Creating Windows Help

Contents

Select an item from the list below to see the contents for that section.

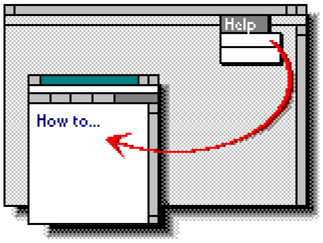
Getting Started About this Help system; Tools and files you need to create Help; How to create a simple Help system

Authoring Guide About WinHelp features and how to implement them. These topics cover many features and techniques not included in Getting Started.

Glossary Glossary of terms used in WinHelp and in this Help system.

Reference Reference topics for WinHelp macros and for the options in the Help project (.HPJ) file. (Reference topics exist in a separate Help file, so these topics look different than the rest of the topics in this system.)

Welcome to Borland's *Creating Windows Help*.



Creating Windows Help v.1.0

from

Borland

International

Getting Started

About *Creating Windows Help*

What You Need to Start

Quick Start - Creating a Simple Help File

Authoring Guide

Collapse contents

Creating Source Files

Compiling and Debugging

Working with Windows

Working with Hotspots

Working with Macros

Working with Footnotes

Providing Navigation Controls

Controlling Scrolling and Wrapping in Topics

Using Graphics in Topics

Working with Buttons

Working with Menus

Working with the How to Use Help Topic

Working with the Contents Topic

Working with Bookmarks

File-Handling Tasks

Creating Context-sensitive Help

Advanced Topics

Authoring Guide

Expand contents

Creating Source Files

Compiling and Debugging

Compiling Source Files into the Help File

Compiling a Subset of Topics (Conditional Compiling)

Compiling and Debugging

Basic Troubleshooting

Working with Windows

Working with Hotspots

Working with Macros

Working with Footnotes

Providing Navigation Controls

Controlling Scrolling and Wrapping in Topics

Using Graphics in Topics

Adding Graphics to a Topic - Overview

About Inline Bitmaps vs. Referenced Bitmaps

Pasting Graphics (Bitmaps) from the Windows Clipboard

Adding Graphics Using the Word Processor's Graphics Tools

Adding Bitmaps by Reference

Adding Boxes and Lines

Using Hypergraphics

Working with Buttons

Creating and Deleting Custom Buttons

Disabling and Enabling Buttons

Changing the Function of a Button

Working with Menus

Working with the How to Use Help Topic

Specifying Your Own How to Use Help File

Jumping to the How to Use Help Topic

Working with the Contents Topic

About the Contents Topic

Creating the Contents Topic

Jumping to a Contents Topic

Setting the Contents to a Specific Topic

Working with Bookmarks

File-Handling Tasks

Printing a Topic

Opening Another Help File

Exiting Windows Help

Creating Context-sensitive Help

Advanced Topics

Reference

Macros, Categorical list
Macros, Alphabetical list
Help Project File
RTF Tokens

by

Ben Gelernter

Lynn Flink ○ **Jeff Powanda** ○ **Holly Von Hendy**

with unwavering support from

Joanne Bealy

About this Help System

This Help system, "Creating Windows Help," tells how to create an online Help system using the Microsoft Windows 3.1 Help compiler and application, called "WinHelp" for short.

"Creating Windows Help" was itself created using WinHelp. It contains working examples of the features being explained. For example, a topic that explains how to create hotspots contains real hotspots and shows how those hotspots were created.

See the following topics for more information about this Help system. Click an item to go directly to that topic, or click the **>>** (Next) button to browse through the topics in sequence.

Jumps to Reference Topics

Graphics Used as Buttons

Files in this Help System

Jumps to Reference Topics

"Creating Windows Help" includes reference documentation for WinHelp macros and the Help project file. Reference topics appear against a gray background, to differentiate them visually from the main Help topics.

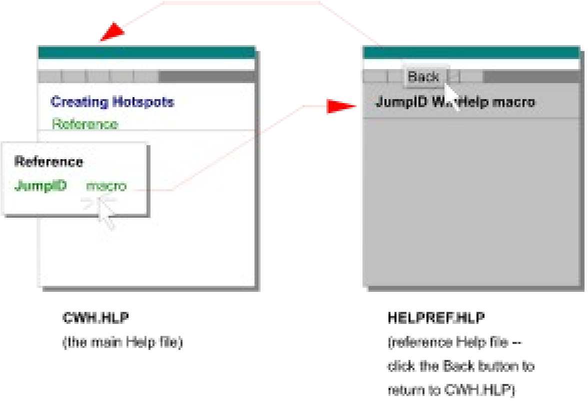
Hotspots that inititate jumps to reference topics appear as follows:

ChangeButtonBinding macro

[WINDOWS]

TITLE option

If you jump to a reference topic from a topic in the main Help file (CWH.HLP), you are jumping to a different Help file. Click the Back button to return to the topic from which you jumped.



Graphics Used as Buttons

In Windows Help, you can simulate a button by creating a graphic that looks like a button and then make it a hotspot that initiates some other action. This Help system includes these standard buttons:

This button...	Does this...
✳	Shows something, or demonstrates something. (This example displays the Copy dialog box. Click the Cancel button to close the dialog box.)
⚑	Shows more infomation. (This example displays a pop-up window explaining how a certain example was created. Click anywhere to close the pop-up window.)
↵	Returns you to the previous topic. This button sometimes appears in secondary windows so you can return to the topic from which you jumped. (This example of the button is inactive.)

Sample Pop-up Window

Click anywhere to close this window.

Files in this Help System

This Help system includes three files:

Filename	Contents
CWH.HLP	Main Help file. This file contains most of the topics in the Help system. This is the file you open to start "Creating Windows Help."
HELPEX.HLP	Topics used as examples in CWH.HLP
HELPREF.HLP	Reference topics for Help macros and the Help project file. These topics are displayed against a gray background, to help distinguish them from the rest of the Help system.

Tools and Files You Need to Create Help

The Basics

You need the following tools and files to create a simple Help file:

- The **WinHelp application** itself: WINHELP.EXE.
  When you create a Help system, you are actually creating a resource file that is used by the WinHelp application. Therefore, you (and your users) also need the WinHelp application to run the Help file.
- A **word processor** for creating topics. The word processor must be able to:
- Save files in Rich Text Format (.RTF).
- Create footnotes that are identified by custom marks.
  For simple Help files, you will create footnotes identified by the custom marks $, #, K, and +.
- A word processor or **text editor** that can save files as ASCII text, for creating the Help project (.HPJ) file.
- The Microsoft Windows **Help compiler** (HCP.EXE or HC31.EXE). Both HCP.EXE and HC31.EXE compile Help files to be used with Windows 3.1. They cannot create Help files to be used with Windows 3.0. HCP.EXE is the "protected mode" compiler, which makes better use of system memory. Use HCP.EXE to compile a Help file from a DOS window under Windows.
- The **error message resource file** for the Help compiler you are using (HCP.ERR or HC31.ERR). This file contains the warning and error messages that WinHelp produces if there are any problems during the compile.

Tools for Advanced Features

These are the tools you will need to create the following advanced features:

- Microsoft **Hotspot Editor** (SHED.EXE).
  Hotspot Editor creates Segmented Hypergraphics (.SHG) files. These files include graphics that have been segmented into hotspots, so that when the user clicks one of the segements, a pop-up window is displayed or another topic is displayed (i.e., the user jumps to another topic.)
- Microsoft **Multiple Resolution Bitmap Compiler** (MRBC.EXE).
  MRBC combines several bitmaps created for use at different resolutions into a single file that can be used by WinHelp. WinHelp checks the resolution of the monitor being used and displays the bitmap created for that resolution.
- A **graphics editor**, like Borland Resource Workshop, that can save graphic files as bitmaps.
  Use the graphics editor to create illustrations and custom buttons. You can also create custom icons.
- **Drawing or painting software** to create illustrations. Besides bitmaps, WinHelp can use files saved as a Windows metafiles (.WMF).

You can also import graphics directly into your source documents, depending on the capabilities of your word processor. In addition, you can paste graphics directly into a document from the Windows Clipboard. Therefore, you can use any graphics software that allows you to cut images and paste them to the Clipboard.

How to Create a Simple Help File - Overview

Topic 1 of 14

This topic is the first in a sequence of topics that describe how to create a simple, non-context-sensitive Help file. Read these topics in sequence for a complete introduction to working with WinHelp.

For the most part, these topics do not describe alternative ways to do things. WinHelp provides several different ways to accomplish some tasks. However, the purpose of these topics is to show the simplest, most straightforward way of creating a typical Help system.

The instructions in these sections assume you are using Microsoft Word to create the topics.

Overview of the Process

These are the basic steps for creating a simple Help file:

1. Write the topics that make up the Help file. Save them as Rich Text Format (.RTF) files.

2. Write a Contents topic. Save it as a Rich Text Format (.RTF) file.

3. Write a Help project (.HPJ) file. Save it as a text file.

4. Compile the topics into a Help resource (.HLP) file.

See the following topics for detailed information:

Click an item to go directly to that topic, or click the **>>** (Next) button to browse through the topics in sequence.

Writing Simple Topics
Writing the Contents Topic
Writing the Help Project File
Compiling the Help File

Writing Simple Topics

Topic 2 of 14

These are the basic steps for creating a simple Help topic:

1. Write a title for the topic.
2. Write the text for the topic.

   Write each topic on a separate page. That is, separate each topic with a hard page break.
3. Add the basic footnotes.
4. Save the file in .RTF format.

You do not have to put all of your topics into one file.

See the following topics for detailed information:

Click an item to go directly to that topic, or click the **>>** (Next) button to browse through the topics in sequence.

Writing the Title of a Topic

Writing the Text for a Topic

Inserting Footnotes for Each Topic

Adding Hotspots to the Topic

Saving the File in RTF format

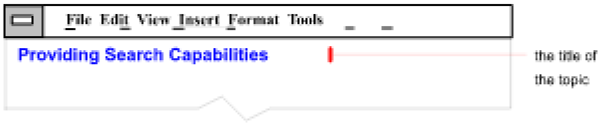
Writing the Title of a Topic

See Also   Topic 3 of 14

Write a title on the first line of the topic.

WinHelp does not require that topics have titles. But titles serve the same purpose online as they do in print: to identify the topic (or chapter) and describe its contents. But beyond that, WinHelp provides ways for a user to search for topics that exist elsewhere in the Help system. The best way to identify a topic is by using this title.

Example


File Edit View Insert Format Tools
Providing Search Capabilities
the title of the topic

Tip

To make the title stand out, format it as you would a title or a section heading in a printed document, using a different type size, bolding, or color.

See Also

Creating a Topic

How Users Navigate Through Help

Providing Search Capabilities

Creating the Contents Topic

Writing the Text for a Topic

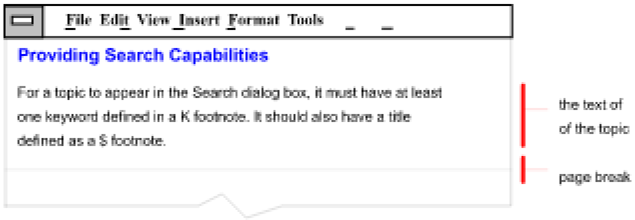
See Also　　　　　　　　　　　　　　　　　　Topic 4 of 14

1. Following the title, write the text for the topic, just as you would for a printed document.

   You do not have to worry about line length for paragraphs. The text in the compiled Help file wraps automatically when it reaches the right edge of the window, no matter how wide the window is.

2. At the end of the topic, insert a page break. WinHelp turns each page of the source file into a separate topic in the compiled Help file.

Example



Providing Search Capabilities

For a topic to appear in the Search dialog box, it must have at least one keyword defined in a K footnote. It should also have a title defined as a $ footnote.

Tips

You can use many of the formatting features that you would for any document, for example, different fonts, type styles, colors, lines, and boxes. But remember that this information will be viewed on the computer monitor. Therefore, it is advisable to use a ten-point, nonserif font such as Arial or Helvetica for most text.

Try to break the text into short paragraphs, lists, and tables as much as possible. This makes it easier to read online.

You should also restrict the length of topics so that the user does not have to scroll through more than two screens to see the entire topic.

See Also

Adding Boxes and Lines

Creating a Topic

Creating Nonscrolling Titles

Creating Nonwrapping Text

Using Bitmaps

Using Hypergraphics

Inserting Footnotes for Each Topic

See Also Topic 5 of 14

WinHelp uses custom footnotes to identify topics and to provide some navigation controls.

These are called "custom" footnotes because they are not identified by sequential numbers. Typically, you will use four footnotes for each topic:

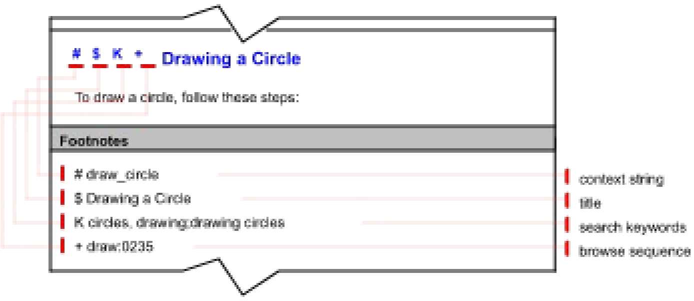
Footnote character	Used to identify...	Purpose
#	Context string	Uniquely identifies the topic.
$	Title	Appears as the topic title in the Search dialog box and in the History list.
K	Keywords (and phrases)	These words and phrases appear in the Search dialog box.
+	Browse sequence	Determines the order of the topics when the user browses through them.

Overview

For each topic, do the following:

1. Move the text insertion point to where you want the footnote identifier to be placed, typically before the first character of the first line of the topic.
2. Specify which footnote mark to use (#, $, K, or +), and insert the footnote.
   (MS-Word provides a dialog box for specifying the footnote identifier).
3. Move the text insertion point to the footnote itself, if it has not already been moved there by the word processor.
4. Type the appropriate information for the footnote, as explained in the following sections.

Example



$ K + Drawing a Circle

To draw a circle, follow these steps:

Footnotes

draw_circle
$ Drawing a Circle
K circles, drawing;drawing circles
+ draw:0235

See the following topics for detailed information:

Click one of the following items to go directly to that topic, or use the >> (Next) button to browse through the topics in sequence.

Assigning a Context String to a Topic: Inserting the # Footnote

Assigning a Name to a Topic: Inserting the $ Footnote

Adding Phrases to the Search List: Inserting the K Footnote

Including the Topic in a Browse Sequence: Inserting the + Footnote

See Also

About Footnotes

Creating Footnotes

Assigning a Context String to a Topic: Inserting the # Footnote

See Also   Topic 6 of 14

To assign a context string to a topic:

1. Move the text insertion point to the first character of the first line of the topic.
2. Insert a **#** footnote.
3. In the footnote, type a string of characters to identify the topic.

   This string is called the "context string." WinHelp uses context strings to identify each topic. The user never sees these strings, but you, the author, will use them to specify jumps to the topic.

   Legal characters are: A-Z (upper- and lowercase), 0-9, period (**.**), and the underscore character (_). Spaces are not allowed. The context string can be no longer than 255 characters.

See Also

Inserting Footnotes for Each Topic

How WinHelp Identifies Topics

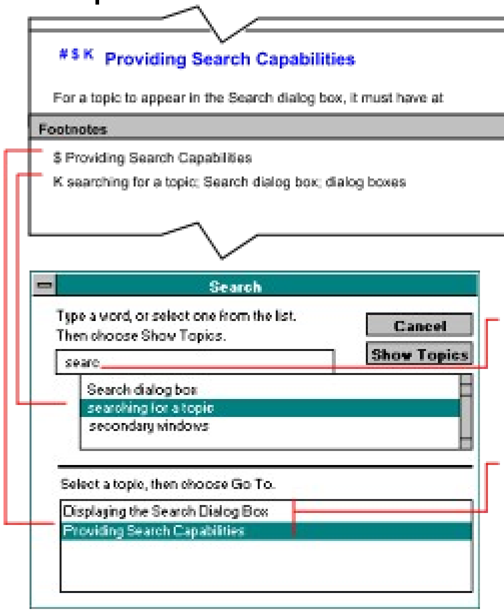
Assigning a Name to a Topic: Inserting the $ Footnote

See Also Topic 7 of 14

To assign a title to a topic:

1. Move the text insertion point to follow the # footnote mark on the line containing the topic's title.
2. Insert a $ footnote.
3. In the footnote, type the title for the topic. This will usually be the same name as the title you typed on the first line of the topic. This name appears to the user in the Search dialog box and in the History list.

Example



#$K Providing Search Capabilities

For a topic to appear in the Search dialog box, it must have at

$ Providing Search Capabilities

K searching for a topic; Search dialog box; dialog boxes

Type a word, or select one from the list. Then choose Show Topics.

searc

Search dialog box
searching for a topic
secondary windows

Select a topic, then choose Go To.

Displaying the Search Dialog Box
Providing Search Capabilities

The user types a word or phrase here..

or chooses one from here...

and the titles (the $ footnotes) of all topics that contain that word or phrase (the K footnotes) appear here. Choosing one of these titles jumps to the topic.

See Also

Inserting Footnotes for Each Topic

Adding Phrases to the Search List: Inserting the K Footnote

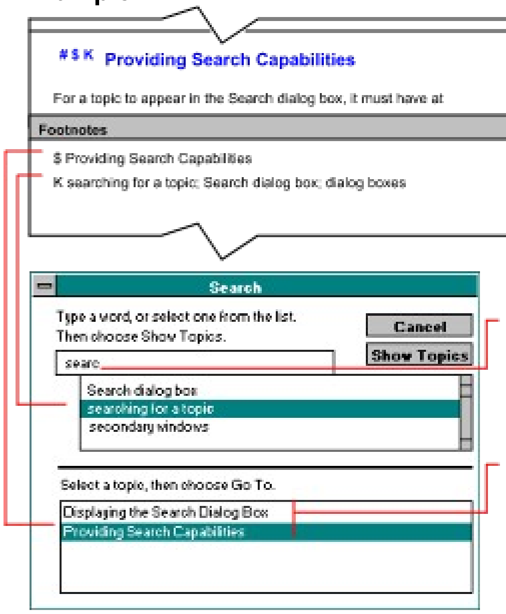
See Also Topic 8 of 14

To add phrases to the search list in the Search dialog box:

1. Move the text insertion point to the first character of the first line of the topic.
2. Insert a K footnote.
3. In the footnote, type words and phrases that you want to appear in the search list. Separate each entry with a semicolon.

   These words and phrases appear in the search list of the Search dialog box. Use the same words and phrases as you would in a traditional index.

Example



$ K Providing Search Capabilities

For a topic to appear in the Search dialog box, it must have at

Footnotes

$ Providing Search Capabilities

K searching for a topic; Search dialog box; dialog boxes

Type a word, or select one from the list.
Then choose Show Topics.

searc

Search dialog box
searching for a topic
secondary windows

Select a topic, then choose Go To.

Displaying the Search Dialog Box
Providing Search Capabilities

The user types a word or phrase here..

or chooses one from here...

and the titles (the $ footnotes) of all topics that contain that word or phrase (the K footnotes) appear here. Choosing one of these titles jumps to the topic.

See Also

Inserting Footnotes for Each Topic

How Users Navigate Through Help

Providing Search Capabilities

Including the Topic in a Browse Sequence: Inserting the + Footnote

See Also Topic 9 of 14

Making browse sequences is a two-part process. You must add the browse buttons (**>>** (Next) and **<<** (Previous)) to the button bar and also specify the browse sequences in the topics.

To add browse buttons to the button bar:

Include the BrowseButton macro in the [CONFIG] section of the project file. See Writing the Help Project File (topic 13 of 14) for more information about this step.

To include a topic in a browse sequence:

1. Move the text insertion point to the first character of the first line of the topic.
2. Insert a + footnote.
3. In the footnote, type the browse sequence identifier. The identifier can be a number (for example, 005) or a group name followed by a colon and a number (for example, CALC:005)

Note: A topic can belong to only one browse sequence.

See Also

Adding Browse Buttons and Browse Sequences

Getting Started: Inserting Footnotes for Each Topic

How Users Navigate Through Help

Adding Hotspots to the Topic

See Also Topic 10 of 14

A hotspot is text or a graphic that the user can click to initiate some action. A hotspot can initiate a jump to another topic, display a topic in a different window, or execute a macro or macros.Most often, words are turned into hotspots so the user can click the word and display another topic.

To create a hotspot that jumps to a different topic:

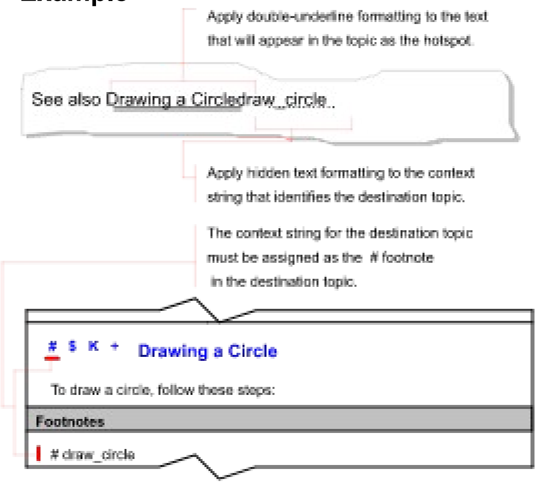
1. Type the text or insert the graphic you want to appear as the hotspot.
2. Highlight the text, and apply double-underline formatting to it.

   In Microsoft Word, highlight the word or words and display the Character format dialog box (press Alt+T, C); then choose Double from the Underline selection list.
3. Immediately following the text or graphic from step 1, type the context string of the destination topic.
4. Highlight the context string, and apply hidden text formatting to it.

   In Microsoft Word, highlight the word or words and display the Character format dialog box (press Alt+T, C); then click the Hidden check box.

Note: To create a Contents topic for your Help system, create a topic that contains the titles of your topics. Then make each title into a hotspot that jumps to the topic named.

Example



Apply double-underline formatting to the text that will appear in the topic as the hotspot.

See also Drawing a Circledraw_circle

Apply hidden text formatting to the context string that identifies the destination topic.

The context string for the destination topic must be assigned as the # footnote in the destination topic.

$ K + Drawing a Circle

To draw a circle, follow these steps:

Footnotes

draw_circle

In the final Help file, the hotspot appears as underlined, green text:

See also Drawing a Circle

See Also

About Hotspots

Creating Hotspots

Creating a Hotspot that...

- Jumps to a Topic in the Same Help File
- Jumps to a Topic in a Different Help File
- Displays a Topic from the Same Help File in a Pop-up Window
- Displays a Topic from a Different Help File in a Pop-up Window
- Displays a Topic from the Same Help File in a Secondary Window
- Displays a Topic from a Different Help File in a Secondary Window

Saving the File in RTF Format

See Also Topic 11 of 14

The WinHelp compiler can compile only Rich Text Format (.RTF) files. Therefore, you must save your source files in Rich Text Format (RTF).

When you save a file as a Rich Text Format file, give it an .RTF filename extension.

It is also a good idea to save source documents in the file format used by your word processor. For example, if you are using Microsoft Word, save your files as .DOC files. This makes it easy to return to the files for further editing later.

About Rich Text Format

A Rich Text Format file is an ASCII file that contains instructions for all the formatting in the document, such as fonts, line breaks, styles, etc. If you look at an .RTF file in a text editor, you can read all the formatting instructions, and all the text in the document.

To save a file as an .RTF file in Microsoft Word:

1. Choose File|Save As...
2. In the dialog box, change the extension of the file to .RTF.
3. Set the Save File As Type... option to Rich Text Format (RTF).
4. Choose OK.

See also

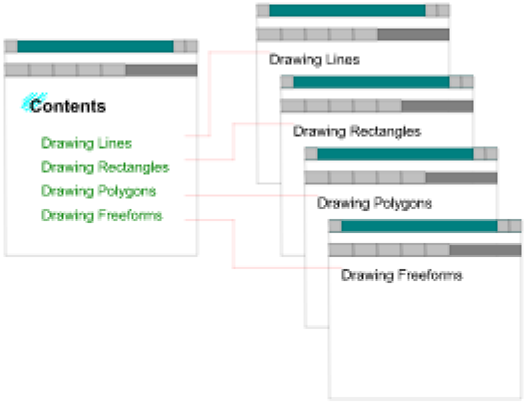
Compiling the Help File

Writing the Help Project File

Writing the Main Contents Topic

See Also   Topic 12 of 14

A contents topic is a topic that lists the sections of your Help system. Each item is hot, so that the user can click it to display the named topic.



WinHelp assumes there is a main contents topic. By default, every Help system has a Contents button on the button bar. When the user clicks that button, the main contents screen is displayed.

To create the main contents topic:

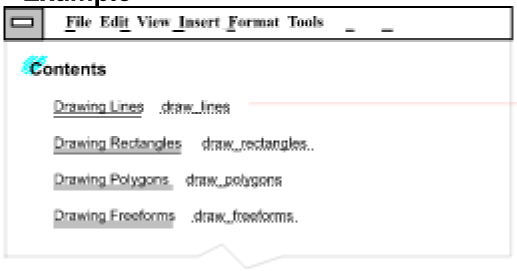
1. Go to the beginning of your first source file.

   By default, WinHelp assumes that the first topic in the first file of your Help source files is the Contents topic.
2. On separate lines, type the titles of the topics you want to appear on the contents screen.

   Although not required, it is advisable that you use the same names for the contents as you used on the first line of your topics. This makes it easier for users to get their bearings.
3. Make each title a hotspot, so that when a user clicks that title, the named topic will be displayed.

   See Adding Hotspots to the Topic for instructions on how to create hotspots.

Example


File Edit View Insert Format Tools
Contents
Drawing Lines draw_lines
Drawing Rectangles draw_rectangles
Drawing Polygons draw_polygons
Drawing Freeforms draw_freeforms

Jumps to topic "Drawing Lines," whose context string is "draw_lines"

See also

About the Contents Topic

Creating the Contents Topic

Jumping to a Contents Topic

Setting the Contents to a Specific Topic

Writing the Help Project File

See Also Topic 13 of 14

The project file is a text file containing information the Help compiler uses to construct the Help file.

The project file can include many instructions that control many aspects of the Help file. This topic only describes what is required and what is typically included to create a simple Help file.

Writing the File

To write a project file for a typical simple help file, follow these steps:

1. Create a text file. Name it as follows:

   - Give the root (i.e., the part of the filename before the extension) the same name you want to be the name of your final compiled Help file.
   - Append the filename extension .HPJ.

   For example, if you want your help file to have the name MYHELP.HLP, name the project file MYHELP.HPJ.

2. Add the following lines to the file. Words in red are placeholders for a value you must supply. These placeholders are explained in "Options," following this list.

   Note: You can insert a comment in the project file by preceding the line with a semicolon. Any line preceded by a semicolon will be ignored.

```
[OPTIONS]
CONTENTS=context_string
TITLE=title
COMPRESS=compress_level
ERRORLOG=log_filename
[CONFIG]
BrowseButtons()
[FILES]
RTF_filename_1
RTF_filename_2
RTF_filename_3
      ⋮
```

3. Save the file as a text file.

Options

`CONTENTS=context_string`

context_string is the context string of the main contents topic. This line is not required. If you do not include it, WinHelp uses the first topic in the first file as the contents topic. When the user clicks the Contents button, the topic named in this line (or the first topic in the first file if this line is not included) is displayed.

`TITLE=title`

title is the name that appears in the title bar of the Help window. Do not enclose the title in quotation marks. This line is not required. If you do not include it, the name "Windows Help" appears on the title bar.

`COMPRESS=compress_level`

compress_level determines how much the file is compressed when it is compiled. It takes longer to compile the file when you compress it.

Use one of the options listed in the Value column of the following table:

Value	To compile...	For this size file...
FALSE	Faster	Larger (no compression)
MEDIUM	Medium	Medium (medium compression)
HIGH	Slower	Smaller (high compression)
0	Faster	Larger (no compression)
1	Slower	Smaller (high compression)
NO	Faster	Larger (no compression)
TRUE	Slower	Smaller (high compression)
YES	Slower	Smaller (high compression)

This line is not required. If you do not include it, the compiler will not compress the file. This is the equivalent of FALSE, 0, or NO.

```
ERRORLOG=log_filename
```

`log_filename` is the name of a file where WinHelp errors are written, if any occur while compiling. This error log is useful for debugging your Help system. When you include this line in the project file, WinHelp creates the file automatically when you compile. This line is not required. If you do not include it, the errors will be displayed on screen but will not be written to a file.

```
BrowseButtons()
```

When you include this line, the browse buttons **>>** (Next) and **<<** (Previous) are added to the Help button bar. This line is not required. If you do not include this line, browse buttons are not added to the button bar.

To implement browse sequences in the Help file, you must include this line, but you must also add browse identifiers to the topics. See Including the Topic in a Browse Sequence: Inserting the + Footnote (topic 9 of 14).

```
[FILES]
```

`RTF_filename_n is` the name of an .RTF source file. Include all the .RTF files that make up your Help system. List each file on a separate line.

You can list the help files in any order. However, be aware of these two characterstics:

- If you do not include a CONTENTS line in the project file, WinHelp assumes that the contents file is the first topic in the first file in this list.
- If you specify a browse sequence but do not include sequential browse identifiers in your topics, WinHelp creates the browse sequence according to the order of the files in this list (and according to the order of the topics within the files).

See Including the Topic in a Browse Sequence: Inserting the + Footnote (topic 9 of 14), for information about adding (or not adding) sequential browse identifiers.

Example

```
; CWH.HPJ
; Help project file for Creating Windows Help

[OPTIONS]
CONTENTS=Main_toc
TITLE=Creating Windows Help
; COMPRESS=NO
COMPRESS=HIGH
ERRORLOG=CWH.OUT

[CONFIG]
BrowseButtons()

[FILES]
TOC.RTF
GETSTART.RTF
BASICS.RTF
TASKS.RTF
REFERENCE.RTF
```

See also
Creating the Help Project File

Compiling the Help File

See Also Topic 14 of 14

About the Help Compiler File

There are currently two Help compilers available for compiling Windows 3.1 Help files:

- HCP.EXE
- H31.EXE

HCP.EXE and HC31.EXE compile Help files to be used with Windows 3.1; they cannot create Help files to be used with Windows 3.0. HCP.EXE is the "protected mode" compiler, which makes better use of system memory. Use HCP.EXE to compile a Help file from a DOS window under Windows.

Compiling Checklist

Before compiling, make sure that all of the following has been done:

- All source topic files must have been saved as Rich Text Format (.RTF) files.
- All of these files must be in the same directory:
- All .RTF files
- The Help compiler (HCP.EXE or HC31.EXE)
- The Help compiler error message resource file (HCP.ERR or HC31.ERR)
- The Help project (.HPJ) file
- Any referenced bitmap (.BMP) or SHED (.SHG) files

> Note: It is possible to put the files in different directories, but you must specify the directories in the project file. See Creating the Help Project File

To compile the Help file:

At the DOS prompt, type:

```
help_compiler_rootname  project_file_rootname
```

where `help_compiler_rootname` is the name of the Help compiler (without the filename extension), and `project_file_rootname` is the name of your Help project file (without the filename extension). For example, type

```
HCP MYHELP
```

to compile a Help file named MYHELP.HLP.

See also

Compiling a Subset of Topics (Conditional Compiling)

Compiling Source Files into the Help File

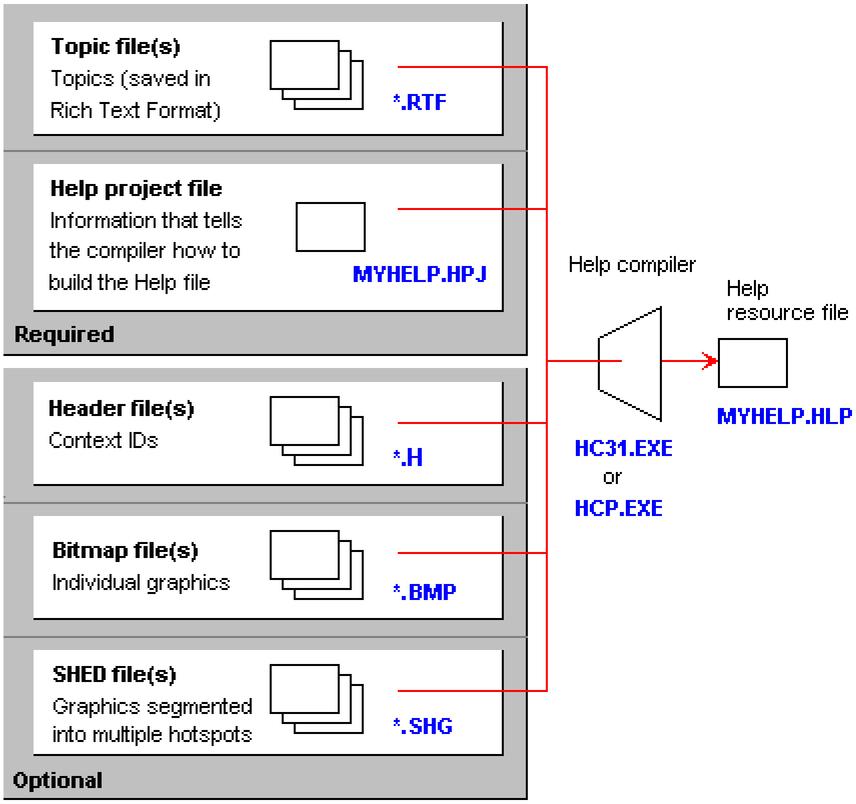
Compiling and Debugging

Basic Troubleshooting

Overview of Source Files Used to Create Help

For More Information...

You use these kinds of files to create a Help system:



Topic file(s)

Topic files contain the words, optional graphics, and the document formatting of your Help system. You can use a word processor to create these files, but they must be saved in Rich Text Format, or else the Help compiler will not be able to compile them.

See Also

Quick Start: Writing Simple Topics

Creating a Topic

Help project file

The Help project file contains information that the Help compiler uses to construct the Help system. This includes such information as the format of the Help windows and controls, what files should be used when compiling, and how the topics should map to the host application for context-sensitive Help. The project file must be an ASCII text file.

See Also

Quick Start: Writing the Help Project File

Creating the Help Project File

Header file(s)

Header files are optional files used to create context-sensitive Help. A header file can be used to map Help topics to user interface objects in the host product or to alias the context string of one topic to the context string of another topic.

See Also

How Applications Are Programmed to Display Context-sensitive Help

Mapping Topic Identifiers for Context-sensitve Help

Mapping Topic Identifiers in a Header File

Bitmap file(s)

Bitmaps are graphics, or pictures. You can paste graphics directly into your topic source files, but if you paste the same bitmap into several places, it unnecessarily increases the size of your final Help file. Instead, you can store the graphics as .BMP files and reference those files in the source topic files.

See Also

Adding Bitmaps by Reference

SHED file(s)

SHED files contain "segmented hypergraphics." These are graphics that contain hotspots, so when the user clicks a hotspot on the graphic, some action is initiated (for example, jumping to another topic).

See Also

Using Hypergraphics

Help compiler

The Help compiler compiles all of your source files into a Help resource file that is used by the WinHelp application to display your Help topics.

See Also

Quick Start: Compiling the Help File

Compiling Source Files into the Help File

Help resource file

When you compile your source files, this is the file that is produced. When you distribute the Help system, you must include the WinHelp application (WINHELP.EXE) and this .HLP file.

How WinHelp Identifies Topics

Each topic in a Help file must be identified by a unique identifier. As the Help author, you use these identifiers to specify jumps to the topics. In a context-sensitive Help system, the host application uses identifiers to associate a context (for example, a dialog box on the user interface or a keyword in a programming environment) with the appropriate help topic.

WinHelp uses two kinds of identifiers, described in the following two topics:

Context Strings

Context Numbers

How WinHelp Identifies Topics - Context Strings

See Also

Context strings are strings of alphanumeric characters used by WinHelp to identify topics.

Where Context Numbers Are Assigned

Every topic must have a context string to identify it. Assign a context string to a topic by putting it in the # footnote for the topic.

Requirements

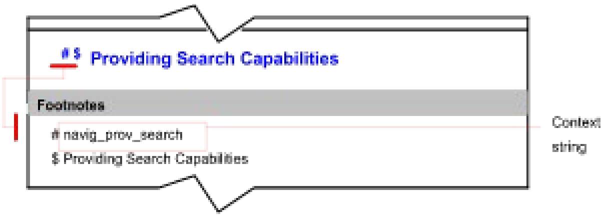
Maximum length: 255 characters

Valid characters: A-Z (upper- and lowercase), 0-9, period (**.**), and the underscore (**_**)

Examples

The following examples show context strings in typical contexts:

A context string used in a # footnote to identify a topic:



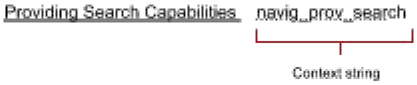
$ Providing Search Capabilities

Footnotes

navig_prov_search

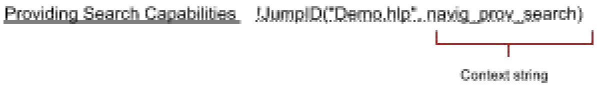
$ Providing Search Capabilities

A context string used to specify a jump in a hotspot definition



Providing Search Capabilities navig_prov_search

A context string used with the JumpID() macro to specify a jump in a hotspot definition:



Providing Search Capabilities !JumpID("Demo.hlp", navig_prov_search)

See Also

Quick Start: Assigning a Context String to a Topic: Inserting the # Footnote

About Footnotes

Creating Footnotes

How WinHelp Identifies Topics - Context Numbers

See Also

Context numbers are numbers used by WinHelp to identify topics.

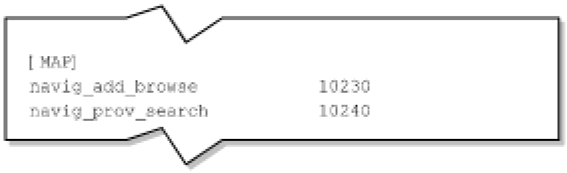
Where Context Numbers Are Assigned

Assign context numbers to topics in the [MAP] section of the Help project file. Or use the [MAP] section to include a header file where the context numbers are assigned.

Examples

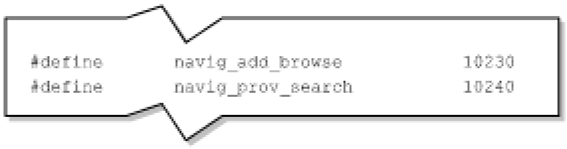
The following examples show context numbers in typical contexts:

Assigning context numbers in the [MAP] section of the Help project file.



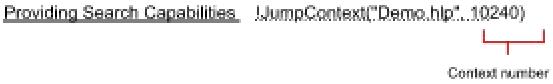
```
[MAP]
navig_add_browse              10230
navig_prov_search             10240
```

Assigning context numbers in a header file.



```
#define       navig_add_browse              10230
#define       navig_prov_search             10240
```

A context number used with the JumpContext() macro to specify a jump in a hotspot definition



```
Providing Search Capabilities  !JumpContext("Demo.hlp", 10240)
```

See Also

How Applications Are Programmed to Display Context-sensitive Help

Mapping Topic Identifiers for Context-sensitve Help

Mapping Topic Identifiers in a Header File

Creating a Topic

See Also

Most Help files consist of many individual Help topics. Each topic is separated from another in the Help topic file by a hard page break.

Note: To add a hard page break in Microsoft Word, press Ctrl+Enter.

Large Help projects often consist of several topic files. Before you begin writing Help topics, try to organize the project into small, manageable topic files. For example, if you are documenting an application's menu structure, create separate topic files for each menu (MNU_FILE.RTF, MNU_EDIT.RTF, and so on).

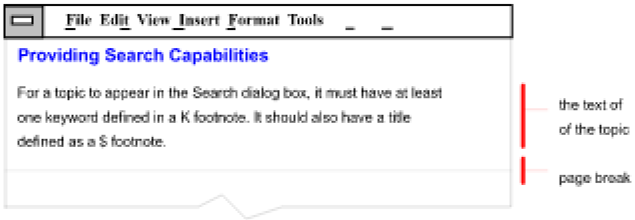
A Help topic usually includes the following elements:

- Footnotes
- Title
- Topic text

In addition, topics can include graphics and hotspots that jump to other topics.

To create topic files, use any word processor or text editor that can save files in Rich Text Format. When you are finished writing topic files, use the Microsoft Help compiler to build a Help resource file.

Example



Providing Search Capabilities

For a topic to appear in the Search dialog box, it must have at least one keyword defined in a K footnote. It should also have a title defined as a $ footnote.

See Also

Quick Start: Writing Simple Topics

Creating Footnotes

Creating a non-scrolling section

Creating a non-wrapping section

Creating the Help Project File

See Also   Reference

The help project file is a text (ASCII) file that contains information that tells the Windows Help compiler how to combine topics and build the help resource file.

Required Format

A project file must be saved as an ASCII text file. You can use any word processor or text editor that can create text files.

Naming a Help Project File

The name for the help project file must follow DOS file-naming conventions, for example, it can be no longer than eight characters.

The filename must include the extension .HPJ.

Use the same root name for the project file that you want used as the root name for your final help file. Give the extension to .HPJ to the project file. The final help file will have an .HLP extension. For example, if you create a project file named MYHELP.HPJ, the final help resource file will be called MYHELP.HLP.

The Parts of the Project File

The project file contains "sections," and a section may contain "options." The section name must be surrounded by square brackets and is by convention written in all uppercase, for example [FILES].

What Must Be Included in a Help Project File

The only required entry in the project file is the list of .RTF files in the [FILES] section.

Help Project File Sections

For complete reference documentation on the sections and options, see the appropriate reference topic. Click the Reference hotspot, above, to display a list of the project file sections..

See Also

Quick Start: Writing the Help Project File

Reference
HPJ statements

Compiling Source Files into the Help File

See Also Reference

The Microsoft Help Compiler is a command line compiler; you compile your source files into a Help file using a command at the DOS prompt.

About the [OPTIONS] Section

The [OPTIONS] section of the Help project file controls how the Help compiler builds your Help file. For more information about the [OPTIONS] section, click Reference at the top of this topic.

To compile your source files into a Help file:

1. Go to a DOS prompt. Note: If you are working in Windows, you can compile a Help file from a DOS window.
2. Change to the directory where the Help compiler and Help project file is.
3. Type the following command:

   hcx project-file

   hcx is the type of compiler you are using (for example, HC31 or HCP).

   project-file is the Help project file (you do not have to type the HPJ extension).

   For example, if you are compiling from within a DOS window, and the name of your project file is MYHELP.HPJ, type:

   ```
   hcp myhelp
   ```

The source files (.RTF files) and graphic files do not have to be in the same directory in which you are compiling; the paths to these files should be specified in the project file.

The compiler runs through a sequence of steps while building the Help file. It compiles files, and checks the integrity of context strings, topic titles, keywords, and browse sequences.

When the compiler is finished building a Help file, the DOS prompt appears. The Help file the compiler builds has the same filename as the Help project file, but with an HLP extension.

See Also

Quick Start: Compiling the Help File

Compiling a Subset of Topics (Conditional Compiling)

Overview of Source Files Used to Create Help

Compiling and Debugging

Basic Troubleshooting

Reference

[OPTIONS] section

HPJ statements

Compiling a Subset of Topics (Conditional Compiling)

See Also Reference

If you want to selectively include or exclude topics from your Help file, you can do so using build tags. Build tags are names that you designate for a conditional build of a Help file, for example, Alpha, Beta, and Gamma.

Coding Topics with Build Tags

You assign build tags to topics by inserting asterisk (*) footnotes at the beginning of the topic. Each topic can be assigned several build tags. Topics without build tags are always included in a build.

Listing Build Tags in the [BUILDTAGS] Section

If you coded topics with build tags, you must list the valid tags in the [BUILDTAGS] section of the Help project file. You can list up to 30 different build tags. For example, the following [BUILDTAGS] section shows five tags:

```
[BUILDTAGS]
ALPHA
BETA
GAMMA
PROFESSIONAL_EDN
STANDARD_EDN
```

Using the Build Option

When you are ready to compile a conditional build, use the Build option to specify the build tags you want to include. You can include only one Build option per project file. The Build option uses the following syntax:

```
BUILD=expression
```

expression is a statement specifying which build tags to include or exclude. The expression can include the following logical operators (in order of precedence):

Operator	Description
~	NOT operator
&	AND operator
|	OR operator

You can use parentheses to override operator precedence.

Examples

Build Expression	Build Instructions
`BUILD = GAMMA & STANDARD_EDN`	Include topics with GAMMA and STANDARD_EDN build tags and topics without build tags
`BUILD = ~PROFESSIONAL`	Include all topics except those with a PROFESSIONAL tag
`BUILD = (BETA|GAMMA) & STANDARD_EDN`	Include all topics with either BETA or GAMMA tags, a STANDARD_EDN tag, and topics without build tags

See Also
Compiling Source Files into the Help File

Reference

Build option

[BUILDTAGS] section

[OPTIONS] section

Compiling and Debugging

See Also Reference

During the compiling phase of building your Help file, you will often receive warnings or error messages. When the Help compiler does not encounter warnings or errors, it displays a series of dots at the DOS command line.

Capturing Compiler Messages

To capture error messages and warnings to a file:

1. Open the project file in a text editor
2. Add the following statement under the [OPTIONS] statement:

   ERRORLOG = filename

Understanding Compiler Messages

Error messages usually occur because the compiler cannot find one or more files including the project file, graphic files, or source RTF files. Errors often then stop the build process.

Warnings are less serious and indicate problems in the file such as formatting errors or misspelled context names. A Help file will be produced but not all of the topics may work properly or be accessable.

See Basic Troubleshooting to find more about the most common causes and solutions to errors and warnings.

Windows 3.1 Help Compiler Message Categories

Message numbers	Source of problem
1000+	Preprocessing
2000+	Project file
3000+	Build tags
3500+	Macros
4000+	Context strings
4200+	Control codes
4600+	Topic file
5000+	Miscellaneous

There are several sources of information about specific error messages including the Microsoft SDK documentation and Windows Help Authoring Guide, Borland language reference manuals, and most third-party books on developing Windows Help.

See Also

Compiling Source Files into the Help File

Basic Troubleshooting

Reference

Build option

[OPTIONS] section

Basic Troubleshooting

See Also

Most errors and warnings can be traced to common mistakes or oversites. Some problems show up in the error log during the build while others are found when viewing the built Help file.

Common mistakes

- Not copying the latest version of the files before doing a build iIf you build files from a different directory than your working directory).
- Not copying all of the needed files (or paths not set up correctly).
- Typographical errors in context IDs (# footnotes).
- Typographical errors in context IDs (formatted as hidden text).

In the project file (.HPJ)

- Missing or misspelled filenames.
- Missing or misspelled HPJ statements such as [OPTTIONS] instead of [OPTIONS].
- Leaving a semicolon at the beginning of a line thereby commenting out that information.
- Putting commands in the wrong section (or omitting the section name), for example listing filenames under [OPTIONS] instead of [FILES].

In the source files

- Formatting part of the context string in a hotspot definition as single or double underline, or formatting part of the hotspot text as hidden text.
- Leaving a space or unformatted characters between the underlined text and hidden text in a hotspot definition.
- Marking page breaks or paragraph marks as hidden.
- Not marking the context string as hidden text in a hotspot definition.
- Not single or double underlining text just before the hidden context string.
- Not using the Keep Lines Together paragraph property to keep information from wrapping.
- Not using tabs or hanging indents correctly.

Tips

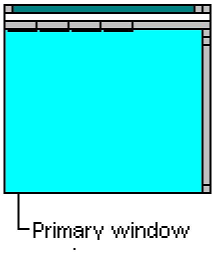
- Search for multiple occurances of an error in a particular context string (formatted as hidden text) in a source file. The Help compiler doesn't report every occurance of an error in a source file.
- Build frequently as you develop the contents of the Help file.
- Avoid renaming context strings when possible.

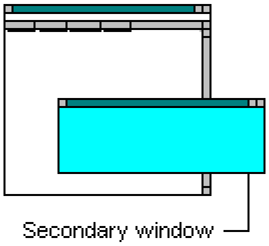
See Also

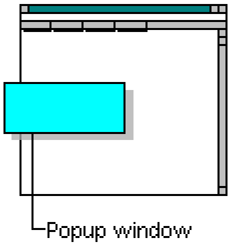
Compiling and Debugging

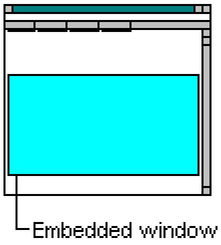
About Different Kinds of Windows

There are four kinds of windows available for displaying topics and other items in Windows Help. Click any of the illustrations below for more information:









Primary window

Ordinarily, the primary window is the default Help window. (Although the application that calls Help *can* be programmed to open Help in a secondary window.) If you do not specify any pop-up windows or secondary windows, all Help topics will appear in this window. The default attributes (size, position,and color) are defined in the project file. Only one primary window can appear at one time, but you can use macros to change its attributes when the user displays a new topic or clicks a hotspot, button, or menu item.

See Also

Defining the Attributes of the Primary Window

Pop-up windows

A pop-up window appears on top of the primary window and is automatically sized to contain the topic that appears in it. Only one pop-up window can be displayed at one time. The pop-up window is removed when the user clicks anywhere on the screen.

See Also

Creating a Hotspot that...

Displays a Topic from the Same Help File in a Pop-up Window

Displays a Topic from a Different Help File in a Pop-up Window

Secondary windows

A secondary window is a separate window that remains on screen until it is explicitly closed, either by the user or by a macro that you build into the Help. The default attributes (size, position, and color) are defined in the project file. You can define several secondary windows in the project file, and several can appear together on screen.

See Also

Closing a Window

Creating a Hotspot that...

Displays a Topic from the Same Help File in a Secondary Window

Displays a Topic from a Different Help File in a Secondary Window

Defining the Attributes of Secondary Windows

Switching Focus to a Different Window

Embedded windows

An embedded window is a rectangular area within a topic that can be used to display or run certain objects that cannot otherwise be included in Help. For example, an embedded window can be used to display 256-color bitmaps, run animation sequences, or play audio. In addition, authors can write their own DLLs to display or run other multimedia objects.

See Also

Creating an Embedded Window

Defining the Attributes of the Primary Window

See Also Reference

Define the title, color, default location, and default size of the primary window by listing them in the Help project file.

Only one primary window can be specified for a Help file.

Quick Reference

Include this line in the [WINDOWS] section of the Help project file:

type = "caption", (x, y, width, height), sizing, (clientRBG), (nonscrollRGB)

Click this button for more Quick Reference on the syntax.

To set a default title for the window, include the following line in the TITLE option of the [OPTIONS] section of the Help project file.If you provide a title for the "caption" option of the [WINDOWS] statement, as shown above, that title will override this default title.

TITLE="titlename"

Example

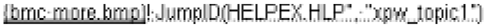
Click this button to display an example of a primary window with different attributes.

Main = , (0, 511, 1024, 512), 0, (0, 255, 255), (128, 128, 128)

How this Example Was Created

Only one primary window can be defined for a Help file. Therefore, the sample primary window in the example is from a separate Help file (named HelpEx.Hlp).

The hotspot used to display this sample window uses the **JumpID** macro to display the different Help file:



```
{bmc more.bmp}!:JumpID(HELPEX.HLP",:"xpw_topic1")
```

See Also

Changing the Size and Position of a Window

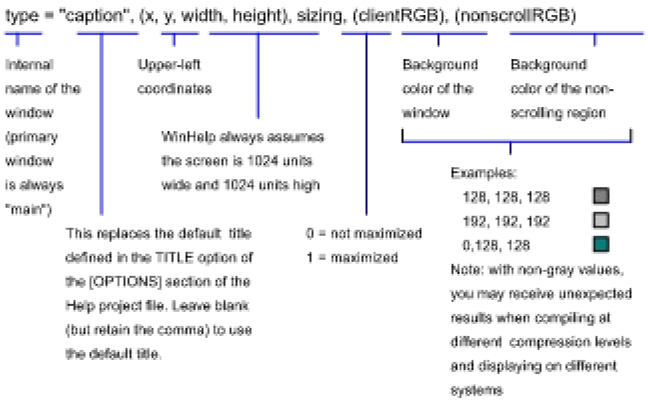
Creating Non-scrolling Titles

Defining the Attributes of Secondary Windows

Reference

TITLE option

[WINDOWS] section



type = "caption", (x, y, width, height), sizing, (clientRGB), (nonscrollRGB)

Internal name of the window (primary window is always "main")

This replaces the default title defined in the TITLE option of the [OPTIONS] section of the Help project file. Leave blank (but retain the comma) to use the default title.

Upper-left coordinates

WinHelp always assumes the screen is 1024 units wide and 1024 units high

0 = not maximized
1 = maximized

Background color of the window

Background color of the non-scrolling region

Examples:

128, 128, 128	▨
192, 192, 192	□
0,128, 128	🟦

Note: with non-gray values, you may receive unexpected results when compiling at different compression levels and displaying on different systems

Defining the Attributes of Secondary Windows

See Also		Reference

Define the title, color, default location, and default size of secondary windows in the Help project file:

You can define many secondary windows for a Help file, but only one can be displayed at one time.

Quick Reference

Include this line in the [WINDOWS] section of the Help project file:

type = "caption", (x, y, width, height), sizing, (clientRBG), (nonscrollRGB)

Click this button for more Quick Reference on the syntax.

Examples

Click these buttons to display examples of secondary windows defined as shown below.

Win1="Secondary Window Win1", (10,50,850,750), 0, (128,128,128), (192,192,192)

Win2="Secondary Window Win2", (25,25, 300,300), 1

How these Examples Were Created

Each of the above examples displays a sample Help topic in a secondary window. All of these secondary windows are defined in the same Help project file (CWH.HPJ). Therefore, the topics that appear in the windows are all part of the same Help file, even though they appear in separate windows.

See Also

Changing the Size and Position of a Window

Creating Non-scrolling Titles

Defining the Attributes of the Primary Window

Reference
[WINDOWS] section

Changing the Size and Position of a Window

See Also Reference About Using Macros

Use the **PositionWindow** macro to change the size and position of the current primary or any secondary window.

Quick Reference

PositionWindow(x, y, width, height, state "name")

or abbreviate as: PW(x, y, width, height, state "name")

Using the Macro

When you change the size and postion of a window, you must explicitly redisplay the window. See the **Example** to see how this is done.

Example

Click the button below to display a secondary window where you can click hotspots to change the size and position of the primary window (this window).

Click here to display the secondary window. Click the Close button in the secondary window to return to this one.

Click here for more information on how this example was created. Click the Back button to return to this topic.

See Also

About Hotspots

About Macros

Creating and Using Compound Macros

Creating a Hotspot that...

- Jumps to a Topic in a Different Help File
- Displays a Topic from the Same Help File in a Secondary Window
- Displays a Topic from a Different Help File in a Secondary Window

Creating Hotspots

Defining the Attributes of the Primary Window

Defining the Attributes of Secondary Windows

Reference

FocusWindow macro

JumpID macro

PositionWindow macro

Close

Example - Changing the Size and Position of a Window

Click this button to move and resize the primary window from its default position and size.

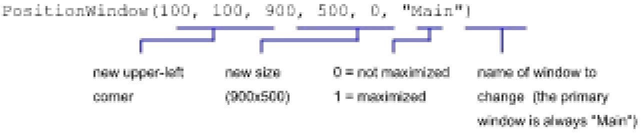
Click this button to return the primary window to its original position and size.

How this Demonstration Was Created

The button that changes the size of the primary window uses the following compound macro:

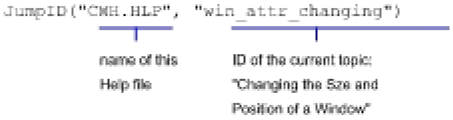
```
PositionWindow(100, 100, 900, 500, 0, "Main"); JumpID("CWH.HLP",
  "win_attr_changing"); FocusWindow("xWin")
```

The **PositionWindow** macro changes the position and size of the primary window:



```
PositionWindow(100, 100, 900, 500, 0, "Main")
```

You must redisplay the window after using **PositionWindow**. The **JumpID** macro redisplays the topic "Changing the Size and Position of a Window" in the resized and repositioned window:



```
JumpID("CWH.HLP", "win_attr_changing")
```

name of this Help file
ID of the current topic: "Changing the Sze and Position of a Window"

When the **JumpID** macro executes, it switches focus back to the primary window. The **FocusWindow** macro switches the focus back to this secondary example window (called "xWin"):

```
FocusWindow("xWin")
```

Switching Focus to a Different Window

See Also Reference About Using Macros

Use the **FocusWindow** macro change the focus to the specified window. You can change focus to the primary window or to any secondary windows.

Quick Reference

FocusWindow("window name")

Example

Click the button below to display a secondary window where you can click hotspots to change the size and position of the primary window (this window). The **FocusWindow** macro is used as part of a compound macro in this example to keep the focus on the secondary window when the primary window is repositioned.

Click here to display the secondary window. Click the Close button in the secondary window to return to this one.

Click here for more information on how this example was created.

See Also

Changing the Size and Position of a Window
Defining the Attributes of a Primary Window
Defining the Attributes of a Secondary Window

Reference
FocusWindow macro

Example - Switching Focus to a Different Window

Click this button to move and resize the primary window and switch the focus back to this window.

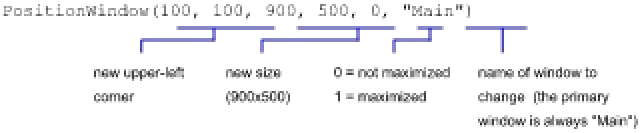
Click this button to return the primary window to its original position and size and, again, switch focus back to this window.

How this Demonstration Was Created

The button that changes the size of the primary window uses the following compound macro:

```
PositionWindow(100, 100, 900, 500, 0, "Main"); JumpID("CWH.HLP",
  "win_attr_changing"); FocusWindow("xWin")
```

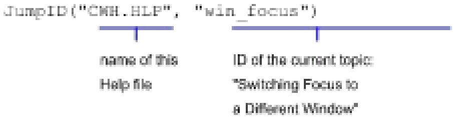
The **PositionWindow** macro changes the position and size of the primary window:



```
PositionWindow(100, 100, 900, 500, 0, "Main")
```

new upper-left corner
new size (900x500)
0 = not maximized 1 = maximized
name of window to change (the primary window is always "Main")

You must redisplay the window after using **PositionWindow**. The **JumpID** macro redisplays the topic "Switching Focus to a Different Window" in the resized and repositioned window:



```
JumpID("CWH.HLP", "win_focus")
```

name of this Help file
ID of the current topic: "Switching Focus to a Different Window"

When the **JumpID** macro executes, it switches focus back to the primary window. The **FocusWindow** macro switches the focus back to this secondary example window (called "xWin"):

```
FocusWindow("xWin")
```

Closing a Window

See Also Reference About Using Macros

Use the **CloseWindow** macro to close either the main Help window or a secondary window.

Quick Reference

CloseWindow("window-name")

Example

Click the button below to go to a topic that demonstrates the CloseWindow macro in a secondary window.

Display the secondary window.

See Also

Defining the Attributes of a Primary Window

Defining the Attributes of a Secondary Window

Reference
CloseWindow macro

Creating an Embedded Window

See Also Reference About Using Macros

An embedded window is a rectangular region within a topic that uses a dynamic-link library (DLL) to display information. You will need to create a custom DLL, or have a developer create one.

Quick Reference

{emx DLL-name, window-class, author-data}

x is one of three values:

l	left aligned
r	right-aligned
c	character-aligned

Note: You will need to consult with the DLL developer to obtain the window-class and author-data information used in the embedded window reference.

Example

How it looks in the .DOC file

```
{eml BORHELP, BHWnd, Str.BH}
{emr BORHELP, BHWnd, Str.BH}
{emc BORHELP, BHWnd, Str.BH}
```

How Embedded Windows Work

Embedded windows do not have any windows features, they are just a rectangular region within a topic window. Users cannot minimize, maximize, or resize an embedded window, and an embedded window cannot be used as a hotspot.

An embedded window will exist as long as the topic containing the embedded window is displayed, even when the topic scrolls past the embedded window. Because it is part of a topic, the embedded window will go away when that topic is no longer displayed.

An embedded window can contain hotspots that are controlled by the DLL, but users will not be able to use keyboard equivalents to access those hotspots.

Positioning Embedded Windows

Embedded windows can be positioned in tables, or within text. Left and right margins between the embedded window and the window margins (or table cell boarders) are set by the paragraph indent. Text will wrap around an **ewr** or **ewl** reference aligned with the upper right or left edge of the embedded window.

Do not put extra spaces between an **ewr** or **ewl** reference and the following text, unless you want the first line of text following the embedded window to indent differently than the subsequent lines of text. You may want to use soft line breaks to ensure that the text wraps properly.

Uses of Embedded Windows

- Display 256-color images.
- Display multi-media elements and their playback controls, such as audio files, animations, and full motion video.
- Assemble and display the contents of a Help topic dynamically based on information stored in an ASCII file or database.

Display images from an application's resource files such as dialog boxes.

See Also

Using Functions from an External Dynamic Link Library (DLL)

Reference

RegisterRoutine macro

About Hotspots

See Also

A hotspot is text or a graphic that the user can click to initiate a jump to another topic, display a topic in a different window, or execute a macro or macros.

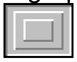
Examples

A text hotspot that jumps to another topic

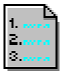
A text hotspot that displays a topic in a pop-up window

A text hotspot that executes the **About** macro (that is, displays the About dialog box)

A graphic hotspot that displays a topic in a popup window:



The same graphic hotspot used to execute the **About** macro:



To show all the hotspots in a topic (in the compiled Help):

Press Ctrl+Tab

See Also

Quick Start: Adding Hotspots to the Topics

Adding Bitmaps by Reference

Creating Hotspots

Using Hypergraphics

Creating a Hotspot that...

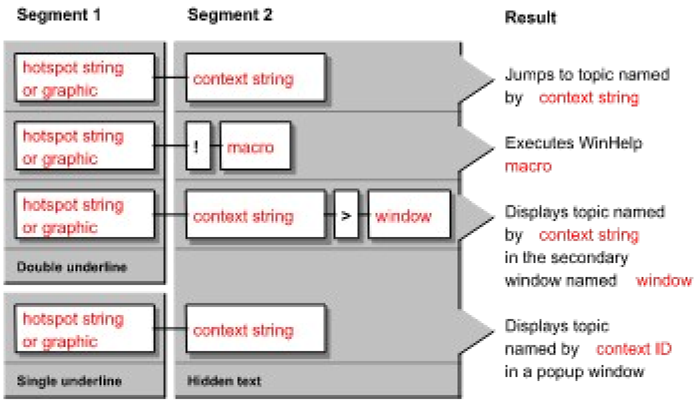
- Jumps to a Topic in the Same Help File
- Jumps to a Topic in a Different Help File
- Displays a Topic from the Same Help File in a Popup Window
- Displays a Topic from a Different Help File in a Popup Window
- Displays a Topic from the Same Help File in a Secondary Window
- Displays a Topic from a Different Help File in a Secondary Window

Creating and Using Compound Macros

Creating Hotspots

See Also   Reference

Quick Reference


Segment 1
Segment 2
Result
hotspot string or graphic
context string
Jumps to topic named by context string
hotspot string or graphic
!
macro
Executes WinHelp macro
hotspot string or graphic
context string
>
window
Displays topic named by context string in the secondary window named window
Double underline
hotspot string or graphic
context string
Displays topic named by context ID in a popup window
Single underline
Hidden text

Type elements that are shown above in **black, bold** exactly as they appear. Supply the appropriate name, ID, or item for the elements that appear in red.

About hotspot definitions

A hotspot definition consists of two segments:

- The text or graphic that will appear in the final Help as the hotspot
- Instructions specifying what will happen when the hotspot is clicked by the user

The table in **Quick Reference**, shows how to format the segments. The first segment is always formatted with a single underline or a double underline. (Overstrike formatting can be used instead of double underlining.) The second segment is always hidden text.

To create a hotspot:

1. Using the appropriate character format, as shown in **Quick Reference**, type the text or insert the graphic you want to appear as the hotspot.
2. Immediately following the text or graphic from step 1, use hidden text format to type the second segment of the definition, as shown in **Quick Reference**.

Note: To apply the formatting, you can either turn the format on and then type the text (or insert the graphic), or you can type the text (or insert the graphic) and then apply the format to it. See the documentation for **Word for Windows** for instructions on applying character formatting.

Do **not** insert any spaces between segment one and segment two.

Examples

How it looks in the .DOC file	How it looks in Help
Jumpx_topic1	Jump
Macro!About()	Macro
Popupx_popup_text	Popup
{bmc square.bmp}x_popup_text	

Creating a hotspot that uses a topic from a different Help file

To create a hotspot that jumps to a topic in a different Help file, use one of the macros below as the second half of the macro hotspot definition:

JumpContents

JumpContext

JumpID

JumpKeyword

To create a hotspot that displays a popup window containing a topic from a different Help file, use one of the macros below as the second half of the macro hotspot definition:

PopupContext

PopupID

See Also

Quick Start: Adding Hotspots to the Topics

About Hotspots

Adding Bitmaps by Reference

Creating a Hotspot that...

- Jumps to a Topic in the Same Help File
- Jumps to a Topic in a Different Help File
- Displays a Topic from the Same Help File in a Popup Window
- Displays a Topic from a Different Help File in a Popup Window
- Displays a Topic from the Same Help File in a Secondary Window
- Displays a Topic from a Different Help File in a Secondary Window

Creating and Using Compound Macros

Reference

JumpContents macro

JumpContext macro

JumpID macro

JumpKeyword macro

PopupContext macro

PopupID macro

About Using Macros

You can associate a macro with hotspots, buttons, menu items, and accelerator keys. You can also execute a macro when the user opens a Help file or displays a topic. See these topics for more information:

About Macros

Assigning a Macro to an Accelerator Key or Keys

Creating a Hotspot that Executes a Macro or Macros

Creating and Using Compound Macros

Creating Hotspots

Executing a Macro when Entering a Topic

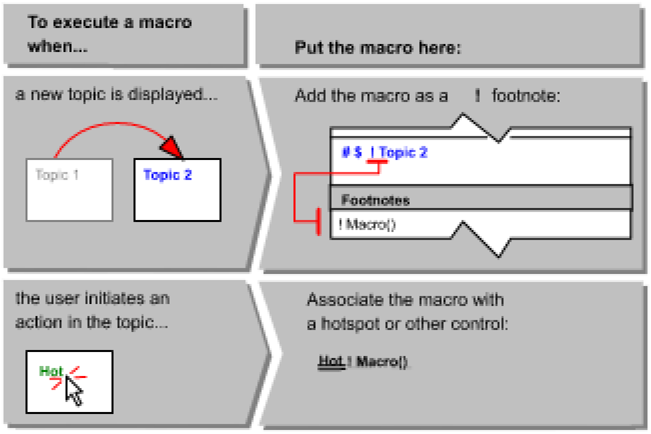
Executing a Macro when Opening a Help File

About Macros

See Also

A macro can be executed when you open a Help file, when you click a hotspot within a topic, or when you display a topic.

Quick Reference



To execute a macro when...	Put the macro here:
a new topic is displayed...	Add the macro as a ! footnote:
the user initiates an action in the topic...	Associate the macro with a hotspot or other control:

Note

You can string several macros together by separating them with semicolons. This is called a compound macro. Some macros have abbreviations, use the abbreviated forms when available when creating a compound macro.

See Also

Assigning a Macro to an Accelerator Key or Keys

Creating a Hotspot that Executes a Macro or Macros

Creating and Using Compound Macros

Executing a Macro when Entering a Topic

Executing a Macro when Opening a Help File

Executing a Macro when Opening a Help File

See Also

Use the [CONFIG] section of the Help project file to list macros you want to execute across an entire Help file. You can also use the [CONFIG] section to register any external DLL functions that your Help file uses.

You should only include macros in the [CONFIG] section that you want to affect the entire Help file, for example, macros that customize menus or the button bar.

To execute macros when you open a Help file:

1. Use a text editor to open the Help project file.
2. Add macros to the [CONFIG] section.
3. Compile the Help file.

Examples

For example, your [CONFIG] section might look like this:

```
[CONFIG]
BrowseButtons()
CreateButton("btn_glos","&Glossary","JumpID(`cwh.hlp',`glossary')")
RegisterRoutine("TOOLSLIB","CHEKFILE","SSS")
```

The macros in this sample [CONFIG] section do the following things:

`BrowseButtons()`

enables browse buttons in the button bar

`CreateButton...`

creates a Glossary button in the button bar

`RegisterRoutine...`

registers a function named CHEKFILE in a DLL called TOOLSLIB.DLL

See Also

About Macros

Assigning a Macro to an Accelerator Key or Keys

Creating and Using Compound Macros

Executing a Macro when Entering a Topic

Reference

[CONFIG] section

BrowseButtons macro

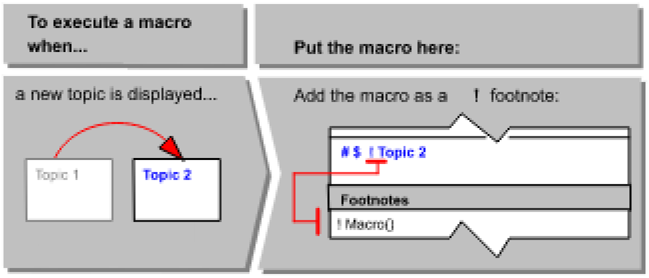
CreateButton macro

RegisterRoutine macro

Executing a Macro when Entering a Topic

Example >>   See Also

Quick Reference


To execute a macro when...
Put the macro here:
a new topic is displayed...
Topic 1
Topic 2
Add the macro as a ! footnote:
# $! Topic 2
Footnotes
! Macro()

To create a macro that executes when entering a topic:

1. Add a **!** footnote to the topic.
2. In the **!** footnote text, type the name of the macro with its arguments.

See Also

About Macros

Assigning a Macro to an Accelerator Key or Keys

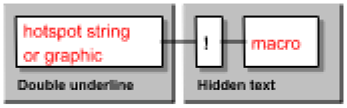
Creating and Using Compound Macros

Executing a Macro when Opening a Help File

Creating a Hotspot that Executes a Macro or Macros

See Also

Quick Reference



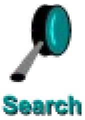
To create a hotspot that jumps to a topic in the same Help file:

1. Using double-underline formatting, type the text or insert the graphic you want to appear as the hotspot.
2. Immediately following the text or graphic from step 1, use hidden text formatting to type the following items. Do not separate any of these iterms by spaces:

An exclamation point (!)

A macro or a compound macro (i.e., several macros separated by semicolons)

Examples

How it looks in the .DOC file	How it looks in Help
About!About()	About
{bml·Search.bmp}!Search()	

See Also

About Macros

Assigning a Macro to an Accelerator Key or Keys

Creating and Using Compound Macros

About Hotspots

Creating Hotspots

Executing a Macro when Entering a Topic

Executing a Macro when Opening a Help File

Assigning a Macro to an Accelerator Key (or Keys)

See Also		Reference

You can assign a macro or group of macros to an accelerator key using the **AddAccelerator** macro. Use the **RemoveAccelerator** macro to remove the macro from the key combination. If you assign a group of macros to a key, separate each macro with a semicolon.

Quick Reference

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

or abbreviate as: AA(key,shift-state,"macro")

RemoveAccelerator(key,shift-state)

or abbreviate as: RA(key,shift-state)

To assign a macro to an accelerator key in a hotspot:

1. Create a hotspot where you want to add hot key access.
2. Following the hotspot, type the AddAccelerator macro using the appropriate arguments as shown in the Quick Reference above.

To assign a macro to an accelerator key in a topic:

1. Create a **!** footnote to the topic where you want to add hot key access.
2. Following the **!** in the footnote pane or type the AddAccelerator macro using the appropriate arguments as shown in the Quick Reference above.

To assign a macro to an accelerator key in the project file:

1. Use a text editor to open the project file.
2. Add the AddAccelerator macro to the [CONFIG] section using the appropriate arguments as shown in the Quick Reference above.

Examples

Click the first button to add an accelerator that displays the About dialog box, then press Alt+Ctrl+F1 to invoke the macro. Then click the second button to remove the accelerator, otherwise the accelerator will continue to work in other topics.

How it looks in the .DOC file **How it looks in Help**

{bmc btn1.bmp}!AA(0x79,6,"About()")

1 then press Alt+Ctrl+F10 to display the About dialog box.

{bmc btn2.bmp}!RA(0x79,6)

2 to remove the accelerator key assignment.

See Also

About Macros

Creating a Hotspot that Executes a Macro or Macros

Creating and Using Compound Macros

Creating Hotspots

Executing a Macro when Entering a Topic

Executing a Macro when Opening a Help File

Reference

AddAccelerator macro

RemoveAccelerator macro

Virtual Key codes

Creating and Using Compound Macros

Example See Also

You can string several macros together so they all are executed as the result of one action. For example, you can create a custom button so that when the button is clicked, a topic is displayed in a secondary window, and all of the navigation controls in the primary window are disabled. Macros that are associated in this way are called compound macros.

You can use compound macros in hotspots and in **!** footnotes.

To create a compound macro:

- Separate each macro in the string with a semicolon.
- Limit the number of characters in all of the macros to 512 characters, including spaces.

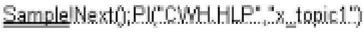
Note: Use abbreviated macro names when available to create compound macros; for example use JI instead of JumpId.

Example

This example jumps to the next topic ("About Footnotes") and displays a pop-up window. If you click the hotspot below, do the following to return to this topic:

1. Click anywhere or press any key to close the pop-up window.

2. Click the Back button to return to this topic.

How it looks in the .DOC file	How it looks in Help
 Sample!Next();PI("CWH.HLP","x_topic1")	Sample

See Also

About Macros

Assigning a Macro to an Accelerator Key or Keys

Creating a Hotspot that Executes a Macro or Macros

Executing a Macro when Entering a Topic

Executing a Macro when Opening a Help File

About Footnotes

See Also   Reference

Use footnotes to code your Help source files to add context strings, topic titles, keywords and phrases, browse sequences, and build tags to your topics, or execute a macro when you display a topic.

There are seven footnote characters you can use to code Help topics:

Footnote character	Used to identify...	Purpose
#	Context string	Uniquely identifies the topic
$	Title	Appears as the topic title in the Search dialog box, the History list, and the Bookmark menu
K	Keywords (and phrases)	These words and phrases appear in the Search dialog box
+	Browse sequence	Determines the order of the topics when the user browses through them
*****	Build tag	Identifies topics that are included or excluded from a compiled Help file
!	Macro	Executes a macro when the topic is displayed
@	Comment	Used to indicate a comment in a topic

You can also use the multikey option to specify custom footnote characters for alternate search keywords.

See Also

Quick Start: Inserting Footnotes for Each Topic

Creating Footnotes

Reference

Multikey option

Creating Footnotes

See Also

To create a footnote for a Help topic:

1. Move the text insertion point to where you want the footnote identifier to be placed, typically before the first character of the first line of the topic.
2. Specify which footnote mark to use (#, $, K, +, *, or !), and insert the footnote.

   (MS-Word provides a dialog box for specifying the footnote identifier).
3. Move the text insertion point to the footnote itself, if it has not already been moved there by the word processor.
4. Type the appropriate information for the footnote.

For information on the specific kinds of footnotes you can create, choose one of the following topics, or use the **>>** (Next) button to browse through the topics in sequence.

Inserting Context String Footnotes

Inserting Topic Title Footnotes

Inserting Keyword Footnotes

Inserting Browse Sequence Footnotes

Inserting Build Tag Footnotes

Inserting Macro Footnotes

See Also

Quick Start: Inserting Footnotes for Each Topic

About Footnotes

Inserting Context String Footnotes

See Also

Create the context string for a topic by inserting a pound sign (**#**) footnote. A context string is a unique text string that WinHelp uses to identify a Help topic. Though not required, context strings are essential if you want to create a true hypertext Help system.

To insert a context string footnote:

1. Move the text insertion point to the first character of the first line of the topic.
2. Insert a **#** footnote.
3. In the footnote, type a string of characters to identify the topic.

   This string is called the "context string." WinHelp uses context strings to identify each topic. The user never sees these strings, but you, the author, will use them to specify jumps to the topic.

   Legal characters are: A-Z (upper- and lowercase), 0-9, period (**.**) and the underscore character (_). The context string must be fewer than 256 characters.

See Also

Quick Start: Inserting Footnotes for Each Topic

About Footnotes

Inserting Topic Title Footnotes

Inserting Keyword Footnotes

Inserting Browse Sequence Footnotes

Inserting Build Tag Footnotes

Inserting Macro Footnotes

Inserting Topic Title Footnotes

See Also

The topic title is the name of a Help topic. It is usually identical to the name that appears on the first line of the topic. The **$** footnote specifies the title that displays in the WinHelp History list, the Search dialog box, and the Bookmark menu.

Generally, all topics should have topic titles. However, if a topic displays exclusively in a pop-up window or a secondary window, a topic title is not needed (because you cannot access topics that display in pop-up windows or secondary windows from the Search dialog box, the History list, or the Bookmark menu).

All topics that have a K footnote should also have a $ footnote. Otherwise, the bottom section of the Search dialog box displays >>Untitled Topic<< for the matching keyword.

To insert a topic title footnote:

1. Move the text insertion point to follow the # footnote mark on the line containing the topic's title.
2. Insert a $ footnote.
3. In the footnote, type a title for the topic (usually the same as the title you typed on the first line of the topic). The title can be up to 128 characters in length.

See Also

Quick Start: Inserting Footnotes for Each Topic

About Footnotes

Inserting Context String Footnotes

Inserting Keyword Footnotes

Inserting Browse Sequence Footnotes

Inserting Build Tag Footnotes

Inserting Macro Footnotes

Inserting Keyword Footnotes

See Also

A keyword is a significant word or phrase that indicates the content of a Help topic. Keywords are similar to index entries. Use the **K** footnote to assign keywords for each topic. The complete keyword list for a Help file appears in the upper section of the Search dialog box.

To insert a keyword footnote:

1. Move the text insertion point to the first character of the first line of the topic.
2. Insert a K footnote.
3. In the footnote, type words and phrases that you want to appear in the search list. Separate each entry with a semicolon.

Note: If a topic does not have a K footnote, it cannot be searched for using the Search button.

See Also

Quick Start: Inserting Footnotes for Each Topic

About Footnotes

Inserting Context String Footnotes

Inserting Topic Title Footnotes

Inserting Browse Sequence Footnotes

Inserting Build Tag Footnotes

Inserting Macro Footnotes

Inserting Browse Sequence Footnotes

See Also

Making browse sequences is a two-part process. You must add the browse buttons (**>>** (Next) and **<<** (Previous)) to the button bar and also specify the browse sequences in the topics.

To add browse buttons to the button bar:

Include the BrowseButton macro in the [CONFIG] section of the project file. See Writing the Help Project File for more information about this step.

To insert a browse sequence footnote:

1. Move the text insertion point to the first character of the first line of the topic.
2. Insert a + footnote.
3. In the footnote, type the browse sequence identifier. The identifier can be a number (for example, 005) or a group name followed by a colon and a number (for example, CALC:005)

Specifying a Simple Browse Sequence

The simplest browse sequence is one where the user can browse, topic by topic, from the first topic to the last. For this kind of sequence, use one sequence of numbers. It is a good idea to space the numbers so you can insert new topics later, if needed. For example:

Desired topic sequence	Sequence identifier
First topic	00005
Second topic	00010
Third topic	00015

Specifying Multiple Browse Sequences

You can also implement multiple browse sequences in a Help file. Use a different prefix for each browse sequence. For example, you might prefix all the topics about graphics with "GRAPH," and prefix all the topics about calculations with "CALC." Then you would have one browse sequence through the graphics topics and another through the calculations topics.

Note: A topic cannot be in more than one browse sequence.

Separate the prefix from the number with a colon. For example:

Desired topic sequence	Sequence identifier
First topic in graphics sequence	GRAPH:00005
Second topic in graphics sequence	GRAPH:00010
Third topic in graphics sequence	GRAPH:00015
First topic in calculations sequence	CALC:00005
Second topic in calculations sequence	CALC:00010
Third topic in calculations sequence	CALC:00015

Using the Order of the Topics to Specify the Browse Sequence

When you include browse sequence identifiers, as in the previous two examples, the topics can exist in any order in the source files.

You can also implement a browse sequence without including sequential browse identifiers in the + footnotes. Use the same identifier in all the topics. When you do this, WinHelp puts the browse sequence in the same order as the topics exist in the source file(s). For example:

Topic sequence	Sequence identifier
First topic in the file	GRAPH:000

Second topic in the file	GRAPH:000
Third topic in the file	GRAPH:000
Fourth topic in the file	CALC:000
Fifth topic in the file	CALC:000
Sixth topic in the file	CALC:000

If you identified the browse sequences as shown in the above example, you would have two browse sequences (one for graphics topics and one for calculations topics) just as in the previous example.

See Also

Quick Start: Inserting Footnotes for Each Topic

About Footnotes

Inserting Context String Footnotes

Inserting Topic Title Footnotes

Inserting Keyword Footnotes

Inserting Build Tag Footnotes

Inserting Macro Footnotes

Inserting Build Tag Footnotes

See Also Reference

Use build tag footnotes (*****) to exclude topics when you compile your Help file. Build tags are names that you designate for a conditional build of a Help file, for example, Alpha, Beta, and Gamma.

To insert a build tag footnote:

1. Move the text insertion point to the first character of the first line of the topic.
2. Insert an asterisk (*) footnote.
3. In the footnote, type the name of the build tag. The build tag can contain any alphanumeric characters, but it cannot contain spaces. If you assign more than one build tag, separate the tags with semicolons. (You can use up to 30 build tags for a Help file.)

The [BUILDTAGS] Section and the Build Option

After you code Help topics with build tag footnotes, you must list the valid tags in the [BUILDTAGS] section of the Help project file.

When you are ready to compile a conditional build, use the Build option to specify the build tags you want to include. You can include only one Build option per project file.

See Also

Quick Start: Inserting Footnotes for Each Topic

About Footnotes

Compiling a Subset of Topics (Conditional Compiling)

Creating the Help Project File

Inserting Browse Sequence Footnotes

Inserting Build Tag Footnotes

Inserting Context String Footnotes

Inserting Keyword Footnotes

Inserting Macro Footnotes

Inserting Topic Title Footnotes

Reference

[BUILDTAGS] section

Build option

Inserting Macro Footnotes

See Also About Using Macros

Use **!** footnotes to execute WinHelp macros when displaying a Help topic. You can insert several **!** footnotes at the beginning of each topic; you can also construct compound macros.

To insert a macro footnote:

1. Add a **!** footnote to the topic.
2. In the **!** footnote text, type the name of the macro with its arguments. If you use more than one macro, separate them with a semicolon. The macro (or compound macro) cannot exceed 512 characters.

See Also

Quick Start: Inserting Footnotes for Each Topic

About Footnotes

Executing a Macro when Entering a Topic

Inserting Browse Sequence Footnotes

Inserting Build Tag Footnotes

Inserting Context String Footnotes

Inserting Keyword Footnotes

Inserting Topic Title Footnotes

Macros, Categorical list

Macros, Alphabetical list

How Users Navigate Through Help

WinHelp provides a number of controls that allow the user to choose how to move from topic to topic. The next two topics show which controls provide which moves:

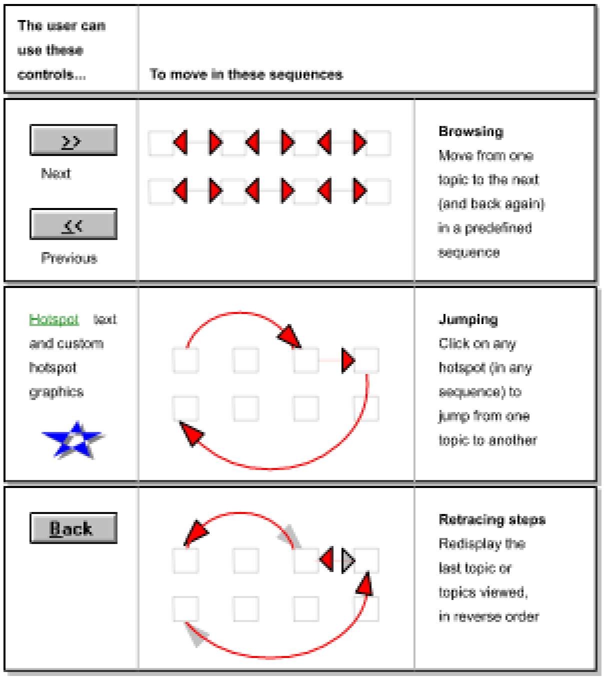
Moving in Sequences

Choosing Specific Topics

Click any of the items in the illustrations to display a pop-up list of topics that explain how to build those navigational paths into your Help. Click any item in the pop-up list to jump to that topic.

How Users Navigate Through Help - Moving in Sequences

For More Information...



The user can use these controls...	To move in these sequences	
>> Next << Previous		**Browsing** Move from one topic to the next (and back again) in a predefined sequence
Hotspot text and custom hotspot graphics		**Jumping** Click on any hotspot (in any sequence) to jump from one topic to another
Back		**Retracing steps** Redisplay the last topic or topics viewed, in reverse order

For More Information...

Click any of the hotspots in the illustration below to display a pop-up list of related topics.

To see what areas are hot:

- Press and hold down Ctrl+Tab to see all the hotspots at once, or
- Press Tab repeatedly to move from one hotspot to the next, or
- Drag the mouse pointer over the illustration: the pointer turns into a hand when it is over a hotspot

Click any item in the pop-up list of related items to jump to that topic.

See Also (Browsing)

Quick Start: Including the Topic in a Browse Sequence: Inserting the + Footnote

Adding Browse Buttons and Browse Sequences

See Also (Jumping)

Quick Start: Adding Hotspots to the Topic

About Hotspots

Creating Hotspots

Creating a Hotspot that...

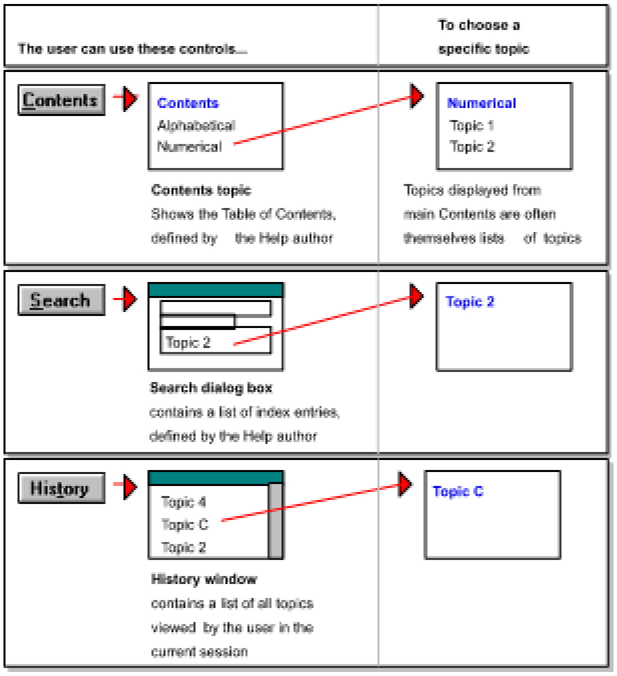
- Jumps to a Topic in the Same Help File
- Jumps to a Topic in a Different Help File
- Displays a Topic from the Same Help File in a Pop-up Window
- Displays a Topic from a Different Help File in a Pop-up Window
- Displays a Topic from the Same Help File in a Secondary Window
- Displays a Topic from a Different Help File in a Secondary Window

See Also (Retracing Steps)

(The Back button appears by default on the WinHelp button bar. You don't have to do anything to make it work.)

How Users Navigate Through Help - Choosing Specific Topics

For More Information...



Contents topic
Shows the Table of Contents, defined by the Help author

Topics displayed from main Contents are often themselves lists of topics

Search dialog box
contains a list of index entries, defined by the Help author

History window
contains a list of all topics viewed by the user in the current session

See also (Contents topic)

Quick Start: Writing the Contents Topic

About the Contents Topic

Creating the Contents Topic

Jumping to a Contents Topic

Setting the Contents to a Specific Topic

See also (Search dialog box)

Displaying the Search Dialog Box

Providing Search Capabilities

See also (History Window)
Displaying the History Window

Adding Browse Buttons and Browse Sequences

See Also Reference

To implement browse sequences, you must add browse buttons to the button bar and define the sequences in the topics.

A Help file can contain several browse sequences, but a topic can be part of only one sequence.

To add Browse buttons to the button bar:

> Add the **BrowseButtons** macro to the [CONFIG] section of the Help project file:

```
[CONFIG]
BrowseButtons()
```

To include a topic in a browse sequence:

1. Make sure the BrowseButtons macro is in the Help project file, as described above.
2. Create a + footnote in the topic.
3. Following the **+** in the footnote pane, type the browse sequence identifier, using alphanumeric characters.

How the Help Compiler Sorts the Browse Sequence Characters

The Help compiler uses ASCII sort to interpret the browse sequence characters in the **+** footnote:

- The string is read character-by-character, left to right.
- Numbers appear before letters.
- Uppercase letters appear before lowercase letters.

For example:

- **`100`** appears before **`99`**
- **`Z`** appears before **`a`**
- **`EditWin: 99999`** appears before **`Editwin: 00001`**

To make the browse sequence follow the sequence of the topics in the file:

1. Create a **+** footnote in each topic of the browse sequence.
2. Following the **+** in the footnote pane, type the same identifier for each topic in the sequence. For example:

   ```
   + FileMenu000
   ```

This way, you do not have to provide a unique browse identifier for each topic, as long as you keep them in order in the file.

See Also

Quick Start: Including the Topic in a Browse Sequence: Inserting the + Footnote

Reference

BrowseButton macro

CreateButton macro

[CONFIG] section

Displaying the History Window

See Also   Reference   About Using Macros

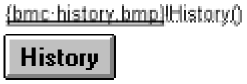
Use the **History** macro to display a window showing a list of the previous 40 Help topics you viewed. You can go to a topic in the list by double-clicking a topic title. The History macro is equivalent to the History button in the WinHelp button bar.

Quick Reference

History()

Example

The History button below executes the History macro.

How it looks in the .DOC file	How it looks in Help
 {bmc·history.bmp}!History()	

Click to display the History list

See Also

How Users Navigate Through Help

Reference
History macro

Displaying the Search Dialog Box

See Also   Reference   About Using Macros

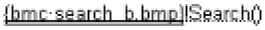
Use the **Search** macro to display the Search dialog box. Executing the Search macro is equivalent to clicking the Search button in the WinHelp button bar.

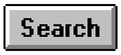
Quick Reference

Search()

Example

The Search button below executes the Search macro.

How it looks in the .DOC file	How it looks in Help
 {bmc·search_b.bmp}!Search()	



Click to display the Search dialog box

See Also

How Users Navigate Through Help

Providing Search Capabilities

Reference
Search macro

Providing Search Capabilities

See Also Reference

For a topic to appear in the Search dialog box, it must have at least one keyword defined in a K footnote. It should also have a title defined as a $ footnote.

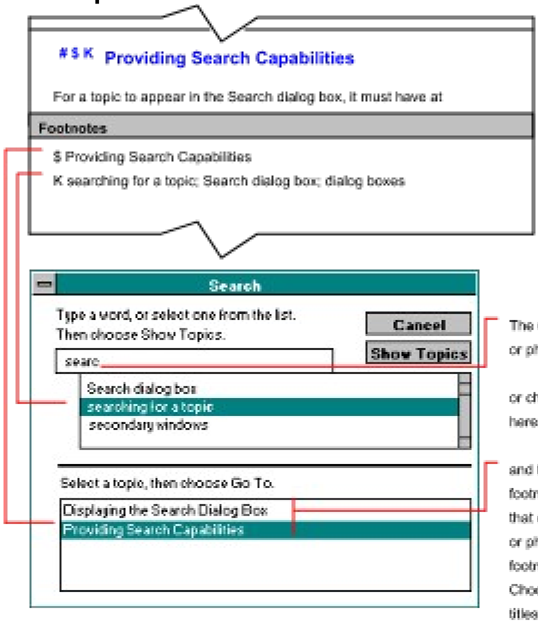
To associate a keyword with a topic:

1. Create a $ footnote in the topic.
2. Following the $ in the footnote pane, type the name of the topic. Typically, this is the same name as the title of the topic, but it can be different. When a user searches for a keyword that is associated with the topic (as described in steps 3 and 4, below), this title appears in the bottom of the Search dialog box.

   (This name is also used elsewhere--e.g., in the History window--but you have to create only one $ footnote.)
3. Create a K footnote in the topic.
4. Following the K in the footnote pane, type words and/or phrases that the user can type or select to search for topics. The K footnotes from all topics combine to make an interactive index that appears in the Search dialog box.

   You can include more than one keyword or key phrase in the footnote. Separate each entry with a semicolon.

Example



#$K Providing Search Capabilities

For a topic to appear in the Search dialog box, it must have at

$ Providing Search Capabilities

K searching for a topic; Search dialog box; dialog boxes

Type a word, or select one from the list.
Then choose Show Topics.

searc

Search dialog box
searching for a topic
secondary windows

Select a topic, then choose Go To.

Displaying the Search Dialog Box
Providing Search Capabilities

The user types a word or phrase here..

or chooses one from here...

and the titles (the $ footnotes) of all topics that contain that word or phrase (the K footnotes) appear here. Choosing one of these titles jumps to the topic.

See Also

Quick Start: Adding Phrases to the Search List: Inserting the K Footnote

How Users Navigate Through Help

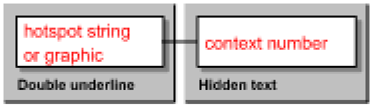
Displaying the Search Dialog Box

Reference
Search macro

Creating a Hotspot that Jumps to a Topic in the Same Help File

See Also　　　　Reference

Quick Reference



To create a hotspot that jumps to a topic in the same Help file:

1. Using double-underline formatting, type the text or insert the graphic you want to appear as the hotspot.
2. Immediately following the text or graphic from step 1, use hidden text formatting to type the context string of the destination topic.

Example

How it looks in the .DOC file	How it looks in Help
Jumpx_topic1	Jump

See Also

About Hotspots

Creating a Hotspot that Executes a Macro or Macros

Creating Hotspots

How Users Navigate Through Help

How WinHelp Identifies Topics

Context ID Numbers

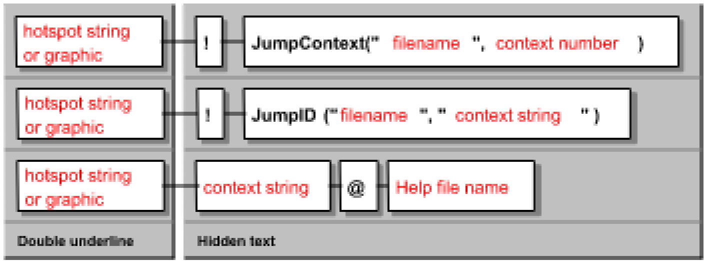
Reference

JumpKeyword macro

Creating a Hotspot that Jumps to a Topic in a Different Help File

See Also   Reference

Quick Reference


hotspot string or graphic
!
JumpContext(" filename ", context number)
hotspot string or graphic
!
JumpID (" filename "," context string ")
hotspot string or graphic
context string
@
Help file name
Double underline
Hidden text

To create a hotspot that jumps to a topic in a different Help file:

1. Using double-underline formatting, type the text or insert the graphic you want to appear as the hotspot.
2. Immediately following the text or graphic from step 1, use hidden text formatting to type the following items. Do not separate any of these items by spaces:

- an exclamation point (!)
- one of the following macros, with the appropriate arguments (as shown in **Quick Reference**, above):

  JumpContext (or JC)
  JumpID (or JI)

You can also use these macros to jump to a topic in the current Help file. Use the name of the current file as the "filename" argument to the macro.

Or:

1. Using double-underline formatting, type the text or insert the graphic you want to appear as the hotspot.
2. Immediately following the text or graphic from step 1, use hidden text formatting to type the following items. Do not separate any of these items by spaces:

- the context string of the destination topic
- an "at" sign (@)
- the name of the Help file that contains the destination topic

Examples

How it looks in the .DOC file	How it looks in Help
Jump!JumpContext("HelpEx.hlp",0002)	Jump
Jump!JumpID("HelpEx.hlp","x_Topic2")	Jump
Jumpx_Topic2@HelpEx.hlp	Jump

See Also

About Hotspots

Creating a Hotspot that Executes a Macro or Macros

Creating Hotspots

How Users Navigate Through Help

How WinHelp Identifies Topics

- Context ID Numbers

Reference

JumpContext macro

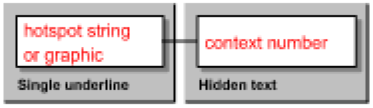
JumpID macro

JumpKeyword macro

Creating a Hotspot that Displays a Topic from the Same Help File in a Pop-up Window

See Also

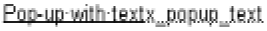
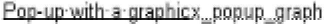
Quick Reference



To create a hotspot that displays a topic from the same Help file in a pop-up window:

1. Using single-underline formatting, type the text or insert the graphic you want to appear as the hotspot.
2. Immediately following the text or graphic from step 1, use hidden text formatting to type the context string of the topic to appear in the pop-up window.

Examples

How it looks in the .DOC file	How it looks in Help
 Pop-up·with·textx_popup_text	Pop-up with text
 Pop-up·with·a·graphicx_popup_graph	Pop-up with a graphic

See Also

About Hotspots

Creating a Hotspot that Executes a Macro or Macros

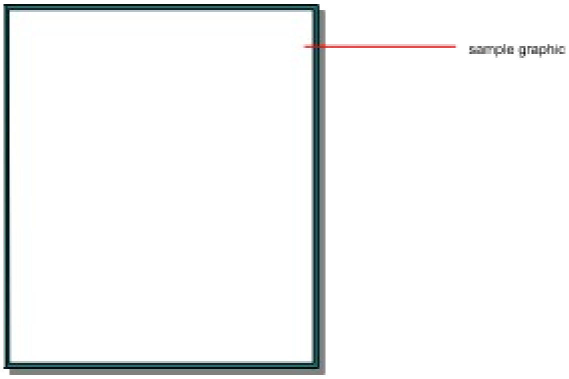
Creating Hotspots

How Users Navigate Through Help

Sample Pop-up Window with Text

This is a sample pop-up window with text.

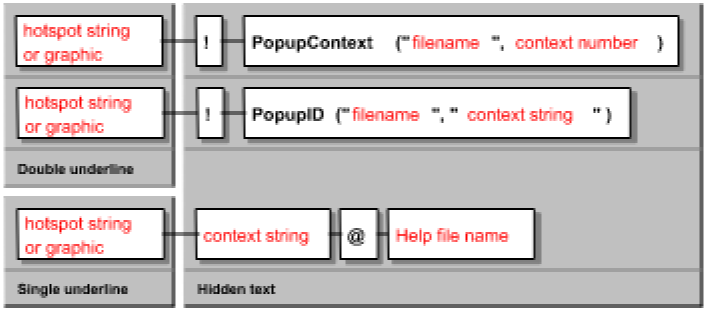
Sample Pop-up Window with a Graphic



Creating a Hotspot that Displays a Topic from a Different Help File in a Pop-up Window

See Also		Reference

Quick Reference


hotspot string or graphic
!
PopupContext ("filename ", context number)
hotspot string or graphic
!
PopupID ("filename "," context string ")
Double underline
hotspot string or graphic
context string
@
Help file name
Single underline
Hidden text

To create a hotspot that displays a topic from a different Help file in a pop-up window:

1. Using double-underline formatting, type the text or insert the graphic you want to appear as the hotspot.
2. Immediately following the text or graphic from step 1, use hidden text formatting to type the following items. Do not separate any of these items by spaces:

- an exclamation point (!)
- one of the following macros, with the appropriate arguments (as shown in **Quick Reference**, above):

  PopupContext (or PC)
  PopupID (or PI)

You can also use these macros to jump to a topic in the current Help file. Use the name of the current file as the "filename" argument to the macro.

Or:

1. Using single-underline formatting, type the text or insert the graphic you want to appear as the hotspot.
2. Immediately following the text or graphic from step 1, use hidden text formatting to type the following items. Do not separate any of these items by spaces:

- the context string of the destination topic
- an "at" sign (@)
- the name of the Help file that contains the destination topic

Examples

How it looks in the .DOC file	How it looks in Help
Pop-up!PopupContext("HelpEx.hlp", 0003)	Pop-up
Pop-upx_Topic2@HelpEx.hlp	Pop-up

See Also

About Hotspots

Creating a Hotspot that Executes a Macro or Macros

Creating Hotspots

How Users Navigate Through Help

Reference

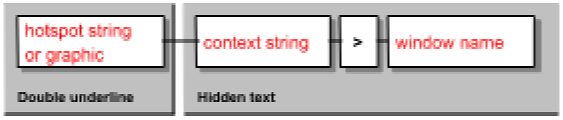
PopupContext macro

PopupID macro

Creating a Hotspot that Displays a Topic from the Same Help File in a Secondary Window

See Also

Quick Reference

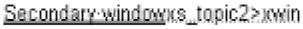


To create a hotspot that displays a topic in a secondary window:

1. Using double-underline formatting, type the text or insert the graphic you want to appear as the hotspot.
2. Immediately following the text or graphic from step 1, use hidden text formatting to type the following items. Do not separate any of these items by spaces:

- context string of the destination topic
- a right caret (>)
- the name of the secondary window, as defined in the project file

Example

How it looks in the .DOC file	How it looks in Help
 Secondary·windowxs_topic2>xwin	Secondary window

See Also

About Hotspots

Creating Hotspots

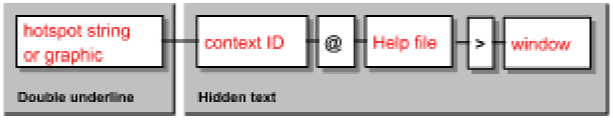
Creating a Hotspot that Executes a Macro or Macros

How Users Navigate Through Help

Creating a Hotspot that Displays a Topic from a Different Help File in a Secondary Window

See Also

Quick Reference

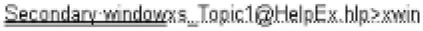


To create a hotspot that displays a topic from a different Help file in a secondary window:

1. Using double-underline formatting, type the text or insert the graphic you want to appear as the hotspot.
2. Immediately following the text or graphic from step 1, use hidden text formatting to type the following items. Do not separate any of these items by spaces:

- the context string of the destination topic
- an "at" sign (@)
- the name of the Help file containing the destination topic
- a right caret (>)
- the name of the secondary window, as defined in the project file

Example

How it looks in the .DOC file	How it looks in Help
 Secondary·windowxs_Topic1@HelpEx.hlp>xwin	Secondary window

See Also

About Hotspots

Creating a Hotspot that Executes a Macro or Macros

Creating Hotspots

How Users Navigate Through Help

Creating Nonscrolling Titles

See Also

To create nonscrolling titles using Word for Windows:

In the .DOC file:

1. Highlight the text that you want to make nonscrolling.
2. Select Paragraph from the Format menu. The Paragraph dialog box is displayed.
3. Select Keep With Next in the Pagination box.

Comments

You can have only one nonscrolling region defined per topic.

Design tip: define a style tag for your topic titles that includes Keep With Next. (See the Word for Windows documentation for more information on formats.)

See Also

Creating Nonwrapping Text

Creating Nonwrapping Text

See Also

To create nonwrapping text using Word for Windows:

In the .DOC file:

1. Highlight the paragraph or paragraphs that you want to make nonwrapping.
2. Select Paragraph from the Format menu. The Paragraph dialog box is displayed.
3. Select Keep Lines Together in the Pagination box.

Comments

Consider that the user might need to scroll or resize the Help window to see all of the information when you use nonwrapping text.

You might want to use tables if the information is not in paragraphs.

Design tip: define a style tag that includes Keep Lines Together.

(See the Word for Windows documentation for more information on formats.)

See Also
Creating Nonscrolling Titles

Adding Graphics to a Topic - Overview

You can include graphics in topics as simple illustrations, as hotspots, and as hypergraphics. You can also add simple formatting to the text in your topic.

Click on an item below to go directly to a topic, or click the **>>** (Next) button to browse through the topics in sequence:

About Inline Bitmaps vs. Referenced Bitmaps

Pasting Graphics (Bitmaps) from the Windows Clipboard

Adding Graphics Using the Word Processor's Graphics Tools

Adding Bitmaps by Reference

Using Hypergraphics

Adding Boxes and Lines

Graphics Supported by WinHelp

WinHelp can display graphics in these formats:

.BMP (standard bitmap)

.DIB (device independent bitmap)

.WMF (Windows metafile)

.SHG (segmented hypergraphics bitmap, created in the Microsoft Hotspot Editor)

.MRB (multiple resolution bitmap, created using the Microsoft Multiple-Resolution Bitmap Compiler)

WinHelp can ordinarily display only monochrome and sixteen-color bitmaps. (See Creating an Embedded Window for information about displaying 256-color images.)

About Inline Bitmaps vs. Referenced Bitmaps

See Also

There are two basic ways to add a graphic to a topic:

- Paste the graphic directly into the topic's source file. This is called an "inline bitmap."
- In the topic, include a reference to a graphic file stored outside the source file. This is called a "referenced" bitmap.

See these topics for more information:

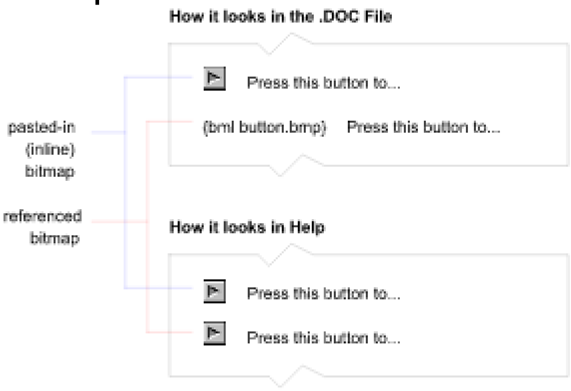
Pasting Graphics (Bitmaps) from the Windows Clipboard

Adding Graphics Using the Word Processor's Graphics Tools

Adding Bitmaps by Reference

(Also click the See Also hotspot, below the title of this topic, for more related topics.)

Example



Advantages to Using Referenced Bitmaps

- Referenced bitmaps take up less memory in the compiled Help file.
- You can store a bitmap file in one place but reference it from many places in the source (.DOC or .RTF) file(s). This reduces the size of the compiled Help file.
- You can wrap text around the graphic.
- You can change the graphic without changing the reference in the source file(s).
- You can use any text editor that supports RTF format to create your source file(s). (That is, you do not have to use a word processor that supports inline bitmaps.)
- You can use hypergraphics.
- You can include graphics larger than 64K. (The Help compiler may not be able to compile inline graphics larger than 64K.)

Disadvantages to Using Referenced Bitmaps

- You cannot see the graphic when you are working in the word processor.
- You must tell the Help compiler where to find the bitmap files when you build the Help file.

Advantages to Using Inline Bitmaps

- You can see the picture whenever you work in the source file.
- You do not have to tell the Help compiler where to find the graphic file.

Disadvantages to Using Inline Bitmaps

- Using the same graphic in more than one place increases the size of the compiled Help file.
- Bitmaps pasted into Word for Windows cannot exceed 64K (or you will receive an Out of Memory error message when you compile).
- You cannot wrap text around an inline graphic.
- The performance of your word processor may be slower when there are embedded graphics in the file.

See Also

About Hotspots

Adding Bitmaps by Reference

Adding Boxes and Lines

Adding Graphics to a Topic - Overview

Adding Graphics Using the Word Processor's Graphics Tools

Creating an Embedded Window

Creating Hotspots

Pasting Graphics (Bitmaps) from the Windows Clipboard

Using Hypergraphics

Pasting Graphics (Bitmaps) from the Windows Clipboard

See Also

If your word processor supports it, you can paste graphic images (bitmaps) from the Windows clipboard directly into your source document. The pasted-in graphic image will be displayed in the compiled Help document just as it appears in the source document. (The graphic must be in a format that can be displayed in the Windows environment.)

Note: Pasted-in bitmaps are sometimes called "inline bitmaps" to distinguish them from "referenced bitmaps," that is, bitmaps that are stored outside the topic and referenced from within the topic. See Adding Bitmaps by Reference for more information.

To create an image and paste it from the clipboard:

1. Create the image in a painting, drawing, or graphics-editing application, for example, Microsoft Paintbrush or Borland Resource Workshop.
2. Use the application's selection tool to select the portion of the image you want to paste into your source file.
3. Use the applications Copy or Cut command to copy the image (or portion of the image) to the Windows clipboard.
4. Go to your source document (.DOC or .RTF file in Word for Windows) in your word processor.
5. Position the cursor where you want to paste the image.
6. Use the word processor's Paste command to paste the image.

The image appears in place.

In Microsoft Word for Windows, the image appears in a frame. You can drag the handles of that frame to change the size and shape of the image. But be aware that the image may not display properly if you adjust the size or shape of the frame.

In Microsoft Word for Windows, you can manipulate the image without returning to the application in which it was created. When you double-click on the image, it is displayed (in a separate window) in the versions of MS-Draw or Paintbrush that are included in Word for Windows.

See Also

About Hotspots

About Inline Bitmaps vs. Referenced Bitmaps

Adding Bitmaps by Reference

Adding Boxes and Lines

Adding Graphics to a Topic - Overview

Adding Graphics Using the Word Processor's Graphics Tools

Creating Hotspots

Using Hypergraphics

Adding Graphics Using the Word Processor's Graphics Tools

See Also

Some word processors include tools for creating and modifying graphics. For example, Microsoft Word for Windows includes versions of MS-Draw and Paintbrush that can be launched from within the main application. Graphics created in these applications appear in place in the source file and thus in the compiled Help file.

To create and insert an image from an embedded application in Word for Windows:

1. In your source (.DOC or .RTF) document, position the cursor where you want the image to appear.
2. Select Insert|Object. The Object dialog box appears.
3. Select the kind of graphic object you want from the list, for example Microsoft Drawing or Paintbrush Picture. The application that creates the desired kind of graphic appears in a separate window.
4. Create an image using the graphics tools available in the application.
5. When finished, exit the program. A prompt asks if you want to update the embedded image. Choose Yes.

The image appears in place.

Importing a Graphic into the Document

If your word processor can import graphic files into the document, and if the graphic is in a format that can be displayed in the Windows environment, that graphic also appears in place in the compiled Help file.

See Also

About Hotspots

About Inline Bitmaps vs. Referenced Bitmaps

Adding Bitmaps by Reference

Adding Boxes and Lines

Adding Graphics to a Topic - Overview

Creating Hot Spots

Pasting Graphics (Bitmaps) from the Windows Clipboard

Using Hypergraphics

Adding Bitmaps by Reference

See Also Reference

Quick Reference

{bmx bitmap-filename}

or

{bmxwd bitmap-filename}

where x is one of three values:

l	left-aligned
r	right-aligned
c	center-aligned
wd	with data (see **Storing Bitmaps with Data**, below)

Adding a Bitmap by Reference

To place a bitmap by reference into the .DOC file:

1.Place the insertion point where you want the graphic to appear, and type the desired bitmap reference, as shown in **Quick Reference**, above.

2. Use a text editor to open the Help project file.

3. Indicate where the bitmap files are located using one of the following methods:

- Add a [BITMAPS] section to the project file and specify the names of the bitmap files.
- Add a BMROOT option to the OPTIONS section of the project file, and specify the directory where the bitmap files can be found.

Adding a Hypergraphic by Reference

To add a hypergraphic (a .SHG file created in the Hotspot Editor) to a topic, you must use the above method to reference the file. Use the name of the .SHG file for the bitmap-filename parameter of the bitmap reference.

Examples

How it looks in the .DOC file	How it looks in Help
{bml·Search.bmp}	
{bmr·Search.bmp}	
{bmc·Search.bmp}	

Storing Bitmaps with Data

Each of the bitmap reference forms **bmc**, **bml**, **bmr** has an alternative form ending with **wd**. Bitmaps positioned with **bmc**, **bml**, or **bmr** appear the same as bitmaps positioned with **bmcwd**, **bmlwd**, or **bmrwd**. However the commands ending with **wd** store the bitmap data differently in the Help file.

When you use **bmc**, **bml**, or **bmr** commands, the bitmaps are stored separately from the data, and a single copy of the bitmap is used for all commands that display the same bitmap file.

The disadvantage is that WinHelp must retrieve the image from the disk before it can be displayed. These bitmaps may display more slowly especially if the Help file is stored on CD-ROM.

When you use **bmcwd**, **bmlwd**, or **bmrwd** commands, the bitmap is stored in the same location in the Help file as the topic text. Each topic that refers to a bitmap will store a copy of that bitmap. This may increase the size of your Help file. If the bitmap is 64K or larger, you will receive an Out-of-Memory error message from the Help compiler. Use this option with smaller bitmaps that appear once or are used infrequently.

See Also

About Hotspots

About Inline Bitmaps vs. Referenced Bitmaps

Adding Boxes and Lines

Adding Graphics to a Topic - Overview

Adding Graphics Using the Word Processor's Graphics Tools

Creating Hotspots

Pasting Graphics (Bitmaps) from the Windows Clipboard

Using Hypergraphics

Reference

[BITMAPS] section

BMROOT option

Adding Boxes and Lines

See Also

To place lines adjacent to text or to surround text with a box:

In the .DOC file, do the following:

1. Highlight the text to which you want to apply the style.
2. Select Border from the Format menu. The Format dialog box is displayed.
3. Select lines or boxes from the dialog box. Click Examples in the topic title bar to see a selection of these styles.

(These instructions are for Word for Windows. See the Word for Windows documentation for more information on formats.)

Comments

In the compiled Help, lines and boxes are automatically sized to the size of the Help window.

Boxes and lines do not show up at all in tables.

The Format|Border|Shading option does not show up in the compiled Help.

The Format|Border|Color option does not show up in the compiled Help.

The Format|Border|Shadow (box shadow) option does not show up in the compiled Help.

Examples

line upper (hairline)

line lower (double hairline)

line left (thick line)

line right (double thick-line)

box (hairline)

This box is narrower because the border of the paragraph is set to 2.5 inches.

See Also

Adding Graphics to a Topic - Overview
About Inline Bitmaps vs. Referenced Bitmaps
Pasting Graphics (Bitmaps) from the Windows Clipboard
Adding Graphics Using the Word Processor's Graphics Tools
Adding Bitmaps by Reference
Using Hypergraphics

Using Hypergraphics

See Also

Hypergraphics are graphics that contain multiple hotspots. Often, the hotspots in a hypergraphic are used to display a pop-up window that describes the part of the graphic covered by the hotspot.

Use the Microsoft Hotspot Editor to turn graphics created as bitmaps into hypergraphics. The Hotspot Editor is also known as SHED, or the Segmented Hypergraphics EDitor. The name of the Hotspot Editor application file is SHED.EXE, and the name of the associated Help file is SHED.HLP. Files created in the Hotspot Editor have an .SHG filename extension.

How to Include Hypergraphics in a Help Topic

Include hypergraphic (SHED) files in a topic the same way you include a referenced bitmap in a topic. For instructions, see Adding Bitmaps by Reference.

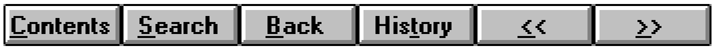
Example

How it appears in the .DOC file

{bml·btnbar.shg}

How it appears in Help

Click on any of the illustrations below for more information:



Getting Help on the Hotspot Editor

Click the button below to jump to the contents of Help for the Hotspot Editor (SHED.HLP). SHED.HLP must be in the same directory as this Help file (CWH.HLP).

Click the Back button or the History button to return to this topic.

The **Contents** button displays the main Help Contents topic.

The **Search** button displays the Search dialog box.

The **Back** button the last topic you viewed.

The **History** button displays the history window from which you can choose among the topics you have viewed in this session of Help.

The **<<** (Previous) button displays the previous topic in the predefined browse sequence.

The **>>** (Next) button displays the next topic in the predefined browse sequence.

See Also

About Hotspots

About Inline Bitmaps vs. Referenced Bitmaps

Adding Bitmaps by Reference

Adding Boxes and Lines

Adding Graphics to a Topic - Overview

Adding Graphics Using the Word Processor's Graphics Tools

Creating Hotspots

Pasting Graphics (Bitmaps) from the Windows Clipboard

Creating and Deleting Custom Buttons

Example See Also Reference About Using Macros

Use the **CreateButton** macro to create a custom button and the **DestroyButton** macro to delete a custom button.

Quick Reference

CreateButton("button ID", "name", "macro")

or abbreviate as: CB("button ID", "name", "macro")

DestroyButton("button ID")

About Creating and Deleting Custom Buttons

You can create custom buttons in one of three ways:

- Place the CreateButton command in the [Config] section of the Help project file. This displays the button in every topic.
- Execute the CreateButton macro in a **!** footnote. This displays the button when the user moves to the topic containing the footnote.
- Create a hotspot that creates a button, either in the current topic or in the destination topic of a jump.

See the examples below for more information on all of these methods.

When you create a custom button with CreateButton, the button remains on the button bar of the current Help file until you remove it with DestroyButton. There are two ways to do this:

- Include the DestroyButton macro in a **!** footnote in every topic where you do not want the button to appear. Do this if the button appears in most of the topics.
- Include the DestroyButton macro as part of a compound macro attached to every navigation control that can be used to move to topics where you do not want the button to appear. Use this method when a button appears in only one or a few topics.

To create custom buttons that display in every topic:

1. Use a text editor to open the Help project file.
2. Add a CreateButton macro to the [CONFIG] Section. In the CreateButton macro, specify the button id, button name, and the macro it will execute.
3. Compile the Help file.

For example, your [CONFIG] section might look like this:

```
[CONFIG]
CreateButton("overv", "Overview", "!JI("CWH.HLP","Helpfile_Overview")")
```

This macro creates a button labeled Overview that will display in every topic (unless macros in that topic override it). The button will jump to the same topic in every instance.

To create a custom button that appears when the user displays a topic:

1. Create a **!** footnote in the topic where you want the button to appear.
2. Following the **!** in the footnote panel, type the CreateButton macro, with the appropriate arguments (as shown in **Quick Reference**, above).

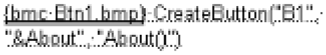
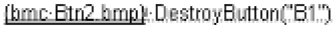
To create a custom button that appears when the user clicks a hotspot:

1. Using double-underline formatting, type the text or insert the graphic you want to appear as the hot spot.
2. Immediately following the text or graphic from step 1, use hidden text formatting to type the following items:

- An exclamation point (!)
- The CreateButton macro, with the appropriate arguments (as shown in **Quick Reference**, above)

Examples

Click button 1 to create a working custom button in this topic. The button displays the **About** dialog box. Then click button 2 to remove the button from the button bar. Otherwise, the custom button will continue to appear in other topics.

How it looks in the .DOC file	How it looks in Help
 {bmc·Btn1.bmp}!·CreateButton("B1",·"&About",·"About()")	Click to create a button
 {bmc·Btn2.bmp}!·DestroyButton("B1")	Click to delete the button

See Also

About Macros

Creating and Using Compound Macros

Disabling and Enabling Buttons

Reference

ChangeButtonBinding macro

CreateButton macro

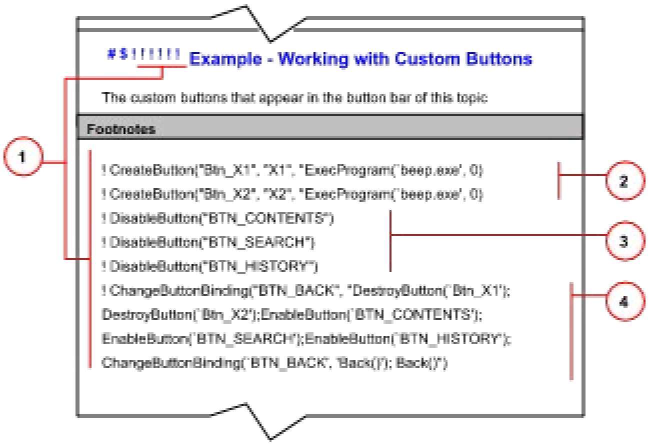
DestroyButton macro

DisableButton macro

EnableButton macro

Example - Working with Custom Buttons

The custom buttons that appear in the button bar of this topic were created using **!** footnotes in the source .DOC file, as shown below:



#$!!!!!! Example - Working with Custom Buttons

The custom buttons that appear in the button bar of this topic

Footnotes

```
! CreateButton("Btn_X1", "X1", "ExecProgram(`beep.exe', 0)
! CreateButton("Btn_X2", "X2", "ExecProgram(`beep.exe', 0)
! DisableButton("BTN_CONTENTS")
! DisableButton("BTN_SEARCH")
! DisableButton("BTN_HISTORY")
! ChangeButtonBinding("BTN_BACK", "DestroyButton(`Btn_X1');
DestroyButton(`Btn_X2');EnableButton(`BTN_CONTENTS');
EnableButton(`BTN_SEARCH');EnableButton(`BTN_HISTORY');
ChangeButtonBinding(`BTN_BACK', `Back()'); Back()")
```

About this Example

1. **!** footnotes are used to create the new buttons and to control how the user can exit this topic.
2. These macros create the two new buttons. The illustration shows the **ExecProgram** macro, but, in fact, this example does not associate any macro with these two sample buttons.
3. These macros disable the default button bar buttons that the user could use to exit this topic. If these were left enabled, without changing their button binding as shown in 4, below, the sample buttons would appear in all subsequent topics. There is no <u>+ footnote,</u> so the Previous (**<<**) and Next (**>>**) buttons are automatically disabled. Therefore, they do not have to be explicitly disabled as shown below.
4. This is one long macro, **ChangeButtonBinding**. It changes the binding of the Back button so that when the user clicks it, the new buttons are deleted, the default buttons are re-enabled, and the most recently viewed topic is displayed:

   `DestroyButton(`Btn_X1'...`

   deletes the new buttons

   `EnableButton(`BTN_CONTENTS'...`

   reenables the previously disabled buttons

   `ChangeButtonBinding("BTN_BACK"...`

   changes the binding of the Back button back to its default, **Back()**

   `Back()`

   executes the **Back()** macro, that is, displays the most recently viewed topic

Disabling and Enabling Buttons

Example See Also Reference About Using Macros

Use the **DisableButton** macro to grey out a button on the button bar. Use the **EnableButton** macro to reenable the button.

Quick Reference

DisableButton("button ID")

or abbreviate as: DB("button ID")

EnableButton("button ID")

or abbreviate as: EB("button ID")

See Also

About Macros

Changing the Function of a Button

Creating and Deleting Custom Buttons

Creating and Using Compound Macros

Reference

ChangeButtonBinding macro

CreateButton macro

DestroyButton macro

DisableButton macro

EnableButton macro

Changing the Function of a Button

Example See Also Reference About Using Macros

Use the **ChangeButtonBinding** macro to change the binding, that is, function, of a button. When you change the binding of a button, you associate a different macro with it so that the macro is executed when the user clicks the button.

Quick Reference

ChangeButtonBinding("button ID", "macro")

or abbreviate as: CBB("button ID", "macro")

See Also

About Macros

Creating and Deleting Custom Buttons

Creating and Using Compound Macros

Reference

ChangeButtonBinding macro

CreateButton macro

DestroyButton macro

DisableButton macro

EnableButton macro

Where to Customize Menus and Menu Items

See Also Reference

You can enter macros that customize the WinHelp menus in the [CONFIG] section of the Help project file or as **!** footnotes at the beginning of topics.

If You Want Help Menus To Be Consistent for All Topics in a File:

Enter macros that customize the WinHelp menus in the [CONFIG] section of the Help project file.

If You Want Help Menus To Change When You Display a Topic:

Enter macros that customize the WinHelp menus in **!** footnotes at the beginning of topics. Because there are multiple ways to jump to a Help topic, you will probably need to enter macros that customize WinHelp menus for nearly every topic in a Help file.

For example, if you want to enable a menu item for only one topic, you will also need to disable it for all other topics in the file.

About Secondary Windows and Pop-up Windows

Secondary windows and pop-up windows do not have menus. Therefore, WinHelp ignores macros that customize menus when they are executed from secondary or pop-up windows.

See Also

About Accelerator Keys

About Menu IDs

About Macros

Creating Menus

Reference
[CONFIG] section

Creating Menus

See Also Reference

Use the **InsertMenu** macro to create a new WinHelp menu.

Quick Reference

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

About Creating Menus

Before you create menu items, that is, commands, you need to create a menu. Ordinarily, menus should remain consistent throughout a Help file. Therefore, the appropriate place to add a menu is in the [CONFIG] section of the Help project file.

Once you add a menu, you cannot delete it. You also cannot remove the standard Help menus (File, Edit, Bookmark, and Help), though you can customize the commands that appear in those menus.

To create a menu for a Help file:

1. Use a text editor to open the Help project file.
2. Add an InsertMenu macro to the [CONFIG] section. In the InsertMenu macro, specify the menu ID, the menu name, and the menu position.
3. Compile the Help file.

For example, your [CONFIG] section might look like this:

```
[CONFIG]
InsertMenu("mnu_def","&Glossary",3)
```

This macro creates a menu named Glossary in the fourth position in the Help menu. The accelerator key for the Glossary menu is G.

See Also

About Accelerator Keys

About Menu IDs

Creating and Deleting Menu Items

Where to Customize Menus and Menu Items

Reference

[CONFIG] section

InsertMenu macro

About Accelerator Keys

See Also

Keyboard accelerators are not required for menus or menu items, though they make the menus easier to access. To assign a keyboard accelerator to a menu or menu item, add an ampersand character (&) before the character you want to use as an accelerator key. Normally, you should use the first character of the menu name as the accelerator key, but that character might already be used by another menu or menu item.

Accelerator keys for menus and menu items must be unique; if the accelerator key you assign is not unique, WinHelp ignores the macro.

See Also

About Menu IDs

Creating and Deleting Menu Items

Creating Menus

About Menu IDs

See Also

Menu IDs that you assign must be different from the IDs already assigned to standard WinHelp menus:

Menu	ID
File menu	mnu_file
Edit menu	mnu_edit
Bookmark menu	mnu_bookmark
Help menu	mnu_help

See Also

About Accelerator Keys

Creating and Deleting Menu Items

Creating Menus

Creating and Deleting Menu Items

Example See Also Reference About Using Macros

Use the **AppendItem** or **InsertItem** macros to add menu items to a WinHelp menu. AppendItem adds a menu item to the end of a menu. InsertItem inserts a menu item within a menu. Use the **DeleteItem** macro to delete a menu item.

To keep WinHelp menus consistent for all topics in a Help file, enter macros that customize the WinHelp menus in the [CONFIG] section of the Help project file.

Quick Reference

AppendItem("menu-id", "item-id", "item-name", "macro")

InsertItem("menu-id", "item-id", "item-name", "macro", position)

DeleteItem("item-id")

See Also

About Accelerator Keys

About Menu IDs

Creating Menus

Enabling and Disabling Menu Items

Where to Customize Menus and Menu Items

Reference

AppendItem macro

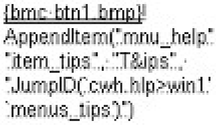
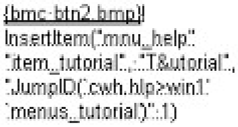
DeleteItem macro

InsertItem macro

Example -- Creating and Deleting Menu Items

To prevent menu changes made in this example from affecting other topics, the Contents, Search, History, and Browse buttons have been disabled.

Click button 1 to append a menu item to the end of the Help menu. Then click button 2 to insert a menu item in the second position of the Help menu. (To see the changes made to the Help menu, click the Help menu or press Alt+H.) Finally, click button 3 to delete the added menu items to prevent them from appearing in other topics.

How it looks in the .DOC file	How it looks in Help
 {bmc btn1.bmp}!AppendItem(`mnu_help', `item_tips', `T&ips', `JumpID(`cwh.hlp>win1', `menus_tips')')	Click to append a Tips command to the Help menu, then click the Help menu (or press Alt+H) to see the new command.
 {bmc btn2.bmp}!InsertItem(`mnu_help', `item_tutorial', `T&utorial', `JumpID(`cwh.hlp>win1', `menus_tutorial')', 1)	Click to insert a Tutorial command in the Help menu, then click the Help menu (or press Alt+H) to see the new command.
	3 Click to remove the added menu items.

Menu Tips

Menus are useful for grouping related functions. Here are some tips to follow when creating WinHelp menus:

- Follow standard Windows user interface guidelines when creating menus. For example, the File menu is usually the first menu and Help is usually the last.
- Keep menus short.
- Group menu commands logically.
- Try to make the first letter of menu commands unique.
- Use ellipses (...) if the menu command displays a dialog box.
- Make menu commands unavailable (gray) only when appropriate.

Tutorial

Sorry, this is just an example; there is not a tutorial for WinHelp. However, note that you can use the **RegisterRoutine** or **ExecProgram** macros to link your Help file to a separate tutorial program.

Enabling and Disabling Menu Items

Example See Also Reference About Using Macros

Use the **DisableItem** macro to disable a menu item. Use the **EnableItem** macro to re-enable the menu item.

Quick Reference

DisableItem("item-id")

or abbreviate as: DI("item-id")

EnableItem("item-id")

or abbreviate as: EI("item-id")

About Disabling and Enabling Menu Items

You can disable custom menu items, that is, menu items that you create using the **InsertItem** or **AppendItem** macros. You cannot disable standard Help menu items such as File|Open. The default color for disabled menu items is gray.

You can enable a menu item only if it is currently disabled. You cannot disable or enable menu items from macros executed in topics that display in secondary windows.

If you want to use a **!** footnote to enable a menu item for only one topic, you will also need to add **!** footnotes to disable the menu item for all other topics in the file.

See Also

Changing the Function of a Menu Item

Creating and Deleting Menu Items

Creating Menus

Where to Customize Menus and Menu Items

Reference

DisableItem macro

EnableItem macro

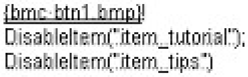
Example -- Disabling and Enabling Menu Items

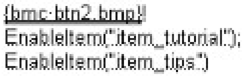
To prevent menu changes made in this example from affecting other topics, the Contents, Search, History, and Browse buttons have been disabled.

Two custom menu items, Tutorial and Tips, have been added to the Help menu for this topic. To see them, click the Help menu or press Alt+H.

Click button 1 to disable the Tutorial and Tips commands. Click button 2 to enable the commands again.

How it looks in the .DOC file	How it looks in Help

 {bmc btn1.bmp}!
DisableItem("item_tutorial");
DisableItem("item_tips")
	Click to disable the Tutorial and Tips commands in the Help menu, then click the Help menu (or press Alt+H) to see the disabled commands.

 {bmc btn2.bmp}!
EnableItem("item_tutorial");
EnableItem("item_tips")
	Click to enable the Tutorial and Tips commands in the Help menu, then click the Help menu (or press Alt+H) to see the enabled commands.

Changing the Function of a Menu Item

Example See Also Reference About Using Macros

Use the **ChangeItemBinding** macro to change the function of a menu item.

Quick Reference

ChangeItemBinding("item-id", "item-macro")

or abbreviate as: CIB("item-id", "item-macro")

About Menu Item Binding

You can use the ChangeItemBinding macro to change the binding, that is, function, of a menu item you have added using the **InsertItem** or **AppendItem** macros. You can also use ChangeItemBinding to change the macro for the How to Use Help menu item on the Help menu. You cannot change the function of other standard Help menu items such as File|Open or Edit|Copy.

See Also

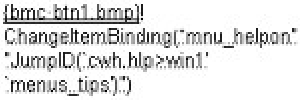
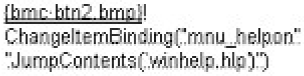
Creating and Deleting Menu Items
Creating Menus
Enabling and Disabling Menu Items
Placing and Removing a Check Mark Beside a Menu Item
Specifying Your Own How to Use Help Topic
Where to Customize Menus and Menu Items

Reference
ChangeItemBinding macro

Example -- Changing the Function of a Menu Item

To prevent menu changes made in this example from affecting other topics, the Contents, Search, History, and Browse buttons have been disabled.

Click button 1 to change the How To Use Help command to display a topic named Tips. (Choose Help| How To Use Help to see the Tips topic.) Click button 2 to change the How To Use Help command back to open the file WINHELP.HLP.

How it looks in the .DOC file	How it looks in Help
 {bmc btn1.bmp}! ChangeItemBinding("mnu_helpon", "JumpID(`cwh.hlp>win1', `menus_tips')")	Click to change the How To Use Help command to display a topic named Tips.
 {bmc btn2.bmp}! ChangeItemBinding("mnu_helpon", "JumpContents(`winhelp.hlp')")	Click to change the How To Use Help command back to open the Help file WINHELP.HLP.

Placing and Removing a Check Mark Beside a Menu Item

Example See Also Reference About Using Macros

Use the **CheckItem** macro to place a check mark beside a menu item. Use the **UncheckItem** to remove a check mark. A check mark indicates that an option has been turned on. For example, Help| Always On Top displays a check mark beside it when you choose it.

Check marks are appropriate only for menu items that toggle between one state and another.

Quick Reference

CheckItem("item-id")

or abbreviate as: CI("item-id")

UncheckItem("item-id")

or abbreviate as: UI("item-id")

See Also

Changing the Function of a Menu Item

Creating and Deleting Menu Items

Creating Menus

Enabling and Disabling Menu Items

Where to Customize Menus and Menu Items

Reference

CheckItem macro

UncheckItem macro

Example -- Placing and Removing a Check Mark Beside a Menu Item

This example demonstrates how to

- add and remove check marks beside a menu item
- use text markers to test a condition before executing a macro

To prevent menu changes made in this example from affecting other topics, the Contents, Search, History, and Browse buttons have been disabled.

A custom menu item, Tips, has been added to the Help menu for this topic. To see it, click the Help menu (or press Alt+H).

To See How the Tips Command Works:

1. Choose Help|Tips. A secondary window appears showing the Help topic Menu Tips.
2. Click the Help menu. A check mark appears beside the Tips command.
3. Choose Help|Tips again. The secondary window showing the Help topic Menu Tips closes.
4. Click the Help menu. The check mark beside the Tips command has been removed.

How the Check Mark Was Created

To place and remove the check mark beside the Tips command, the following **!** footnote was added to the beginning of this topic:

```
AppendItem("mnu_help", "item_tips", "T&ips", "IfThenElse(IsMark(`tips'),
`DeleteMark("tips"); CloseWindow("AboutWin"); UncheckItem("item_tips")',
`SaveMark("tips"); CheckItem("item_tips"); JI("cwh.hlp>AboutWin",
"menus_tips");')")
```

This is one long macro, AppendItem. It appends the menu item Tips to the Help menu, and places or removes a check mark beside the command when you choose it:

```
IfThenElse(IsMark(`tips')...
```

if there is a text marker named "tips," run the first macro (DeleteMark...); if there is not a text marker named "tips," run the second macro (JumpID...)

```
DeleteMark("tips")
```

deletes the "tips" text marker created when you choose the Tips command

```
CloseWindow("AboutWin")
```

closes the secondary window in which the Menu Tips topic appears

```
UncheckItem("item_tips")
```

removes the check mark beside the Tips command

```
SaveMark("tips")
```

saves a text marker named "tips" for the Menu Tips topic

```
CheckItem("item_tips")
```

places a check mark beside the Tips command

```
JumpID("cwh.hlp>AboutWin",...
```

jumps to the Menu Tips topic, displaying it in the secondary window AboutWin

Specifying Your Own How to Use Help File

See Also Reference About Using Macros

Use the **SetHelpOnFile** macro to change the Help file that the How To Use Help command opens.

Quick Reference

SetHelpOnFile("filename")

About the How to Use Help Command

By default, the How To Use Help command opens the WINHELP.HLP Help file that comes with Windows. You can specify your own Help file to use instead. Because menus and menu items should behave consistently throughout a Help file, the appropriate place to specify the Help file used for the How to Use Help command is in the [CONFIG] section of the Help project file.

To specify the file used by the How to Use Help command:

1. Use a text editor to open the Help project file.
2. Add a SetHelpOnFile macro to the [CONFIG] section, specifying the Help file you want to use for How to Use Help.
3. Compile the Help file.

Example

This macro in the [CONFIG] section of the Help project file changes the Help file that the How To Use Help command opens from WINHELP.HLP (the default file) to HOWHELP.HLP:

```
[CONFIG]
SetHelpOnFile("howhelp.hlp")
```

Note: You can also use the ChangeItemBinding macro to change the Help file that the How To Use Help command opens. For example, you could add the following macro to the [CONFIG] section of the Help project file:

```
ChangeItemBinding("mnu_helpon","JumpContents(`howhelp.hlp')")
```

See Also

Jumping to the How to Use Help Topic

Changing the Function of a Menu Item

Reference

SetHelpOnFile macro

ChangeItemBinding macro

[CONFIG] section

Jumping to the How To Use Help Topic

See Also Reference About Using Macros

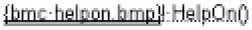
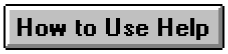
Use the **HelpOn** macro to display the Help file that explains how to use Help. (The default Help file for WinHelp is WINHELP.HLP.)

Quick Reference

HelpOn()

Example

The following How To Use Help button executes the HelpOn() macro to display the How To Use Help file:

How it looks in the .DOC file	How it looks in Help
 {bmc helpon.bmp}! HelpOn()	 Click to display the How To Use Help file.

See Also

Specifying your Own How to Use Help File

Reference
HelpOn macro

About the Contents Topic

See Also Reference

Much like the table of contents in a book, the Contents topic establishes the hierarchy of topics and the overall structure of your Help file. A well-designed Contents topic reveals what information is included in a Help file and how it is organized.

The Contents topic is also an important navigation device. Many applications have a Help menu with a Contents command; the Contents topic, therefore, is one of the primary entry points for a Help system. You can jump to the Contents topic by clicking the Contents button in the WinHelp button bar.

Specifying the Contents Topic

When you open a Help file, WinHelp displays the Contents topic by default. You specify the Contents topic using the Contents option in the [OPTIONS] section of the Help project file. If you do not use the Contents option, the Contents topic is the first topic in the first file listed in the [FILES] section.

See Also

Writing the Contents Topic

Designing the Contents Topic

Jumping to a Contents Topic

Setting the Contents to a Specific Topic

Reference

[OPTIONS] section

[FILES] section

Designing the Contents Topic

See Also

There are many approaches to designing the Contents topic of a Help file. Here are some general guidelines:

Present a Manageable Hierarchy

For fast access to Help information, limit the number of hierarchical levels in your Help file (try not to exceed four levels). The user should not have to jump eight times from the Contents topic before arriving at the topic he wants. Conversely, avoid flat hierarchies that result in unusually long Contents topics.

Keep it Short

For navigation purposes, the Contents screen should be able to be read and understood quickly. Try to keep the topic brief enough so that it does not require scrolling.

Avoid Vague Headings

To avoid confusion, headings should be clear and direct. For example, vague headings such as "Other Features" and "Advanced Topics" do not adequately describe the topics that they contain. If headings in the Contents topic are vague or too general, it may mean that you need to reorganize or restructure your Help file.

Use Graphics to Illustrate Concepts

Use appropriate graphics to help clarify complex headings. Graphics can also serve as signposts in your Help topics, illustrating your position in the Help structure. (Click the Contents button to see the graphics used to illustrate the Contents screen for this Help file.)

See Also

About the Contents Topic

Writing the Contents Topic

Jumping to a Contents Topic

Setting the Contents to a Specific Topic

Jumping to a Contents Topic

See Also Reference About Using Macros

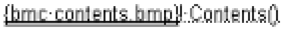
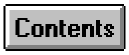
Use the **Contents** macro to jump to the Contents topic.

Quick Reference

Contents()

Example

The following Contents button executes the Contents() macro:

How it looks in the .DOC file	How it looks in Help
 {bmc·contents.bmp}!·Contents()	 Click to jump to the Contents.

See Also

About the Contents Topic

Writing the Contents Topic

Designing the Contents Topic

Setting the Contents to a Specific Topic

Reference
Contents macro

Setting the Contents to a Specific Topic

See Also Reference About Using Macros

Use the **SetContents** macro to set the Contents to a specific topic.

Quick Reference

SetContents("filename",context-number)

To set the Contents to a specific topic:

Use the SetContents macro in the [CONFIG] section of the Help project file or in a **!** footnote in a topic. Specify the filename and the context number for the topic.

Note: You can also use the **ChangeButtonBinding** macro to set the Contents button to jump to a specific topic.

Example

The following SetContents macro in the [CONFIG] section sets the Contents topic for a help file to the topic in the file CWH.HLP with the context number 100.

```
[CONFIG]
SetContents("cwh.hlp",100)
```

See Also

About the Contents Topic

Writing the Contents Topic

Designing the Contents Topic

Jumping to a Contents Topic

Reference

SetContents macro

ChangeButtonBinding macro

Defining a Bookmark

See Also Reference About Using Macros

Use the **BookmarkDefine()** macro to display the Bookmark Define dialog box. This is equivalent to choosing Bookmark|Define.

Quick Reference

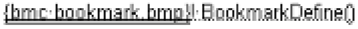
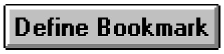
BookmarkDefine()

About Bookmarks

Bookmarks let you mark a Help topic for later reference. When you define bookmarks, the bookmark names appear in the Bookmark menu in the order you create them. (Bookmarks are different from WinHelp text markers, which are used to test a condition before executing a macro; see Working with Text Markers.)

Example

The following Define Bookmark button executes the BookmarkDefine() macro:

How it looks in the .DOC file	How it looks in Help
 {bmc bookmark.bmp}! BookmarkDefine()	 Click to display the Bookmark Define dialog box.

See Also

Going to a Bookmark

Reference

BookmarkDefine macro

CreateButton macro

Going to a Bookmark

See Also Reference About Using Macros

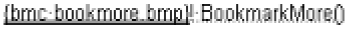
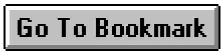
To go to a bookmark, use the **BookmarkMore()** macro, which displays the Bookmark More dialog box. The BookmarkMore macro is equivalent to choosing Bookmark|More.

Quick Reference

BookmarkMore()

Example

The following Go To Bookmark button executes the BookmarkMore() macro:

How it looks in the .DOC file	How it looks in Help
 {bmc·bookmore.bmp}!·BookmarkMore()	 Click to display the Bookmark More dialog box.

See Also

Defining a Bookmark

Reference

BookmarkMore macro

CreateButton macro

Printing a Topic

See Also Reference About Using Macros

Use the **Print** macro to print a Help topic. The Print macro is equivalent to choosing File|Print. You can print only one Help topic at a time.

Quick Reference

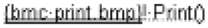
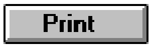
Print()

To print a topic that displays in a secondary window:

Secondary windows do not have WinHelp menus, so they can be printed only using the Print macro. To print from a secondary window, add a hotspot Print button (like the one in the following example) that executes the Print macro.

Example

The following Print button executes the Print() macro:

How it looks in the .DOC file	How it looks in Help
 {bmc print.bmp}!Print()	 Click to print this topic.

See Also

Copying Help text to the Clipboard

Reference

Print macro

PrinterSetup macro

Opening Another Help File

See Also Reference About Using Macros

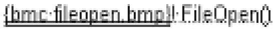
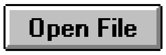
Use the **FileOpen** macro to display the File|Open dialog box. The FileOpen macro is equivalent to choosing File|Open.

Quick Reference

FileOpen()

Example

The following Open File button executes the FileOpen() macro:

How it looks in the .DOC file	How it looks in Help
 {bmc·fileopen.bmp}!·FileOpen()	 Click to display the File|Open dialog box.

See Also

Creating a Hotspot that...

- Jumps to a Topic in a Different Help File
- Displays a Topic from a Different Help File in a Popup Window
- Displays a Topic from a Different Help File in a Secondary Window

Exiting Windows Help

Reference
FileOpen macro

Exiting Windows Help

See Also Reference About Using Macros

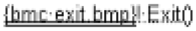
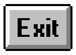
Use the **Exit** macro to exit from WinHelp. The Exit macro is equivalent to choosing File|Exit.

Quick Reference

Exit()

Example

The following Exit button executes the Exit() macro:

How it looks in the .DOC file	How it looks in Help
 {bmc exit.bmp}! Exit()	 Click to exit WinHelp.

See Also

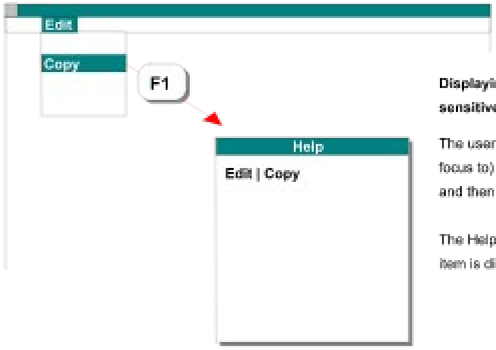
Opening Another Help File

Reference
Exit macro

About Context-sensitive Help

See Also

It is possible to create help topics that are associated with a context in the host application. When that context has input focus, and the user asks for help on the object (usually by pressing F1 or by clicking a Help button), the associated topic is displayed. This kind of implementation is called "context-sensitive help."



Displaying Context-sensitive Help

The user highlights (gives focus to) the menu item and then presses F1.

The Help topic for that menu item is displayed.

A context can be a user-interface object such as a menu, menu item, window, dialog box, or dialog box option. A context can also be a predefined string of text, for example a reserved programming keyword, that appears in a text field or in an editor window.

See Also

Implementing Context-sensitive Help - Overview

Implementing Context-sensitive Help - Overview

Follow these steps to implement context-sensitive Help for an application.

In the following list, each step contains a reference to one or more topics. You can click the name of a topic to go directly to it, or you can use the browse buttons, **<<** (Previous) and **>>** (Next), to browse in sequence through them all.

1. Determine how you want the Help topics to correspond to the host application's user interface. For example, do you want a Help topic for every menu command? every dialog box? every option in a dialog box?

   See Planning and Designing Context-sensitive Help for more information.

2. Determine how the application will call Help. If you are not programming the application yourself, consult with the application programmer(s) to determine how this is or should be done.

   There are two kinds of identifiers the application can use to call WinHelp when help is requested for a user-interface object. The application can pass to the WinHelp application either

   a context string or

   a context number.

You must proceed differently, depending on which method is used. See steps 3 and 4, below, for the alternatives.

See also How Applications Are Programmed to Display Context-sensitive Help for more information.

3. If the application passes context strings, you (the Help author) must make sure that each object in the application is programmed with the correct one. If you are not programming the application yourself, confer with the application programmer(s) to ensure this is done correctly.

   See these topics for more information:

   Planning and Designing Context-sensitive Help

   How Applications Are Programmed - Passing Context Strings

4. If the application passes context numbers, you must map the numbers to the context strings in your Help system.

   See these topics for more information:

   How Applications Are Programmed - Passing Context Numbers

   Mapping Topic Identifiers for Context-sensitve Help

   Mapping Topic Identifiers in a Header File

   Sharing a Header File with the Application

5. You may not want to provide a separate topic for every object for which the application is programmed to provide context-sensitive Help. For example, the application may be calling WinHelp with a unique identifier for every option in a dialog box, but you want to provide only one topic for the entire dialog box.

   In this case, you can alias all of the dialog box's options to the main dialog box topic. Then, when the application calls WinHelp with the identifier for one of the options, the topic for the dialog box is displayed instead.

   See Assigning Aliases for Context Strings for more information.

See Also

About Context-sensitive Help

Planning and Designing Context-sensitive Help

See Also

The first step in creating context-sensitive Help is to decide how the Help system will be organized and how it will respond to the user's requests for Help. You must decide:

- What parts of the application will have Help topics associated with them.
- When and how can the user ask for help on an object.

Candidates for Having Context-sensitive Help

- Menus
- Menu items
- Dialog boxes
- Dialog box options
- Windows or special screen regions
- Other user-interface objects, like button bars or palettes
- Highlighted text, or text that is touched by the text insertion point

Typical Controls an Application Can Provide for Calling Help

- A Help button in a dialog box
- A Help button on a button bar
- A command on the application's Help menu, such as Help on Current Object
- Certain keystrokes: F1 is the Windows standard for calling Help
- Certain mouse clicks, for example Ctrl+right-click

Further Considerations

- Can the user get help on disabled menu items (or other unavailable objects)?
- If two means of calling Help are available in a context (for example, a Help button and F1), will both display the same Help topic?

See Also

About Context-sensitive Help

Implementing Context-sensitive Help - Overview

How Applications Are Programmed to Display Context-sensitive Help

See Also

Different programming languages and development environments provide different means for calling and displaying WinHelp. Consult the documentation for your programming environment for details.

To create Help, you do not have to understand all the details of how the host application is programmed to call WinHelp. But in order to implement context-sensitive Help, you do have to know what kind of identifier the application is passing to WinHelp when the user asks for help on an object.

How the Application Identifies an Object for WinHelp

When the user asks for context-sensitive Help for the object with input focus, the application does the following:

- Sends a message to start the WinHelp application (or redisplay it),
- Tells WinHelp which Help file to use, and
- Passes to WinHelp a unique number or string that identifies the topic to display.

You must know whether the application is passing context strings or context numbers. This determines how you must proceed to implement context-sensitive Help. See the following topics for more information about the different methods:

How Applications Are Programmed - Passing Context Strings

How Applications Are Programmed - Passing Context Numbers

See Also

About Context-sensitive Help

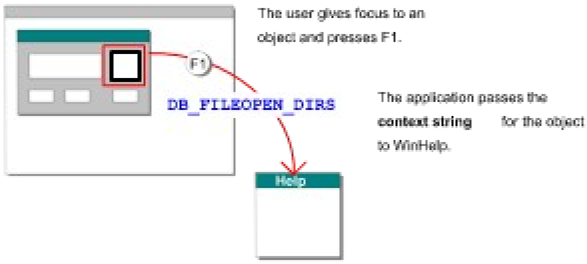
Implementing Context-sensitive Help - Overview

How Applications Are Programmed - Passing Context Strings

See Also Reference

Some programming languages and application development environments provide a way to assign Help context strings directly to user-interface objects. When the application calls Help in these environments, it can pass to WinHelp the same context strings you use to identify topics in your Help system.

Passing a Context String to WinHelp


The user gives focus to an object and presses F1.
F1
DB_FILEOPEN_DIRS
The application passes the **context string** for the object to WinHelp.
Help

What the Help Author Needs To Do

In these environments, you (the Help author) do not have to do anything to interpret the context identifier that the application sends to WinHelp for the object with input focus.

(Contrast this with the need to create a [MAP] section in the project file when the application passes context numbers. See Mapping Topic Identifiers for Context-sensitive Help for more information about what you have to do if the application passes context numbers.)

If someone else is programming the application using one of these methods, you must give the programmer(s) a list of all the context strings for all the topics that will be used for context-sensitive Help. The programmer(s) then must use those strings, as appropriate for the method used.

About Languages and Environments that Can Pass Context Strings

There are several ways that an application can be programmed to use context strings, as described below:

Specifying the Help Topic Identifier as a Property of an Object

In some programming languages, you can define a user interface object and specify as one of its properties the Help context string of the associated Help topic.

In some visual development environments, you can display an object/property inspector for a user-interface object and enter into one of the inspector's fields the context string or context number for the associated topic. (An object/property inspector is a dialog box or window that lets you see and sometimes change the values of certain properties of an object.)

Consult the documentation for your programming language or development environment for more information on assigning Help as a property, if available.

Using the JumpID Macro to Call WinHelp

If the host application for your Help system is written in a language like C or Pascal, the application programmer can use the WinHelp **JumpID** macro with the Windows API (Application Programming Interface) function **WinHelp** to specify which Help topic to display for an object.

The JumpID macro accepts the Help context string as one of its parameters, so the programmer can specify the Help context string directly in the source code for the object.

Consult the documentation for your programming language for more information about how to use the WinHelp function.

See Also

About Context-sensitive Help

Implementing Context-sensitive Help - Overview

Reference

JumpID macro

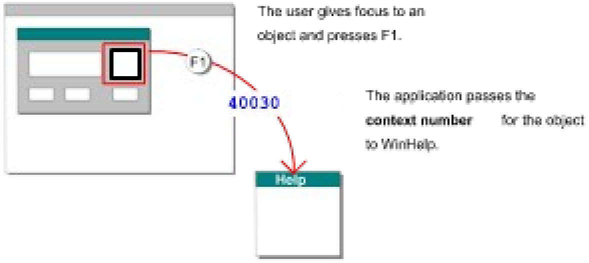
[MAP] section

How Applications Are Programmed - Passing Context Numbers

See Also Reference

In languages like C and Pascal, the programmer can assign unique Help context numbers to user-interface objects. When a user asks for context-sensitive Help on one of these objects, the application passes the context number for the object to WinHelp.

Passing a Context Number to WinHelp


The user gives focus to an object and presses F1.
F1
40030
The application passes the **context number** for the object to WinHelp.
Help

What the Help Author Needs To Do

When the application is programmed this way, you (the Help author) must map all the context numbers to the appropriate Help context strings in your source files. Do this in the [MAP] section in the Help project file.

See Mapping Topic Identifiers for Context-sensitive Help for more information.

See Also

About Context-sensitive Help

Implementing Context-sensitive Help - Overview

Reference

[MAP] section

JumpID macro

Mapping Topic Identifiers for Context-sensitive Help

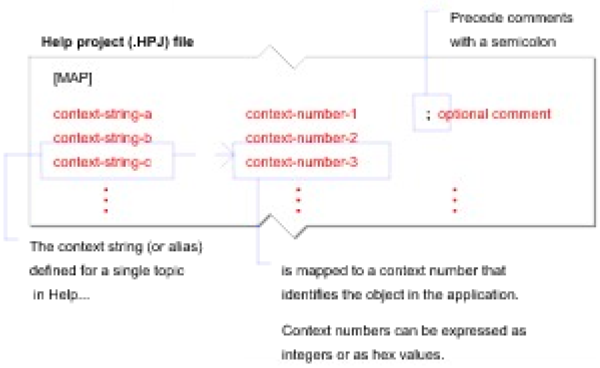
Example See Also Reference

If the application calls WinHelp with context numbers, you must map the context numbers to the context strings used in your Help system.

(See How Applications Are Programmed to Display Context-sensitive Help for a discussion of how the application calls WinHelp.)

To map context numbers to their associated context strings:

1. Open the project file with a text editor.
2. Create a [MAP] section if one does not already exist.
3. Write the context string (or alias) used to identify the Help topic for an object. Following the context string or alias, insert at least one space or tab.
4. Continuing on the same line, write the context number used by the application for the object mentioned in step 3.
5. OPTIONAL: Insert at least one space or tab, insert a semicolon (;), and add a comment. The comment cannot wrap to the next line. unless you add another semicolon to precede it.
6. Repeat steps 3 through 5 on a new line for every mapping. The following illustration shows this format:



Precede comments with a semicolon

The context string (or alias) defined for a single topic in Help...

is mapped to a context number that identifies the object in the application.

Context numbers can be expressed as integers or as hex values.

7. Save the file.

Using a Header File Instead of Adding Mappings to the [MAP] Section

Instead of adding all the mappings to the project file itself, you can put them all in a separate header file and include in the project file a reference to the header file. See Mapping Topic Identifiers in a Header File for more information.

Alternative Formats for the [MAP] Section

Click the "Example" hotspot under the title of this topic for examples of alternative formats you can use in the [MAP] section. (Click the Back button in the Examples topic to return to this one.)

See Also

About Context-sensitive Help

Implementing Context-sensitive Help - Overview

Reference

[MAP] section

Mapping Topic Identifiers in a Header File

See also Reference

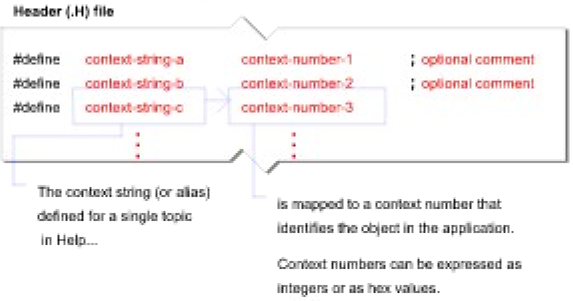
If the application does not call WinHelp with the same context strings as you use in your Help source files, you must use the [MAP] section of the project file to establish the connection between a user-interface object and the Help topic for that object.

(See How Applications Are Programmed to Display Context-sensitive Help for a discussion of how the application calls WinHelp.)

You can list all of the mappings in the [MAP] section, or you can list them all in a separate header file and include in the project file a reference to the header file.

To create a header file that maps context strings to context numbers:

1. Use a text editor to create a new text file. Give it any name (following DOS naming conventions), and use the filename extension "h," for example UI_Map.h.
2. Write `#define`, followed by at least one space or tab.
3. Continuing on the same line, type the context string used to identify the Help topic for an object, followed by at least one space or tab.
4. Continuing on the same line, write the context number used by the application to identify the object mentioned in step 3. You must obtain context numbers from whomever is programming the application.
5. Repeat this on a new line for every mapping. The following illustration shows this format:



Header (.H) file

```
#define     context-string-a     context-number-1     ; optional comment
#define     context-string-b     context-number-2     ; optional comment
#define     context-string-c     context-number-3
                   :                    :
```

The context string (or alias) defined for a single topic in Help...

is mapped to a context number that identifies the object in the application.

Context numbers can be expressed as integers or as hex values.

6. Save the file.
7. Add a reference to the header file in the project file, as described below.

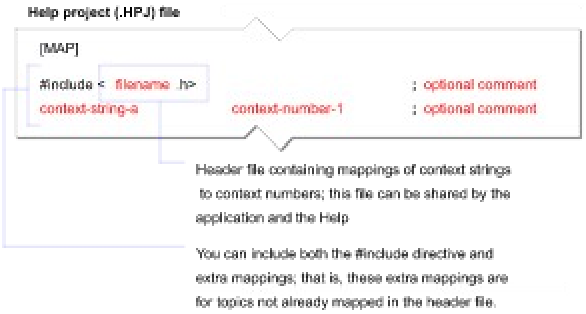
To reference the header file in the project file:

1. Open the Help project file with a text editor.
2. Create a [MAP] section if one does not already exist.
3. Write `#include`, followed by the name of the file you created in step 1 in the previous section ("To create a header file that maps context strings to context numbers:").

   Enclose the filename in angle brackets, for example:

   ```
   #include <UI_Map.h>
   ```

   The following illustration shows this format:



Help project (.HPJ) file

```
[MAP]

#include < filename .h>                          ; optional comment
context-string-a            context-number-1     ; optional comment
```

Header file containing mappings of context strings to context numbers; this file can be shared by the application and the Help

You can include both the #include directive and extra mappings; that is, these extra mappings are for topics not already mapped in the header file.

4. Save the file.

Sharing a Header File with the Application

In some cases, you may be able to share a header file with the application. See Sharing a Header File with the Application for more information.

See Also

About Context-sensitive Help

Implementing Context-sensitive Help - Overview

Reference

[MAP] section

Sharing a Header File with the Application

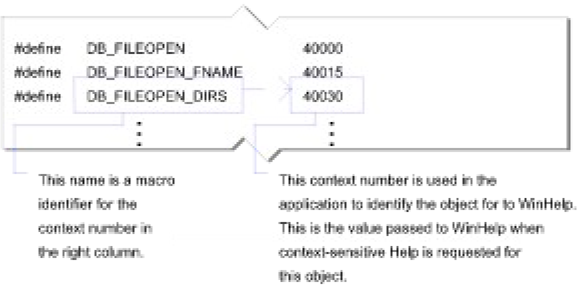
See also Reference

In a C program, it is possible to create a header file that assigns names to context numbers. The programmer can then use these names in the source code, instead of the numbers, which are not as easy to remember or debug. You may be able to use this header file to assign mappings in your Help system.

About #define Directives in the Application Source Code

The application's header file contains #define statements that assign names to Help context numbers (or, more precisely, create macro identifiers for the context numbers). For example,

Header (.H) file used by the application



```
#define    DB_FILEOPEN          40000
#define    DB_FILEOPEN_FNAME    40015
#define    DB_FILEOPEN_DIRS     40030
```

This name is a macro identifier for the context number in the right column.

This context number is used in the application to identify the object for to WinHelp. This is the value passed to WinHelp when context-sensitive Help is requested for this object.

These names are evaluated to the correct numbers when the program is compiled. Therefore, even though the programmer has created names for the Help identifiers, the application still passes the number, not the name, to WinHelp.

Note: Header files like this are not required in the application source code. Programmers use them for convenience: it is easier to remember and work with names than numbers. Also, the programmer can change the number once in the header file, without having to change all the references throughout the source code.

Sharing Context Names with the Application

Even though the application does not pass the names from the header file to WinHelp, you can take advantage of the header file to map the context numbers from the application to the context strings that are used in Help.

The header file used by the application is similar to the header file used by Help (see Mapping Topic Identifiers in a Header File). Therefore, in some cases, you can use the same header file (or a copy of the header file) to establish the mappings for your Help system. To do this, you must meet these requirements:

- The names used in the application must be identical to the context strings in the Help source file.
- The header file must list the actual context numbers. The numbers cannot be "offsets." For example:

This would work both for the application and for Help:

```
#define DB_FILEOPEN           40000
#define DB_FILEOPEN_FNAME     40015
#define DB_FILEOPEN_DIRS      40030
```

This may work for the application, but not for Help:

```
#define DB_FILEOPEN           ID(40000)
#define DB_FILEOPEN_FNAME           15
#define DB_FILEOPEN_DIRS            30
```

If the application source code uses offsets, you must replace them in your header file with the correct numbers.

If you are including comments in your header file, you must use a comment character that is compatible with both the programming language and WinHelp. For example, "//" is a valid comment character in C and in WinHelp.

See Also

About Context-sensitive Help

Implementing Context-sensitive Help - Overview

Reference

[MAP] section

Assigning Aliases for Context Strings

See also Reference

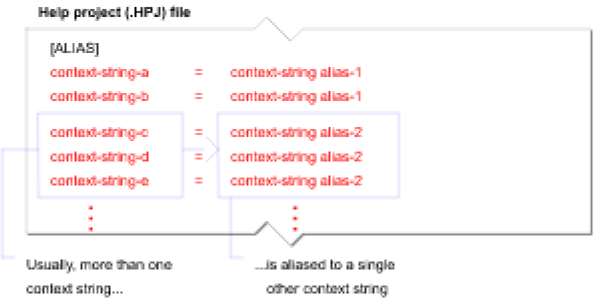
A context string cannot be assigned to more than one topic. However, there are times when you may want or need to use several different context strings to refer to the same topic. For example,

- You combine several topics into one, and you do not want to change all the references in your source files to the new context string.
- The host application is programmed to provide context-sensitive Help, and it is passing to your Help system more context identifiers than is appropriate for your Help design. For example, the application may be calling WinHelp with a unique identifier for every option in a dialog box, but you want to provide only one topic for the entire dialog box.

In cases such as these, you can use the [ALIAS] section of the project file to alias one context string to another. When a context string is aliased to another, WinHelp substitutes the second one for the original whenever that original is referenced in Help.

To assign aliases for context strings:

1. Open the project file with a text editor.
2. Create an [ALIAS] section if one does not already exist.
3. Write the context string you want to alias, followed by at least one space or tab.
4. Continuing on the same line, insert an equal sign (=), followed by at least one space or tab.
5. Continuing on the same line, write the context string to which you want to alias the first one (from step 3).
6. Repeat this on a new line for every context string you want to alias to another. The following illustration shows this format:


Help project (.HPJ) file
[ALIAS]
context-string-a = context-string alias-1
context-string-b = context-string alias-1
context-string-c = context-string alias-2
context-string-d = context-string alias-2
context-string-e = context-string alias-2
Usually, more than one context string...
...is aliased to a single other context string

7. Save the file.

Using a Header File Instead Adding Aliases to the [ALIAS] Section

Instead of adding all your alias statements to the project file, you can create them in a separate header file, and reference that header file from the project file. See Assigning Aliases in a Header File for more information.

See Also

About Context-sensitive Help

Implementing Context-sensitive Help - Overview

Reference
[ALIAS] section

Assigning Aliases in a Header File

See also Reference

You can alias one context string to another, so that when the first context string is referenced in Help, WinHelp substitutes the second one for the original. You can assign these aliases in the [ALIAS] section of the project file or you can assign them in a header file, as described below:

To create a header file for aliases:

1. Use a text editor to create a new text file. Give it any name (following DOS naming conventions), and use the filename extension "h," for example Aliases.h.
2. Add all the aliases, following the same format you would use to add them to the [ALIAS] section, as described in Assigning Aliases for Context Strings. Do not add the line with the heading [ALIAS].
3. Save the file.
4. Add a reference to the header file in the project file, as described below.

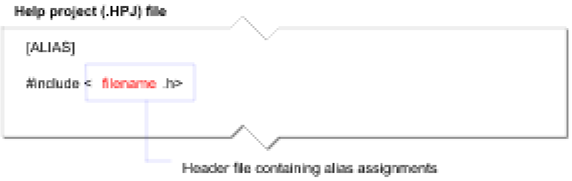
To reference the header file in the project file:

1. Open the Help project file with a text editor.
2. Create an [ALIAS] section if one does not already exist.
3. Write `#include`, followed by the name of the file you created in step 1 in the previous section, "To create a header file for aliases:"

   Enclose the filename in angle brackets, for example:

   ```
   #include <Aliases.h>
   ```

   The following illustration shows this format:



4.Save the file.

See Also

About Context-sensitive Help

Implementing Context-sensitive Help - Overview

Reference

[ALIAS] section

Alternative Formats for the Project File [MAP] Section - Examples

Using Integer Context Numbers

This is the simplest and most common way to map context strings to context numbers.

```
[MAP]                     ;using context numbers
;File menu
FileOpen           0020   ;File Open dialog box
FileSaveAs         0022   ;File SaveAs dialog box
;Help menu
HelpContents       0001   ;Contents topic
;Screen Regions
EditWindow         0077   ;How to use the Editor window
PreviewWindow      0079   ;How to use the Print Preview window
;SpeedBar buttons
FileOpenButton     30000  ;File Open button
HelpButton         30035  ;Help button
```

Using Hexadecimal Context Numbers

This is format is essentially the same as the one above, but uses numbers in hexadecimal format instead of integers.

```
[MAP]                      ;using context numbers
;File menu         0x10A
FileOpen           0x10E   ;File Open dialog box
FileSaveAs         0x10F   ;File SaveAs dialog box
;Help menu
HelpContents       0x0111  ;Contents topic
;Screen regions    0x0112
EditWindow         0x0113  ;How to use the Editor window
PreviewWindow      0x0114  ;How to use the Print Preview window
;SpeedBar buttons
FileOpenButton     0x0121  ;File Open button
HelpButton         0x0122  ;Help button
```

Using a #include (Header*) File and #define Statements

This is format is essentially the same as the one above, but uses the #define directive for each line in the file. This could be convenient if you are copying and pasting from the application's header file, which must use #defines.

```
[MAP]                           ;using #define statements.
#include hlpctxt.h              ;file with #defines and values
#define File menu       0x10A
#define FileOpen        0x10E   /*File Open dialog box */
#define FileSaveAs      0x10F   /*File SaveAs dialog box*/
#define Help menu       0x102
#define HelpContents    0x0111  /*Contents topic*/
#define ScreenRegions   0x0112
#define EditWindow      0x0113  /*How to use the Editor window*/
```

```
#define PreviewWindow   0x0114  /*How to use the Print Preview win*/
;SpeedBar buttons
#define FileOpenButton  0x0121  /*File Open button*/
#define HelpButton      0x0122  /*Help button*/
```

*The header file specifies the Help context names used by the application program and their corresponding values.

Working with Text Markers

WinHelp provides several macros that let you mark and unmark Help topics, jump to marked topics, or test whether a mark exists before executing a macro.

Text markers are different from bookmarks. They are used only in WinHelp macros and are transparent to the user viewing a Help file.

Using Text Markers to Test a Condition Before Executing a Macro

The primary purpose for text markers it to test whether a condition exists before executing a macro. You can test if a text marker exists or if it does not exist. The process involves two steps:

1. First, use the **SaveMark** macro to save a text marker upon executing a macro from one Help topic.
2. When you jump to another topic, use the **IfThen** or **IfThenElse** macros to test if the text marker exists before executing a macro.

You can construct conditional macros that

- Execute *Macro1* if a text marker exists.
- Execute *Macro1* if a text marker does not exist.
- Execute *Macro1* if a text marker exists; if it does not exist, execute *Macro2*.

Using Text Markers to Prevent Macro Execution Errors

Another reason to use text markers is to avoid errors. Because WinHelp is a hypertext system, often there is no way to know the sequence of macros that have been executed. You can use text markers to indicate that a condition exists because a macro has been executed.

For example, if you create a button in several Help topics, you may want to remove it (using the **DestroyButton** macro) in all others. Before removing the button, you should make sure that it has been created. Otherwise, WinHelp displays an error message if the macro cannot be executed. If you save a text marker when you create the button, in other Help topics you can test whether the marker exists before removing it.

Using Text Markers for Navigation Purposes

Text markers mark a location in a Help file and can be used like other navigation controls in WinHelp. Use the **GotoMark** macro to jump to a topic with a text marker.

For more details on using text markers, see the following topics:

Saving and Deleting Text Markers

Jumping to Topics with Text Markers

Using Text Markers to Test Conditions

Saving and Deleting Text Markers

Example See Also Reference About Using Macros

Use the **SaveMark** macro to save a text marker in a topic. Use the **DeleteMark** macro to remove a text marker.

Quick Reference

SaveMark("marker-text")

DeleteMark("marker-text")

Using SaveMark and DeleteMark to Identify Conditions

Use SaveMark and DeleteMark macros to identify the conditions used to execute conditional **IfThen** or **IfThenElse** macros. First, use the **SaveMark** macro to save a text marker upon executing a macro from one Help topic. When you jump to another topic, use the **IfThen** or **IfThenElse** macros to test if the text marker exists before executing a macro. If execution of a macro ends the condition, use DeleteMark to remove the text marker.

See Also

Jumping to Topics with Text Markers

Using Text Markers to Test Conditions

Reference

DeleteMark macro

IfThen macro

IfThenElse macro

SaveMark macro

Jumping to Topics with Text Markers

See Also Reference About Using Macros

Use the **GotoMark** macro to jump to a Help topic with a text marker.

Quick Reference

GotoMark("marker-text")

Note: WinHelp ignores the GotoMark macro if the specified text marker does not exist.

Examples

The following macro, which would follow a hotspot and be formatted as hidden text, jumps to a topic with a text marker named "glossary".

```
  ! GotoMark("glossary")
```

The following ! footnote uses the CreateButton macro to create a button that jumps to a topic with the text marker "glossary".

```
  ! CreateButton("btn_notepad", "&Glossary", "GotoMark(`glossary')")
```

See Also

Creating and Deleting Custom Buttons
Creating and Deleting Menu Items
Saving and Deleting Text Markers
Using Text Markers to Test Conditions

Reference

CreateButton macro

GotoMark macro

IfThen macro

IfThenElse macro

Using Text Markers to Test Conditions

Example See Also Reference About Using Macros

Use the **IfThen**, **IfThenElse**, **IsMark**, and **Not** macros to test whether a text marker exists before executing a macro. (Create text markers using the **SaveMark** macro.)

Quick Reference

IfThen(IsMark("marker-text"), "macro")

IfThenElse(IsMark("marker-text"), "macro1", "macro2")

IsMark("marker-text")

Not(IsMark("marker-text"))

About IfThen and IfThenElse

IfThen and IfThenElse are similar macros. IfThen executes a macro only if a text marker exists. IfThenElse executes one of two macros; it executes one macro if the text marker exists or another macro if the marker does not exist.

About IsMark and Not

The IsMark macro tests whether a text marker exists. The Not macro, used in conjunction with the IsMark macro, tests whether a text marker does not exist. Both IsMark and Not macros are used within IfThen and IfThenElse macros.

See Also

Jumping to Topics with Text Markers

Saving and Deleting Text Markers

Reference

IfThen macro

IfThenElse macro

IsMark macro

Not macro

SaveMark macro

Copying Help Text to the Clipboard

See Also   Reference   About Using Macros

Use the **CopyDialog** and **CopyTopic** macros to copy text from a Help topic to the Clipboard. (CopyDialog and CopyTopic do not copy graphics to the Clipboard.)

Quick Reference

CopyDialog()

CopyTopic()

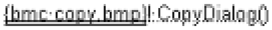
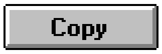
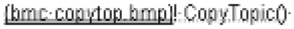
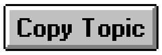
Copying Text from Secondary Windows and Pop-up Windows

You cannot use Winhelp's Edit|Copy command to copy text from secondary windows or pop-up windows to the Clipboard (secondary windows and pop-up windows do not have menus). However, the CopyDialog and CopyTopic macros let you sidestep this limitation of WinHelp.

Note: The CopyTopic macro is equivalent to pressing Ctrl+Ins from the main Help window.

Examples

The following two hotspot buttons execute the CopyDialog and CopyTopic macros.

How it looks in the .DOC file	How it looks in Help
 {bmc·copy.bmp}!·CopyDialog()	 Click to display the Copy dialog box
 {bmc·copytop.bmp}!·CopyTopic()·	 Click to copy all text in this topic to the Clipboard

See Also

About Different Kinds of Windows

About Hotspots

Creating a Hotspot that Executes a Macro or Macros

Printing a topic

Reference

CopyDialog macro

CopyTopic macro

Running Other Windows Applications from Help

See Also   Reference   About Using Macros

Use the **ExecProgram** macro to execute another Windows application from Help.

Quick Reference

ExecProgram("command-line",display-state)

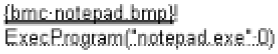
or abbreviate as: EP("command-line",display-state)

Examples

The following ! footnote uses the CreateButton macro to create a button that executes the Windows Notepad program:

```
  ! CreateButton("btn_notepad", "&Notepad", "ExecProgram(`notepad.exe', 0)")
```

The following hotspot button executes the Windows Notepad program.

How it looks in the .DOC file	How it looks in Help
 {bmc notepad.bmp}! ExecProgram(`notepad.exe',0)	 Click to run the Notepad program

See Also

Creating and Deleting Custom Buttons

Creating and Deleting Menu Items

Using Functions from an External Dynamic Link Library

Reference

CreateButton macro

ExecProgram macro

Using Functions from an External Dynamic Link Library (DLL)

See Also Reference About Using Macros

Use the **RegisterRoutine** macro to register a Dynamic Link Library function with WinHelp. Once the function is registered, you can use it the same way you use standard WinHelp macros.

Quick Reference

RegisterRoutine("DLL-name","function-name","format-spec")

or abbreviate as: RR("DLL-name","function-name","format-spec")

Using DLLs to Extend WinHelp

The RegisterRoutine macro lets you use DLL functions to extend the power of WinHelp. For example, any of the following tasks could be done by using functions contained in a DLL:

- add a full-text search engine
- print multiple Help topics at once
- display animation graphics
- use sound effects

If you want to use a DLL function throughout a Help file, you should register it in the [CONFIG] section of the Help project file.

To register a DLL:

1. Use a text editor to open the Help project file.
2. Add a RegisterRoutine macro to the [CONFIG] section. In the RegisterRoutine macro, specify the DLL, the function, and the format specifications for the function parameters.
3. Compile the Help file.

Examples

This macro in the [CONFIG] section of the Help project file registers a function called CHEKFILE in the DLL named TOOLSLIB.DLL. The function has three parameters, each with format S (for far char *)

```
[CONFIG]
RegisterRoutine("TOOLSLIB","CHEKFILE","SSS")
```

See Also

Creating an embedded window

Running Other Windows Applications from Help

Reference
[CONFIG] section

RegisterRoutine macro

Glossary

* A B C D E F G H I J K L
M N O P Q R S T U V W X Y Z

Symbols

! footnote

# footnote

#define directive

#include directive

$ footnote

* footnote

+ footnote

@ footnote

A

accelerator key

alias

[ALIAS] section

B

[BAGGAGE] section

[BITMAPS] section

bookmark

browse sequence

[BUILDTAGS] section

button bar

button binding

C

compound macro

[CONFIG] section

Contents topic

context number

context-sensitive

context string

D

double-underline formatting

Dynamic Link Library (DLL)

E - G

error log

[FILES] section

footnote

H

header file

Help compiler

Help resource file

hidden-text formatting

.HLP files

host application

hotspot

Hotspot Editor

hypergraphics

hypertext

I - L

input focus

item binding

K footnote

keyword

M

macro

[MAP] section

Multiple-Resolution Bitmap Compiler (MRBC)

N

navigation controls

nonscrolling region

nonwrapping text

O

[OPTIONS] section

P

pop-up window

primary window

project file (or Help project file)

Q - R

Rich Text Format (RTF)

S

secondary window

.SHG files

single-underline formatting

T

text marker

title

topic

U - Z

[WINDOWS] section

WINHELP.EXE

WINHELP.HLP

! footnote

Footnote used to execute a macro when you display a Help topic. You can reference more than one macro by separating each macro with a semicolon.

To enter a ! footnote in Word for Windows, choose Insert|Footnote, and enter ! in the Custom Footnote Mark box.

# **footnote**

Footnote used to identify the context string for a Help topic. Context string footnotes may contain letters, numbers, periods, and underscore characters. Context strings are not case sensitive.

To enter a # footnote in Word for Windows, choose Insert|Footnote, and enter # in the Custom Footnote Mark box.

#define directive

A C directive used in programming. A file containing the #define statements is used both by the application and the Help compiler. The #define directive assigns the given *value* to the specified *name*. All subsequent occurrences of the name are replaced by the value.

Parameter	Description
name	Specifies the name to be defined. This value is any combination of letters, digits, and punctuation.
value	Specifies any integer, character string, or line of text.

#include directive

A C directive used in programming. Use the #include directive in the Help project file to include text from another file (such as a header file that lists context strings and context numbers).

The syntax for the #include directive is

#include <*filename*>

filename must be enclosed within angle brackets.

$ footnote

Footnote used to identify the title of a Help topic. Topic titles appear when you use the Search dialog box, the History list, or the Bookmark menu.

To enter a $ footnote in Word for Windows, choose Insert|Footnote, and enter $ in the Custom Footnote Mark box.

*** footnote**

Footnote used to specify build tags for a Help topic.

To enter a * footnote in Word for Windows, choose Insert|Footnote, and enter * in the Custom Footnote Mark box.

+ footnote

Footnote used to set the browse sequence for a Help topic.

To enter a + footnote in Word for Windows, choose Insert|Footnote, and enter + in the Custom Footnote Mark box.

@ footnote

Footnote used to enter comments for a Help topic.

To enter a @ footnote in Word for Windows, choose Insert|Footnote, and enter @ in the Custom Footnote Mark box.

accelerator key

A keyboard shortcut for displaying a menu, executing a command, clicking a button, or running a macro. To assign accelerator keys, place an ampersand (&) before the character you want to use as a shortcut key when entering the name of the menu, menu item, or button. To assign accelerator keys for macros, use the AddAccelerator macro.

alias

Context string to which other context strings are equated, so that when the other context strings are referenced in Help, the context-string assigned as an alias is used instead. For example, the context strings for all the options in a dialog box can be aliased to the context string for the dialog box. When Help is requested for any of the options, the Help topic for the dialog box is displayed instead.

Assign aliases in the [ALIAS] section of the Help project file.

[ALIAS] section
Optional section of the Help project file that allows you to reference a Help topic using more than one context string. The [ALIAS] section can save you much time in recoding context strings.

[BAGGAGE] section

Optional section of the Help project file that lists files (such as bitmaps or multimedia files) that you want included in the Help file. The advantage of including these files in the .HLP file is improved performance; the disadvantage is a larger .HLP file. You can list up to 1,000 files in the [BAGGAGE] section.

To reference a [BAGGAGE] file in a topic, precede the file with an exclamation point (!).

[BITMAPS] section

Section of the Help project file that lists the bitmap files included in the Help file. This section is required, unless you specifed a path for bitmap files using the BMROOT or ROOT options in the [OPTIONS] section.

bookmark

A marked location for a Help topic. You define bookmarks in WinHelp using the Bookmark menu. Bookmarks let you mark several topics in a Help file for later reference.

browse sequence

A navigational aid that establishes a linear path through a group of Help topics. There can be multiple browse sequences in a Help file, but each Help topic can belong to only one sequence. To add browse sequences, add a + footnote at the beginning of the topic. To display browse buttons in the WinHelp button bar, add the BrowseButtons() macro to the [CONFIG] section of the Help project file.

[BUILDTAGS] section

Optional section of the Help project file that lets you conditionally compile a Help file, or build different versions of a Help file. You can list up to 30 build tags in the [BUILDTAGS] section. Build tags are not case sensitive and cannot contain spaces.

button bar

A row of buttons located along the top of the Help window below the menu bar. The Help buttons let you navigate easily through a Help file. Standard Help buttons include Contents, Search, Back, History, << (Previous), and >> (Next). You can customize the button bar and create your own buttons.

button binding

The function of a button in the WinHelp button bar. The button can be one of the standard Help buttons or one you created using the CreateButton macro. To change the function of a WinHelp button, use the ChangeButtonBinding macro.

compound macro

A string of two or more macros separated by semicolons. A compound macro can be used in association with a hotspot or in an exclamation point (!) footnote. The macros in a compound macro are executed one after another.

[CONFIG] section

Optional section of the Help project file that specifies Help macros to run when the Help file is opened. For example, you might include macros in the [CONFIG] section to create buttons, menus, or menu commands that are not standard in WinHelp.

Contents topic

The first topic displayed when a user opens a Help file. The Contents topic is similar to a table of contents in a book, and serves as an important navigational tool for the Help file. The Help button bar has a Contents button, which displays the Contents topic. In the [OPTIONS] section of the Help project file, use the Contents option to specify the context string of the Contents topic.

context number

A unique number that is assigned to a context string. The application can then pass the number to WinHelp to display context-sensitive Help.

You assign context numbers to context strings in the [MAP] section of the Help project file. There can be only one context number assigned to each context string or alias.

context sensitive

Pertaining to the current state of a program or a selected window, menu, command, dialog, or object onscreen. Context-sensitive Help is usually available by pressing the F1 key or clicking a Help button.

context string

A unique string that WinHelp uses to identify a Help topic. Context strings can consist of any combination of letters, numbers, periods, and underscore characters, but they cannot contain spaces. A context string must be unique, although the same context string can be used in more than one Help file, even if the files are linked through jumps. Create the context string for a topic by adding it as the topic's # footnote.

double-underline formatting

Character formatting that creates a hotspot link to another topic, or for a macro. Double-underline text must be followed by a context string or macro formatted as hidden or strike-out text. After you compile the Help file, the double-underline text appears as green, underline text in the Help window. When you click the underline text, Help jumps to another topic.

To format double-underline text in Word for Windows, select the text and press Ctrl+D, or choose Format|Character, then choose Double in the Underline combo box.

Dynamic Link Library (DLL)

A library of programming code that is separate from an application and loaded or unloaded into memory as needed. DLLs are created using a compiled programming language such as C. You can expand the functionality of Help by registering a DLL function using the RegisterRoutine macro.

error log
A text file to which errors are written during the process of compiling the Help file. Use this file to help debug your Help system. To create an error log file, include the ERRORLOG option in the [OPTIONS] section of the Help project file.

[FILES] section

Required section of the Help project file that specifies the .RTF files included in the Help file.

If your Help file includes many .RTF files, you can list them in a separate header file and reference the file using the #include directive.

footnote

Footnotes serve as control codes in the Help topic files.

Footnote	Purpose
#	Context string
$	Title
+	Browse sequence
K	Keyword
*	Build tag
!	Macro reference
@	Comment

You can also create custom footnotes with the Multikey option in the Help project file.

header file

A text file referenced using an #include directive in the Help project file. A header file can be used to map Help topics to user interface objects in the host product or to use aliases (instead of actual context strings) to reference a Help topic. Header files are often used to make the project file smaller and easier to maintain.

A header file is sometimes called an "include file" because the #include directive is used to reference it.

Help compiler
A program that turns the source .RTF files and graphics files into a binary data file that can be read by WINHELP.EXE. There are several versions of the Microsoft Windows Help compiler:

Compiler	Description
HC30.EXE	Windows 3.0 compiler
HC31.EXE	Windows 3.1 compiler
HCP.EXE	Windows 3.1 protected mode compiler (makes use of extended and virtual memory)

Help resource file

A binary file (with a filename extension .HLP) created by the Windows Help compiler that can be read by the Windows Help program (WINHELP.EXE).

hidden-text formatting

Character formatting that designates a context string or macro. Hidden text context strings appear immediately after a hotspot (single-underlined or double-underlined text). In some word processing programs, you can also use strike-out text to designate a context string.

To format hidden text in Word for Windows, select the text and press Ctrl+H, or choose Format| Character and check Hidden in the Style box.

.HLP files

Binary resource files created by the Windows Help compiler that can be read by the Windows Help program (WINHELP.EXE).

host application

The application for which the Help system is created. A host application and its Help system can be programmed so that when an object in the application has input focus, and the user requests help, context-sensitive Help is displayed for that object.

hotspot

A text or graphic that the user can click to initiate some action. A hotspot can initiate a jump to another topic, display a topic in a different window, or execute a macro or macros.

Hotspot Editor

A utility that lets you define multiple hotspots in one graphic file. The Hotspot Editor (SHED.EXE) can read bitmaps (monochrome or 16-color) or Windows metafiles. It creates files with a .SHG file extension.

hypergraphics
A graphic file with multiple hotspots. You can define multiple hotspots in a graphic using the Hotspot Editor.

hypertext
An online database of interlinked topics that provides multiple methods of information access and navigational controls. WinHelp is a form of hypertext.

input focus

The object in the user interface of an application that will receive the next input (if no other object is given focus before the input is generated). Input focus can be indicated by

- text insertion point in a text field
- highlighted bar of a menu
- dotted box around certain controls
- highlighted title bar of a window

item binding

The function of a WinHelp menu item. You can use the ChangeItemBinding macro to change the function of a menu item you have added using the InsertItem or AppendItem macros. You cannot change the function of other standard WinHelp menu items such as File|Open or Edit|Copy.

K footnote

Footnote used to identify the keywords of a Help topic. The complete list of keywords for a Help file appear in the upper portion of the Search dialog box.

To enter a K footnote in Word for Windows, choose Insert|Footnote, and enter K in the Custom Footnote Mark box.

keyword

A significant word or phrase that indicates the content of a Help topic. Keywords are similar to index entries. The complete keyword list for a Help file appears in the upper section of the Search dialog box.

In Word for Windows, you enter keywords for a Help topic by inserting a K footnote at the beginning of the topic.

macro

WinHelp commands that allow you to customize Help menus, commands, and the button bar; jump to topics; display pop-up topics; manage and position windows; register DLLs or run other executable programs; and add keyboard shortcuts.

[MAP] section

Optional section of the Help project file that associates context strings or aliases with context numbers (decimal or hexadecimal). The context number is a number the application passes to display a Help topic. There can be only one context number assigned to each context string or alias.

Multiple-Resolution Bitmap Compiler (MRBC)

A utiltity that combines bitmaps of different resolutions into one graphics file that can be used by WinHelp. The MRBC eliminates sizing problems and color distortion that result from creating a graphic file using one video display resolution (such as VGA) and displaying it on another (such as EGA).

navigation controls

Buttons and other controls that let you move easily through a Help file. The common navigation controls available in most Help files are Contents, Search, Back, History, << (Previous), and >> (Next).

nonscrolling region

A region at the top of a Help topic (below the button bar) that remains visible when you scroll. Nonscrolling regions allow you to display the topic title at all times. You can also include graphics and hotspot jumps in the nonscrolling region.

nonwrapping text

Text that is not automatically reformatted to place words on the next line when you resize the Help window. If nonwrapping text does not fit horizontally in the Help window, a horizontal scroll bar appears at the bottom of the window. Tables are often nonwrapping.

To format nonwrapping text in Word for Windows, choose Format|Paragraph and check Keep Lines Together (in the Pagination group box).

[OPTIONS] section

Optional section of the Help project file that specifies such options as the root directories for bitmaps and .RTF files, the title for the Help window, the Contents topic, the topics included in the Help file, the copyright message used in the About dialog box, the type of compression used during compiling, and other options. The [OPTIONS] section should appear first in your Help project file.

pop-up window

A separate window from the primary Help window that displays a pop-up topic. A pop-up hotspot in the Help window appears as green, dotted-underline text. Click a pop-up hotspot to display a pop-up window. To close a pop-up window, press Esc or click anywhere onscreen.

primary window

The main Help window. Pop-up windows and secondary windows appear onscreen in addition to the primary Help window. When you jump from one topic to another, the topic you are jumping to replaces the current topic in the primary window. You can define the name, size, and position of the primary window in the [WINDOWS] section of the Help project file.

project file (or Help project file)

A text file with an .HPJ file extension that contains information necessary to compile a Help file. A project file can consist of up to nine sections, each enclosed in brackets ([]). See the topic Help Project File in the Reference section of this Help system for more details.

Rich Text Format (RTF)

A type of file format that provides a variety of formatting capabilities for creating footnotes, and single-underline, double-underline, and hidden characters. The Help compiler requires source files in Rich Text Format.

secondary window

A window that displays in addition to the primary Help window. To display a Help topic in a secondary window, specify the name of the window along with the context string of a hotspot jump. The name, size, and position of the secondary window must be defined in the [WINDOWS] section of the Help project file. WinHelp menus and the button bar are not available in secondary windows, and topics that appear in secondary windows are not recorded in the History list.

.SHG files
Segmented hypergraphics files created with the Hotspot Editor.

single-underline formatting

Character formatting that creates a link to a pop-up topic. Single-underline text must be followed by a context string formatted as hidden or strike-out text. After you compile the Help file, the single-underline text appears as green, dotted-underline text in the Help window. When you click the dotted-underline text, another topic appears in a pop-up window.

To format single-underline text in Word for Windows, select the text and press Ctrl+U, or choose Format|Character, then choose Single in the Underline combo box.

text marker

A placeholder in a Help topic created using the SaveMark macro. Text markers are used within WinHelp macros to mark and unmark Help topics, jump to marked topics, or test whether a mark exists before executing a macro.

title
The name of a Help topic, which usually appears on the first line. The $ footnote specifies the title that displays in the History list, the Search dialog box, and the Bookmark menu.

topic
A discrete chunk of information within a Help file. Usually, a Help file consists of many Help topics. In a hypertext Help system, each topic provides several navigational controls for jumping from one topic to another. Each topic in the Help source files is separated from another topic by a hard page break.

[WINDOWS] section
Section of the Help project file that defines the placement, size, and color of the primary Help window and any secondary Help windows.

WINHELP.EXE
The Windows Help program. WINHELP.EXE reads .HLP files created by the Windows Help compiler.

WINHELP.HLP

The Help file for WINHELP.EXE. By default, the How to Use Help command in the Help menu opens WINHELP.HLP. To change the Help file that the How to Use Help command opens, use the SetHelpOnFile macro.

Macro Reference, Categorical Lists

Macro Reference, Alphabetical

Click the name of the macro to jump to the reference topic for that macro.

Button macros

Back	Displays previous topic.
BrowseButtons	Adds Browse buttons.
ChangeButtonBinding	Changes the assigned function of a Help button.
Contents	Displays the Contents topic in 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.
History	Displays the history list.
Next	Displays next topic in browse sequence.
Prev	Displays previous topic in browse sequence.
Search	Displays Search dialog box.
SetContents	Designates a specific topic as the Contents topic.

Keyboard macros

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

Linking macros

JumpContents	Jumps to the contents topic for the active file.
JumpContext	Jumps to a topic with the specified context number.
JumpHelpOn	Jumps to the Contents of the How to Use Help file.
JumpId	Jumps to the specified topic in the specified Help file.
JumpKeyword	Jumps to the first topic containing the specified keyword.
PopupContext	Displays the topic with the specified context number in a pop-up window.
PopupId	Displays the topic with the specified context id in a pop-up window.

Menu macros

About	Displays the About Windows Help dialog box.
Annotate	Displays the Annotate dialog box.
AppendItem	Appends a menu item to the end of a custom menu.
BookmarkDefine	Displays Bookmark Define dialog box
BookmarkMore	Displays the Bookmark dialog box.
ChangeItemBinding	Changes the assigned function of a menu binding.
CheckItem	Displays a check mark next to a menu item.
CopyDialog	Displays the Copy dialog box.
CopyTopic	Copies all text in the current topic to the Clipboard. (Use

	the CopyDialog macro to show what is in the Clipboard.)
DeleteItem	Removes a menu item from a menu.
DisableItem	Disables a menu item.
EnableItem	Enables a menu item.
Exit	Exits Windows Help.
FileOpen	Displays the Open dialog box.
HelpOn	Displays the Help file for the Windows Help application.
InsertItem	Inserts a menu item at a given position on a menu.
InsertMenu	Adds a new menu item to the Help menu bar.
Print	Prints the current topic (from the File menu).
PrinterSetup	Displays the Printer Setup dialog box from the File menu.
SetHelpOnFile	Specifies a custom How to Use Help file.
UncheckItem	Removes a check mark from a menu item.

Program macros

ExecProgram	Starts an application from within WinHelp.
RegisterRoutine	Registers a function within a DLL as a Help macro.

Text-marker macros

DeleteMark	Removes a marker added by SaveMark.
GotoMark	Executes a jump to a marker set by SaveMark.
IfThen	Executes a Help macro if a given marker exists.
IfThenElse	Executes one of two Help macros if a given marker exists.
IsMark	Tests whether a marker set by SaveMark exists.
Not	Reverses the result set by IsMark.
SaveMark	Saves a marker for the current topic and Help file.

Window macros

CloseWindow	Closes the main or secondary Help window.
FocusWindow	Changes the focus to a specific Help window.
HelpOnTop	Toggles the on-top state of Windows Help.
PositionWindow	Sets the size and position of a Help window.

Sample Topic x_topic1

This is a sample topic. To return to the previous topic, click the Back button.

This topic exists in the Help file CWH.HLP. Its context string is x_topic1.

Sample Topic x_topic2

This is a sample topic. To return to the previous topic, click anywhere or press any key.

This topic exists in the Help file HELPEX.HLP. Its context string is x_topic2.

Sample Topic x_topic3

This is a sample pop-up topic. To return to the previous topic, click anywhere or press any key.

This topic exists in the Help file HELPEX.HLP. Its context string is x_topic3, and its context number is 0003

Sample Topic xs_topic1

This is a sample topic. To return to the previous topic, click the Close button.

This topic exists in the Help file HELPEX.HLP. Its context string is xs_topic1.

Sample Topic xs_topic2

This is a sample topic. To return to the previous topic, click the Close button.

This topic exists in the Help file CWH.HLP. Its context string is xs_topic2.

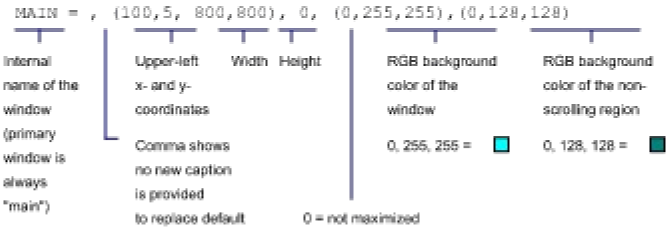
Sample Primary Window

This is a sample topic in a new primary window. To return to the previous topic, click the Back button.

This topic exists in the Help file HELPEX.HLP. Its context string is xpw_topic1.

How This Window is Defined

This primary window is defined in the [WINDOWS] section of the project file as follows:



```
MAIN = , (100,5, 800,800), 0, (0,255,255),(0,128,128)
```

Internal name of the window (primary window is always "main")
Comma shows no new caption is provided to replace default
Upper-left x- and y-coordinates
Width
Height
0 = not maximized
RGB background color of the window
0, 255, 255 = 🟦
RGB background color of the non-scrolling region
0, 128, 128 = 🟩

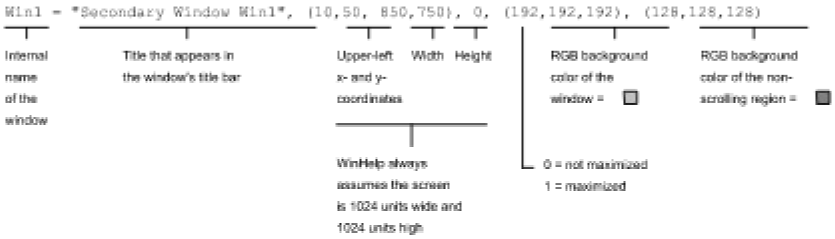
Sample Secondary Window 1

Click here to close this window and return to the primary window.

This topic exists in the Help file CWH.HLP. Its context string is xsw_topic1.

How This Window is Defined

This secondary window is defined in the [WINDOWS] section of the project file as follows:



```
Win1 = "Secondary Window Win1", (10,50, 850,750), 0, (192,192,192), (128,128,128)
```

Internal name of the window
Title that appears in the window's title bar
Upper-left x- and y-coordinates
Width
Height
WinHelp always assumes the screen is 1024 units wide and 1024 units high
0 = not maximized
1 = maximized
RGB background color of the window = □
RGB background color of the non-scrolling region = ■

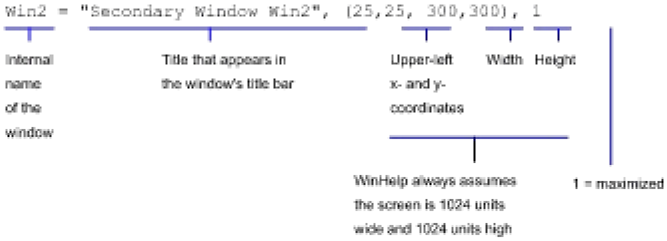
Sample Secondary Window 2

Click here to close this window and return to the primary window.

This topic exists in the Help file HELPEX.HLP. Its context string is xsw_topic2.

How This Window is Defined

This secondary window is defined in the [WINDOWS] section of the project file as follows:



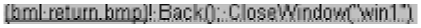
```
Win2 = "Secondary Window Win2", (25,25, 300,300), 1
```

Internal name of the window
Title that appears in the window's title bar
Upper-left x- and y-coordinates
Width
Height
WinHelp always assumes the screen is 1024 units wide and 1024 units high
1 = maximized

Example - Using the CloseWindow Macro to Close a Window

Click here to close this window and return to the primary window.

The button above is defined as follows.



```
{bml return.bmp}! Back(); CloseWindow("win1")
```

where:

Back redisplays the last topic viewed.

CloseWindow closes the secondary window.

"win1" is the name of the window as defined in the [WINDOWS] section of the project file.