry macro footnote was specified after text was specified for the topic.

Result: Help Workshop ignores the entry macro footnote.

Solution: Move the entry macro footnote to the beginning of the topic, and then recompile.

{button ,AL(`BAS_ADD_ENTRY_MACRO')} Related Topics

HC3104: Warning: The topic ID footnote (#) does not specify a topic ID.

Problem: You have a specified a topic ID footnote that does not contain any characters.

Result: Help Workshop does not assign the topic an identifier.

Solution: Add a valid topic ID to the footnote, and then recompile.

{button ,AL(`BAS_ADDING_TOPIC_ID')} Related Topics

HC3105: Warning: The minor browse string exceeds 50 characters: [ ].

Problem: The browse string to the right of the colon is larger than 50 characters.

Result: Help Workshop truncates the string which may affect the browse sequence itself.

Solution: Shorten the browse string in the appropriate topics.

{button ,AL(`BAS_ADDING_BROWSE_CODE')} Related Topics

HC3106: Warning: Browse footnote (+) does not appear before any text.

Problem: The browse sequence footnote (+) does not precede the first paragraph of the topic.

Result: Help Workshop ignores the browse sequence footnote.

Solution: Move the browse sequence footnote so that is at the beginning of the topic, and then recompile.

{button ,AL(`BAS_ADDING_BROWSE_CODE;BAS_TOPIC_FOOTNOTE')} Related Topics

HC3107: Warning: A browse sequence has already been defined for this topic.

Problem: The browse sequence footnote (+) has been specified twice in the same topic.

Result: Help Workshop uses the first browse sequence footnote and ignores the duplicate footnote.

Solution: Remove the duplicate browse sequence footnote from the topic, and then recompile.

{button ,AL(`BAS_ADDING_BROWSE_CODE')} Related Topics

HC3108: Warning: The title footnote ($) does not appear before any text.

Problem: The title footnote does not precede the first paragraph of the topic.

Result: Help Workshop ignores the title footnote, and the topic does not have any title.

Solution: Move the title footnote to the beginning of the topic, and then recompile.

{button ,AL(`BAS_ADD_TITLE;BAS_TOPIC_FOOTNOTE')} Related Topics

HC3109: Warning: The title footnote ($) does not contain any text.

Problem: A title footnote was specified that does not contain any text.

Result: Help Workshop ignores the title footnote, and the topic does not have any title.

Solution: Add a title string next to the topic title footnote, and then recompile.

{button ,AL(`BAS_ADD_TITLE')} Related Topics

HC3110: Warning: A title has already been defined for this topic.

Problem: A title footnote has already been specified for the topic.

Result: Help Workshop uses the first title footnote and ignores the duplicate title footnote.

Solution: Remove the second title footnote from the topic, and then recompile.

{button ,AL(`BAS_ADD_TITLE')} Related Topics

HC3111: Warning: This topic contains keywords but no title.

Problem: One or more keywords have been specified for the topic, but a title footnote was not specified.

Result: If no title is specified for the topic, and the keyword appears in more than one topic (including other Help files), WinHelp will display "Untitled topic #n" in the Topics Found dialog box.

Solution: If this is not a pop-up topic, add a title using the title footnote ($), and then recompile. If the topic will only be displayed in a pop-up window, remove the keyword footnotes.

{button ,AL(`BAS_ADDING_KEYWORDS')} Related Topics

HC3112: Warning: The build footnote (*) is not the first footnote in the topic.

Problem: The build footnote does not precede all other text and footnotes in the topic.

Result: Help Workshop ignores the build tag footnote, and the topic is not assigned a build tag.

Solution: Move the build tag footnote to the very beginning of the topic, before any other footnotes, and then recompile.

{button ,AL(`BAS_INSERTING_BUILD_TAG')} Related Topics

HC3113: Warning: Build tag is longer than 32 characters: [ ].

Problem: The build tag is too long when used in conjunction with the BUILD option in the project file.

Result: Help Workshop ignores the build tag assigned to the topic.

Solution: Shorten the build tag to 32 or fewer characters, and then recompile. Alternatively, specify the build tags to include/exclude in the Build Tags tab of the Options dialog box. This form of including/excluding topics places no restrictions on the characters, length, or number of build tags.

{button ,AL(`BAS_INSERTING_BUILD_TAG')} Related Topics

HC3114: Warning: The bitmap [ ] has been used as part of another {bmx} command.

Problem: Two {bmx} commands that specify bitmaps for multiple color depths have a bitmap in common but are otherwise different or in a different order.

Result: WinHelp will treat the shared bitmap files as separate files, and this effectively doubles the space required for each of the specified bitmaps.

Solution: Make sure that bitmap commands that are intended to be identical include the same bitmaps in the same order.

{button ,AL(`BAS_MULT_COLOR_GRAPHICS;bmx')} Related Topics

HC3116: Warning: Table has more than [ ] columns.

Problem: The table has more columns than WinHelp is able to display.

Result: Help Workshop treats the additional columns as one table cell.

Solution: Reduce the number of table columns, and then recompile.

{button ,AL(`overview')} Related Topics

HC3117: Warning: Side-by-side paragraphs are not supported.

Problem: Side-by-side paragraph formatting has been used.

Result: Help Workshop ignores the side-by-side paragraph formatting.

Solution: Convert the side-by-side paragraphs to a table or a tabbed paragraph, and then recompile.

{button ,AL(`RTF_OVERVIEW')} Related Topics

HC3118: Warning: Unrecognized RTF (Rich Text Format) graphics format.

Problem: An embedded bitmap contains a format that Help Workshop does not recognize. Help Workshop supports only Windows bitmaps (.bmp), Windows device-independent bitmaps (.dib), and Windows metafiles (.wmf)

Result: Help Workshop ignores the specified graphic.

Solution: Convert the graphic to a supported format, and then recompile.

{button ,AL(`BAS_ADDING_GRAPHICS')} Related Topics

HC3119: Warning: Hash conflict between [ ] and [ ]. One of these topic IDs must be changed.

Problem: The hash algorithm used by Help Workshop to convert a topic ID into an internal number that WinHelp uses, has generated the same hash value for both of the listed topic IDs.

Result: WinHelp displays the topic with the first topic ID whenever the user chooses a hotspot containing either topic ID.

Solution: Change one of the specified topic IDs, and then recompile.

{button ,AL(`BAS_ADDING_TOPIC_ID')} Related Topics

HC3120: Warning: The LANGUAGE option [ ] is not supported.

Problem: The LANGUAGE option has been specified. This option is no longer supported.

Result: Help Workshop uses the default English (U.S.) sorting order.

Solution: Open the project file in Help Workshop, click the Options button, click the Sort tab, and then specify the language of the Help file.

{button ,AL(`LANGUAGE;LCID;PRO_HPJ_SORT_SETTINGS')} Related Topics

HC3121: Warning: Invalid version of Ftsrch.dll.

Problem: The version of Ftsrch.dll on your system is invalid.

Result: Neither Hall nor Maximum compression can be used and no index file is created for full text search.

Solution: Reinstall Ftsrch.dll from your Windows 95 or Windows NT installation disks, and then recompile.

{button ,AL(`BAS_FULLTEXT_INDEX')} Related Topics

HC3123: Warning: Missing '=' character in REPLACE option: [ ].

Problem: The syntax for the REPLACE option is invalid.

Result: Help Workshop ignores the entire option.

Solution: Open the project file in Help Workshop, click the Options button, click the Files tab, and then enter the path in the Substitute Path Prefix box.

{button ,AL(`REPLACE;PRO_HPJ_REPLACE_PATH')} Related Topics

HC3124: Warning: The TMP folder [ ] is invalid.

Problem: The folder name is invalid.

Result: Help Workshop uses the default folder for temporary files.

Solution: Open the project file with Help Workshop, click the Options button, click the Files tab, and then click the Browse button beside the TMP folder box. Use the Browse dialog box to locate the correct folder.

{button ,AL(`TMPDIR;PRO_HPJ_TMP_DIR')} Related Topics

HC3126: Warning: The window (>) footnote does not appear after the topic ID (#) footnote.

Problem: The window footnote was placed before the topic ID footnote.

Result: Help Workshop ignores the window footnote.

Solution: Reposition the window footnote after the topic ID footnote, and then recompile.

{button ,AL(`BAS_ADD_WINDOW_TYPE')} Related Topics

HC4001: Warning: Cannot find or load Ftsrch.dll.

Problem: Help Workshop cannot find the Ftsrch.dll file.

Result: Neither Maximum nor Hall compression can be used and no index file is created for full text search.

Solution: Reinstall Ftsrch.dll from your Windows 95 or Windows NT installation disks, and then recompile.

{button ,AL(`BAS_FULLTEXT_INDEX')} Related Topics

HC4002: Warning: The RTF file [ ] is corrupted at offset [ ].

Problem: Help Workshop encountered an error while processing the topic (.rtf) file. This problem can occur when the number of opening and closing braces don't match, when the end of the file is reached before all columns of a table have been specified, or when the number specified for an RTF token is invalid.

Result: Help Workshop ignores the rest of the topic file.

Solution: Open the file and save it again as Rich Text Format (RTF), and then recompile. If the problem persists, look for errors in the topic file.

{button ,AL(`BAS_CREATE_TOPIC_FILE;RTF_OVERVIEW')} Related Topics

HC4003: Warning: An error occurred attempting to read the file [ ].

Problem: Help Workshop can not read the specified file. Another program may have locked the file, or if the file is on a network, the network may be down.

Result: Help Workshop ignores the file.

Solution: Check file and its path. Correct errors, and then recompile.

HC4004: Warning: File is not an RTF (Rich Text Format) file.

Problem: The specified file has not been saved as an RTF file.

Result: Help Workshop ignores the file.

Solution: Resave the file as RTF, and then recompile.

{button ,AL(`BAS_CREATE_TOPIC_FILE;RTF_OVERVIEW')} Related Topics

HC4005: Warning: The [ ] Help file has not been created.

Either there are no topics to compile, or the build expression is incorrect for all topics.

Problem: Help Workshop reached the end of all topic files without encountering any text.

Result: Help Workshop does not create a Help file.

Solution: Confirm that you have specified at least one topic file that contains text. If all topic contain build tags, you must specify at least one build tag to include (use the Build Tags tab in the Options dialog box of Help Workshop).

{button ,AL(`BAS_SPECIFY_BUILD')} Related Topics

HC4006: Warning: The folder [ ] specified for the [ ] option does not exist.

Problem: A nonexistent folder has been specified. Either the folder name was specified incorrectly, or it points to a network location that does not (currently) exist.

Result: Help Workshop uses the current working folder.

Solution: Use Help Workshop to correct the folder name for the specified option, and then recompile.

{button ,AL(`OPTIONS;BAS_ABOUT_HPJ_FILES')} Related Topics

HC4007: Warning: The [ ] option [ ] is not a valid value.

Problem: The value specified for the option is invalid.

Result: Help Workshop ignored the option.

Solution: Load the project file into Help Workshop and then resave it. Help Workshop will correct the error (usually by prompting you first).

{button ,AL(`BAS_ABOUT_HPJ_FILES;OPTIONS')} Related Topics

HC4008: Warning: Cannot find or load Ftsrch.dll. Hall compression turned off.

Problem: Help Workshop either cannot find the Ftsrch.dll file, or it is out of date or corrupted.

Result: Neither Maximum nor Hall compression can be used and no index file is created for full text search.

Solution: Reinstall Ftsrch.dll from Windows 95 or Windows NT installation disks, and then recompile.

{button ,AL(`BAS_FULLTEXT_INDEX;BAS_COMPRESSION')} Related Topics

HC4009: Warning: The topic ID specified in the project file as the default topic does not exist.

Problem: You have specified a default topic ID in the project file, but none of your topics contains that topic ID.

Result: Help Workshop uses the first topic in the build as the default topic.

Solution: Verify that the topic ID exists in the desired topic and that it is spelled correctly, and that the topic file is included in the [FILES] section of the project file, and then recompile.

{button ,AL(`CONTENTS_hpj;BAS_ADDING_TOPIC_ID')} Related Topics

HC4010: Warning: There are more opening braces than closing braces.

Problem: When Help Workshop reached the end of the file, the number of opening and closing braces did not match.

Result: The Help file may be corrupt.

Solution: Load and resave the topic (.rtf) file, and then recompile. If the problem persists, you will need to edit the topic file directly.

HC4011: Warning: There are 20 opening braces without intervening closing braces.

Problem: More opening braces without matching closing braces have been encountered than should ever occur in a normal topic (.rtf) file.

Result: The rest of the file is ignored.

Solution: Load and resave the topic file, and then recompile. If the problem persists, you will need to edit the topic file directly.

HC4012: Warning: The full-text search index cannot be created because neither Phrase nor Hall compression has been selected.

Problem: In order to create a full-text search index, you must compile with either Maximum or Hall compression.

Result: Help Workshop is unable to generate an full-text search (.fts) index file.

Solution: Open the project file in Help Workshop, click the Options button, click the Compression tab, and check the Maximum radio button. Save the project file and recompile.

HC4013: Warning: Invalid default font number in [ ]. Using [ ] as the default font.

Problem: The RTF token \deff specifies as a default font a font number that is not defined in the \fonttbl section.

Result: Help Workshop will ignore the specification and use the first font declared in the \fonttbl section as the default font.

Solution: In the topic (.rtf) file change the number after the \deff token to specify a font declared in the \fonttbl section. Then recompile.

{button ,AL(`RTF1')} Related Topics

HC4014: Warning: There is text after the closing brace in the RTF file.

Problem: Help Workshop encountered the closing brace that should signify the end of the topic (.rtf) file, and then encountered additional text.

Result: Help Workshop adds the text to the Help file, but the text cannot be accessed.

Solution: In the raw topic file, move or delete the text after the closing brace, and then recompile.

{button ,AL(`BAS_CREATE_TOPIC_FILE')} Related Topics

HC5000: Error: The filename [ ] is too long.

Problem: The specified filename is too long.

Result: The Help file is not created.

Solution: Use Help Workshop to shorten the path, and then recompile.

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

HC5001: Error: File is a folder, not a file.

Problem: You have specified a filename, but the name specified is actually a folder.

Result: The Help file is not created.

Solution: Move or rename the folder or the file, and then recompile.

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

HC5002: Error: The text for the button in the CreateButton macro is too long.

Problem: You have specified more than 96 characters in the CreateButton macro.

Result: The Help file is not created.

Solution: Reduce to 96 or less the number of characters in the CreateButton macro, and then recompile.

{button ,AL(`BAS_AUTH_BTN;CreateButton')} Related Topics

HC5003: Error: Permission to open the file [ ] is denied. Another program has probably locked the file.

Problem: You do not have the required file privileges to open the requested file. This can happen if the file has been opened by another program. For example, if WinHelp has the Help file open, Help Workshop is not be able to write to that Help file.

Result: The Help file is not created.

Solution: If the file has been loaded in another program, switch to that program and either close the file or close the program. If you are trying to write a file to a networked drive, make certain you have write permission on that server.

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

HC5005: Error: The file [ ] cannot be found.

Problem: The specified file cannot be located.

Result: The Help file is not created.

Solution: Verify that you have specified the correct filename and that the file exists in the correct folder. Then recompile the project file.

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

HC5006: Error: Invalid RTF tokens for a table.

Problem: A \row command appeared before a \cell command in the topic (.rtf) file.

Result: The Help file is not created.

Solution: Change the order of commands in the topic file, and then recompile.

{button ,AL(`RTF1')} Related Topics

HC5007: Invalid use of the WinHelp menu [ ] in the [ ] macro..

Problem: You have attempted to use a standard WinHelp menu ID.

Result: The Help file is not created.

Solution: Use a menu ID created with a call to the ExtInsertItem, ExtInsertMenu, or InsertMenu macros.

{button ,AL(`syntax_more;AddAccelerator;RemoveAccelerator')} Related Topics

HC5009: Error: [ ] does not contain a comma separating the parameters in the macro [ ].

Problem: Either a parameter has been left out of the macro or parameters are not separated with a comma.

Result: The Help file is not created.

Solution: Add the missing comma or parameter to the macro in the topic file, and then recompile.

{button ,AL(`syntax_more')} Related Topics

HC5010: Error: [ ] is an invalid parameter for the macro [ ].

Problem: You cannot use a standard WinHelp menu as a parameter for this macro.

Result: The Help file is not created.

Solution: Use the name of your own custom menu.

{button ,AL(`syntax_more')} Related Topics

HC5011: Error: Cannot open the file [ ].

Problem: The file cannot be opened. This can occur when another program has the file locked, or the file is on a network and the network is down.

Result: The Help file is not created.

Solution: If another program has the file open, close the file. In some cases, it may be necessary to close the program.

{button ,AL(`BAS_ABOUT_HPJ_FILES')} Related Topics

HC5012: Error: Too many nested #include files. Cannot include the file [ ].

Problem: A file specified in a #include statement can contain an included file which can in turn contain an included file, up to five levels. This limit has been exceeded.

Result: The Help file is not created.

Solution: Merge the contents of the deepest #include file into its parent, and then recompile.

{button ,AL(`BAS_CNT_OVERVIEW;BAS_ABOUT_HPJ_FILES')} Related Topics

HC5013: Error: Invalid DBCS escape sequence: [ ].

Problem: The DBCS character is invalid.

Result: The Help file is not created.

Solution: If this is not a DBCS topic file, change the language of the Help file. Otherwise you will need to use a different word that does not use this DBCS character.

{button ,AL(`DBCS;PRO_HPJ_DBCS')} Related Topics

HC6000: Error: Help Workshop is out of memory. If you have any other programs running, close them and try compiling again.

Problem: There is insufficient memory and swap file space.

Result: The Help file is not created.

Solution: If you are running Windows 95, increase the amount of hard disk space available on the drive Windows is installed on. Otherwise, increase the size of your permanent swap file.

{button ,AL(`BAS_THE_HELP_COMPILER')} Related Topics

HC6001: Error: Out of disk space writing to the temporary file [ ]. Free up disk space on this drive or change your TMP environment variable.

Problem: There is insufficient space on the drive where Help Workshop is creating temporary files.

Result: The Help file is not created.

Solution: Either free space on the specified drive, or tell Help Workshop to create temporary files on a different drive (or network connection).

{button ,AL(`BAS_THE_HELP_COMPILER')} Related Topics

HC6002: Error: An error occurred while reading the file [ ].

Problem: Help Workshop was not able to read the file. The file may be corrupt.

Result: The Help file is not created.

Solution: Confirm that the file is valid, replacing if not. If the file appears to be valid, run scandisk to verify that your hard disk does not have problems.

{button ,AL(`BAS_THE_HELP_COMPILER')} Related Topics

HC6003: Error: Out of file handles. Increase the FILES= line in CONFIG.SYS.

Problem: There are not enough file handles available.

Result: The Help file is not created.

Solution: If possible, increase the FILES setting in the CONFIG.SYS file on your computer to FILES=50 or greater. Reboot your computer, then recompile.

{button ,AL(`BAS_THE_HELP_COMPILER')} Related Topics

HC6004: Error: The file [ ] is a read-only file.

Problem: Help Workshop cannot write to the specified file.

Result: The Help file is not created.

Solution: Change the file's read-only attribute if you want Help Workshop to overwrite the file. Otherwise, rename the file and try again to compile the project file.

{button ,AL(`BAS_THE_HELP_COMPILER;ROOT;BMROOT')} Related Topics

HC6005: Error: [ ] is a folder, not a file.

Problem: The name you have specified for a file is actually a folder.

Result: The Help file is not created.

Solution: Move or rename the folder or the Help file, and then recompile.

{button ,AL(`BAS_THE_HELP_COMPILER')} Related Topics

HC6006: Error: [ ] is a device name and cannot be used as a filename.

Problem: The name you have specified for a file is a device.

Result: The Help file is not created.

Solution: Change the name to a file, not a device, and then recompile.

{button ,AL(`BAS_THE_HELP_COMPILER')} Related Topics

HC6007: Error: Cannot write to the file [ ].

Problem: An error occurred while Help Workshop was writing to a file.

Result: The Help file is not created.

Solution: Run scandisk to fix any problems on your hard drive. Or trying compiling on a different drive or network connection.

{button ,AL(`BAS_THE_HELP_COMPILER')} Related Topics

HC6008: Error: No files have been specified in the [FILES] section of the project file.

Problem: The [FILES] section of your project file did not specify any files.

Result: The Help file is not created.

Solution: Open the project file using Help Workshop, click the Files button, click the Add button and enter the name of a topic file.

{button ,AL(`FILES;PRO_HPJ_RTF_FILES')} Related Topics

HC6009: Error: The language identifier [ ] is not supported in this version of Windows.

Problem: The language specified for the Help file is not supported on your version of Windows. For example, you cannot compile a Japanese Help file on an English version of Windows.

Result: The Help file is not created.

Solution: Confirm that correct language has been specified. If the language is correct, you must compile the Help file on a version of Windows appropriate to that language.

{button ,AL(`BAS_THE_HELP_COMPILER')} Related Topics

Click this to specify general options in addition to options for compressing help files, sorting keywords, specifying which files to use while compiling, determining full text search functionality, specifying macros and build tags, and changing fonts. You cannot compile Help files without specifying the location of topic (.rtf) files.

Click this to specify the location of topic (.rtf) files or to ignore revision marks in the topic files you compile. You cannot compile Help files without specifying the location of topic files.

Click this to specify the type, attributes, color, and position of Help windows. You can also specify which buttons should appear at the top of Help windows and whether macros run when a Help window opens. You can compile Help files without specifying any of these options.

Click this to specify the location of bitmaps that you want to use in your Help files. If your Help file contains bitmaps, you may need to specify where those bitmaps are located before compiling the file.

Click this to map topic IDs to numeric values. Context-sensitive Help can function only if IDs are mapped. This Help topic, for example, is context-sensitive.

Click this if you want to replace one set of topic IDs with an alternate set of topic IDs. You can compile Help files without specifying these options.

Click this if you want to specify menus or buttons for your Help file or register DLLs and DLL functions used as macros. You need to specify configuration settings only if you use these features in your Help file.

Click this to specify files for Help Workshop to store within the Help file's internal file system. Windows Help can access data files stored in the Help file system more efficiently than it can access files stored in the Windows file system. You need to specify these settings only if you use these features in your help file.

Saves changes to this project (.hpj) file, and then compiles it.

Displays all the settings that are currently specified for this project (.hpj) file. To change a setting, double-click it.

Provides a place for you to type the ID of the topic that WinHelp displays whenever the user initiates a jump to a topic that is unavailable. For Help files that do not have an associated contents (.cnt) file, WinHelp displays the default topic as the first topic when the user first opens the Help file and when the Contents button is clicked. If you do not specify a default topic, WinHelp uses the first topic of the first file listed in the project (.hpj) file.

Provides a place for you to type text that will appear in the title bar of the main Help window if no main window type is specified in the Windows Properties dialog box. If you leave this box blank and the contents (.cnt) file does not specify a title, the words "Windows Help" appear in the title bar of your main windows.

If this box is checked, Help Workshop displays messages about problems that probably will not adversely affect your Help file. Clear this box only if you are certain that the messages Help Workshop displays are not about problems that you need to correct.

If this box is checked, a progress report appears while you compile a Help file. The report includes the name of each topic (.rtf) file as it is processed.

Provides a place for you to type text that will appear when a user clicks the Help menu in a main window, and then clicks Version.

To include the date on which the Help file was compiled, add **%date** to the end of your text.

Provides a place for you to type text that will be appended to Help text that is copied or printed.

Does not compress Help files as they are compiled. Although Help files are larger if you choose this option, they compile more quickly. You might want to specify this option until you are ready to compile the final version of the file.

Uses both Hall and Zeck compression. If you use maximum compression, it takes approximately three times longer to compile your Help file than it would if you used no compression at all. You can temporarily turn off compression In the Compile a Help File dialog box.

Enables you to choose between Hall and Zeck compression. If you choose only Zeck compression, you can also choose Phrase compression and specify whether you want to use an existing phrase (.ph) file. More Help is available for each of the items below.

If this box is checked, Help Workshop uses Hall compression, which is more effective than Phrase compression for files that are 100 K or larger in size. If you use Hall compression, your file will compile more quickly than if you use Phrase compression--unless you use Phrase compression with an existing phrase (.ph) file, in which case Phrase compression is significantly faster.

Hall compression is most effective when used with Zeck compression. However, if your Help file will ultimately be compressed by a utility before it is copied to distribution disks, the utility might achieve higher compression if you use only Hall compression.

If this box is checked, Help Workshop uses Zeck compression, which, combined with Hall compression, is the most effective compression you can use. You can use Zeck compression without Hall compression to reduce compile time, but doing so will increase the size of your Help file.

If your Help file will ultimately be compressed by a utility before it is copied to distribution disks, the utility might achieve higher compression if you do not use Zeck compression on your Help files. In this case, clear this check box, and use Hall compression instead. Although this will cause the Help files on distribution disks to be smaller, the Help file on users' hard disks will be larger as a result.

If this box is checked, Help Workshop uses Phrase compression, which is sometimes more effective than Hall compression for files that are 100K or smaller in size. You might also want to use Phrase compression if you want to achieve only minimal compression without significantly increasing compile time. Note, however, that compile speed is faster only if you use an existing phrase file. If your file is larger than 100 K, or you do not have an existing phrase file, use Hall compression instead.

To reduce compile time if you have compiled this file using Phrase compression before, make sure the Use Existing Phrase (.ph) File box is checked. Do this only while you are fine-tuning your file, however. You will achieve the best results by using Phrase compression only if the Use Existing Phrase (.ph) file box is clear.

If this box is checked, and you have compiled this file using Phrase compression before, Help Workshop uses the existing Phrase (.ph) file instead of generating a new one. This can significantly reduce compile time. Use this option only while you are fine-tuning your file, however. You can achieve maximum compression by using Phrase compression only if the Use Existing Phrase (.ph) file box is clear.

Help is available for each item in this group. Click ? at the top of the dialog box, and then click the specific item you want information about.

Specifies the language of the Help file. The language is used by Help Workshop to determine how to display typographer's quotes (“”) and to determine whether topic files use DBCS characters.

When you compile this project (.hpj) file, the driver for the language you select must be on your computer. If it is not, you will be unable to compile this file.

Click this if the language you want to use is not in the list, and if you know the hexadecimal number that identifies the language

Provides a place for you to type the hexadecimal number that identifies the language whose sorting conventions you want to use. If you do not enter a valid number, you will be unable to compile your Help file.

If this box is checked, Help Workshop ignores non-spacing marks. A non-spacing mark is typically an accent mark—a character that prints over a preceding character . For example, the letter "a" can be followed by a "`", to create the character "à".

If this box is checked, Help Workshop ignores symbols such as punctuation marks when determining the sort order of keywords.

If this box is blank, commas or colons identify second-level index entries. To use one or more other characters, type those characters in this box.

Provides a place for you to type the name of the Help file that will be created when you compile this project (.hpj) file. You can give the Help file a name that is different from the name of your project file, but you must use the .hlp extension for the Help file. You can also use a long filename.

Provides a place for you to type the name of a log file that will be created when you compile this project (.hpj) file. The log file contains the text that appears on your screen as a file compiles. If you do not want to generate a log file, leave this box blank.

Lists the names and locations of topic (.rtf) files whose topics will be included in the compiled Help file. To add or remove topic files, click Change.

Provides a place for you to type the name of the contents (.cnt) file. The text in this file appears on the Contents tab when the Help file is opened.

Provides a place for you to type the name of the folder that will contain temporary files that are created as you compile a Help file. By default Help Workshop uses the folder that contains your topic (.rtf) files. Help Workshop does not use this temporary location unless bitmaps are missing or your compiled Help file is larger than 8 MB.

Specifies the path that you want Help Workshop to use instead of the path that is specified in this project (.hpj) file. If you move topic (.rtf) or bitmap (.bmp) files to a different drive or server, you can update this field instead of editing each individual path name. To specify or change the path prefix, click Edit.

Click this to specify or change the path that you want Help Workshop to use instead of the path that is specified in this project (.hpj) file .

Click this to add topic (.rtf) files to or remove them from the list. You can also click this to change other settings for your topic files.

Click this if you are not sure of the name or location of a file or folder. You will be able to browse through folders to find the file you want.

Lists the topic (.rtf) files whose topics make up the content of your Help file. Also lists files that are included in your Help file. Included files are ASCII text files that contain lists of topic files.

Click this to add topic (.rtf) files to this project (.hpj) file. After you click this button, locate the topic file that you want to add, and then double-click it.

Removes the selected topic (.rtf) file from this project (.hpj) file.

Click this to add an ASCII text file that contains a list of topic (.rtf) files. This provides an alternative to adding individual topic files in this dialog box.

Click this if you need to specify the location of topic (.rtf) files. Help Workshop searches for topic files in your project folder, which is the folder that contains your .hpj file. If a topic file is not in the project folder and its entry on the Topic Files list does not include a full path, or a path relative to the project folder, you must specify its location so Help Workshop can find it.

Help Workshop recognizes most DBCS languages and processes them correctly whether or not this box is checked. If your topic (.rtf) files use a double-byte character set, and the language is not Chinese, Japanese, or Korean, check this box.

If this box is checked, Help Workshop accepts any revision marks in your topic (.rtf) files. If unchecked, Help Workshop compiles the Help file as if no revisions were made.

Provides a place for you to type the name and location of an ASCII text file that contains a list of items. Depending on the circumstances, the file can include a list of topic (.rtf) files and their location, macros, window types, mapped topic IDs, or a set of topic IDs that are aliased to an alternate set of topic IDs. Including an ASCII text file provides an alternative to adding these items individually. If you do not know the name or location of the file you want to include, click Browse.

Lists the paths to your topic (.rtf) files. Help Workshop searches for topic files in your project folder, which is the folder that contains your .hpj file. If a topic file is not in the project folder and its entry on the Topic Files list does not include a full path, or a path relative to the project folder, you must specify its location here so Help Workshop can find it.

Click this to specify a path to your topic (.rtf) files.

Removes the selected path from the list.

Provides a place for you to type the path prefix that is currently specified in the project (.hpj) file. The path you type is probably no longer valid, or is a path that you do not want to use anymore. For example, if you have moved topic (.rtf) or bitmap (.bmp) files to a different folder or server, you can type the old path here and the new one in the Substitute This Path Prefix box. This prevents you from having to update each individual path in your project (.hpj) file.

Provides a place for you to type an updated path prefix, which replaces the prefix specified in the Current Prefix box above. For example, if you have moved topic (.rtf) or bitmap (.bmp) files to a different folder or server, you can type the new path here and the old one in the Current Prefix box. This prevents you from having to update each individual path in your project (.hpj) file.

Provides a place for you to type a short comment about the change. As it compiles, Help Workshop omits this comment, so it does not increase the size of your Help file. The comment can, however, help you track when or why you substituted a different path prefix.

If this box is checked, Help Workshop generates a full-text search index for the Help file. If you create a full-text search (.fts) file for users, the Find tab in Help is populated the first time users click it. In other words, users will not have to use the Find Setup wizard to create a full-text search file, which can take quite a while for larger Help files.

The disadvantage to creating an full-text search file for users is that it significantly increases compile time and takes up space on the distribution disks for your product. If you are shipping your product on a CD-ROM, you can probably provide the convenience of an .FTS file at little cost to you. If you are shipping on floppy disks, the size of the full-text search file can affect the number of disks needed to ship your product.

If this box is checked, Help Workshop includes topics that have no titles in the full-text search index. In most cases, untitled topics are context-sensitive help or pop-ups within topics. Untitled topics might be of little use to users.

Enables users to search only for individual words. Although this keeps a full-text search (.fts) file's size to a minimum, it prevents users from searching for phrases, matching phrases, or similar topics.

Enables users to search for complete phrases, such as a person's first and last name. Although this makes the full-text search index more usable, it increases the size of the full-text search (.fts) file.

Enables users to see matching phrases as they type them, which can make it easier for users to find a phrase. Although this makes the full-text search index more usable, it increases the size of the full-text search (.fts) file. If your Help file is large, or a user's computer is slow, specifying this option can significantly increase the amount of time it takes for Help to search the full-text search file.

Enables users to mark Help topics for later reference. Help can then search for information that is related to the information in the marked topics.

This feature is most useful for searching large or multiple Help files. Although it makes the full-text search index more usable, it increases the size of the full-text search (.fts) file.

Lists all the keywords in the Help file that have macros associated with them. These macros run whenever a user selects the specified keywords in the Help Index. To add an entry to this list, click Add.

Click this to associate a keyword with a macro. The macro you associate will run whenever a user selects the specified keyword in the Help Index.

Removes the selected keyword and its associated macro from the Keywords list.

Click this if you want to change the selected keyword or its associated macro.

Displays the title that will appear in the Topics Found dialog box if the selected keyword leads to multiple topics. In this case, the specified macro will run when a user double-clicks the topic in the dialog box.

Displays the macro or macros that are associated with the selected keyword. If you use multiple macros, separate them using a colon (**:**).

Provides a place for you to type or modify the keyword(s) that you want to associate with the macro(s) in the box below. You can specify multiple keywords in this box. The macros specified below will run whenever a user double-clicks any of these keywords. Separate multiple keywords with semicolons.

Provides a place for you to type or modify the macro(s) that you want to associate with the keyword(s) in the box above. You can specify multiple macros in this box. The specified macros will run whenever a user double-clicks any of the keywords specified above. Separate multiple keywords with semicolons.

Provides a place for you to type the topic title that will appear in the Topics Found dialog box if the selected keyword leads to multiple topics. In this case, the specified macro will run when a user double-clicks the topic title in the dialog box.

Lists the build tags of topics that you want to exclude when you compile your Help file. A build tag is an asterisk (*) footnote in your topic (.rtf) file that precedes all other footnotes, and it identifies a topic or topics that you want to include in or exclude from a build.

If a build tag footnote appears in this list and in the list of build tags to include, this list will take precedence and topics containing the footnote will be excluded from the Help file. Topics without build tag footnotes are automatically included in a Help file.

Click this to specify the build tag footnote (*) that identifies topics you want to exclude from the compiled Help file.

Removes the selected build tag footnote from the list.

Lists the build tags of topics that you want to include when you compile your Help file. A build tag is an asterisk (*) footnote in your topic (.rtf) file that precedes all other footnotes, and it identifies a topic or topics that you want to include in or exclude from a build.

If you want a topic with a build tag footnote to be included in a Help file, you must add the build tag to this list. Topics without build tag footnotes are automatically included in a Help file.

Click this to specify the build tag footnote (*) that identifies topics you want to include in the compiled Help file.

Removes the selected build tag footnote from the list.

Provides a place for you to type the build tag footnote.

Provides a place for you to type a short comment about the build tag footnote. The comment does not increase the size of your Help file.

Lists character sets you can use instead of the one that your computer is currently using. You might want to specify a different character set if you are creating a Help file in a language other than the one that your computer is using.

Displays the font for topic titles and keywords that appear on the Contents, Index, and Find tabs, and in the Topics Found dialog box. If no font is appears in this box, WinHelp determines which font is used. To change the font, click Change.

Changing the font is useful if your Help file will be shipped to international markets unlocalized. For example, WinHelp is localized to use larger fonts in Far East versions, and unlocalized Help files will use the larger fonts unless you specify your own.

Click this to change the font for topic titles and keywords that appear on the Contents, Index, and Find tabs, and in the Topics Found dialog box.

Specifies the fonts in topic (.rtf) files that you want to replace with a different font, point size, or character set. To make a substitution, click Add.

Click this to substitute different fonts for the fonts that are specified in your topic (.rtf) file.

Click this to remove the selected font substitution from the list.

Click this to change the selected font substitution.

Displays the font that is currently being used for topic titles and keywords that appear on the Contents, Index, and Find tabs, and in the Topics Found dialog box. If no font appears in this box, WinHelp determines which font is used.

Changing the font is useful if your Help file will be shipped to international markets unlocalized. For example, WinHelp is localized to use larger fonts in Far East versions, and unlocalized Help files will use the larger fonts unless you specify your own.

Specifies the point size of the font that is currently being used for topic titles and keywords that appear on the Contents, Index, and Find tabs, and in the Topics Found dialog box. If a bitmapped default font is specified, the range of available point sizes might be limited.

Lists fonts that you can replace with other fonts. To replace a font that appears in your Help file, click it, and then select the font you want to replace it with in the Replacement Font area.

Lists character sets that you can replace with other character sets. To replace a character set that is used in your Help file, click it, and then select the character set you want replace it with in the Replacement Font area.

Provides a place for you to specify the point size of the font you want to replace. To change the size of a font that appears in your Help file, specify its original size here, and then specify the size you want to replace it with in the Replacement Font area.

Lists fonts that you can use to replace the font that is specified in the Original Font area.

Lists character sets that you can use to replace the character set that is specified in the Original Font area.

Provides a place for you to type the point size that you want to use instead of the point size specified in the Original Font area.

Provides a place for you to type a short comment about the change. The comment does not increase the size of your Help file. The comment can, however, help you track when or why you substituted one font for another.

Lists the secondary window types that are specified for this Help file. If you want all topics in a Help file to be displayed in this window, you can specify the window type in the contents (.cnt) file or the topic (.rtf) file. After you have added a window type, you can change its properties on the other tabs in this dialog box. To add a window type, click the General tab, and then click Add.

Click this to add a new secondary window type in which you can display the topics in your Help file.

Removes the selected window type from the list.

Provides a place for you to type the text that will appear in the title bar of this Help window. If you do not specify title bar text, and there is no contents (.cnt) file that specifies title bar text, nothing appears in the title bar of this Help window.

Provides a place for you to type a short comment about the window type. For example, you might want to specify the circumstances under which this window type should be used. The comment does not increase the size of your Help file.

If this box is checked, the specified window remains in the foreground, even when users switch to a different window or program. This feature is especially useful for procedures because the Help topic remains visible while the user carries out a task.

If this box is checked, the height of the specified window changes automatically to accommodate the amount of text in it. The main Help window cannot be automatically sized.

If this box is checked, the specified window covers a user's entire screen. This box and the Auto-Size Height box cannot be checked at the same time.

If this box is checked, the size of the specified window is based on a 1024 x 1024 screen resolution. WinHelp automatically adjusts the window's size and position so it retains its relative size, regardless of the resolution of the user's screen. Lines of text will wrap differently, depending on a user's screen resolution.

If this box is clear, the exact coordinates specified below are used, regardless of a user's screen resolution. In this case, a window that seems reasonably sized on a high-resolution monitor will appear much larger on a low-resolution monitor. Lines of text wrap in the same place, regardless of a user's screen resolution.

Click this to display a sample window that you can position and size by dragging the window's borders. When you are finished, the Left, Top, Width, and Height boxes are automatically filled in, based on the size and position of the sample window.

Cancels customized size and position settings and allows WinHelp to determine the size and position of the window.

If the Adjust For User's Screen Resolution box is checked, these values determine the size and position of the specified window based on a 1024 x 1024 screen resolution. WinHelp automatically adjusts the window's size and position so it retains its relative size, regardless of the resolution of the user's screen. Lines of text will wrap differently, depending on a user's screen resolution.

If the Adjust For User's Screen Resolution box is clear, the coordinates specified here are used regardless of a user's screen resolution. In this case, a window that occupies half the screen of a low-resolution monitor will occupy much less space on a high-resolution monitor. Lines of text will wrap in the same place, regardless of a user's screen resolution.

If this box is checked, a Contents button appears on the button bar in the specified window. If a contents (.cnt) file is present, the Contents button displays the Contents tab in the Help Topics dialog box. If a contents file is not present, the Contents button displays either the default topic or the first topic in the first file that is listed in your project (.hpj) file. You can specify a default topic and the order of files in the Options dialog box.

If this box is checked, a Print button appears on the button bar in the specified window. The Print button prints the current topic.

If this box is checked, an Index button appears on the button bar in the specified window. The Index button displays the Index tab in the Help Topics dialog box.

If this box is checked, a Back button appears on the button bar in the specified window. The Back button displays the previously viewed topic in the current window.

If this box is checked, a Find button appears on the button bar in the specified window. The Find button displays the Find tab in the Help Topics dialog box.

If this box is checked, an Options button appears on the button bar in the specified window. The Options button displays a menu containing the following commands: Annotate, Copy, Print Topic, Font, Keep Help On Top, and Use System Colors. This menu also appears when users click a topic using their right mouse button.

If this box is checked, a Help Topics button appears on the button bar in the specified window. If a contents (.cnt) file is present, clicking the Help Topics button for the first time displays the Contents tab in the Help Topics dialog box. Thereafter, clicking the Help Topics button displays the tab that a user most recently worked with.

If there is no contents file, the Help Topics button displays either the default topic or the first topic in the first file that is listed in your project (.hpj) file. You can specify a default topic and the order of files in the Options dialog box.

If this box is checked, a pair of browse buttons appears on the button bar in the specified window. The browse buttons enable users to move from one topic to another, as specified by your browse sequence. For these buttons to work property, you must specify a plus sign (+) footnote and browse sequence information in the topics that you want to include in the browse sequence.

Shows the background color for the nonscrolling region in the specified window. A nonscrolling region is an area that is always visible while users scroll through the information in a topic. This is especially useful for titles or other information that should always be available. To specify a nonscrolling region, apply the Keep With Next paragraph style (or its equivalent) to the paragraph(s) that you want to place in a nonscrolling region.

Shows the background color for the specified window.

Click this to change the background color of the specified window.

Click this to change the background color for the nonscrolling region in the specified window.

Lists the macros that will run when the specified window opens for the first time. WinHelp runs the macros in the order shown in this list.

Click this to add a macro that you want to run when the specified window opens for the first time.

Removes the selected macro from the list.

Click this to change the selected macro.

Click this to add an ASCII text file that contains a list of macros. This provides an alternative to adding macros individually.

Provides a place for you to type the name of the macro that you want WinHelp to run when the specified window opens for the first time.

Provides a place for you to type a short comment about the macro. The comment does not increase the size of your Help file.

Provides a place for you to type or modify a macro that will run when the specified window opens for the first time.

Provides a place for you to type or modify a short comment about the macro. The comment does not increase the size of your Help file.

Lists the folders in which Help Workshop can find the bitmap (.bmp) files that you use in your topic (.rtf) files. Help Workshop searches for bitmap files in the folder that contains your project file, in the folder that contains the topic (.rtf) file that specified the bitmap, and in all the folders specified for your topic files. If a bitmap file is not in any of these folders, you must specify the folder here so Help Workshop can find it.

Click this to add a folder that contains bitmap (.bmp) files to the list.

Removes the selected folder from the list.

Lists topic IDs that are mapped to numeric values. If you want a program to call a Help topic directly (as in the case of context-sensitive Help), you must map the topic ID to a numeric value. The program passes this value to WinHelp to display the associated topic.

Click this to add a topic ID and its associated numeric value to the list.

Removes the selected topic ID and its associated numeric value from the list.

Click this to add an ASCII text file that contains a list of topic IDs and their associated numeric values. This provides an alternative to adding individual topic IDs and values.

Click this if you want to change the selected topic ID or its associated numeric value.

Provides a place for you to specify a prefix other than IDH_ for mapped topic IDs. A prefix other than IDH_ generates errors as you compile your help file — unless you specify that prefix here.

Provides a place for you to type the topic ID that you want to map to a numeric value. The topic ID is specified by a number sign (#) footnote in your topic (.rtf) file.

Provides a place for you to type the numeric value that you want to map to the topic ID above. You must map IDs to numeric values if a program calls them directly, as in the case of context-sensitive Help. The numeric value corresponds to the value that a program passes to WinHelp to display the associated topic.

Provides a place for you to type a short comment about the mapped topic ID. The comment does not increase the size of your Help file.

Provides a place for you to type or modify the topic ID that you want to map to a numeric value. The topic ID is specified by a number sign (#) footnote in your topic (.rtf) file.

Provides a place for you to type or modify the numeric value that you want to map to the topic ID above. You must map IDs to numeric values if a program calls them directly, as in the case of context-sensitive Help. The numeric value corresponds to the value that a program passes to WinHelp to display the associated topic.

Provides a place for you to type or modify a short comment about the mapped topic ID. The comment does not increase the size of your Help file.

Lists aliases, which are topic IDs that are replaced by alternate topic IDs. Topic IDs are specified by a number sign (#) footnote in your topic (.rtf) file.

Aliases enable you to reassign topic IDs to other topics, without having to edit your topic (.rtf) files. They are especially useful if you have made a change that causes a mapped numeric value to refer to a topic other than the one you originally intended. An alias enables you to display the intended topic without having to change the code in the program.

Click this to associate a topic ID with an alternate topic ID.

Removes the selected alias from the list.

Click this to add an ASCII text file that contains a list of alias statements. This provides an alternative to adding individual aliases in this dialog box.

Click this to modify the selected alias.

Provides a place for you to type the original topic ID. Although topic IDs are specified by a number sign (#) footnote in your topic (.rtf) file, the ID you type here does not need to be. If a program calls this ID, the ID you type below will appear instead.

Provides a place for you to type the replacement topic ID. Whenever a program calls the topic ID above, this topic will appear instead.

Provides a place for you to type a short comment about why you substituted one topic ID for another. The comment does not increase the size of your Help file.

Provides a place for you to type or modify the original topic ID. Although topic IDs are specified by a number sign (#) footnote in your topic (.rtf) file, the ID you type here does not need to be. If a program calls this ID, the ID you type below will appear instead.

Provides a place for you to type or modify the replacement topic ID. Whenever a program calls the topic ID above, this topic will appear instead.

Provides a place for you to type or modify a short comment about why you substituted one topic ID for another. The comment does not increase the size of your Help file.

Lists the macros that apply to your entire Help file. WinHelp runs these macros whenever the Help file is opened.

Click this to add a configuration macro that will run whenever your Help file is opened.

Removes the selected configuration macro from the list.

Click this to add an ASCII text file that contains a list of configuration macros. This provides an alternative to adding individual macros in this dialog box.

Click this to change the selected configuration macro.

Provides a place for you to type a macro that applies to your entire Help file. WinHelp runs this macro whenever the Help file is opened.

Provides a place for you to type a short comment about this configuration macro. The comment does not increase the size of your Help file.

Provides a place for you to type or modify a macro that applies to your entire Help file. WinHelp runs this macro whenever the Help file is opened.

Provides a place for you to type or modify a short comment about this configuration macro. The comment does not increase the size of your Help file.

Lists any data files needed by dynamic-link library (DLL) files that are used in your Help file.

Click this to add a data file that will be stored in the Help file's internal file system.

Removes the selected data file from the list.

Displays the name of the default Help file. WinHelp searches the default Help file for the topic IDs listed in the contents (.cnt) file (unless the topic statement specifies another Help file). Also specifies whether all the topics in the Help file should be displayed in a secondary window. To change this setting, click Edit.

Displays the name that will appear in the title bar of the Help Topics dialog box and Help windows if no title bar text is specified in the project (.hpj) file. To change this setting, click Edit.

Click this to specify a default Help file, a default window type if applicable, and default title bar text.

Provides a place for you to type the name of the default Help file. WinHelp searches the default Help file for the topic IDs listed in the contents (.cnt) file (unless the topic statement specifies another Help file).

Provides a place for you to specify a default secondary window type, which causes all the topics in this Help file to appear in the specified window. Specifying a default window type here is optional. If you specify a window type, it must be defined in your project (.hpj) file.

Provides a place for you to type the name that will appear in the title bar of the Help Topics dialog box and Help windows if no title bar text is specified in the project (.hpj) file.

Displays the headings and topics that will appear on the Contents tab. If no items are listed here, click Add Above or Add Below.

Click this to change the topic title, topic ID, Help file, or window type for the selected entry.

Removes the selected entry from the list.

Click this to add a heading or topic above the selected entry.

Click this to add a heading or topic below the selected entry.

Moves the selected item to the right one level in the hierarchy.

Moves the selected item to the left one level in the hierarchy.

Click this to specify the Help files whose keywords should appear on the Index tab of the Help Topics dialog box.

Click this if your Help file contains ALink and KLink macros that jump to topics in other files. The keywords for link files are not included in the combined index.

Click this if you want to display custom tabs in the Help Topics dialog box. To add a custom tab, you must have a dynamic-link library (.DLL) file that was written specifically for that purpose.

Adds a heading to the contents. Headings appear as book icons and can contain other headings or topics. Users can double-click headings to view their contents.

Adds a topic to the contents. Topics appears as page icons. Users can double-click topics to display them.

Includes a second contents (.cnt) file in this contents file. If WinHelp finds the included contents file on a user's disk, the topics in it appear in this contents. Including contents files enables you to supply additional information later or to easily replace an existing component of your Help system.

Provides a place for you to type the title that will appear in the contents.

Provides a place for you to type the name of the contents (.cnt) file that you want to include. If WinHelp finds this file on a user's disk, the topics in it appear in the currently selected spot in the contents.

Provides a place for you to type the ID for the topic that will appear when users double-click this title. Instead of an ID, you can also specify a macro or macros. For example, you might want to specify a macro that starts a program when users double-click this title. Separate multiple macro entries with colons.

Provides a place for you to type the name of the Help file that contains this topic. If the topic is in the default Help file, leave this field blank.

Provides a place for you to specify the window type that you want this topic to appear in. The window type you specify must be defined in your project (.hpj) file. If you want the topic to appear in the default window, leave this field blank.

Lists the files whose keywords will appear on the Index tab of the Help Topics dialog box.

Click this to add a Help file whose keywords you want to appear on the Index tab of the Help Topics dialog box.

Removes the selected file from the list. This prevents the keywords in that file from appearing in the combined index.

Click this to change the Help title, filename, or comment for the selected entry.

Provides a place for you to type a title for the Help file whose keywords you want to appear in this Help index. The title you type here will appear if users choose Custom in the Find Setup Wizard. It also appears in the Topics Found dialog box if identical topic titles in different Help files are found.

Provides a place for you to type the name of the Help file whose keywords will appear in this Help index.

Provides a place for you to type a short comment. The comment does not increase the size of your Help file.

Lists Help files that WinHelp will search for A- or K-keywords. If your Help file contains ALink and KLink macros that jump to topics in other files, make sure those files are listed here. The keywords in the listed files do not appear in the combined index.

Click this to add Help files that WinHelp will search for A- or K-keywords.

Removes the selected file.

Lists the files that are included in the combined index. If a file that contains ALink and KLink references is listed here, you do not need to add it to the box above.

Lists the custom tabs that will appear in the Help Topics dialog box. To add a custom tab, you must have a dynamic-link library (.DLL) file that was written specifically for that purpose.

Click this to add a custom tab to the Help Topics dialog box.

Removes the selected tab from the list.

Click this to modify the tab name, dynamic-link library (DLL) filename, or comment for the selected tab.

Provides a place for you to type a name for the custom tab. This name will appear on the tab in the Help Topics dialog box.

Provides a place for you to type the name of the dynamic-link library (DLL) file that contains the executable routines for the custom tab.

Provides a place for you to type a short comment. The comment does not increase the size of your Help file.

Provides a place for you to type or modify a name for the custom tab. This name will appear on the tab in the Help Topics dialog box.

Provides a place for you to type or modify the name of the dynamic-link library (DLL) file that contains the executable routines for the custom tab.

Provides a place for you to type or modify a short comment. The comment does not increase the size of your Help file.

Specifies the project (.hpj) file that you want to compile. To display a list of recently compiled project files, click the down arrow button in this box. To specify a new file, type its path and name, or click Browse.

Click this to locate the project (.hpj) file that you want to compile.

Begins compiling the specified project (.hpj) file.

If this box is checked, the Help Workshop window minimizes while it compiles the specified project (.hpj) file. Minimizing the Help Workshop window reduces compile time.

If this box is checked, Help Workshop does not compress Help files as they are compiled. Although Help files are larger if you choose this option, they compile more quickly. You might want to specify this option until you are ready to compile the final version of the file.

If this box is checked, the Help file opens immediately after it is compiled.

Specifies the Help file for which you want to create a report. To specify a file, type its path and name, or click Browse. Reports are especially useful if you want to extract information from a Help file that has already been compiled. For example, you can determine what the A- and K-keywords in a compiled Help file are, and then jump to them from your Help file.

Click this to locate the Help file for which you want to create a report.

Provides a place for you to type a location and name for the report file. Because the report will be an ASCII text file, you might want to give it a .txt filename extension.

Creates a report for the specified Help file.

Lists all the topic titles (as defined by dollar sign ($) footnotes) in the specified Help file.

Lists the title and hash number for every topic in the specified Help file. A hash number is a unique number that Help Workshop assigns to topics based on their topic IDs. It is this number, and not the topic ID itself, that WinHelp uses. If a topic doesn't contain an A-footnote or its K-footnotes are not specific enough, you can jump to the topic by using the JumpHash macro and its hash number.

Lists the keywords (as defined by K-footnotes) assigned to each topic in the specified Help file. You can jump to these keywords by using the KLink macro in your Help file.

Lists the keywords (as defined by A-footnotes) in the specified Help file. You can jump to these keywords by using the ALink macro in your Help file.

Lists all the text in the specified Help file from the first topic of the first topic (.rtf) file to the last.

Lists the types of files that you can create by using Help Workshop.

Specifies the contents (.cnt) file that you want to test. To specify a file, type its path and name, or click Browse. Help Workshop checks the syntax in the specified file to make sure it is correct. It then attempts to jump to every topic in the contents file and displays a message if a jump fails.

Starts testing the specified contents (.cnt) file.

Click this to locate the contents (.cnt) file that you want to test.

Specifies the macro(s) that you want to send to WinHelp. WinHelp will run the macros so that you can make sure they work. Separate multiple macro entries with colons.

Specifies the Help file that you want to test the macro on.

Sends the specified macro(s) to WinHelp.

Specifies the name and location of the Help file that you want to test.

Click this to locate the Help file that you want to test. To specify a file, type its path and name, or click Browse.

Lists the API commands that you can send in the WinHelp API. You can use this feature to mimic the commands a program sends to WinHelp.

Specifies the value that you want to send in the dwData parameter of the WinHelp API.

Initiates the API call to WinHelp.

If a project (.hpj) file is specified above, this box lists the mapped topic IDs that are specified in that file. To view a mapped topic, click its ID, and then click View Help. If you do not want to view a mapped topic, clear this box.

If this box is checked, and you display a mapped topic, you can view each subsequent topic by repeatedly clicking View Help.

Specifies the path and name of the project (.hpj) file. You need to specify this only if you want to view the mapped topics specified in the file. To specify a file, type its path and name, or click Browse next to this box.

Specifies the location and name of the Help file that you want to view.

Opens the specified Help file, based on the settings you selected in this dialog box.

Opens the specified Help file as if it were called from the WinHelp API HELP_FINDER command.

Displays the topic specified in the Mapped Topic IDs box as a pop-up.

Displays the specified Help file as a training card. Training cards are a set of Help topics in secondary windows that are configured to send information to and receive it from another program.

Opens the specified Help file as if it were a double-clicked file icon or as if it were invoked from an MS-DOS command prompt.

Click this to locate the file you want to include

Refreshes the list of mapped topic IDs for the project (.hpj) file specified above. You should click this whenever you specify a different project file.

Enter up to 8 characters for the name of the window type. This name can be used in jumps, macros, and the window footnote.

Procedure: This window type is normally used for displaying procedures. It is auto-sizing, contains three buttons on the button bar, and is positioned in the upper right corner of the screen.

Reference: This window type is normally used for displaying reference material. It is auto-sizing, contains three buttons on the button bar, and is positioned on the left side of the screen and takes up approximately two thirds of the width of the screen.

Error message: This window type is normally used for displaying error messages. It is auto-sizing, contains no buttons, and lets WinHelp determine the position (upper right corner of the screen unless the user changes the position).

Provides a place for you to type the prefix in your project file that you want to replace.

If this box is checked and Help Author is enabled, two new fields appear in the Topic Information dialog box, which appears when you use your right mouse button to click a topic, and then click Topic Information. The additional fields show which topic (.rtf) file the topic is in and what the topic ID is. This feature is especially useful for testing, debugging, and editing topics — but it does increase the size of your Help file. Therefore, make sure this box is clear when you compile your file for the last time.

Lists most of the project information for the Help file, including Windows, Configuration, and most of the Options information. You can use this information to create a project file, by saving the report with an .hpj extension, and opening it in Help Workshop as a project file.

Click this for more information about the settings in this dialog box.

Click this to include an ASCII text file that contains a list of defined window types.

An ASCII text file, called a “project file,” that contains information Help Workshop uses to combine topic (.rtf) files and other elements into a Help file.

An ASCII text file, called a “contents file,” that contains information Help Workshop uses to create contents and indexing information for a Help file.

Tips of the Day

Tips of the Day introduce new Help Workshop features that make it easier to create, compile, and test your Help files. Each tip provides information that will help you get the most out of Help Workshop. To turn Tips of the Day off or on, click Help, and then click Tip Of The Day. To print a Tip of the Day, use your right mouse button to click it, and then click Print Topic.

Tip: Including and excluding topics

You can exclude specific topics when you compile a Help file. This is useful if you want to create different versions of a help file based on specific criteria. For example, you can exclude incomplete topics or topics that are only relevant under specific circumstances. For more information, look up “Build tags” in Help for Help Workshop.

Tip: Extracting information from a Help file

You can create jumps to topics in other Help files by determining what ALinks and KLinks were used in the file. To extract ALink and KLink information from a compiled Help file, click the File menu, and then click Report. For help on an item in the Report dialog box, click ? in its title bar, and then click the item.

Tip: Viewing WinHelp messages

Help Workshop provides information about almost every command a program sends to WinHelp, every macro that WinHelp runs, the topic number and title of every topic that appears, the command for every hotspot you click, and more. To view this information, click the View menu, and then click WinHelp Messages.

Tip: Smarter compiling

You can compile a project file without opening it. To do this, click the File menu, and then click Compile. In the dialog box, you can choose settings that make compiling faster and testing easier. For example, you can temporarily turn off compression so files compile faster or include each topic's topic ID and the name of the .rtf file the topic is in. For help on an item in the Compile A Help File dialog box, click ? in its title bar, and then click the item.

Tip: Specifying bitmaps for different display types

You can combine bitmaps with different color depths by specifying all the bitmaps in your topic (.rtf) file, separated by semicolons. WinHelp will choose the bitmap whose color depth most closely matches a user's display. For example:

{bmc mono.bmp; 16.bmp; 256.bmp}

For more information, see Help in Help Workshop.

Tip: Testing the jumps in your contents file

You can determine whether the jumps in your contents (.cnt) file are correct by clicking the Test menu, and then clicking Contents File. This command tests the syntax of the specified contents file, and then tests each jump. If an error occurs, a message that contains information about the incorrect jump appears. For more information, see Help in Help Workshop.

Tip: Displaying general topic information

You can display information about each Help topic, in addition to more detailed error messages, by clicking the File menu and making sure Help Author is checked. This causes a topic number to appear in the title bar of each Help topic. It also provides two additional commands on the context menu that can help you test your files. For more information, see Help in Help Workshop.

Tip: Navigating within a Help file

To easily navigate within a Help file, click the File menu, and then make sure Help Author is checked. If a Help topic is open, you can move between topics by using key combinations. To move forward or backward one topic at a time, use CTRL+SHIFT+RIGHT ARROW or CTRL+SHIFT+LEFT ARROW. To jump to the beginning or the end of your Help file, use CTRL+SHIFT+HOME or CTRL+SHIFT+END. To jump to a specific topic, use CTRL+SHIFT+J. For more information, see Help in Help Workshop.

Tip: Determining a topic's ID and source file

If the Help Author command on the File menu is checked, and you use your right mouse button to click any topic (including pop-ups), you can click Topic Information to view information about a topic. If the Include .Rtf Filename And Topic ID box in the Compile A Help File dialog box was checked when the Help file was compiled, information about the .rtf file a topic is in, in addition its topic ID also appears. For more information, see Help in Help Workshop.

Tip: Jumping to specific topics

You can jump to a specific topic in a Help file by pressing CTRL+SHIFT+J in a topic window, and then specifying a topic ID or number. You can also run any macro by typing an exclamation mark (!), and then typing the short name of the macro.

Tip: More flexibility in assigning topic IDs

You are no longer limited to using alphabetic and numeric characters for unmapped topic ID names. (Unmapped topic IDs identify topics that are not called from a program.) You can now use spaces, some punctuation, and extended characters in unmapped topic ID names. For more information, see Help in Help Workshop.

Tip: Viewing mapped Help topics

You can easily jump to every mapped topic ID in your project file by clicking the File menu, and then clicking Run WinHelp. After you specify Help (.hlp) and project (.hpj) filenames, and then click the Refresh button, all the mapped topic IDs appear in the Mapped Topic IDs list. You can view individual topics or view all topics in sequence. For more information, see context-sensitive Help in the View Help File dialog box.

Tip: Adding the compile date to the Version dialog box

You can add the date on which your Help file was compiled to WinHelp's version dialog box. To do this, open your .hpj file, and then click the Options button. In the Display This Text In The Version Dialog Box area, add the following to the end of any text that appears there:

%date

The date will appear in English and cannot be translated.

Tip: Specifying a window type for individual topics

You can specify the window type that a topic should appear in when it is displayed from the Index or Find tab. To do this, use a greater-than sign (>) footnote in your .rtf file, and then specify the window type you want the topic to appear in. For example:

> main

For more information, see Help in Help Workshop.

Tip: Making a bitmaps background transparent

You can make the background of a 16-color bitmap transparent by adding the letter "t" to the bitmap statement in your .rtf file, as in the following example:

{bmlt 16color.bmp}

For more information, see Help in Help Workshop.

Tip: Associating macros with keywords

You can easily specify a macro that runs whenever a user chooses a particular keyword in the Help Index. To do this, open your .hpj file, and then click Options. Click the Macros tab, and then click Add. You can then specify the keyword and title that will appear on the Help Index tab and in the Topics Found dialog box. When a user double-clicks the keyword or title, WinHelp launches the macro. For help on an item in the Keyword Macros dialog box, click ? in its title bar, and then click the item.

Tip: Displaying pop-ups if ALinks or KLinks fail

You can specify a pop-up topic that appears if an ALink or KLink jump fails. For example, if the destination file is unavailable, you can display information about how users can obtain or install the file. For more information, see Help in Help Workshop.

Tip: Specifying the background color for pop-ups

You can choose a background color for pop-up topics by using the SetPopupColor macro. To change all pop-up – topic colors, open your project (.hpj) file, click Config, and then specify the SetPopupColor macro. To specify the color for an individual pop-up topic, specify the SetPopColor macro with the PopupId macro for the pop-up's hotspot. For more information, see Help in Help Workshop.

Tip: Getting Help on error messages

If you receive an error message while you compile a file and aren't sure what it means, look up the error in Help. Help provides a description of all error messages and recommends solutions. For example, if you receive error HC1003, look up "HC1003" in the Help Index for Help Workshop.

Tip: Localizing typographer's quotes

Help Workshop can convert the typographer's quotes (“”) in your .rtf file to quotation marks that are appropriate for specific languages. To localize these quotes, open your project (.hpj) file, and then click Options. Click the Sorting tab, and then specify a language. For more information, see Help in Help Workshop.

Tip: Using bitmaps supplied by Help Workshop

You can use standard bitmaps that are supplied by Help Workshop. Although these bitmaps may not be in your bitmaps directory, you can specify them in your .rtf file. For more information, see Help in Help Workshop.

Tip: No need for quotation marks in macros

As a general rule, you do not need to use quotes in macros if they appear in either your topic file or project file. Help Workshop supplies all the quotes for you and determines which type of quote to use in nested macros.

Help Workshop automatically converts all macros to their shortest form. You can use the long name of a macro to add legibility to your .rtf files without affecting the size of your Help file.

Tip: Running macros when a secondary window opens

You can specify macros that run the first time a secondary window opens. For example, you may want to specify macros that close other secondary windows or that add buttons to the button bar. To specify a macro for a secondary window, open your project (.hpj) file, click the Windows button, and then click the Macros tab. For help on an item in the Window Properties dialog box, click ? in its title bar, and then click the item.

Tip: Viewing messages that programs send to WinHelp

You can determine exactly what Topic IDs a program sends to WinHelp by clicking the View menu in Help Workshop, and then clicking WinHelp Messages. If the project file has never been opened or compiled by Help Workshop, only the mapped numeric values appear. However, if the project file has been opened or compiled at least once, the Topic ID is displayed along with its mapped numeric value.

Tip: Specifying whether a window should auto-size

You can set up topic windows so that their length is automatically adjusted to accommodate the amount of text in the topic. To do this, open your project (.hpj) file in Help Workshop, click the Windows button, and then select or clear the Auto-Size Height check box. For help on an item in the Window Properties dialog box, click ? in its title bar, and then click the item.

To create a jump to a topic or pop-up topic

1 Place the insertion point directly after the text or bitmap that you want to be the hotspot, and then type the topic ID of the topic you want to jump to. Do not put a space between the hotspot and the topic ID.

2 Select the hotspot and then:

- Apply the double-underline character style if you want another topic to be displayed.
- Apply the single-underline character style if you want a pop-up topic, to be displayed.

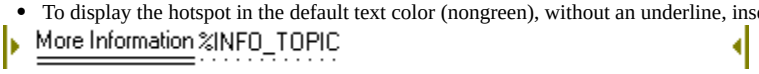
3 Select the topic ID, and then apply the hidden character style.

Example

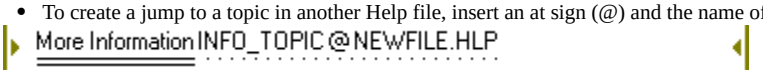
Tips

- If the hotspot text ends in a punctuation mark, include the mark when you apply the double-underline character style. Otherwise, a line break could cause the mark to appear at the beginning of a line.
- To display the hotspot in the default text color (nongreen), insert an asterisk (*) at the beginning of the topic ID.

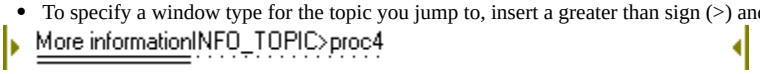
- To display the hotspot in the default text color (nongreen), without an underline, insert a percent sign (%) at the beginning of the topic ID.



- To create a jump to a topic in another Help file, insert an at sign (@) and the name of the other Help file at the end of the topic ID.



- To specify a window type for the topic you jump to, insert a greater than sign (>) and the window name at the end of the topic ID.



To add a hotspot that runs a macro

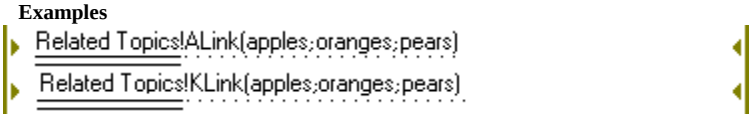
1 Place the insertion point directly after the text or bitmap reference that you want to be the hotspot, and then type an exclamation point (!). Do not leave a space between the exclamation point and the hotspot.

2 Type the macro command. Multiple macros can be specified by separating them with a colon (:) or semicolon(;).

3 Select the hotspot, and then apply the double-underline character style.

4 Select the exclamation point and the macro command, and then apply the hidden character style.

Examples



{button ,AL(`MACRO_REF_OVR;BAS_ASSOCIATIVE_JUMPS')} Related Topics

To create a bitmap that contains multiple hotspots

Click here to start Hotspot Editor.

For information about how to create a bitmap that contains multiple hotspots, click the Help menu in Hotspot Editor.

To add a bitmap to a topic

1 Place the insertion point where you want to display the bitmap.

2 To add a single instance of a bitmap, simply copy and paste it.

If you will use multiple instances of the bitmap, create a link to its file by using the following syntax:

{bmx filename.bmp**}**

For x, specify one or two of the values in the table below. For filename.bmp, specify the name of your bitmap file.

Value	Description
c	Aligns the bitmap as a text character.
l	Aligns the bitmap with the left margin.
r	Aligns the bitmap with the right margin.
t	Used with any of the above values, this indicates that the white pixels of the bitmap should be converted to the solid color closest to the background color of the window. This can be used only with 16-color bitmaps.

Note

- You can specify multiple bitmaps by separating their names with a semicolon. In this case, WinHelp displays the bitmap that most closely matches the number of colors that can be displayed on a user's computer. For more information, click Related Topics.

{button ,AL(`PRO_MULT_COLOR_GRAPHICS;PRO_COMP_RESOLUTIONS;BAS_ADDING_GRAPHICS;BAS_MULT_RESOLUTION')} Related Topics

To add video or animation to a topic

1 Place the insertion point where you want to add the multimedia file.

2 Type the reference in the following syntax:

{mci filename.ext**}**

For filename.ext, specify the name of the multimedia file you want to add.

Note

- You can specify options for the multimedia file and can align the embedded window. For more information, see the mci command reference.

To configure a bitmap for multiple color depths

1 Create a bitmap file for each color depth that you want to make available.

2 Place the insertion point where you want the bitmap to be displayed, and then create a bitmap statement that includes each filename. Separate filenames with semicolons, as shown in the example.

WinHelp automatically uses the bitmap file that is most appropriate for a user's computer.

Example

{bml dog_16.bmp;dog_256.bmp;dog_mono.bmp}

{button ,AL(`PRO_ADD_PICTURE;BAS_MULT_COLOR_GRAPHICS;BAS_MULT_RESOLUTION;BAS_ADDING_GRAPHICS')}
Related Topics

If you use a bitmap in more than one place in your Help file, use a bitmap statement to create a link to it instead of copying and pasting the image. This reduces the size of your Help file because only one copy of the bitmap needs to be included in the file.

The absolute position is a setting that tells WinHelp that the coordinate values you specify for a window definition are to reflect the number of pixels, regardless of screen resolution. As a result, the size and position of the window will vary widely on different monitors, but the text in the windows will always wrap at the same places. If this setting is not specified, WinHelp adjusts the settings for the resolution of the monitor on which the Help file is displayed, and the window size and position will be roughly the same on all monitors, regardless of resolution.

An accelerator key is a keystroke that provides a keyboard equivalent to commands displayed on menus or a button bar.

An ALink macro provides a link to topics by means of keywords listed in the A-footnotes of topic (.rtf) files. When a user clicks a hotspot that runs an ALink macro specifying a keyword, WinHelp jumps to the topic that contains the keyword, or, if more than one topic contains the same keyword, WinHelp displays the Topics Found dialog box listing the titles of the topics. Keywords used in A-footnotes are reserved for use by ALink macros only.

A contents (.cnt) file provides the information for the Contents tab, such as the titles of topics. You can also use a contents file to group multiple files, and to specify additional files to search in ALink and KLink macros. The contents file is specified in the project (.hpj) file.

An authorable button is a button in the text area of a Help window that can be used as a hotspot or to run macros. You can specify the text label for an authorable button.

Browse buttons are right-arrow (>>) and left-arrow (<<) buttons that the user clicks to display the next topic or the previous topic in a browse sequence.

A browse code is a number or character string in the A-footnote of a topic to specify the topic's position in a browse sequence. The topics are sorted in ascending order by using browse codes.

A browse sequence is a group of topics that the user can display one at a time, forward or backward, by clicking the browse buttons.

A build tag is an optional label that you use to specify whether a topic should be included in or excluded from a specific build. Build tags let you prepare several different Help files from a single set of topic files. After you have assigned build tags to topics, you can specify in the project (.hpj) file which topics to include in the build.

A citation is a message that you specify to appear at the end of the text when a user copies or prints a topic.

Color depth is the number of solid colors a computer can display on its monitor at one time.

A context menu is the menu that appears when the user clicks the right mouse button in a Help window.

A context number (or map number) is a unique value you assign to a topic that you want another program to call directly (as in the case of context-sensitive Help). The program passes this value to WinHelp to display the associated topic. You specify context numbers for topics in the [MAP] section of the project (.hpj) file. Context numbers are used by the JumpContext macro.

Custom buttons are buttons on a Help window's button bar created by using the CreateButton macro.

The default Help file is the file WinHelp searches for the topic IDs listed in the contents (.cnt) file (unless the topic statement specifies another help file). The default Help file is specified in the contents file.

The default topic is the topic WinHelp displays whenever the user initiates a jump to a topic that is unavailable. For Help files that do not have an associated contents (.cnt) file, WinHelp displays the default topic as the first topic when the user first opens the Help file and when the Contents button is clicked. If you do not specify a default topic, WinHelp uses the first topic of the first file listed in the project (.hpj) file.

An entry macro is a macro that you specify to run whenever a user displays a topic. You specify an entry macro for a topic in an exclamation mark (!) footnote in the topic (.rtf) file. You can define a series of entry macros by separating them with colons or semicolons. Multiple macros are run in the order they are listed.

A footnote is used in a topic to provide a unique topic identifier, identify index keywords, and provide other special information to Help Workshop.

Full-text search is the ability to search for any word or phrase contained in Help file topics. Users can initiate a full-text search by using the Find tab in the Help Topics dialog box.

A hash number is a unique number generated from a topic ID. It is this number, and not the topic ID itself, that WinHelp uses. You can use the JumpHash macro to jump to a topic identified by a hash number, or you can use the Report command to list the hash numbers of all topics in a Help file.

Help Author mode is a way of running a Help file for testing purposes, providing the author with special navigational ability and other information about the structure of topics, links, windows, and macros. The Help Author command is available on the Help Workshop File menu.

A Help makefile contains a list of project (.hpj) files that you want to compile. Help Workshop compiles the project files in the order they are listed. Help makefiles have an .hmk extension.

A nonscrolling region is a specified portion at the top of a topic that always remains visible, even if the user scrolls to the bottom of a long topic. The nonscrolling region is separated from the scrolling region by a thin, black line, and it can contain text, bitmaps, and hotspots. You specify a nonscrolling region by applying the Keep With Next paragraph style (or equivalent) to the paragraph(s) in the topic (.rtf) file.

The Help Topics dialog box serves as the initial interface for the entire Help system. Typically, it contains Contents, Index, and Find tabs. The Help Topics dialog box appears when Help is run by using the HELP_FINDER command of the WinHelp API, or when a user double-clicks a Help file that is associated with a contents (.cnt) file that contains information for the Contents tab.

High Contrast is a display option that directs Windows to use colors and fonts that are designed for easy reading. Users can set this option on the Display tab in the Accessibility Options program in Control Panel.

The history is a list of topics displayed in the main Help window, in the order the user viewed them. To see this list, click the Options menu, and then click Display History Window. This list is kept separately from the history list used by the Back button (and macro).

A hotspot is an area on a screen that the user can click to jump to another topic or pop-up window, or to execute one or more macros.

A keyword is a word or phrase associated with a topic, usually for indexing purposes. The same keyword can be associated with several topics. You assign keywords to topics in footnotes in the topic (.rtf) file.

A KLink macro provides a link to topics by means of keywords listed in the K-footnotes of topic (.rtf) files. Keywords used in K-footnotes are also used in the index, so a KLink jump to a keyword has the same effect as clicking that keyword in the index. By contrast, ALink macros use keywords listed in A-footnotes, and these keywords are reserved for use by the ALink macro only. When a user clicks a hotspot that runs a KLink macro, WinHelp displays the Topics Found dialog box listing all the titles of topics associated with the K-keywords specified by the macro. The user can then jump to the desired topic by double-clicking the topic title.

The main Help window, unlike a secondary window, has a menu and a default button bar, and it cannot be autosized. WinHelp has default settings for the main window, but you can customize it by specifying it as "main" in the [WINDOWS] section of the project (.hpj) file. Only one main window can be displayed at a time.

Multi-index keywords are words or phrases listed in special topic footnotes for use in indexes other than the one designated in the K-footnotes. With the exception of the A-keywords, these keywords can be accessed only by a program calling WinHelp with the HELP_MULTIKEY command.

A second-level index entry is a subset of a first-level index entry. On the Index tab, second-level index entries appear indented.

A secondary window, unlike a main window, does not have a menu bar. Secondary windows must be specified in the [WINDOWS] section of the project (.hpj) file. The user can display up to nine secondary windows at one time.

A topic ID is a string that identifies a topic or a location within a topic to display.

A transparent bitmap is a 16-color bitmap in which WinHelp replaces the white pixels with the color of the background of the current Help window. Typically, marking a bitmap as transparent enables it to blend in better with the background.

Training cards are a set of Help topics in secondary windows that you can configure to communicate with another program. Authors can design training cards that display different topics based on what the user does within that other program.

A pop-up topic is a topic that appears in a pop-up window. A pop-up topic can contain text, graphics, multimedia files, or hotspots. Typically, you use pop-up topics to convey additional information, such as the definition of a term or the description of a dialog box component.

A pop-up window is a window in which a pop-up topic is displayed. Pop-up windows have no menu or button bars and appear on top of the topic from which they are called. Pop-up windows disappear when a user clicks anywhere on the screen. A pop-up window can contain text, graphics, multimedia files, or hotspots. However, if you have a hotspot in a pop-up window, the first pop-up window closes when the user clicks the hotspot. WinHelp can display only one pop-up window at a time.

About project files

The project (.hpj) file is an ASCII text file that is used to compile your Help file. The project file contains all the information Help Workshop needs to combine topic (.rtf) files and other elements into a Help file.

You use Help Workshop to create and modify project files. When you create a new project file, the minimum settings you need are already made for you. The remaining settings depend on the size and complexity of your Help file.

Each section and option in the project file has a different purpose, and each can be used to improve the effectiveness and usability of your Help file. For more information about the project files settings that are available in Help Workshop, open a project (.hpj) file or create a new one, and then do one of the following:

- For Help on items in dialog boxes, click ? at the top of the dialog box, and then click the item.
- For Help on items in a window, click the item or its label by using your right mouse button, and then click What's This.

{button ,AL(`BAS_HPJ;NEW_AUTHOR_HPJ;PRO_HPJ')} Related Topics

About the contents file

The contents (.cnt) file is an ASCII text file that provides the instructions for the Contents tab in the Help Topics dialog box, and directs WinHelp to display the keywords of specified Help files on the Index and Find tabs.

You design your table of contents by specifying the following in your contents file.

- Headings
- Topics
- Commands

To create and edit contents files, use Help Workshop. For more information, click Related Topics.

Note

- The Contents tab does not appear in the Help Topics dialog box unless the contents file contains at least one topic that jumps to a Help file on a user's computer. The valid jump must be present in the contents file when a user first opens the associated Help file.

{button ,AL(`BAS_CNT_31;3X_CONTENTS_DESIGN;NAVIGATION_DESIGN;CONTENTS_DESIGN;MULTIPLE_HELP_DESIGN;BAS_CNT;MORE_ABOUT_CNT_FILES;CONTENTS_DESIGN')} Related Topics

Headings

Headings are represented by book icons and can contain a group of related topics and other headings. Users display the contents of a heading by double-clicking it.

Topics

Topics are represented by page icons and either display a topic or run a macro when a user double-clicks them. Along with the topic title and topic ID, you can specify which Help file a topic is in, and which window type you want the topic to appear in.

Commands

Commands specify the scope and appearance of your table of contents and index, including:

- The name of the default Help file (and window)
- The title to display in the Help Topics dialog box
- The names of Help files whose keywords you want to include on the Index tab and whose text you want to include on the Find tab of the Help Topics dialog box
- The names of Help files WinHelp should search for keywords if the ALink and KLink macros are used in topics
- The names and locations of other tabs to display in the Help Topics dialog box.

Compiling Help files

You create Help files by compiling the project file. This brings together all the information in your project (.hpj) file, topic (.rtf) files, bitmap (.bmp) files, and DLL data files.

When you compile your Help file, you can choose from several compression options that make your Help file smaller. The greater the compression, the longer it takes to compile your file. You can temporarily turn off compression to reduce the amount of time it takes to compile.

{button ,AL("BAS_THE_HELP_COMPILER;BAS_ABOUT_HPJ_FILES;PRO_HPJ_COMPRESSION")} Related Topics

Features for testing Help files

WinHelp and Help Workshop have the following testing features that make it easier to test your Help files:

- Help Author mode
- Monitoring WinHelp messages
- Test Contents File command
- Compare macro

{button ,AL("Test;Compare;BAS_HELP_AUTHOR_MODE;BAS_COMPILE_MSG;NEW_COMPILE")} Related Topics

Help Author mode

If a check mark appears beside the Help Author command on the File menu in Help Workshop, a line enabling Help Author features is added to your Win.ini file. Help Author mode provides additional information about the topics in a Help file, and enables you to more easily navigate from one topic to the next.

{button ,AL(`PRO_HELPAUTHOR;BAS_HELP_AUTHOR_MODE')} Related Topics

Compare macro

The Compare macro enables you to display side-by-side instances of two Help files. This macro is especially useful for comparing original and translated versions of the same Help file. You can run the Compare macro while you are viewing your Help by pressing CTRL+SHIFT+J, which displays the Jump dialog box.

{button ,AL(`Compare')} Related Topics

Test Contents File command

You can test your contents (.cnt) file in Help Workshop by clicking the Test menu, and then clicking Contents File. Help Workshop checks to make sure that the syntax in your contents file is correct. Help Workshop then jumps to every topic specified in the contents file and displays a message if one of the jumps does not work.

{button ,AL(`PRO_TEST_CNT')} Related Topics

Monitoring WinHelp messages

You can view WinHelp messages to determine whether actions in a Help file are working correctly. Using this feature, you can monitor WinHelp API calls, context-sensitive calls, which macros are being run, hotspot jumps, which topics and pop-ups are current, and which configuration (.gid) file is being read.

To monitor WinHelp messages in Help Workshop, click the View menu, and then click WinHelp Messages. Then open a Help file.

Tips

- Messages are added to the log file at the insertion point, so if you move the insertion point during a session, make sure you return it to the end of the text.
- Help Workshop tracks WinHelp messages for all instances of WinHelp. Make sure that only one instance of WinHelp is running.

{button ,AL(`PRO_TRACK_WINHELP_MSGS')} Related Topics

About Help Author mode

You can display information about a Help topic in addition to more detailed error messages by clicking the File menu in Help Workshop, and then making sure Help Author is checked. Help Author mode provides the following features:

Topic numbers
Easier navigation
Information about topics
Information about hotspots

{button ,AL(`BAS_TESTING_HELP_FILES;BAS_COMPILE_MSG;')} Related Topics

Topic numbers

If Help Author is enabled, title bar text is replaced by unique topic numbers that identify each topic's position in the Help file. The first topic in the first topic (.rtf) file that is listed in your project (.hpj) file appears as topic number one. Subsequent topics are listed in the order specified in your project file. The number for a topic may change if the sequence of topics in your topic file changes or if you add or remove topics. To display a specific topic by its number, press CTRL+SHIFT+J in a topic, and the type the topic number.

Easier navigation

If Help Author is enabled, you can use key combinations to navigate within your Help file. If a Help topic is open, you can move to the following or preceding topic by pressing CTRL+SHIFT+RIGHT ARROW or CTRL+SHIFT+LEFT ARROW. To move to the beginning or end of your Help file, press CTRL+SHIFT+HOME or CTRL+SHIFT+END.

Information about topics

If Help Author is enabled, a Topic Information command appears when you click a topic by using your right mouse button. Clicking this command displays the topic's title (as specified by its $ footnote), any macros specified by an entry macro (!) footnote, the current window type, and the Help filename and path. Additional information appears in this dialog box if you selected the Include .Rtf Filename And Topic ID In Help File check box in the Compile a Help File dialog box before you compiled. This adds the name of the .rtf file that contains the topic and the topic's ID to the Topic Information dialog box.

Information about hotspots

If Help Author is enabled, an Ask On Hotspots command appears when you click a topic by using your right mouse button. If a check mark appears beside this command, WinHelp does not run the hotspot jump or macro when you click it. Instead, it displays the hidden text associated with the hotspot jump or macro. You can then choose to run the jump or macro, or return to the topic.

Note

- The Ask On Hotspots command might not work correctly if there are no keywords in your Help file.

Compiler messages

Windows 95 can display up to 64K of compiler messages (Windows NT can display up to 1 MB). However, if you specify a log file in which to save messages, there is no size limit for the file. For more information about a message and how to resolve it, use the Index or Find tab in Help Workshop Help, and look up the corresponding message ID, for example "HC1003." While you compile your Help file, Help Workshop may display messages of the following types:

? Notes
? Warnings
? Errors

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_THE_HELP_COMPILER;EXEC_HCW;EXEC_HCRTF;PRO_HPJ_ERRORLOG;PRO_HPJ_ERRORLOG')} Related Topics

Notes are conditions that you should be aware of. They will probably not cause serious problems when you open your Help file. For example, "HC1003: Note: A paragraph mark is formatted as a hidden character."

Warnings are conditions that will result in a defective Help file. For example, "HC3059: Warning: A hotspot is specified without a macro or topic ID."

Errors are conditions that prevent the Help file from compiling. For example, "HC5005: Error: The file cannot be found."

Checklist for translating Help files

WinHelp can display Help files compiled in any of 29 different languages. In addition, WinHelp provides tools for translating a Help file from one language to another while retaining the same structure.

You typically carry out the following tasks sequentially when you translate Help files:

1 Translate topic (.rtf) files

2 Change settings in project (.hpj) files

3 Translate contents (.cnt) files

4 Compare original and translated versions

{button ,AL(`test;BAS_SORT_LANGUAGE;BAS_ABOUT_HPJ_FILES;BAS_PROJ_FILE_EDITOR;BAS_CNT_OVERVIEW;BAS_CONTENTS_FILE_EDITOR;BAS_TESTING_HELP_FILES;BAS_PROJECT_FILE_CHECKLIST;BAS_CONTENTS_CHECKLIST;BAS_THE_HELP_COMPILER;translate_hpj')} Related Topics

Translate topic (.rtf) files

Create copies of the original topic files before you translate the text. You will probably want to translate all the text in topics, including K-keywords, titles, and the text on authorable buttons. When translating K-keywords, retain one-to-one relationships for the sake of any KLink macros in the Help file. Be sure that keywords specified in KLink macros are updated to specify translated keywords. Be sure to retain the original topic IDs in the # footnotes, any hidden jump identifiers, browse tags, build tags, and macros.

Change settings in project (.hpj) files

Create a copy of the original project file. In Help Workshop, open the new project file, and then check the Translation command on the File menu. Most settings that should not be translated become unavailable. Make any necessary changes, including:

- The sorting order for keywords
- The default character set
- The default fonts
- Any path changes to point to the translated topic files
- Whether the topic (.rtf) files use a double-byte character set (DBCS)

Translate contents (.cnt) files

Create a copy of the original contents files. In Help Workshop, open the new contents file, and then check the Translation command on the File menu. Any settings that should not be translated become unavailable. Change headings and topic titles as needed.

Compare original and translated versions

The Compare macro enables you to view two Help files at once. This enables you to view the original topics and the corresponding translated topics side-by-side. To run the Compare macro, press CTRL+SHIFT+J when a topic is open, and then type the following:

!compare("filename.hlp")

For filename.hlp, substitute the name of the Help file you want to compare with the file that is open.

Customizing topic windows

WinHelp includes a predefined window called the main window, which has a menu bar and button bar. You can customize the main window or create new secondary windows.

You can define as many as 255 window types (including the main window) in your project (.hpj) file. WinHelp can display up to 10 windows at a time (the main window and nine secondary windows). You can customize the following window properties:

- Size and position
- Background color
- Title bar text
- Button bar
- Menus and menu commands
- Stay-on-top status
- Automatic window sizing
- Macros that run when a window opens

{button ,AL(`BAS_AUTH_MENU;BAS_CONFIG_MACRO;NAVIGATION_DESIGN;MACRO_REF_OVR;BAS_AUTH_BTN;BAS_AUTH_BTN;WINDOW_DESIGN')} Related Topics

Changing a window's size and position

You can specify the size and position of a window by opening your project (.hpj) file in Help Workshop, and then clicking the Windows button. Size and position settings are on the Position tab.

Tip

- For help on an item in the Window Properties dialog box, click ? in its title bar, and then click the item.

{button ,AL("PRO_HPJ_WINDOW_POSITION")} Related Topics

Changing a window's background color

You can specify the background color of a window by opening your project (.hpj) file in Help Workshop, and then clicking the Windows button. Background color settings are on the Color tab.

Tip

- For help on an item in the Window Properties dialog box, click ? in its title bar, and then click the item.

{button ,AL("PRO_HPJ_BACKGROUND_COLOR")} Related Topics

Changing a window's title bar text

You can specify the background color of a window by opening your project (.hpj) file in Help Workshop, and then clicking the Windows button.

Tip

- For help on an item in the Window Properties dialog box, click ? in its title bar, and then click the item.

{button ,AL("PRO_ADD_WINDOW_TITLE")} Related Topics

Changing a window's button bar

You can add buttons to or remove them from a window's button bar by opening your project (.hpj) file in Help Workshop, clicking the Windows button, and then clicking the Buttons tab.

Notes

- For help on an item in the Window Properties dialog box, click ? in its title bar, and then click the item.
- In addition to the buttons provided by Help Workshop, you can also use Help macros to add up to 22 buttons to the button bar in a Help window. For more information, click Related Topics below.

{button ,AL("PRO_HPJ_MAIN_BUTTON_BAR;PRO_HPJ_ADD_BUTTON_BAR;MACRO_REF_BUTTON")} Related Topics

Changing the main window's menu commands

You can add menus and menu commands to the menu bar in a main window by adding the necessary macros to the configuration section of your project (.hpj) file. To do this, open your project file in Help Workshop and click the Config button.

Tip

- For help on an item in the Configuration Macros dialog box, click ? in its title bar, and then click the item.

{button ,AL("PRO_HPJ_WINDOW_CONFIG_MACROS;REF_CUST_MENU;MACRO_REF_MENU")} Related Topics

Changing a window's on-top state

You can specify that a window remains in the foreground, even when a user jumps to another window or program. To do this, open your project (.hpj) file in Help Workshop, click the Windows button, and then make sure Keep Help Window On Top is checked.

Note

- Users can override the Keep Help Window On Top setting by clicking a topic using their right mouse button, and then clicking Keep Help On Top.

Specifying whether a window should auto-size

You can set up secondary windows so that their length automatically adjusts to accommodate the amount of text in the topic. To do this, open your project (.hpj) file in Help Workshop, click the Windows button, and then select the Auto-Size Height check box.

Notes

- The Auto-Size Height setting overrides all other height settings and can be used only for secondary windows.
- For help on an item in the Window Properties dialog box, click ? in its title bar, and then click the item.

Specifying macros that run when a window opens

You can run macros every time a particular window type opens by associating macros with that window type. For example, you may want to associate macros that add special menus, menu commands, and buttons to a window. To do this, open your project (.hpj) file in Help Workshop, click the Windows button, and then click the Macros tab.

Tips

- By adding entry macros to certain topics, you can make a particular command or button appear when a user opens some topics but not others.
- For help on an item in the Window Properties dialog box, click ? in its title bar, and then click the item.
- You can also assign macros to individual window types by opening your project (.hpj) file in Help Workshop, clicking the Windows button, and then clicking the Macros tab. Any macros you assign to the main window will override macros assigned to the main window in the Help Configuration dialog box. However, any macros you assign to a secondary window apply only to that window type.

{button ,AL(`MACRO_OVERVIEW;MACRO_REF_MENU;PRO_HPJ_WINDOW_CONFIG_MACROS')} Related Topics

Specifying macros that apply to your entire Help file

You can specify macros that run whenever a user opens a Help file. For example, you can use macros to specify optional buttons, menus, or menu items for the main window and to access programs and capabilities outside the Help system. To do this, open your project (.hpj) file in Help Workshop, and then click the Config button.

Tips

- For help on an item in the Configuration Macros dialog box, click ? in its title bar, and then click the item.
- You can also assign macros to individual window types by opening your project (.hpj) file in Help Workshop, clicking the Windows button, and then clicking the Macros tab. Any macros you assign to the main window will override macros assigned to the main window in the Help Configuration dialog box. However, any macros you assign to a secondary window apply only to that window type.

{button ,AL(`BAS_HPJ_CREATE_WINDOW;BAS_AUTH_BTN;MACRO_OVERVIEW;MACRO_REF_OVR')} Related Topics

Overview of full-text search features

WinHelp version 4.0 includes a full-text search feature, which enables users to search through every word in Help files to find a match.

In most cases, when a user clicks the Find tab in the Help Topics dialog box for the first time, the Find Setup wizard appears. The wizard helps users set up a full-text search index. A full-text search index lists all the unique words in the Help file.

Instead of having users create their own full-text search index, you can create it for them and ship it with your Help files. The advantage of shipping a full-text search index is that full-text search features are immediately available to users. The disadvantage is that a full-text search index takes up space on your product's distribution disks.

To create a full-text search index for your users, open your project (.hpj) file, and then click the Options button. Click the FTS tab, and change settings as needed. This causes a full-text search (.fts) file to be created whenever you compile your Help file with maximum or Hall compression turned on.

{button ,AL(`NEW_WINHELP_INTERFACE;nofind_cnt;PRO_FULLTEXT_INDEX')} Related Topics

Help system titles

By specifying titles for different elements in your Help system, you can orient your users. For information about creating these titles, click one of the buttons below.

Title bar text for the Help Topics dialog box
Headings in the Topics Found dialog box
Title bar text for Help windows
File titles in the Find Setup wizard
Heading and topic titles on the Contents tab

{button ,AL(`BAS_PROJ_FILE_EDITOR;BAS_CONTENTS_FILE_EDITOR;BAS_HPJ_CREATE_WINDOW;PRO_ADD_WINDOW_TITLE;PRO_ADD_TITLE;PRO_CNT_ADD_BOOK;PRO_CNT_ADD_TOPIC;PRO_HPJ_TITLE;PRO_CNT_CREATE;PRO_CNT_COMBINE_HELP_FILES')} Related Topics

Title bar text for the Help Topics dialog box

You specify text for the title bar of the Help Topics dialog box by opening your contents (.cnt) file, and then typing the title in the Default Title box. If your Help system has more than one Help file, specifying a default title ensures that the Help Topics dialog box has the same title regardless of which Help file is open.

If no default title is specified in the contents file, WinHelp uses the Help title specified for the current Help file. You can specify the Help title by opening your project (.hpj) file, clicking the Options button, and then typing the title in the Help Title box.

If no default title is specified in the contents file and no Help title is specified in the project file, WinHelp displays the filename of the current Help file in the title bar of the Help Topics dialog box.

Headings in the Topics Found dialog box

From the Help Index, the Topics Found dialog box appears when a user double-clicks a keyword that is associated with more than one topic. From within topics, the Topics Found dialog box appears when users click text that is associated with an ALink or KLink macro. The topic titles in this dialog box correspond to the titles you specify in the $ footnote in your topic (.rtf) file.

Title bar text for Help windows

You can specify text for the title bar of each window type by opening your project (.hpj) file, and then clicking the Windows button.

If no title bar text is specified for a window type, WinHelp uses the default title specified in your contents file. You can specify a default title, by opening your contents (.cnt) file, and then typing the title in the Default Title box.

If no default title is specified in the contents file, WinHelp uses the Help title specified for the current Help file for the main window, and leaves the title blank in secondary windows. You can specify the Help title by opening your project (.hpj) file, clicking the Options button, and then typing the title in the Help Title box.

If no default title is specified in the contents file and no Help title is specified in the project file, "Windows Help" appears in the title bar of the main Help window and secondary window titles will be blank.

Note

- A Help window's title bar text also appears on a taskbar button when a Help window is open. The taskbar is the bar on your Windows 95 desktop that has the Start button on it. Buttons representing running programs appear on this bar.

File titles in the Find Setup wizard

When users click the Find tab in the Help Topics dialog box for the first time, the Find Setup wizard typically appears. If a user chooses Custom in this wizard, a list of current Help files appears. From this list, users can choose which files they want to include in their full-text search index. The items in the list correspond to the Help titles specified for index files in your contents (.cnt) file. You can specify Help titles for index files by opening your contents file, and then clicking the Index Files button.

Heading and topic titles on the Contents tab

You specify the text for the headings and topics on the Contents tab by opening a contents (.cnt) file in Help Workshop, clicking Add Above or Add Below, and then filling in the appropriate text boxes. The text you specify for topics is usually the same as the titles specified by the $ footnotes in your topic (.rtf) files.

Designing context-sensitive Help

In WinHelp version 4.0, you can provide context-sensitive Help for many of your program's interface elements. In dialog boxes, users typically display context-sensitive help by clicking ? at the top of a dialog box, and then clicking the item. In windows, users typically display context-sensitive help by clicking an item using their right mouse button, and then clicking What's This.

Help authors and program developers typically work together to implement context-sensitive Help. Help authors write the relevant Help topics, and program developers modify the program to call the correct Help topic. As part of this modification, developers assign numeric values to items that users are likely to want more information about. You can then map these numeric values to the appropriate topic IDs in your project (.hpj) file.

Note

- When assigning topic IDs to context-sensitive Help topics, make sure the first four characters in the ID are IDH_. Help Workshop recognizes that topic IDs beginning with these characters are ones that a program will call. While compiling, Help Workshop displays a list of topic IDs that are in your topic (.rtf) files, but have not been mapped to numeric values. Help Workshop also lists IDs that are mapped to numeric values but are not in your topic files. Using the IDH_ naming convention will help you find and resolve problems in your context-sensitive Help.

{button ,AL(`CONTEXT_SENSITIVE_DESIGN;MAP;NEW_CONTEXT;cs_help;PRO_HPJ_ADD_MAP')} Related Topics

Keyword jumps

Keyword jumps are new macros that provide indirect links to one or more Help topics. When users click a hotspot, button, or menu command that is associated with a keyword jump, WinHelp displays the Topics Found dialog box, which lists topics that contain the same keyword as the one specified by the hotspot. Keyword jumps are made possible by KLink and ALink macros. For more information about these macros, click one of the following buttons.

? KLink keyword jumps
? ALink keyword jumps

{button ,AL(`MACRO_REF_OVR;MULTIPLE_HELP_DESIGN;BAS_MACRO_JUMP;BAS_INSERTING_JUMP;BAS_INSERTING_POPUP;ALink;KLink;PRO_INSERT_MACRO;PRO_CNT_LINKS;BAS_TOPIC_FOOTNOTE;BAS_ADDING_KEYWORDS')} Related Topics

KLink keyword jumps

KLink jumps are based on keywords specified by K-footnotes in your topic (.rtf) files. For example, a KLink macro that specifies the keywords "macros" and "graphics" will present users with a Topics Found dialog box containing a list of all topics that have the word "macro" or "graphics" in their K footnotes. As you author additional topics that contain these keywords, the items in the Topics Found dialog box increase accordingly. However, if the topic from which a user runs the KLink macro contains the designated keyword in its K-footnote, that topic does not appear in the Topics Found dialog box.

The main benefit of KLink macros is that they are automatically updated as keywords are added to or deleted from topics. They also enable you to jump to topics in Help files that may be on a user's hard disk at some point in the future.

Note

- Keywords specified in the KLink macro must exactly match K-keywords. For example, KLink(network) will not match the first-level index keyword "network," because the macro specification lacks a trailing comma.

ALink keyword jumps

ALink jumps are identical to KLink jumps except that the keywords they jump to are specified by A-footnotes in topics. A-keywords are used only for ALink macros and, unlike K-keywords, they never appear on the Index tab in the Help Topics dialog box. As with KLink macros, if the topic from which a user runs the ALink macro contains the designated keyword in its A-footnote, that topic does not appear in the Topics Found dialog box.

The main benefit of ALink macros is that they do not change, regardless of any changes to the K-keywords in topic (.rtf) files.

Using graphics in your Help files

In your topic files, you can include graphics of the following types: Windows bitmaps (.bmp or .dib), Windows metafiles (.wmf), Windows Help multiple-hotspot (SHED) bitmaps (.shg), and Windows Help multi-resolution bitmaps (.mrb). For more information about using graphics in your Help files, click one of the following buttons.

? Graphics color support

? Transparent bitmaps

Graphics color support

WinHelp version 4.0 can display graphics with any number of colors. WinHelp automatically reduces the color depth as necessary to the maximum color depth allowed by a user's display driver. The result is sometimes unsatisfactory, especially in cases where more than one graphic appears on a user's screen. For greater control, you can use optimized 16-color images (which work well over a wide range of platforms) or the {bmx} statement (which enables you to specify a series of graphics that have different color depths).

Transparent bitmaps

If you are using a colored background in your topics, you may want to use transparent bitmaps. To do this, add the letter t to the {bmx} statement, for example, {bmct}, {bmlt}, or {bmrt}. WinHelp replaces the white pixels in a transparent bitmap with the background color of the currently active window the first time the bitmap is displayed. If the background color is dithered, WinHelp uses a solid color closest to the dither for the bitmap's background color.

{button ,AL(`GRAPHICS_DESIGN;BAS_MULT_RESOLUTION;bmx;SHED')} Related Topics

Multiple aspect ratio graphics

If a specified bitmap's aspect ratio is different from the aspect ratio of a user's monitor, WinHelp attempts to compensate for the difference. The result is a slightly distorted bitmap. If a graphic includes text, the text may not be readable.

Help Workshop automatically removes the aspect ratio from bitmap (.bmp) files so that WinHelp will not stretch or shrink them. If you want WinHelp to distort your bitmaps, you must create them using the Multi-Resolution Bitmap Compiler. Doing so will preserve the aspect ratio of the bitmap, but the bitmap typically does not compress as well.

To include Multi-Resolution Bitmap Compiler (.mrb) files in a Help file, use the {bmx} bitmap reference statement.

Note

- If you want text in a graphic to change depending on the aspect ratio of a user’s monitor, use a metafile.

{button ,AL(`MRBC;bmx')} Related Topics

Multiple color depth graphics

If a specified bitmap's color depth is different from the color depth of a user's monitor, WinHelp attempts to compensate for the difference. This has no effect when a low color-depth graphic is displayed on a high color-depth monitor, but a high color-depth graphic may look bad when it is displayed on a low color-depth monitor.

To more precisely control the way a bitmap appears on computers that support different color depths, you can create a separate file for each bitmap — one file for each color depth users are likely to have. After you have created these files, you can list them in a single {bmx} bitmap statement, separated by semicolons. WinHelp determines the color depth capability of a user's computer, and then uses the most appropriate bitmap file.

If you specify multiple bitmaps in a {bmx} bitmap statement, you must specify the same number of bitmaps in the same order in all your topic (.rtf) files. If you do not do this, some of the bitmaps you specify may not appear. Or the bitmaps you specify may take twice as much room in your Help file. For example:

- If you specify {bmc Ralph_16.bmp; Petra_256.bmp} and {bmc Ralph_16.bmp; Tony_256.bmp} in your topic files, only the first occurrence of the bitmap statement will be used in all cases. The Ralph_16.bmp and Petra_256.bmp bitmaps will appear in your Help file, but the Tony_256.bmp file will not.
- If you specify {bmc Nancy_16.bmp; Tami_256.bmp} and {bmc Tami_256.bmp; Nancy_16.bmp}, both bitmaps will appear in your Help file, but will take twice as much space. Help Workshop considers each bitmap statement as unique because the bitmaps are specified in a different order.
- If you specify {bmc Ralph_16.bmp; Laura_256.bmp}, and later specify only {bmc Ralph_16.bmp} in your topic files, Ralph_16.bmp and Laura_256.bmp will be used in both instances.

{button ,AL(`BAS_ADDING_GRAPHICS;BAS_MULT_RESOLUTION;bmx')} Related Topics

About training card Help

Training cards are a set of Help topics in secondary windows that you can configure to send instructions to and receive instructions from another program. This is particularly useful in walking users through a procedure step by step. Whenever users correctly carry out a step, you can automatically display the next step. If users incorrectly carry out a step, you can display information that specifically addresses their mistakes.

Developers can use the WinHelp API in their programs to send information to WinHelp. Help authors can use the TCard macro in their Help files to send information to programs. Most Help authors will have to work closely with a program developer in designing an effective set of training cards.

{button ,AL(`TCARD_DESIGN;TCard;winhelp_api;WM_TCARD')} Related Topics

Customizing main window menus

Unlike a secondary window, a Help main window features a menu bar. You can use standard menus and commands, or you can add new menus and menu items by using macros.

You can also use menus to display commands that can be organized into logical groups. For example, you can change the items on menus to reflect the type of information contained in a group of related topics. WinHelp can place a check mark next to a menu item to indicate that the item is off or on.

Menu items run macros when users click them, press a key combination, or press an accelerator key. You can make menu items available or unavailable during a Help session.

To use a menu macro, you must run it from the main window. If a topic contains a macro that adds an item to the Help menu, but that topic is displayed in a secondary window, the macro will have no effect.

{button ,AL(`MACRO_REF_MENU')} Related Topics

Customizing Help buttons

In WinHelp version 4.0, both main and secondary windows can have a button bar with as many as 22 buttons on it. You can use the standard button bar set or use a custom set of buttons. To define custom buttons, you use macros.

Buttons are best used to display a limited number of frequently used functions. They enable users to access features by a single mouse click or key combination.

Help buttons run macros when users click them, press a key combination, or press an accelerator key. You can make Help buttons available or unavailable during a Help session.

Notes

- WinHelp automatically sizes buttons to fit the text displayed on the button.
- Buttons are added to the button bar in the order in which you specify them.
- Buttons can contain only text; they cannot contain graphics.

{button ,AL(`MACRO_REF_BUTTON;PRO_HPJ_ADD_BUTTON_BAR;PRO_ADD_BROWSE_BUTTONS;PRO_HPJ_MAIN_BUTTON_BAR')} Related Topics

Help system limits

The 32-bit technology of Windows 95 enables WinHelp version 4.0 to dramatically expand the capability of the Help system you author.

Item	Limit
Help file size	2 gigabytes
Topics per topic (.rtf) file	No practical limit
Topics per Help file	No practical limit
Topics per keyword	800
Topic footnote length	16,383 characters
Keyword length	255 characters
Hotspot hidden text	4,095 characters
Help title string	127 characters
Topic title string	127 characters
Custom window title string	50 characters
Custom window name	8 characters
Copyright string	255 characters
Browse string	50 characters
Referenced bitmaps	65,535 bitmaps per Help file
Filename	259 characters
Font name	31 characters
Font ranges	20 ranges
Error log file	No limit (Help Workshop can display 64K of text in Windows 95 and 1 MB of text in Windows NT, but the log file itself has no size limit.)
Citation string	2,000 characters
Window definitions	255 per Project file
Window caption	50 characters
contents file entries	No practical limit
Contents headings	9 levels (indented)
Contents topic strings	255 characters (127 in a DBCS language)
Contents heading text	No practical limit

{button ,AL(`DESIGN')} Related Topics

Designing for multiple Help files

You do not need to include all your Help topics within one Help file. For example, if the program you are documenting contains modular components, authoring separate Help files for each component is often a good idea. If a user decides not to install a component during setup or adds a new component later, the Help system still appears as a single unit. WinHelp version 4.0 can present a single index and a single table of contents for all the Help files you specify.

By planning ahead, you can make it possible to add information to your family of Help files after it has been released. By including the appropriate information in the contents file, you can automatically add new sections to the Contents tab and new keywords to the Index tab. You can also create jumps to topics that do not exist at the time you release your product but that might be added later.

When designing a Help system that uses multiple Help files or that may be augmented by additional Help files, consider the following:

- In the contents file you can specify jumps to topics that are located in other Help files. If WinHelp does not find the referenced Help file on a user's computer, the topics located in that file do not appear on the Contents tab. This feature is useful if your program has several components that are optional for users to install. In this case, simply create separate Help files for each component, and if one is not installed, the topics for that component do not appear on the Contents tab. This minimizes distraction and confusion for users.
- To prevent inconsistencies in the index, such as verb tense, capitalization, and the use of plurals, use the Report command in Help Workshop to generate a list of the keywords that are in the Help files you intend to combine.
- You can use keyword jumps to jump to topics in your family of Help files or in another family of Help files. KLink macros provide jumps to multiple topics that contain specified indexed K-keywords. ALink macros provide jumps to topics that contain specified nonindexed A-keywords. These macros are especially useful if you have multiple Help files because the links are resolved at run-time, based on the Help files that are actually available when the macro is run.

Note

- If you add files to your family of Help files after it has been released, the additions will not appear on the original Contents or Index tab until the Help file is reinitialized. To reinitialize the Help file, a setup program or your user must run the following command:

  winhelp -g filename.hlp

  For filename.hlp, substitute the name of the Help file you want to reinitialize.

{button ,AL(`BAS_OVERVIEW;BAS_ABOUT_HPJ_FILES;BAS_CNT_OVERVIEW;BAS_WRITE_INDEX;BAS_ASSOCIATIVE_JUMPS')} Related Topics

Designing a table of contents

You design the information on the Contents tab by configuring a contents (.cnt) file. You may want to show all topics or show only those topics that beginning users are likely to need.

When designing a table of contents, keep the following in mind:

- Book icons represent headings. Double-clicking a book icon enables users to see the subentries under that heading. Subentries can be other book icons or page icons.
- Page icons represent topics. Double-clicking a page icon enables users to display a topic or run a macro. You can specify what window type a topic should appear in.
- Page icons can jump to topics in other Help files.
- Users can print the contents of a book icon by clicking the icon, and then clicking the Print button on the Contents tab.
- You can include other contents files in your main contents file.

{button ,AL(`BAS_OVERVIEW;BAS_CNT_OVERVIEW;PRO_CNT_ADD_BOOK;PRO_CNT_ADD_TOPIC;PRO_HPJ_SPECIFY_CNT_FILE;PRO_CNT_DYN;PRO_CNT_DEFAULT_HELP_FILE')} Related Topics

Designing pop-up topics

A pop-up is a window that "pops up" on top of a window or dialog box. WinHelp automatically sizes pop-up windows to fit the text or graphic in them. Pop-ups remain on screen until users click anywhere on their screen.

Typical uses for pop-ups

Considerations for writing pop-up topics

{button ,AL(`BAS_OVERVIEW;BAS_CREATE_TOPIC;BAS_INSERTING_POPUP')} Related Topics

Typical uses for pop-ups

Pop-ups are generally used for the following purposes:

- Definitions
- Context-sensitive Help in dialog boxes
- Graphics
- Multimedia clips
- Tips and notes

Considerations for writing pop-up topics

When you write pop-up topics, you may want to consider the following:

- Unless you include an authorable Print button in pop-ups, users can print them only by clicking them with their right mouse button, and then clicking Print.
- Topics that do not have titles specified by $ footnotes are typically excluded from the full-text search index. Because pop-ups are best used in context, it would be more confusing than helpful if users were able to find the text in them.
- For context-sensitive Help topics that will be called from a program, make sure the first four characters in the topic ID are IDH_. This enables Help Workshop to detect whether topic IDs are mapped to numeric values in your project (.hpj) file.

Guidelines for using video and animation

Video and animation files can be included in any topic, including pop-up topics. WinHelp version 4.0 creates an embedded window for multimedia files and automatically sizes the window to accommodate the image or controller (if any). If the window is not large enough to display the controller, WinHelp crops the right and bottom edges. Users can resize the window to show any part of the controller that does not initially appear in the window.

Multimedia files tend to be large and can significantly increase a Help file's size. If your Help file is likely to be copied to a user's hard disk (rather than run from a CD-ROM), file size might be a an issue.

Note

- You can ship multimedia files without including them in your Help file. This feature may be useful if you want to run sample files that are already included as part of the product, or that currently exist on a user's computer.

{button ,AL(`BAS_OVERVIEW;BAS_ADDING_VIDEO;BAS_ADDING_SOUND;TMPDIR;mci;PRO_ADD_VIDEO')} Related Topics

Designing an index

A comprehensive keyword list is one of the most helpful navigating tools you can provide for users. In an index, you can provide synonyms that help users find information without requiring them to be familiar with your terminology. Even though WinHelp 4.0 supports full-text search features, you will probably want to prepare a keyword list. If users have only a full-text search index available to them, they can search for and find only words that are actually present in the text of the Help file.

? Types of keywords
? Keyword guidelines

{button ,AL(`PRO_ADD_KEYWORD;PRO_ADD_LEVEL2_KEYWORDS;PRO_CNT_COMBINE_HELP_FILES')} Related Topics

Types of keywords

It is typically best to include several types of keywords in each topic. When you specify keywords, you may want to include:

- Nontechnical terms that are likely to occur to beginning users
- Technical terms that are likely to occur to advanced users
- Common synonyms for technical terms
- Words that describe the topic generally
- Words that describe specific subjects within the topic
- Inverted forms of keyword phrases. For example, "combining Help files" and "Help files, combining"

Keyword guidelines

If you want your index to look similar to those in Windows 95 Help, you may want to consider the following:

- Always use the gerund form of verbs for keywords that begin with a verb
- Always use the plural form of nouns
- Use synonyms consistently throughout a family of Help files
- Make sure that keyword phrases beginning with the same word are consistently general or consistently specific

You can create first and second-level index entries. Users can either double-click second-level index entries to view particular topics or double-click a first-level entry to display the titles of all subentries in the Topics Found dialog box.

Designing Help windows

When considering what type of information should go in main and secondary windows, consider the following features and limitations of each window type:

- Secondary windows do not have a menu bar.
- Secondary windows do not provide access to a history window or to bookmarks.
- You cannot automatically size the main window to accommodate the amount of text in a topic.

If you are keeping your Help window on top, consider its size and placement. Ideally, users should be able to simultaneously read and carry out a procedure.

{button ,AL(`BAS_OVERVIEW;BAS_HPJ_CREATE_WINDOW')} Related Topics

HELPINFO

```
typedef   struct   tagHELPINFO {            // hi
    UINT     cbSize;
    int      iContextType
    int      iCtrlId;
    HANDLE hItemHandle;
    DWORD   dwContextId;
    POINT   MousePos;
} HELPINFO, FAR *LPHELPINFO;
```

The **HELPINFO** structure contains information about an item for which context-sensitive Help has been requested.

Parameter	Description
cbSize	Size of this structure, in bytes.
iContextType	Type of context for which Help is requested. This member can be one of these values: **Value** — **Meaning** HELPINFO_MENUITEM — Help requested for a menu item. HELPINFO_WINDOW — Help requested for a control or window.
iCtrlld	Identifier of the window or control if the iContextType parameter is HELPINFO_WINDOW, or identifier of the menu item if the iContextType parameter is HELPINFO_MENUITEM.
hItemHandle	Identifier of the child window or control if iContextType is HELPINFO_WINDOW, or identifier of the associated menu if iContextType is HELPINFO_MENUITEM.
dwContextId	Help context identifier of the window or control.
MousePos	A **POINT** structure that contains the screen coordinates of the mouse cursor. This parameter is useful for providing Help based on the position of the mouse cursor.

{button ,KL("WM_HELP message;POINT")} Related Topics

HELPWININFO

```
typedef struct {        // hwi
    int    wStructSize;
    int    x;
    int    y;
    int    dx;
    int    dy;
    int    wMax;
    TCHAR rgchMember[2];
} HELPWININFO;
```

The **HELPWININFO** structure contains the size and position of either a primary or a secondary Help window.

Parameter	Description
wStructSize	Size of this structure, in bytes.
x	X-coordinate of the upper left corner of the window, in screen coordinates.
y	Y-coordinate of the upper left corner of the window, in screen coordinates.
dx	Width of the window, in pixels.
dy	Height of the window, in pixels.
wMax	Value specifying how to show the window. This member must be one of these values:

Value	Meaning
SW_HIDE	Hides the window and passes activation to another window.
SW_MINIMIZE	Minimizes the specified window and activates the top-level window in the Z order.
SW_RESTORE	Same as SW_SHOWNORMAL.
SW_SHOW	Activates a window and displays it in its current size and position.
SW_SHOWMAXIMIZED	Activates the window and displays it as a maximized window.
SW_SHOWMINIMIZED	Activates the window and displays it as an icon.
SW_SHOWMINNOACTIVE	Displays the window as an icon. The window that is currently active remains active.
SW_SHOWNA	Displays the window in its current state. The window that is currently active remains active.
SW_SHOWNOACTIVATE	Displays a window in its most recent size and position. The window that is currently active remains active.
SW_SHOWNORMAL	Activates and displays the window. Whether the window is minimized or maximized, Windows restores it to its original size and position.

rgchMember	Name of the window.

Comments

WinHelp divides the display into 1024 units in both the x- and y-directions. To create a secondary window that fills the upper left quadrant of the display, for example, a program would specify zero for the **x** and **y** members and 512 for the **dx** and **dy** members.

A program can set the size and position information by calling the **WinHelp** function with the HELP_SETWINPOS value.

{button ,AL(`winhelp_api')} Related Topics

MULTIKEYHELP

```
typedef struct tagMULTIKEYHELP {        // mkh
    DWORD mkSize;
    BYTE    mkKeylist;
    TCHAR szKeyphrase[1];
} MULTIKEYHELP;
```

The **MULTIKEYHELP** structure specifies a keyword table and an associated keyword to be used by the WinHelp program.

Parameter	Description
mkSize	Size of this structure, in bytes.
mkKeylist	A single character that identifies the keyword table to search.
szKeyphrase	A null-terminated text string that specifies the keyword to locate in the keyword table.

{button ,AL(`winhelp_api')} Related Topics

WinHelp

BOOL WinHelp(HWND hwnd**,** LPCTSTR lpszHelpFile**,** UINT fuCommand**,** DWORD dwData**)**

The **WinHelp** function starts Windows Help (WINHELP.EXE) and passes additional data indicating the nature of the help requested by the program.

Parameter	Description
hwnd	Handle of the window requesting Help. The **WinHelp** function uses this handle to keep track of which programs have requested Help. If fuCommand specifies HELP_CONTEXTMENU or HELP_WM_HELP, hwnd identifies the control requesting Help.
lpszHelpFile	Address of a null-terminated string containing the path, if necessary, and the name of the Help file that WinHelp is to display. The filename may be followed by an angle bracket (>) and the name of a secondary window if the topic is to be displayed in a secondary window rather than in the primary window. The name of the secondary window must have been defined previously in the **[WINDOWS]** section of the Help Project (.HPJ) file.
fuCommand	Type of help requested. For a list of possible values and how they affect the value to place in the dwData parameter, see the Comments section.
dwData	Additional data. The value used depends on the value of the fuCommand parameter. For a list of possible values, see the Comments section.

Return Value

If successful, the return value is TRUE; otherwise, it is FALSE.

Comments

The program specifies the name and, where required, the directory path of the Help file to display.

Before closing the window that requested Help, the program must call **WinHelp** with the fuCommand parameter set to HELP_QUIT. Until all programs have done this, WinHelp will not terminate. Note that calling WinHelp with the HELP_QUIT command is not necessary if you used the HELP_CONTEXTPOPUP command to start Help.

The following table shows the possible values for the fuCommand parameter and the corresponding formats of the dwData parameter:

fuCommand	Action	dwData
HELP_COMMAND	Runs a Help macro or macro string.	Address of a string that specifies the name of the Help macro(s) to run. If the string specifies multiple macro names, the names must be separated by colons or semicolons . You must use the short form of the macro name for some macros because WinHelp does not support the long name.
HELP_CONTENTS	Displays the topic specified by the Contents option in the **[OPTIONS]** section of the .HPJ file. This command is for backward compatibility. New programs should provide a .CNT file and use the HELP_FINDER command.	Ignored; set to 0.
HELP_CONTEXT	Displays the topic identified by the specified context identifier defined in the **[MAP]** section of the .HPJ file.	Unsigned long integer containing the context identifier for the topic.
HELP_CONTEXTMENU	Displays the Help menu for the selected window, and then displays (in a pop-up window) the topic for the selected control.	Address of an array of double word pairs. The first double word in each pair is a control identifier, and the second is a context number for a topic.

HELP_CONTEXTPOPUP	Displays, in a pop-up window, the topic identified by the specified context identifier defined in the **[MAP]** section of the .HPJ file.	Unsigned long integer containing the context identifier for a topic.
HELP_FINDER	Displays the Help Topics dialog box.	Ignored; set to 0.
HELP_FORCEFILE	Ensures that **WinHelp** is displaying the correct Help file. If the incorrect Help file is being displayed, **WinHelp** opens the correct one; otherwise, there is no action.	Ignored; set to 0.
HELP_HELPONHELP	Displays Help on how to use WinHelp, if the **WINHLP32.HLP** file is available.	Ignored; set to 0.
HELP_INDEX	Displays the topic specified by the **CONTENTS** option in the **[OPTIONS]** section of the .HPJ file. This command is for backward compatibility. New programs should provide a .CNT file and use the HELP_FINDER command.	Ignored; set to 0.
HELP_KEY	Displays the topic in the keyword table that matches the specified keyword, if there is an exact match. If there is more than one match, this command displays the Topics Found list box.	Address of a keyword string. Multiple keywords must be separated by semi-colons.
HELP_MULTIKEY	Displays the topic specified by a keyword in an alternative keyword table.	Address of a **MULTIKEYHELP** structure that specifies a table footnote character and a keyword.
HELP_PARTIALKEY	Displays the topic in the keyword table that matches the specified keyword if there is an exact match. If there is more than one match, this command displays the Topics Found dialog box. To display the Index without passing a keyword, you should use a pointer to an empty string.	Address of a keyword string. Multiple keywords must be separated by semi-colons.
HELP_QUIT	Informs the WinHelp program that it is no longer needed. If no other programs have asked for Help, Windows closes the WinHelp program.	Ignored; set to 0.
HELP_SETCONTENTS	Specifies the Contents topic. The WinHelp program displays this topic when the user clicks the Contents button if the Help file does not have an associated .CNT file.	Unsigned long integer containing the context identifier for the Contents topic.
HELP_SETPOPUP_POS	Sets the position of the subsequent pop-up window.	Address of a **POINTS** structure. The pop-up window is positioned as if the mouse cursor were at the specified point when

		the pop-up window is invoked.
HELP_SETWINPOS	Displays the Help window, if it is minimized or in memory, and sets its size and position as specified.	Address of a **HELPWININFO** structure that specifies the size and position of either a primary or secondary Help window.
HELP_TCARD	Indicates that a command is for a training card instance of the WinHelp program. A program combines the HELP_TCARD flag with other commands by using the bitwise OR operator.	Depends on the command with which the flag is combined.
HELP_WM_HELP	Displays, in a pop-up window, the topic for the control identified by hwnd.	Address of an array of double word pairs. The first double word in each pair is a control identifier, and the second is a context identifier for a topic.

Comments

The following values may also be used, however, since they are not in windows.h, you must add them to your own header file:

```
#define HELP_FORCE_GID    0x000e
#define HELP_TAB                  0x000f
```

fuCommand	**Action**	**dwData**
HELP_FORCE_GID	Changes to the .GID file associated with the help file passed in as the lpszHelpFile parameter.	Ignored; set to 0.
HELP_TAB	Opens the Help Topics dialog box, displaying the specified extensible tab.	Zero-based index of the extensible tab to display (0 is the first tab, 1 the second, etc.).

{button ,KL("HELPWININFO structure;MULTIKEYHELP structure;POINTS;training card Help;WM_HELP message;WM_TCARD message;MAP section")} Related Topics

WM_HELP

WM_HELP
lphi = (LPHELPINFO) lParam;

The WM_HELP message is sent whenever the user presses the F1 key.

Parameter	Description
lphi	Address of a **HELPINFO** structure that contains information about the menu item, control, dialog box, or window for which Help is requested.

Default Action

The **DefWindowProc** function passes WM_HELP to the parent window of a child window, or to the owner of a top-level window.

Comments

If a menu is active when F1 is pressed, WM_HELP is sent to the window associated with the menu. Otherwise, WM_HELP is sent to the window that has the keyboard focus. If no window has the keyboard focus, WM_HELP is sent to the currently active window.

{button ,KL("HELPINFO structure;context-sensitive Help")} Related Topics

WM_TCARD

WM_TCARD
idAction = wParam; // user action
dwActionData = lParam; // action data

The WM_TCARD message is sent to a program that has initiated a training card with the WinHelp program.

Parameter	Description
idAction	Value of wParam. Indicates the action that the user has taken. This parameter can be one of the following values:

Value	Meaning
IDABORT	The user clicked an authorable Abort button.
IDCANCEL	The user clicked an authorable Cancel button.
IDCLOSE	The user closed the training card.
IDHELP	The user clicked an authorable Help button.
IDIGNORE	The user clicked an authorable Ignore button.
IDOK	The user clicked an authorable OK button.
IDNO	The user clicked an authorable No button.
IDRETRY	The user clicked an authorable Retry button.
IDYES	The user clicked an authorable Yes button.
HELP_TCARD_DATA	The user clicked an authorable button. The lParam parameter contains a long integer specified by the Help author.
HELP_TCARD_NEXT	The user clicked an authorable Next button.
HELP_TCARD_OTHER_CALLER	Another program has requested training cards.

dwActionData	Value of lParam. If idAction specifies HELP_TCARD_DATA, this parameter is a long integer specified by the Help author. Otherwise, this parameter is zero.

Return Value

The return value is ignored; use zero.

Comments

The message informs the program when the user clicks an authorable button or hotspot that runs the **TCard** macro. A program initiates a training card by specifying the HELP_TCARD command in a call to the **WinHelp** function.

WinHelp can run only one training card at a time. If another program requests a training card, the program currently using training cards receives a WM_TCARD message with wParam == HELP_TCARD_OTHER_CALLER. The program receiving this message should either terminate the training card (HELP_QUIT | HELP_TCARD), or the next time it gets the focus, it should call WinHelp with a HELP_FORCEFILE | HELP_TCARD command.

{button ,AL(`winhelp_api;TCard')} Related Topics

WinHelp error numbers

When Help Author mode is turned on, WinHelp will usually give the author enough information to solve the problem in the help file. However, if a user encounters a problem in your help file, only the error number will be available. The following table lists the error numbers and the probably cause of the error. Errors that only occur when Help Author mode is on are not included in this list.

1024	A macro was specified that doesn't exist.
1026	A macro parameter is incorrect (for example, using a number when a quoted string was expected).
1027	Missing close quote.
1028	Unrecognized macro or variable name.
1029	Parameter prototype is invalid (RegisterRoutine problem).
1031	Missing closing parenthesis in a macro.
1032	Invalid separator between macros.
1033	Invalid return type specified (RegisterRoutine problem)
1034	Invalid macro syntax.
1035	Parameter doesn't match the expected type (for example, using a negative number for a paramter that expects an unsigned number).
1036	A variable used in a macro is unrecognized.
1037	The current macro cannot be executed until a previously called macro completes. For example, until the FileOpen macro completes, other macros cannot execute.
1049	A macro called itself.
1050	Missing separator in an authorable button command.
1053	Invalid multi-key keyword.
1061	Too many secondary windows open at once.

Creating a project file

This training card sequence will guide you through the steps of creating a basic project file.

Click Next to continue.

{button Next,TC(16,30009)}

? On the File menu, click New.

? Click Help project, and then click OK.

? Specify the filename and path for your project file, and then click Save.

Now that you've created a project file, you need to add the information about your topic files.

Have you created your topic (.rtf) files yet?

{button Yes,TC(16,30005)} {button No,TC(16,30011)}

When you create your topic files later, open your project file and click the Files button to specify the name and location of your topic files.

Click Next to continue.

{button Next,TC(16,30027)}

? Click the Files button.

? Click Add.

Specify the name and path of your topic file, and then click Open.

If you want to add another topic file, click Add.
When you are done adding files, Click OK.

Are you using any bitmaps in your Help file?

{button Yes,TC(16,30000)} {button No,TC(16,30010)}

If you later decide to use bitmaps in your Help file, open your project file and click the Bitmaps button to add the location of your bitmap files.

Click Next to continue.

{button Next,TC(16,30018)}

? Click Bitmaps.

? Click Add.

? Click the folder that contains your bitmaps, and then click OK.

? Click OK.

Do you want to define a custom window?

{button Yes,TC(16,30016)} {button No,TC(16,30012)}

If you later decide that you want to define a custom window, just open your project file and click Windows.

Click Next to continue.

{button Next,TC(16,30036)}

? Click the Windows button.

? Click Add.

Type a name for the window type, and then click OK.

? Click the Position tab.

? Click Auto-Sizer.

? Drag the sample window to the position you want and then drag its window borders to create the size you want. When you are done, click OK.

? Click the Buttons tab.

? Select the buttons you want to add to your window. When you are done, click the Color tab.

1. To change the color of the non-scrolling or the scrolling region, click the Change button next to the sample window. Choose the color you want, and then click OK.
2. When you are done changing colors, click the General tab.

1. In the Title Bar Text box, type text that you want to appear in the title bar for this window.
2. If you want to use the auto-sizing feature, click Auto-Size Height.
3. If you want your Help window to stay visible in front of other window, click Keep Help Window On Top.
4. When you are done, click OK.

You can use compression when you compile so that your Help file will take up less disk space. However, it may take longer to compile.

Do you want to use compression?

{button Yes,TC(16,30035)} {button No,TC(16,30034)}

If you decide to turn on compression later, just open your project file and click Options. Then, click the Compression tab.

Click Next to continue.

{button Next,TC(16,30050)}

? Click Options.

? Click the Compression tab.

? Choose your compression options, and then click OK.

Done!

You have created a basic project file. You can change any of these settings at any time by opening your project file in Help Workshop.

{button Close,TC(8,0)}