Table of Contents

Close

Introduction

Up a Level

Chapter 1 The Windows Help Application

Up a Level

Chapter 2 Getting Started with Help Authoring

Up a Level

Chapter 3 Designing Your Help System

Up a Level

Chapter 4 Help Authoring Guidelines

Chapter 5 Using Help Author

Chapter 6 Creating Topics

Chapter 7 Formatting Topics

Chapter 8 Creating Links and Hot Spots

Chapter 9 Defining Topic Windows

Chapter 10 Adding Graphics

Chapter 11 Creating Hypergraphics

Chapter 12 Creating Graphics for Different Displays

Chapter 13 Customizing the Help File

Chapter 14 Help Macros

Chapter 15 Help Macro Reference

Chapter 16 The Help Project File

Chapter 17 Building the Help File

Chapter 18 Help Error Messages

Chapter 19 The WinHelp API

Chapter 20 Writing DLLs for Windows Help

Appendix A Windows Virtual-Key Codes

Appendix B Help RTF Statements

Appendix C Baggage Access Functions

Chapter 1 The Windows Help Application

The Main Help Window

Chapter 1 The Windows Help Application

Windows Help Menus

Chapter 1 The Windows Help Application

The Windows Help Button Bar

Chapter 1 The Windows Help Application

Customizing the Help Application

Chapter 1 The Windows Help Application

The Mouse and the Keyboard Interfaces

Chapter 2 Getting Started with Help Authoring

Windows Help Files

Chapter 2 Getting Started with Help Authoring

Developing Help Files

Chapter 2 Getting Started with Help Authoring

Building a Simple Help File

Chapter 3 Designing Your Help System

Designing for People

Chapter 3 Designing Your Help System

Design Issues

Chapter 4 Help Authoring Guidelines

Help Organization

Chapter 4 Help Authoring Guidelines

Formatting and Style

Chapter 5 Using Help Author

Installing Help Author

Chapter 5 Using Help Author

Getting Started

Getting Started
Getting Help

Chapter 5 Using Help Author

Windows Help Project Editor

Chapter 5 Using Help Author

The Windows Help Authoring Templates

Chapter 6 Creating Topics

Getting Started

Chapter 6 Creating Topics

Creating New Topics

Chapter 6 Creating Topics

Editing Topics

Assigning Context Strings

Chapter 6 Creating Topics

Inserting Topic Titles

Assigning Keywords

Specifying Topic-Entry Macros

Specifying Browse Sequences

Assigning Build Tags

Inserting Topic Comments

Chapter 7 Formatting Topics

Understanding Help Formatting

Chapter 7 Formatting Topics

Creating Tables

Chapter 8 Creating Links and Hot Spots

Linking Topics Using Jumps

Chapter 8 Creating Links and Hot Spots

Creating Hot Spots

Chapter 8 Creating Links and Hot Spots

Inserting Standard Jump Hot Spots

Creating Pop-Up Hot Spots

Chapter 8 Creating Links and Hot Spots

Creating Macro Hot Spots

Chapter 8 Creating Links and Hot Spots

Creating Links to Online Tutorials

Chapter 8 Creating Links and Hot Spots

Changing the Standard Appearance of Hot Spots

Chapter 9 Defining Topic Windows

The Main Help Window

Chapter 9 Defining Topic Windows

Secondary Windows

Defining Nonscrolling Regions

Chapter 9 Defining Topic Windows

Pop-Up Windows

Chapter 9 Defining Topic Windows

Embedded Windows

Chapter 10 Adding Graphics

Creating Pictures

Chapter 10 Adding Graphics

Placing Graphics in Topic Files

Chapter 11 Creating Hypergraphics

Hotspot Editor

Chapter 12 Creating Graphics for Different Displays

Multi-Resolution Bitmap Compiler

Chapter 13 Customizing the Help File

Defining Menus and Buttons

Chapter 13 Customizing the Help File

Defining Buttons

Chapter 13 Customizing the Help File

Creating a Custom How To Use Help File

Chapter 14 Help Macros

Executing Help Macros

Chapter 14 Help Macros

Constructing Help Macros

Chapter 14 Help Macros

Using DLL Calls as Help Macros

Chapter 14 Help Macros

Macro Quick Reference

Chapter 15 Help Macro Reference

Macro Reference

Chapter 16 The Help Project File

Project-File Features

Chapter 16 The Help Project File

Help Project File Reference

Chapter 17 Building the Help File

Building a Help File

Chapter 17 Building the Help File

Building Help Projects with Multiple Files

Chapter 17 Building the Help File

Debugging the Help File

Chapter 17 Building the Help File

Testing the Built Help File

Chapter 18 Help Error Messages

Help Compiler 3.1 Error Messages

Chapter 18 Help Error Messages

Help Compiler 3.0 Error Messages

Choosing Help from the Help Menu

Chapter 20 Writing DLLs for Windows Help

Creating Custom Help Macros

Chapter 20 Writing DLLs for Windows Help

Calling Windows Help Internal Functions

Chapter 20 Writing DLLs for Windows Help

Writing DLLs for Embedded Windows

Appendix B Help RTF Statements

RTF Semantics

Appendix B Help RTF Statements

The RTF File

Appendix B Help RTF Statements

Overview of Help RTF Statement Support

Appendix B Help RTF Statements

Help RTF Statement Reference

\pict
\picw
\picwgoal
\plain
\qc
\qj
\ql
\qr
\ri
\row
\rtf
\sa
\sb
\scaps
\sect
\sl
\strike
\tab
\tb
\tqc
\tqr
\trgaph
\trleft
\trowd
\trqc
\trql
\tx
\ul
\uldb
\v
\wbitmap
\wbmbitspixel
\wbmplanes
\wbmwidthbytes
\windows
\wmetafile

Introduction

The Microsoft® Windows™ Help application and compiler let you provide online Help to users of your Windows-based application. Microsoft Windows Help version 3.1 supports hypertext and context-sensitive links and integrates many advanced hypertext capabilities, making Windows Help an optimal tool for developing small Help systems and large, full-featured online documents.

See Also

About This Guide

About This Guide

The Microsoft Windows Help Authoring Guide provides information for writers and developers of Help systems for Microsoft Windows-based applications. This section introduces the Windows Help environment and provides background information you should review before developing a Help file.

This guide is written for anyone who is creating a Help file, whether that person is a technical writer or a developer of applications for Windows. The word you as used in this guide refers either to the Help author or to the developer creating the Help system. The term user, on the other hand, refers to the person who will eventually use the Help files you create.

The Microsoft Windows Help Authoring Guide contains the following chapters:

- Chapter 1, The Windows Help Application, explains the basic features of the Windows Help application that displays Help files.
- Chapter 2, Getting Started with Help Authoring, explains the basic process for creating and building Help files. In this chapter youll create a simple Help file to learn the basics of Help authoring.
- Chapter 3, Designing the Help System, discusses issues and tradeoffs to consider when designing your Help system.
- Chapter 4, Help Authoring Guidelines, provides a set of general and specific guidelines you can follow as you create topics.
- Chapter 5, Using Help Author, explains Microsoft Help Author, a tool that makes creating Help files simpler and easier. The first part of the chapter explains how to use the Help Project Editor to create Help files. The second part explains how the Windows Help Authoring Templates customize Microsoft Word for Windows to make creating topic files easier.
- Chapter 6, Creating Topics, explains how to create Help topic files and how to add the Help-specific information that the Help compiler uses to identify topics, create a keyword index, organize topics into browse sequences, and run macros.
- Chapter 7, Formatting Topics, explains how to format the text and graphics that you include in topic files. The chapter describes various formatting techniques, such as selecting fonts, creating tables, creating margins, and using special characters.
- Chapter 8, Creating Links and Hot Spots, explains how to create links between topics in the Help system and how to create hot spots that run Help macros.
- Chapter 9, Defining Topic Windows, explains how to display topics in different Help windows, such as pop-up, secondary , and embedded windows. It also explains how to add nonscrolling regions to topic windows.
- Chapter 10, Adding Graphics, explains how to add pictures to topic files, how to control the layout of text and graphics within a topic, and how to display information activated from hot graphics.
- Chapter 11, Creating Hypergraphics, describes hypergraphics and explains how to use Hotspot Editor to create graphics with multiple hot spots.
- Chapter 12, Creating Graphics for Different Displays, explains how to use the Multi-Resolution Bitmap Compiler to combine graphics with different display resolutions into a single file format so that they will look good on different machines.
- Chapter 13, Customizing the Help File, explains how to use Help features to change the way your Help file appears and how users work with it. The first part of the chapter discusses ways to customize Help menus and buttons. The second part shows other ways to change Help features.
- Chapter 14, Help Macros, describes Help macros and explains the rules for constructing macros and using them in the Help file.
- Chapter 15, Help Macro Reference, contains a list of all the Help macros you can use to customize your Help file or the Windows Help feature set.
- Chapter 16, The Help Project File, describes the format and contents of the Help project file, which defines how a Help file is built, and explains how to use it to customize Windows Help for your Help

file.

- Chapter 17, Building the Help File, explains how to build a Help file, how to fix problems that arise during the build process, and how to display a Help file after it has been built.
- Chapter 18, Help Error Messages, explains each of the Help error messages that you might encounter when building a Help file with the Help compiler version 3.1 or 3.0.
- Chapter 19, The WinHelp API, explains the WinHelp API. It describes how to create context-sensitive links within the application and how to call WinHelp to perform Help-related operations.
- Chapter 20, Writing DLLs for Windows Help, explains how to write a dynamic-link library (DLL) for Windows that extends the functionality of Windows Help. This chapter shows how an application can use custom DLLs to provide additional functionality to Help authors or to control the behavior of topic elements placed in embedded windows.
- Appendix A, Windows Virtual-Key Codes, shows the symbolic constant names, hexadecimal values, and keyboard equivalents for the virtual-key codes used by Microsoft Windows version 3.1. You use these virtual-key codes to provide keyboard access for Help macros within the Help file.
- Appendix B, Help RTF Statements, describes the syntax and purpose of rich text format (RTF) statements supported by the Microsoft Help compiler. The RTF statements define the formats used to encode Help features in the source topic files.
- Appendix C, Baggage Access Functions, provides specialized source code that can be built into an application or custom DLL so that it can retrieve the appropriate data file from the Help files [BAGGAGE] section.

About Windows Help

Windows Help lets users view Help files and other online documents in the Microsoft Windows graphical environment. Windows Help files present online information using the following elements:

- Text, with multiple fonts, type sizes, and colors
- Graphics (in several formats and using as many as 16 colors)
- Segmented hypergraphics (bitmaps with embedded hot spots)
- Cross-reference jumps for linking information
- Pop-up windows for presenting additional text and graphics in a nonintrusive way
- Secondary windows for presenting information in a controlled format
- Keyword search capability for finding specific information

Windows Help provides a practical way to combine different types of information into a format users can access easily from Windows and from Windows-based applications. Use Windows Help to:

- Create Help files for a Windows-based application.
- Supplement or replace printed product documentation.
- Create stand-alone online documents.
- Convert hard-to-access printed manuals into electronic format.

> Note: Even though you can use the Windows Help tools to create different types of electronic documents, this guide refers to all created files as Help files. The information, however, applies equally to these other documents.

What's New in Version 3.1?

Windows Help maintains upward compatibilityHelp files built with Help version 3.0 are compatible with Help version 3.1, but Help files built with the version 3.1 Help compiler will not work in version 3.0 Help applications.

The basic process for developing Help files remains the same; however, the number of features has increased dramatically. You can do everything you did using version 3.0and much more. Some of the new features in version 3.1 of Windows Help are:

- You can create new menus and menu items or modify existing menus and menu items. See Chapter 14, Help Macros, and Chapter 16, The Help Project File.
- The Help button bar has a new look and a new button called History. You can add your own buttons to the standard buttons, or you can modify the function of existing buttons. See Chapter 13, Customizing the Help File, Chapter 14, Help Macros, and Chapter 16, The Help Project File.
- The Copy command has a dialog box that lets users select the exact text they want to copy to the Clipboard. See Chapter 1, The Windows Help Application.
- In addition to the main Help window, you can define secondary windows that also display Help information. See Chapter 9, Defining Topic Windows, and Chapter 16, The Help Project File.
- You can control the size, placement, and background color of all Help windows, both main and secondary. See Chapter 13, Customizing the Help File, and Chapter 16, The Help Project File.
- Help windows can have a nonscrolling region that does not move when the user scrolls information. The nonscrolling region can include the same elements as the main window. See Chapter 9, Defining Topic Windows, and Chapter 16, The Help Project File.
- Help includes a set of macros that you can use to modify and extend the functionality of Windows Help. The Help macros can affect an entire Help file, or they can be limited to a single topic or a single hot spot within a topic. See Chapter 14, Help Macros.
- Pop-up windows stay up until the user closes them with another action, so you can use pop-up windows for a wider variety of information with text, graphics, and hot spots. See Chapter 1, The Windows Help Application.
- In addition to links between topics, you can create links to information within the same topic or to topics in other Help files. See Chapter 6, Creating Topics, Chapter 8, Creating Links and Hot Spots,and Chapter 14, Help Macros.
- Topics can include dynamically sized, multiple-column tables. See Chapter 7, Formatting Topics.
- Help includes the Hotspot Editor application so that your Help graphics can include hot spots for pop-up windows, hypertext jumps, or macros. See Chapter 10, Adding Graphics, and Chapter 11, Creating Hypergraphics.
- Help includes the Multi-Resolution Bitmap Compiler so that you can create graphics that display correctly on different video resolutions. See Chapter 12, Creating Graphics for Different Display.
- The Help project file includes several new sections and many new options, which gives you increased control over how the Help file is created and built. See Chapter 16, The Help Project File.
- The Help compiler has been rewritten for improved performance; error messages have also been revised and rewritten. See Chapter 17, Building the Help File, and Chapter 18, Help Error Messages.
- The Help compiler supports three levels of compression. See Chapter 16, The Help Project File.
- To make developing Help files easier and more efficient, Windows Help includes the Help Authoring Templates and the Help Project Editor. These additions let you use menu commands and dialog boxes to create Help files within the Windows graphical environment instead of entering Help codes in your RTF editor. See Chapter 5, Using Help Author.
- You can write custom DLLs that extend Helps functionality by including features that you want to be part of your Help system. See Chapter 14, Help Macros, and Chapter 20, Writing DLLs for Windows Help.
- You can create an embedded window within a Help topic and use a DLL to display an object, such as an animation or a 256-color bitmap, in the window. See Chapter 20, Writing DLLs for Windows Help.

What Should You Know to Begin?

Because Windows Help integrates text and graphics, you might use many skills during the creation of a Help filedocument analysis, writing, editing, graphic design and production, and for programming and compiling Windows. This guide assumes you have this expertise or that you work with others who do.

The actual skills necessary to create the RTF files from which the Help file is generated, however, are much simpler. To start using Windows Help and this authoring guide, you should have:

- Experience with MS-DOS®.

  Although most of your development work takes place within the Windows graphical environment, you should know the basics of the MS-DOS operating system. Managing a Help project requires some understanding of MS-DOS commands and directory structures.
- Experience using Windows and an understanding of the Windows user interface.

  Before starting development on your Help system, you should install Windows version 3.0 or 3.1 on your computer and learn how to use it. Be sure to learn the name, purpose, and operation of each part of a Windows-based application (such as windows, dialog boxes, menus, controls, and scroll bars). Because the Windows Help application incorporates these features, it is very important that you understand them so you can implement them properly in your application and Help system.
- Experience using Word for Windows or another word processor.

  Microsoft Word for Windows is the preferred word processor for creating Windows Help files. Many of the features in Windows Help were designed with Word for Windows in mind. You can develop Help files using other editors, but they may present challenges that you typically avoid when using Word for Windows.
- An understanding of the user-interface style guidelines for Windows.

  One goal of Microsoft Windows is to provide a common user interface for all applications, including the Help application. This ultimately helps the user by reducing the effort required to learn the user interface, and it helps you by clarifying the choices you have to make when designing your application and Help system. Even though the content of the Help system varies from application to application, the user expects Help to be the same. Therefore, it is important that you build in a certain amount of consistency with other Help systems to make it easier for users to learn your Help system.

System Requirements

The system required to create a Windows Help file is different from the system required to view the Help file.

Authoring System Requirements

The following is the recommended configuration to create and test a Help file.

Hardware

- 80386-based computer running at 25 MHz
- 4 megabytes RAM
- 70 megabytes hard-disk storage
- VGA (16-color) monitor
- Tape drive or other storage device to create backups

Software

- MS-DOS version 3.3 or later
- Microsoft Windows version 3.0 or 3.1
- Microsoft Windows Help version 3.1
- Microsoft Word for Windows version 1.1 or 2.0

Viewing System Requirements

The following is the recommended configuration to display a Windows Help file.

Hardware

- 80286-based computer running at 10 MHz
- 1 megabyte RAM
- 20 megabytes hard-disk storage
- EGA (monochrome or color) monitor

Software

- MS-DOS version 3.3 or later
- Microsoft Windows version 3.0 or 3.1
- Microsoft Windows Help version 3.1

What Tools Do You Need?

Windows Help version 3.1 includes the following tools and files that you need to build your Help files.

Tool	Description
Microsoft Windows Help: WINHELP.EXE	The Windows Help application is the application that users open to display the Help files you create. Help is built into the Windows operating system; therefore, it is a shared resource available to all applications running in the Windows environment.
Microsoft Windows Help Compiler: HC31.EXE	The Help compiler compiles RTF files into binary Help files (.HLP) that can then be displayed in the Windows Help application.
Microsoft Hotspot Editor: SHED.EXE	Hotspot Editor lets you create graphics with multiple hot spots. Using Hotspot Editor, you can define hot spots that link to other Help graphics, to Help topics, or to multimedia events (if provided for by external DLLs).
Microsoft Multi-Resolution Bitmap Compiler: MRBC.EXE	The Multi-Resolution Bitmap Compiler lets you create bitmaps with different resolutions and combine them into a single graphic to compensate for differences between the aspect ratio of bitmaps you create and the users display.
Microsoft Windows Help Authoring Templates: WHAT30.DOT and WHAT31.DOT	The Help Authoring Templates are word-processing templates that modify Word for Windows. You use the templates to create and edit Help topic files and save them as RTF so they can be compiled. The templates offer a simplified way to add Help features to your Help file.
Microsoft Windows Help Project Editor: WHPE.EXE	The Help Project Editor is a tool that you use to create and edit Help project files (.HPJ). You can also compile Help files from within the Windows environment using the editor.
Microsoft Help Example: HELPEX.EXE	The Help Example application is written in the C programming language and conforms to the user-interface style recommended by Microsoft for Windows-based applications. It is an example of a simple Windows-based application that uses context-sensitive Help created with the Windows Help development tools. It is a good idea to review the source code for this application if you are planning to include context-sensitive Help for your application.

Related Documentation

In addition to this guide, there are several other documentation sources that relate to Help. They vary in the type of information they contain, but you may find one or more of them useful.

> Note: The Microsoft Windows version 3.1 Software Development Kit (SDK) includes information on Windows Help that is substantially different from the information in this guide in one respect: it is intended primarily for a programming audience. Therefore, you or the developers in your company should refer to the indicated SDK manuals for additional technical information about Windows Help.

Title	Contents
Programming Tools (SDK): Chapter 3, Creating Help Files, and Appendix B, Help Compiler Error Messages	Detailed information about the RTF file format and syntax used in Help topics and the Help project file. Use this information to learn the technical structure of Help or to develop your own custom application that reads and writes RTF files.
Programmers Reference, Volume 2: Functions (SDK)	Information about the HELP_* constants of the WinHelp function.
Programmers Reference, Volume 4: Resources (SDK): Chapter 15, Windows Help Statements and Macros	Syntax and purpose of RTF statements and macros used in Help topics and project files. Use this information to learn the technical structure of Help or to develop your own custom application that reads and writes RTF files.
Microsoft Windows version 3.1 Guide to Programming: Chapter 20, Dynamic-Link Libraries	Step-by-step explanation of how to create a Windows-based dynamic-link library.
The Windows Interface: An Application Design Guide	Principles of user interface design. This information will help you make decisions about the Help system you are creating and will ensure that the changes you make to the standard Windows Help interface are consistent with Microsoft Windows and applications for Windows.

Chapter 1 The Windows Help Application

This chapter introduces the Windows Help application and user interface. It explains the basic features of Windows Help as they appear to users. Because the Help application will display the Help files you develop, you must understand Windows Helps features from the users perspective.

The Windows Help User Interface

Windows Help is a standard Windows-based application that displays online Help files and is consistent with other applications that run in the Windows operating system. In addition to basic Windows skills, such as using the mouse, choosing menu commands, working with dialog boxes, using scroll bars, and so on, users must become familiar with unique elements of the Windows Help interface to use Help effectively. The following sections describe those elements.

Starting Help

Most applications for Windows offer Help; generally, all you do to start Help is choose a command, such as Contents, from the applications Help menu. You can also press F1 at any time to start Help. Or you can use any of the other methods Windows offers for starting applicationsthe Run command in Program Manager or File Manager, for example.

The Main Help Window

The main Help window appears when a user requests Help, and it stays open until the application or the user closes it. Information appears within the main Help window, which has the standard window elements: a title bar, a menu bar, a button bar, a display area (with scrolling and nonscrolling regions), and scroll bars (Figure 1.1).

Main Window Attributes

The first time the user requests Help, the main window appears in its default state on the screen. The default settings determine the size, placement, and background color of the main window. After the Help window displays, the user can change any of those attributes. Help authors can also define attributes for the main window that override the predefined defaults.

Title Bar

The title bar at the top of the Help window displays the name of the Help file or any title text that the author specifies in the [OPTIONS] or [WINDOWS] section of the Help project file (see Chapter 16, The Help Project File, for details). If the author doesnt specify a title, Windows Help appears in the title bar (Figure 1.2).

A Help title can have as many as 50 characters. Help centers the title in the available title bar width; however, if the user resizes the Help window so that the title text is wider than the window, the title is truncated from the right (Figure 1.3).

If the user opens a version 3.0 Help file, the title appears in Help 3.0 format: [Title string] Help - <Help filename.hlp>, as in Figure 1.4.

Size

The initial size of the main Help window is approximately two-thirds of the total screen width. Windows Help reserves a 2-pixel space around the Help window borders to keep the window from jamming against the edge of the screen display. The default size is large enough to accommodate the Help button bar without forcing the standard buttons to wrap (Figure 1.5).

> Note: To create international versions of Windows Help, the default size of the main window varies slightly depending on the length of the foreign-language text on the standard Help buttons. This variable is stored in the WINHELP.EXE resource file.

Users can resize the Help window, like other windows, by dragging the window borders or by using the Size command on the Control menu. They can also minimize or maximize the Help window. Any changes the user makes to the size of the main Help window are saved in the [Windows Help] section of the WIN.INI file.

> Note: Because Help does not define minimum limits for the size of the main window, the user can make the Help window so small that it is unusable.

Placement

When it first appears, the main window is positioned to the far right of the screen and extends the full height of the screen. Users can move the main window as they would move any other window in Windowsby dragging the title bar or by using the Move command on the Control menu. If the user moves the main window, the new position is recorded in the [Windows Help] section of the WIN.INI file.

Background Color

The default background color for the main window is the color the user defines using the Color application in the Windows Control Panel. However, the main window may appear with a custom background color if the author defines one for the Help file (Figure 1.6).

Authoring Predefined Attributes

Depending on the type of Help file you are creating, you may want to change Helps predefined defaults for the main window. For example, you may want to have the main Help window start up in a certain location on the screen when users request Help. Also, many types of information have different presentation requirements. Step-by-step procedures may work well in a tall, fairly narrow window. Tables of key equivalents, on the other hand, require a wider window. Other systems, like certain instructional models, require a fixed page size in which users view the information without scrolling.

Help authors can define any of the main window attributes discussed in this section:

- The window title
- The size of the window
- The location of the window
- The background color

Main window attributes are set in the [WINDOWS] section of the Help project file (see Chapter 16, The Help Project File, for details).

The Topic Area

In most applications for Windows, the part of the window where the user interacts with the computer and performs work is called the application workspace. In Windows Help, the workspace is where information is displayed. Help information is divided into chunks called topics. When you open a Help file, the topic (text and graphics) appears in the workspace of the main window (Figure 1.7).

> Note: If the Windows Help application is started without opening a Help file, the Windows Help logo appears in the topic area, and the button bar and scroll bars do not appear.

The Nonscrolling Region

Between the button bar and the scrolling region Help authors can create a nonscrolling region of the main Help window. The nonscrolling region is simply a portion of the window that does not scroll with the rest of the topic information; therefore, no scroll bars are necessary. If a vertical scroll bar displays in the topic, it does not extend into the nonscrolling region. The nonscrolling region is separated from the scrolling region by a hairline (Figure 1.8).

Because the nonscrolling region is an optional region that a Help author defines when creating the topic files, it may or may not appear in the Help window. If it does appear, it can contain the same kind of information as the scrolling region.

The Scrolling Region

The part of the main window that contains the topic information is called the scrolling region. Text within the scrolling region wraps dynamically along the right edge of the window, even when the user changes the size of the window, unless the author creates nonwrapping text.

Scroll Bars

The scrolling region of the main Help window can include horizontal or vertical scoll bars. However, unlike Help version 3.0, scroll bars do not appear in the scrolling region unless they are necessary. Not displaying scroll bars gives Help authors more space in the scrolling region for Help information.

Windows Help determines whether to display scroll bars when the window is first displayed and each time the user resizes the window. If the topic information extends below the lower window border, Help displays a vertical scroll bar (Figure 1.9).

If the topic information extends beyond the right edge of the Help window, a horizontal scroll bar appears (Figure 1.10).

Once a horizontal scroll bar appears, it remains in the topic, even if the user scrolls to another part of the topic that does not require a horizontal scroll bar.

> Note: Scroll bars behave independently in each Help window. Therefore, scroll bars that appear in the main window do not affect secondary windows.

Help scroll bars work the same as scroll bars in other Windows-based applications with one exception: information in the main window is not updated when the user drags the scroll box. Instead, Help updates the information only after the user stops moving the scroll box and releases the mouse button. This scroll-bar behavior is consistent with Windows-based applications that display large amounts of information in the main window, such as Word for Windows. For a complete description of the scroll bar interface, see The Mouse and the Keyboard Interfaces section later in this chapter.

Topic Text and Graphics

The scrolling and nonscrolling regions of the main Help window can display text and graphics, as well as other multimedia elements. These elements are created in separate applications, such as word processors and paint programs, and are included in the Help file during the build process. Help merely serves as a display engine for the text and graphics you create.

By controlling the way you create text and graphics, you control the way a topic looks to your users. Text attributesfont, point size, type style, colorare determined by Help authors when they create the topic files in a rich-text format (RTF) editor such as Word for Windows. The same is true for other elements. For example, graphics can be full-size or reduced, complete or partial, full-color or monochrome (Figure 1.11).

Topic Hot Spots

The scrolling and nonscrolling regions can include text and graphics that are hotthey provide links to other Help topics and to more information about the current topic.

Text hot spots are green and have either a solid or dotted underline. Graphics that are hot have no special visual cue. However, whether text or graphics, the pointer changes shape whenever it is over a hot spot (Figure 1.12).

To choose a hot spot

1. Position the pointer on the green text or hot graphic.
   The pointer changes to a hand.
2. Click the mouse button once.
   Or press TAB to highlight the hot spot, and then press ENTER.

   Note: Do not double-click the mouse button when you choose a hot spot because Help processes both clicks separately. Depending on the Help file, the double-click may cause unexpected behavior, choosing two hot spots or executing a macro twice, for example.

   Note: Pressing SHIFT+TAB highlights the hot spots in reverse order, and CTRL+TAB highlights all the hot spots within a topic simultaneously.

   If the hot spot links to another topic, that topic appears in the Help window.

   If the hot spot links to more information, Help displays the information in a pop-up window on top of the Help window. To close the pop-up window, click anywhere or press any key.

Windows Help Menus

Windows Help has four standard menus: File, Edit, Bookmark, and Help. Menu commands open Help files, assist users as they navigate through and request information from Help files, or display Help on specific tasks.

Customizing the Menu Bar

The descriptions that follow refer to the standard Help menus and commands. Except for the How To Use Help command, you cannot modify the standard Help menus or menu items. However, you can add additional menus and commands. Customizing the Help menu bar involves a number of Help macros, such as InsertMenu, InsertItem, DeleteItem, EnableItem, and others. For complete details about how to create your own custom menus and commands, see Chapter 13, Customizing the Help File, and Chapter 15, Help Macro Reference.

The File Menu

Commands on the File menu let you open a Help file, print a topic, set up your printer, or quit Windows Help.

Open

The Open command opens an existing Help (.HLP) file.

You can open the Help file for any application that uses Help, even if you are not using the application. For example, you can be working in Notepad and open the Help file for Program Manager to read about using group windows.

If you currently have a Help file open, Help closes that file before opening the new file.

Help uses the File Open dialog box stored in the COMMDLG.DLL file if it is available on the users system. Otherwise, Help uses the 3.0 version of the Open dialog box.

To open a Help file

1. From the File menu in Help, choose Open.
   The Open dialog box appears (Figure 1.13).
2. If the file is on a different drive, select the drive you want from the Drives box.
3. In the Directories box, double-click the directory that contains the Help file.
   Or press the UP ARROW or DOWN ARROW key to select the directory, and then press ENTER.
4. In the File Name box, double-click the name of the Help file you want to open.
   Or select the file, and then choose OK.
   Instead of using the File Name and Directories boxes, you can type the complete path in the File Name box, and then choose OK.

Print Topic

The Print Topic command prints the text in the topic that is currently displayed in the main Help window.

Help prints the entire topic, including graphics, but it does not print any annotations you might have made. If the topic has a nonscrolling region, it is printed at the top of the page. If the topic has an embedded window, the DLL is responsible for printing the information in the embedded window.

When a user prints a topic, Help displays a print status dialog box that contains the following information: Printing <topic title>. (Help uses the topic title text as defined in the title footnote.) If the topic does not have a name, Help displays untitled as the topic title. The user can continue printing or cancel the print job from this dialog box.

If you have connected to and installed a printer in Windows, Help prints to the default printer.

To print a Help topic

1. From the File menu in Help, choose Print Topic.
   The Print Topic dialog box appears while Help sends the topic information to the printer (Figure 1.14).

To cancel printing

1. Choose the Cancel button in the Printing dialog box.
   Or press the ESC key.

Print Setup

The Print Setup command selects a printer for Help and changes printer setup information.

Help prints topics on the default printer. If you have installed more than one printer, you can make any of

your installed printers the default printer. You can also change the default printer options. The available options in the dialog box depend on the type of printer you have connected to your computer.

Help uses the Print Setup dialog box stored in the COMMDLG.DLL file if it is available on the users system. Otherwise, Help uses the 3.0 version of the Print Setup dialog box. When using the version 3.0 dialog box, Help retains the version 3.1 menu command and Print Setup dialog box title rather than Printer Setup.

To set up a printer or change printer options

1. From the File menu in Help, choose Print Setup.
   The Print Setup dialog box appears (Figure 1.15).
2. Select the options you want for this printer.
   The options will vary depending on your installed printer.
3. If available, choose the Options button to select more options for the printer, and then choose OK when you have finished selecting options.
4. Choose OK.

Exit

The Exit command quits Windows Help.

When closing, Windows Help saves any annotations or bookmarks you have created.

To exit Help

1. From the File menu in Help, choose Exit.
   The Help window closes.

The Edit Menu

Commands on the Edit menu affect the text in the Help file, letting you copy a topic into a text editor or add notes to a Help topic.

Copy

The Copy command copies some or all of the text in a Help topic to the Clipboard. From the Clipboard, users can paste the text into another application or document. In this way, your Help system can provide examples and other information that the users might want to copy for themselves.

Choosing the Copy command places the text of the current Help topic in the Copy dialog box. (This command does not copy graphics.) Line breaks are inserted at the end of each line, unless the topic was authored as nonwrapping.

In the Copy dialog box, you can select a portion of the topic text that you want to copy to the Clipboard, or you can copy the entire topic to the Clipboard by not selecting anything. If you make a selection in the Copy dialog box, only the selected text is copied to the Clipboard.

Once displayed, the Copy dialog box can be moved and resized. Changes to the dialog box size or location are recorded in the [Windows Help] section of the WIN.INI file.

To copy text in the current Help topic to the Clipboard

1. From the Edit menu in Help, choose Copy.
   The Copy dialog box appears (Figure 1.16).
2. To copy all the text to the Clipboard, choose the Copy button.
   Or select the text you want to copy to the Clipboard, and then choose the Copy button.
   You can paste the text in the Clipboard into a Help annotation or into a document from another application.

You can also copy the entire contents of a Help topic directly to the Clipboard without displaying the Copy dialog box.

To copy the entire topic directly to the Clipboard

1. Press CTRL+INS.

Annotate

The Annotate command attaches a note to the currently displayed topic. This feature lets users add their own comments to the information you provide in the Help file.

In the Annotate dialog box, you can type the annotation you want to add to the Help topic, edit an existing annotation, delete an annotation, copy all or part of an annotation to the Clipboard, or paste the contents of the Clipboard into the annotation window.

When you save an annotation, Help places a paper-clip icon to the left of the topic title to remind you that you have added text to this topic. Clicking the paper-clip icon opens the Annotate dialog box so that you can see the annotation.

Copy and Paste buttons are included in the Annotate dialog box so that users can select a portion of the text to copy to the Clipboard. Or they can copy the entire annotation to the Clipboard by not selecting anything. If they make a selection, only the selected text is copied to the Clipboard. The Paste button allows users to place information in the Clipboard and then paste it into the Annotate edit box and save it with a topic. In this way, users can enhance their annotations with information from different sources. For example, a user might copy one Help topic to the Clipboard and then go to a different topic and paste the copied topic into an annotation to have the information from both topics accessible from one topic. The Paste button becomes active only when there is something in the Clipboard that might be pasted into the Annotate edit box.

Annotations are saved in a file with an .ANN extension. The annotation filename is based on the root

name of the Help file and the annotation extension. For example, annotations to File Manager Help would be stored in the WINFILE.ANN file. Help stores annotation files in the users Windows directory. Annotations in the current Help file remain in effect between Windows Help sessions.

Once displayed, the Annotation dialog box can be moved and resized. Changes to the dialog box size or location are recorded in the [Windows Help] section of the WIN.INI file.

To add an annotation to the current Help topic

1. From the Edit menu in Help, choose Annotate.
   The Annotate dialog box appears (Figure 1.17).
2. Type the text you want to add.
   If you make a mistake, press BACKSPACE to erase any unwanted characters, and then continue typing.
   Text wraps automatically in the edit box, but you can end a line before it wraps by pressing ENTER.
3. When you have finished creating the annotation, choose the Save button.
   A paper-clip icon appears in the topic title, indicating that you have made an annotation (Figure 1.18).

To view an annotation

1. Go to the topic where you made the annotation.
2. Click the paper-clip icon.
   Or press TAB to highlight the paper-clip icon, and then press ENTER.
3. When you have finished reading the annotation, choose the Cancel button.

To remove an annotation

1. Go to the topic where you made the annotation you want to delete.
2. Click the paper-clip icon.
   Or press TAB to highlight the paper-clip icon, and then press ENTER.
3. Choose the Delete button.

You can also copy text from an annotation and paste it into another annotation or into another document. Likewise, you can paste text from other documents into annotations.

To copy an annotation

1. Go to the topic where you made the annotation you want to copy.
2. Click the paper-clip icon.
   Or press TAB to highlight the paper-clip icon, and then press ENTER.
   To copy the entire annotation to the Clipboard, choose the Copy button or press CTRL+INS.
3. Or select the text in the annotation that you want to copy to the Clipboard, and then choose the Copy button.
4. To close the Annotate dialog box, choose the Save button.

To paste an annotation

1. Copy to the Clipboard the text that you want to paste into the annotation.
2. Go to the topic where you want to paste the annotation.
3. Click the paper-clip icon.
   Or press TAB to highlight the paper-clip icon, and then press ENTER.
4. To paste the contents of the Clipboard at the beginning of the topic, choose the Paste button or press SHIFT+INS.
   Or place the insertion point where you want to insert the new text, and then choose the Paste button.
5. To save the changes and close the Annotate dialog box, choose the Save button.

The Bookmark Menu

Commands on the Bookmark menu create and remove bookmarks placed at specific Help topics. The Bookmark menu initially contains one command: Define. As users define bookmarks, they are added to the menu until a total of nine appear. Help then adds a More command to give the user access to the additional bookmarks.

Define

The Define command places a bookmark in the Help file at the current topic or removes a bookmark from a topic. Just as you can place bookmarks in a book to mark specific references, you can place bookmarks at Help topics you use frequently. When you place a bookmark at a topic, you can get that topic quickly from Helps Bookmark menu.

In the Bookmark Define dialog box, you can type the name you want to assign to the bookmark, edit the suggested name, or delete a bookmark. When you define a bookmark, Help suggests the topic title as the bookmark name. You can use the default name or type your own name in the edit box. If the topic does not have a title, you must type your own bookmark name for it. When you save the bookmark, Help adds the bookmark name to the Bookmark menu and records the topic title and the users location within the topic for future reference. The Delete button in the dialog box lets you remove any bookmark that appears in the bookmark list box.

After you define a bookmark, you can choose the bookmark name from the Bookmark menu to move directly to the topic.

Help saves all user-defined bookmarks in the WINHELP.BMK file. Bookmarks are saved to a global file to make it easy for users to return to any marked location in any Help file. Help stores WINHELP.BMK in the users Windows directory. Bookmarks remain in effect between Windows Help sessions.

Bookmarks may or may not be compatible across different versions of a product. For example, if a user upgrades the Windows Help application from version 3.0 to version 3.1, all defined bookmarks will remain valid. However, if a user upgrades the Help file that uses the bookmarks, the bookmarks may become invalid. If the new Help file has many changesadded and removed topicsthe bookmarks will go to the wrong places. Therefore, it is best to remove a users .BMK file when your setup program installs a new version of the Help file.

To place a bookmark at the current topic

1. From the Bookmark menu in Help, choose Define.
   The Bookmark Define dialog box appears (Figure 1.19).
2. Type the name you want to give to the bookmark in the Bookmark Name box, and then choose OK.
   Or just choose OK to use the topic title as the bookmark name.
   The bookmark name now appears on the Bookmark menu in Help (Figure 1.20).

To remove a bookmark

1. From the Bookmark menu in Help, choose Define.
2. Select the bookmark you want to remove.
   Or type the bookmark name in the edit box.
3. Choose the Delete button.
   The bookmark name is removed from the bookmark list box and from the Bookmark menu.

Bookmark Names

After a user defines a bookmark, its name appears on the Bookmark menu so that the user can go directly to the bookmarked topic. The first nine bookmarks are listed on the Bookmark menu in the order that they are created. The number appearing to the left of the item name provides the keyboard access.

To go to a bookmarked topic

1. From the Bookmark menu, choose the name of the bookmarked topic you want to view.
   Underlined numbers precede the first nine bookmark names. You can press the corresponding number key to go quickly to a marked topic.

More

The More command appears as the last item on the Bookmark menu when you have defined more than nine bookmarks.

Choosing More displays the Bookmark dialog box, which contains a list of all the bookmarks you have defined in the current Help file. Help selects the first item in the bookmark list when the dialog box opens.

To choose from the complete list of bookmarked topics

1. From the Bookmark menu in Help, choose More.
   The Bookmark dialog box appears (Figure 1.21).
2. In the Go To Bookmark box, double-click the name of the bookmarked topic you want to view.
   Or select the bookmark name, and then choose OK.

The Help Menu

Commands on the Help menu display the How To Use Help file for Windows Help, place Help windows on top of other application windows, and identify the Help application.

How To Use Help

The How To Use Help command displays the Contents screen of the Help file for Windows Help (WINHELP.HLP). This file provides instructions on how to use the Windows Help application and Help topics.

If the user opens Help as a stand-alone application, choosing this command displays the How To Use Help file in a second Help window. This second Help window remains open until the user closes it or until the user closes the first instance of Help (if there are no other instances of the Help window open).

In Windows Help version 3.1, Help authors can create their own How To Use Help file instead of using the default file by using the SetHelpOnFile macro in the [CONFIG] section of the Help project file. Replacing the standard file is important if you want the instructional file to reflect the particular features of your applications Help system.

> Note: Use the SetHelpOnFile macro to specify a custom How To Use Help file, the DeleteItem macro to remove the standard How To Use Help item, and the InsertItem macro to place the new menu item on the Help menu. See Chapter 15, Help Macro Reference, for information about each of these macros and Chapter 16, The Help Project File, for information about how to use them in the [CONFIG] section of the Help project file.

To learn how to use the Help application

1. From the Help menu, choose How To Use Help.
   Help displays the Contents topic of the standard file or of a custom How To Use Help file if one has been authored (Figure 1.22).

Always On Top

The Always On Top command causes all Help windows to appear on top of other application windows on the screen. If you minimize a window that is on top, its icon also appears on top of other windows.

> Note: Other applications that use the topmost attribute, such as Windows Task List, may compete with Help for the top position. In that case, the Help window may appear beneath another window.

When you first ask for Help, the Help window appears on top of other windows on the screen. If you click inside another window, the Help window goes behind and the window you clicked appears on top.

You can choose to keep the Help window on top of other windows even when switching to other application windows. This can be especially useful if you are using Help to follow a step-by-step procedure in your application.

> Note: This command is available only with Windows version 3.1.

To make the Help window stay on top

- From the Help menu, choose Always On Top.
  A checkmark appears next to the command, and a 2-pixel drop shadow appears around the window border to indicate that the Help window is on top (Figure 1.23)

To return Help to normal window behavior

- From the Help menu, choose Always On Top again.

Note: If the Always On Top command is not checked, a secondary window that has been defined as an on-top window may remain on top. The main Help window can be made on top only by choosing this command.

About Help

The About Help command displays a dialog box that identifies the Windows Help application. It includes the Help application name, the Help icon, the version number, and the Microsoft Help copyright notice. In addition, the dialog box lists information about the user and the users hardware configuration.

Using the COPYRIGHT option in the Help project file, you can provide a custom copyright notice or other message about your applications Help file. For details, see Chapter 16, The Help Project File.

Help uses the About Help dialog box stored in the SHELL.DLL file if it is available on the users system. Otherwise, Help uses the 3.0 version of the About Help dialog box.

To see information about Windows Help

1. From the Help menu, choose About Help.
2. The About Help dialog box appears (Figure 1.24).

The Windows Help Button Bar

Windows Help has four standard buttons: Contents, Search, Back, and History. The buttons appear in the button bar, just below the menu bar in the Help window. The standard buttons function as navigation controls, helping users access the information in the Help file (Figure 1.25).

To choose a Help button

1. Click the button you want.

   Or press the underlined letter in the button name.

The Help button bar in version 3.1 is different from the version 3.0 button bar in two ways: the Browse buttons (<< and >>) have changed from standard buttons to optional buttons, and the buttons have changed from graphical buttons to text-only buttons. The latter change was necessary because:

- Text-only buttons are half as large as graphical buttons, so more information fits in the Help window.
- Text-only buttons reduce the size of the Help application (WINHELP.EXE), giving you more space for your Help file.
- Text-only buttons simplify the process of adding custom buttons to the interface and of translating the existing buttons into foreign languages for international markets of your application and Help.

The Help button bar behaves the same as a standard menu bar: buttons wrap if the window becomes too narrow to fit all the buttons horizontally. The buttons continue to wrap until they are all stacked vertically. If the buttons still do not fit in the window, Windows Help reduces the size of the buttons and truncates the button text, if necessary.

Customizing the Button Bar

The descriptions that follow refer to the standard Help buttons. You cannot remove the standard Help buttons. However, you can change the function of the standard buttons or add additional buttons to the button bar. Help authors can create up to 16 additional buttons for a total of 22 buttons (including Browse buttons).

Custom buttons can have as many as 29 characters, after which Help truncates the button name. Help determines the size of all the buttons based on the longest name and the default button size (determined by the longest standard button nameContents). So if you assign long button names to custom buttons, the size of all the buttons in the button bar will increase.

Customizing the Help button bar involves a number of Help macros, such as ChangeButtonBinding to modify the standard buttons, CreateButton, DestroyButton, EnableButton, and others to create and modify custom buttons. For complete details about how to create your own custom buttons, see Chapter 13, Customizing the Help File.

The Contents Button

The Contents button is the first, or leftmost, button in the Help button bar. Choosing the Contents button displays the topic designated by the Help author as the Help Contents (usually the top-level topic in a Help file). If the author does not specify a contents topic by using the Contents= entry in the [OPTIONS] section of the Help project file, Windows Help displays the first physical topic in the Help file (the first topic in the first .RTF file listed in the [FILES] section of the Help project file).

The purpose of the Contents button in Windows Help version 3.1 is to create a consistent navigation point within a single Help file and across all Windows-based applications that use Help. When using hypertext systems, users often feel lost and need a way to reorient themselves in this abstract hyperspace. The Contents button gives users an easy way to return to a familiar starting place.

> Note: When displaying a version 3.0 Help file, Help uses the Index= entry as the Contents topic.

All applications included with Windows version 3.1, such as Program Manager, File Manager, Write, and so on, have a Contents command on the Help menu. Choosing the Contents command from an applications Help menu is the same as choosing the Contents button in the Help button bar.

To see the Contents topic of the current Help file

- Choose the Contents button.

The Search Button

Windows Help lets users search for Help information by keyword, which is similar to searching through a book index. The purpose of the Search button in Windows Help version 3.1 is to create a consistent way to create an online index for Help files. Book indexes have proven to be very effective ways for people to look up information. In an electronic document, such as a Help file, users also need an easy way to find specific information. The Search button gives users the electronic equivalent of a book index.

Users search for information by looking up the keywords an author has defined for the Help topics. For example, a user might look up all the topics that have the keyword save associated with them. Those topics should provide information about saving different kinds of items, such as files and documents (Figure 1.26).

The Search dialog box has an edit field, two list boxes, and three command buttons:

- The edit box is used to type a keyword for the search. As the user types in the edit field, the keyword list scrolls automatically to match the users entry or the closest letter to that entry. For example, typing the letter S scrolls the keyword list to the first keyword beginning with S. The keyword highlight moves forward in the list box whenever it has a match closer to what the user is typing. If a user performs a search and goes to the topic, the previous search keyword will be entered automatically in the edit box.
- The top list box contains all the keywords in the Help file. Six keywords show in the list box at one time. Clicking a keyword in the list box enters it in the edit box. Double-clicking an entry selects the keyword and performs the search.
- The Show Topics button performs the search using the keyword in the edit box. A user can press ENTER after selecting a keyword.
- The lower list box shows all the topics found by the search. Approximately seven entries will fit in the list box. If more topics are found, the user can scroll to see more topics. Clicking a topic selects it. Double-clicking a topic title displays that topic (same as selecting the topic and choosing Go To).
- The Go To button displays the topic the user has selected in the Topics Found list box and closes the Search dialog box. After a user chooses Show Topics, the user can press ENTER again to go to the topic. This double-return action in the dialog box allows for quick searches. Or, users can search quickly by typing a keyword and pressing ENTER twice to go to the first topic displayed in the Topics Found list.
- The Cancel button dismisses the dialog box. (The Cancel button changes to Close if the user has already performed a search.)

Users can move the Search dialog box if they want. Changes to the dialog box location are recorded in the [Windows Help] section of the WIN.INI file.

All applications included with Windows version 3.1, such as Program Manager, File Manager, Write, and so on, have a Search For Help On command on the Help menu. Choosing the Search For Help On command from an applications Help menu is the same as choosing the Search button in the Help button bar.

To search for a Help topic by keyword

1. Choose the Search button.

   The Search dialog box appears (Figure 1.27).

   If the user has performed a search during this session, the previous keyword is displayed in the edit box.
2. Type a keyword, or select one from the keyword list.

   You can use your keyboard to move quickly to different parts of the list. The list scrolls to those keywords that most closely match what you are typing in the text box.
3. Choose the Show Topics button.

   A list of topics associated with that keyword appears in the lower list box, as in Figure 1.28.
4. Select the topic you want, and then choose the Go To button to jump to the selected topic.

   Or double-click the topic you want.

The Back Button

The Back button has the same functionality as in Help version 3.0: it keeps track of every topic users look at from the time they open Help until they close it and stores these topics in a back list. Choosing the Back button displays the topic that was previously displayed in the Help window (Help goes back one item in the back list). Choosing Back repetitively causes Help to back up one screen at a time along the path the user has followed.

In other words, Back lets users go back to topics they have already viewed, in the order they viewed them, without forcing them to remember anything about the topic or how they got there. Back simply retraces the steps the user has taken. So, in one sense, Back is like a screen recorder and playback mechanism combined into a single button.

Back functions in the current file as well as across files, so if a user opens or jumps to a different file, backing up reopens an inactive file to display a previously viewed topic. However, the Back list is cleared when the user closes Help. If there is no previous topic to view, the Back button is dimmed.

To go back to the previous topic

- Choose the Back button.

To back up to a previously viewed topic

- Choose the Back button repeatedly until you return to the topic you want to see.

The History Button

Choosing the History button displays a window that shows the topics (a maximum of 40) you have viewed during the current session. You can select any topic from the list to view it again (Figure 1.29).

The purpose of the History button is to give users a visual representation of the topics that they have viewed during a session and let them return easily to any of those topics. Because the space that users navigate in hypertext systems is abstract and nonlinear, they often suffer from disorientation. As users move through the information, however, they create a linear path. The history list records this linear movement and lets users visualize the path they have taken through the information and return quickly to a particular place. In this sense, History is similar to Back in that they both are backtracking devices. But because the History window presents this path visually and stays open while users interact with it, History offers advantages to the browsing user.

Windows Help uses the following guidelines to build the history list:

- The topic title of the topic displayed in the Help window when the user chooses the History button appears first in the history list.
- The second item in the history list is selected when the user chooses the History button so that the user can just press ENTER to return to the previous topic. This gives History the same functionality as Back.
- Each entry in the history list occupies one line. If the topic title does not fit in the History window, it is truncated rather than wrapped.
- The history list records the users location within a topic as well as the topic title, even though only the title appears in the list. When users choose an item from the history list, Help returns them to the exact place in the topic where they were when they left it (scrolled half-way down, for example).
- The history list records only topics that are displayed in the main Help window. Topics displayed in secondary windows or pop-up windows do not appear in the history list.
- The history list records only jump-type movements in Help. Hot-spot events such as launching an application or a tutorial are not recorded. Movements caused by using the Back button or requesting How To Use Help are recorded, however.
- Help does not remove duplicate entries from the history list. If a user views a topic several times, each instance appears in the history list.
- Help maintains the history list across Help files. If the user chooses an interfile jump, the history list prefixes the root name of the Help file before the topic title.

When a user chooses a topic from the history list, the History window does not close, but it loses the focus. To switch back to the History window, the user can click inside the window, choose the History button, or press ALT+F6. To cancel the History window without choosing a topic, the user can choose the Close command from the Control menu or press ALT+F4. If the user minimizes the main Help window, the History window closes, and restoring the Help window does not restore the History window. The user must choose the History button again to restore the History window.

The default size of the History window is approximately one-third the height and width of the screen. The default position is centered. Once displayed, the History window can be moved and resized. If the user changes the size or location of the History window, it will retain those changes the next time it appears. Changes to the windows size or location are recorded in the [Windows Help] section of the WIN.INI file.

To use the History button to return to a topic

1. Choose the History button.
   The History window appears (Figure 1.30).
2. Double-click the topic to which you want to return.
   Or select the topic and press ENTER.
   If necessary, use the scroll bar to see more topics.

Browse Buttons

The Browse buttons, which were standard in Help version 3.0, are now optional in version 3.1. Even though they are optional buttons, they can be considered as standard buttons because they are a feature of the Windows Help application. The Browse buttons have been made optional because Microsoft feels that this functionality can often be better presented in the context of the topic information rather than as part of the Help application. It is up to each individual author to decide whether to include the standard Browse buttons or create a unique, custom browsing feature in their Help system.

Browse buttons can be included as part of the standard Windows Help button bar by adding the BrowseButtons macro to the [CONFIG] section of the Help project file. For more information about adding the Browse buttons, see Chapter 15, Help Macro Reference, and Chapter 16, The Help Project File. The procedure for creating browse sequences within the topic files has not changed from version 3.0 to 3.1. See Chapter 6, Creating Topics, for the authoring information.

If the author includes the Browse buttons in the Help file, or if Help opens a version 3.0 Help file, the Browse buttons are displayed to the right of the four standard buttons (Figure 1.31).

> Note: When displayed in Windows Help version 3.1, version 3.0 files use the 3.1 text-only buttons rather than the graphical buttons seen in version 3.0.

To browse to the previous topic in a browse sequence

- Choose the Browse << button.

To browse to the next topic in a browse sequence

- Choose the Browse >> button.

Customizing the Help Application

When working with Help files, you might want to alter the look and behavior of Windows Help. You can customize Help by:

- Creating a program item to start Help and a specific Help file.
- Starting Help from MS-DOS.
- Changing the standard Help icon to a custom icon.
- Changing Helps default color scheme.

The following sections explain how to customize the way Windows Help looks and works.

Creating a Program Item for Your Help File

While you are creating your own Help file, you may want to have a more customized setup for starting Help. The best thing to do is to create one or more program items in a Program Manager group that will start Help and, if you want, a specific Help file.

To create a program item for the Help application

1. In Program Manager, open the group where you want to add the Help item.
2. Press and hold down the ALT key while double-clicking a blank area of the group window.
   The Program Item Properties dialog box appears.
3. Fill in the Program Item Properties dialog box as necessary, and then choose the OK button.
   For help with the Program Item Properties dialog box, choose the Help button in the dialog box.
   You can specify any of the command-line options for starting Help in the Command-Line box. See the next section for details.

Starting Help from the MS-DOS Command Line

You can start Windows Help directly from the command line in MS-DOS, or you can use command-line options within Windows to start Help using a program item or using the Run command in Program Manager.

To start Help without opening a Help file, type:

```
c:\windows\winhelp
```

To start Help and open a specific Help file, add the Help file name after Help:

```
c:\windows\winhelp myhelp.hlp
```

If the Help file is located on a network server, specify the network path to the file:

```
n:\server\share\help\winhelp myfile.hlp
```

Use the -n option followed by a context ID number to start Help and display a topic with a context ID that you have defined in the [MAP] section of the Help project file:

```
c:\windows\winhelp -n 801 pifedit.hlp
```

Use the -i option followed by a context string to start Help and display a topic with the context string identifier that you have assigned using the pound footnote (#):

```
c:\windows\winhelp -i cg_how_pm progman.hlp
```

Changing the Standard Help Icon

The standard Windows Help icon is a yellow question mark. You can replace this icon with a custom icon; use any .ICO icon file that you create with Windows Imagedit or another icon program.

To change the Help icon

1. Open the group in Program Manager that contains the Help program item and select the Help icon.
2. From the File menu, choose Properties.
3. In the Program Item Properties dialog box, choose the Change Icon button.
4. If more than one icon is available in the Current Icon area, scroll through the icons by clicking the scroll arrows or by using the LEFT and RIGHT ARROW keys.

   If you would like more icons to choose from, you can scroll through all the available Program Manager icons. Or you can use the Browse button to select an icon from a directory on your hard disk or floppy drive, if you have other icons to choose from.
5. When the icon you want to use is selected, choose OK.
6. In the Program Item Properties dialog box, choose OK.

Changing Help's Default Color Scheme

Because Windows Help is a standard Windows-based application, users can modify its default appearance the same way they modify the look of Windowsby using the Control Panel Colors setting to change their screen colors. Control Panel lets users change the color of the standard Help window elements: title bar, menu bar, button bar, scroll bars, window background, window text, and window border. In addition to these standard window elements, Help has two special elements: hot-spot text and the Help startup screen. By default, text hot spots in Help are green and the startup screen is blue. Windows Help provides custom WIN.INI entries that users can use to modify these two visual elements of Help.

> Note: Generally, applications should not change any of the WIN.INI settings described in this section because they are global settings designed for end users. They are not provided so that an applications setup program can edit the WIN.INI file and insert or update these settings during installation of their product. If an application wants to change the way hot spots appear in Help, it should use the authoring procedures described in Chapter 8, Creating Links and Hot Spots.

Changing Hot-Spot Colors

If users want to change the way hot spots appear in Help, they can edit the Windows WIN.INI file and specify custom hot-spot colors. Help supports the following WIN.INI entries:

[Windows Help]JumpColor=(RRR, GGG, BBB)PopUpColor=(RRR, GGG, BBB)MacroColor=(RRR, GGG, BBB)IFJumpColor=(RRR, GGG, BBB)IFPopUpColor=(RRR, GGG, BBB)

RRR, GGG, and BBB represent the red, green, and blue (RGB) components of the hot-spot color. These color values can be a number from 0 to 255. The meaning of each entry is explained in the following table.

Command	Defines the color for	Default color
JumpColor	Jump hot spots	Green
IFJumpColor	Interfile jump hot spots	Same as JumpColor
PopUpColor	Pop-up hot spots	Same as JumpColor
IFPopUpColor	Interfile pop-up hot spots	Same as JumpColor
MacroColor	Macro hot spots	Same as JumpColor

To change the color of Help hot spots

1. Open the WIN.INI file in Notepad or some other text editor.
2. Go to the [Windows Help] section.
3. Type one or more of the hot-spot color entries.

   > Note: You can specify just the JumpColor entry to change the color of both jump and pop-up hot spots.

4. Save the WIN.INI file and exit the text editor.
5. Restart Windows to have your changes take effect.
   When you open Help again, your hot-spot colors will be set to the colors you specified.

Overriding Author-Defined Hot-Spot Colors

As Chapter 8 explains, Help authors can also change the color of hot-spot text. These author-defined colors may cause problems for some users (those with gas plasma VGA screens, for example), making it difficult or even impossible to read the hot-spot text. If the Help file contains an author-defined color for

hot spots, the above entries will have no effect. To override an author-defined hot-spot color, users can type the following entry in the WIN.INI file:

[Windows Help]Colors=NONE

If Help finds this entry in the WIN.INI file, it uses the system default colors for foreground and background colors, regardless of how the Help file was authored.

Chapter 8, Creating Links and Hot Spots, explains how Help authors can change the default appearance of hot spots.

Changing Help's Startup Screen Colors

When you start the Windows Help application without opening a Help file, Help displays a startup logo that identifies itself. If users want to change the starting and ending colors used in the Help logo screen, they can edit the WIN.INI file. Help supports the following WIN.INI entries:

[Windows Help]LogoStart=(RRR, GGG, BBB)LogoEnd=(RRR, GGG, BBB)

As with the hot-spot entries, you specify the RGB color you want for the logo screen color on the right side of the equal sign (a number from 0 to 255). The meaning of each entry is explained in the following table.

Command	Defines the color for	Default color
LogoStart	Windows Help startup logo screen	Blue
LogoEnd	Windows Help ending logo screen	Blue

The Mouse and the Keyboard Interfaces

You can use the mouse or the keyboard to work with Help, or you can use a combination of the two. The following sections provide details about both interfaces.

Mouse Interface

Windows Help uses the standard mouse interface for common actions: choosing commands, moving or resizing the Help window, choosing buttons, interacting with dialog boxes, scrolling information, and choosing hot spots. There are, however, some of anomalies in the mouse interface.

The first involves choosing hot spots. In many Windows-based applications, the user must double-click the mouse to execute an action. This is true in Help also. To choose a topic from the History window, you can double-click the topic. But this is not true with hot spots. Choosing a hot spot requires only one mouse click. Using a double-click to choose a hot spot may result in a double action. For example, double-clicking a jump may cause Help to execute an action in the two topics: the one containing the jump and the one that is displayed as a result of the jump.

The second difference involves scrolling. In many application windows and list boxes, the information in the window or list box is updated as the user moves the scroll box. In the main Help window, information is not updated when the user moves the scroll box. Instead, the information is updated only when the user finishes moving the scroll box and releases the mouse button.

Scrolling with the Mouse

Use the following techniques to scroll text within the Help window.

To scroll	Do This
Up or down one line	Click the up or down scroll arrow on the vertical scroll bar.
Right or left one line	Click the right or left scroll arrow on the horizontal scroll bar.
Up or down one window	Click the vertical scroll bar above or below the scroll box.
Right or left one window	Click the horizontal scroll bar to the right or left of the scroll box.
Continuously	Point to one of the scroll arrows, and hold down the mouse button until the information you want comes into view.
To a particular place in the topic	Drag the scroll box to the position you want.

Keyboard Interface

As in other Windows-based applications, Help provides keyboard equivalents for most mouse actions. The following tables describe the keys you can use in Help.

Help Access Keys

Use the following keys to get Help from the application.

To	Press
Start Help and display the Contents for the application. If the Help window is already open, pressing F1 displays the Contents for the How To Use Help topics.If the application provides context-sensitive Help (such as Program Manager and File Manager), pressing F1 displays a Help topic on the selected command or dialog box option.This feature is available only if the application supports it. It is available in all of the applications included with Windows.	F1
Change to Help mode so you can choose the command, click the screen region, or press the key or key combination on which you want Help.This feature is available only in certain applications (such as Microsoft Word for Windows). It is not available in any of the applications included with Windows.	SHIFT+F1

Help Button Keys

Use the following key equivalents for the Help buttons.

To	Press
Display the Help Contents for the application	C
Display the Search dialog box, which lists all the search keywords for the application	S
Go back to the last topic you viewed	B
Display the History window, which lists the last 40 topics you have viewed	T
Display the next topic in a browse sequence (Optional Browse >> button)	period (.)
Display the previous topic in a browse sequence (Optional Browse << button)	comma (,)

Help Window Keys

Use the following keys while working in Help.

To	Press
Select a hot spot. Pressing TAB repeatedly moves you clockwise to other hot spots in the topic.	TAB
Select a hot spot. Pressing SHIFT+TAB repeatedly moves you counterclockwise to other hot spots in the topic.	SHIFT+TAB
Select all the hot spots in a topic.	CTRL+TAB
Copy the entire contents of the current Help topic directly to the Clipboard without displaying the Copy dialog box. Or copy an entire annotation or a portion of it to the Clipboard.	CTRL+INS

Paste the contents of the Clipboard into the Annotation dialog box.	SHIFT+INS
Switch between the main Help window and the History window.	ALT+F6
Close the Help window.	ALT+F4

Help Movement Keys

Use the following keys to scroll text within the scrolling region of the Help window.

To scroll	Press
Up one line	UP ARROW
Down one line	DOWN ARROW
Right one character	RIGHT ARROW
Left one character	LEFT ARROW
To the leftmost position	HOME
To the rightmost position	END
To the first line in the scrolling region of the topic	CTRL+HOME
To the last line in the scrolling region of the topic	CTRL+END
Up one window	PAGE UP
Down one window	PAGE DOWN
To the beginning of the topic	CTRL+PAGE UP
To the end of the topic	CTRL+PAGE DOWN

Chapter 2 Getting Started with Help Authoring

This chapter introduces you to the basic process of authoring and building Help files (.HLP). The Building a Simple Help File section presents two short walk-throughs that guide you through the authoring process by having you create and build a simple Help file yourself. Because each walk-through uses a different process, you can become familiar with different ways to create Help files and choose the authoring method that works best for you.

Windows Help Files

When developing a Help system, the objective is to create one or more Help files. A Help file is the binary file that is actually displayed by the Windows Help application. In essence, the Help authors work parallels the programmers development of an application. All the work and coding are produced for the express purpose of creating an application that a user can use. Similarly, all the writing and developing of Help files is done to produce a Help file that the user can view in Help.

That describes the purpose of a Help file, but what exactly is a Help file? In simple terms, it is a document. Like word-processing documents or spreadsheet documents, Help documents are defined in part by the information they contain. Spreadsheet documents are electronic ledger sheets that can perform calculations and other mathematical operations. Word-processing documents are electronic pages that can manipulate text and graphics. Help files are like word-processing documents in that they too can manipulate text and graphics. But Help documents also can include hypertext elements, such as links and hot spots, and multimedia elements, such as sound and video. And, unlike the other two kinds of documents, Help documents have no real-world equivalent. They exist only in electronic form on the computer. You can create a spreadsheet or a letter without a computer, but you cant create a Help document without a computer.

Help Topics

A Help file is made up of distinct units of information called topics. A topic in a Help file is similar to a page in a word-processing document. Each topic focuses on a specific piece of information such as a concept, a step-by-step procedure, a command summary, an illustrative graphic, a keyboard table, an example, or any other unit of information that you define. A topic can include text or graphics or both text and graphics. By definition topics are an arbitrary length. Either the content itself or the Help author determines the length of a particular topic. The same is true of Help files. Since a Help file is made up of all the topics it contains, its size is determined wholly by the number of topics in it.

The first topic that appears when the user opens a Help file is the Contents topic, which acts as a table of contents for the Help file. It may also serve as the home screen from which all other topics or topic categories are accessed. The user can return to the Contents topic by choosing the Contents button (the first button on the Windows Help button bar). Figure 2.1 shows the Contents topic for the sample Help filePLACEHOLDER.

> Note: You can view the sample application by clicking the PLACEHOLDER icon or by running Windows Help and opening the PLACEHOLDER.HLP file.

If the information in a Help topic doesnt fit in the current window, you can use the scroll bars or use the UP ARROW, DOWN ARROW, PAGE UP, or PAGE DOWN key to see the rest of the information.

Hypertext Jumps

Help topics are linked by hypertext jumps. When you choose text or a bitmap that activates a jump (called a hot spot), Windows Help clears the current topic and displays the topic indicated by the jump. Jumps are indicated in a Help file as follows:

- The mouse pointer changes from an arrow to a hand when it is over a hot spot.
- Text jumps appear in green with a solid underline.
- Bitmaps coded as jumps appear the same as regular bitmaps; however, the pointer changes to a hand when it is over the bitmap hot spot.

Figure 2.2 shows a topic that contains a jump at the underlined text PLACEHOLDER (this text is green on a color monitor).

To make the jump

- Position the mouse pointer over the hot spot and click the left mouse button.

Windows Help finds and displays the topic referenced by the jump, as in Figure 2.3.

After jumping to a topic, you can return to the original topic by choosing the Back button on the Windows Help button bar.

Pop-Up Windows

Within topics, certain terms or concepts often require further explanation. Instead of having Windows Help jump away from the current topic to provide the additional information, you can have it display the information in a pop-up window on top of the current topic. Pop-up windows are stripped-down windows that have only a window border. They can display text and graphics, and they can include hot spots. Pop-up window hot spots are indicated in the Help file as follows:

- The mouse pointer changes from an arrow to a hand when it is over the hot spot.
- Text hot spots appear in green with a dotted underline.
- Bitmaps coded as pop-up hot spots appear the same as regular bitmaps; however, the pointer changes when it is over the bitmap hot spot.

In Figure 2.4, the first paragraph includes a hot spot that, when chosen, displays a pop-up window.

When you click the hot spot, the pop-up window appears (Figure 2.5).

Pop-up windows remain displayed until you click the mouse button again or press any key.

Developing Help Files

You create Help files using the Help tools. At a minimum, you will use Word for Windows or another text editor, the Windows Help compiler, and the Windows Help application. (For a complete list of Help tools, see the Introduction to theis authoring guide.)

Developing a Help file involves several processes:

1. Develop a documentation plan for the Help project.
2. Program the application to access Help.
3. Create the topic files.
4. Create the Help project file.
5. Compile the topic files into a binary .HLP file.
6. Test and debug the Help file.

Developing a Documentation Plan

The documentation plan provides a way for Help authors to organize the Help project and to begin thinking about design issues. The document defines how the Help system will teach users about the product. Usually, one or more writers or instructional designers create the plan with input and suggestions from other team members, writing managers, developmental editors, product marketing managers, and program managers.

The documentation plan should describe:

- The product definition
- The target audience and market
- The documentation objectives and instructional model
- The components in the documentation package
- A content outline of each component
- Technical considerations and assumptions
- Resources and tools
- The delivery schedule and intermediate milestones
- Project issues and concerns

Programming the Application

In order for users to access your Help file, the application developers must program the application to access Help. Ideally, the application should provide context-sensitive Help. When users ask for Help, they expect the application to know where they are and what they have been doing. They become impatient when they ask for Help and are given irrelevant information or a long list of topics (from which they must choose the correct information). Context-sensitive Help is convenient, saves users time, and prevents users from overlooking something important.

Context-sensitive Help must be programmed into the application using the WinHelp API function (explained in Chapter 19). When creating context-sensitive Help, a software engineer assigns special ID numbers (often called hooks) to each context. After these context IDs have been assigned, the Help author can create the Help topics that the user will see when requesting Help on a particular interface component.

The application should be programmed early in the development cycle so that Help authors have adequate time to create the topics. Otherwise, writers will have to postpone their writing or reorganize the Help topics later in the process.

Creating the Topic Files

Windows Help files usually consist of several rich-text format (RTF) files, with each file divided into one or more topics. A topic is any distinct unit of information that is separated by a page break, such as a Contents screen, a conceptual description, a set of instructions, a keyboard table, a glossary definition, a list of jumps, a picture, and so on.

You create topic files using a word processor or a text editor that can generate RTF files. (To ensure the highest compatibility, we recommend that you use Microsoft Word for Windows.) Creating Help topics involves roughly the same phases as creating printed books: gathering information, writing, and editing.

Generally, when gathering information for the Help topics, a Help author can rely on these sources:

- The product specification
- The applications printed documentation
- Help files from a previous release of the application
- Hands-on experience using the alpha and beta software
- The software engineers creating the application

Topic files also contain special Help codes that pass organizational and build information to the Windows Help compiler. Within the topic text, you create cross-reference links, pop-up window hot spots, macro commands, references to graphics files, and other codes particular to your Help file. The finished topic files contain the content and build information for the Help file.

Creating the Help Project File

After you create the topic files, you build the Help file with the Windows Help compiler (HC31.EXE), which uses the topic files and the Help project file (a special build file with an .HPJ extension). The Help project file tells the compiler exactly how to build the Help file.

Building the Help File

After you create the topic files and you have a Help project file, you use the Help compiler to build the Help file. The Help compiler uses the Help project file and topic files to create a single Help file (with an .HLP extension).

During the build, the Help compiler displays error messages when it encounters a problem. Help authors can use these build errors to debug the Help file.

Displaying the Built Help File

You can display the compiled Help file using the Windows Help application (WINHELP.EXE). Windows Help interprets and displays the elements of the Help filetext, pictures, hypertext links, and keyword search indexand lets the user interactively explore the information presented. In a built file, all the features designed into the Help file come into use; for example, users can browse topics as they would in a printed book or use hypertext links to jump around.

> Note: With the exception of being able to add notes and bookmarks, users can only view a compiled Help file; they cant change the actual content or structure of the file.

Because certain types of errors and authoring mistakes only show up when viewing the built file, the Help author should perform additional testing and debugging at this time. If errors do appear in the built version, the Help author should correct the problem and rebuild the file.

Building a Simple Help File

Now that youve seen what a Help file looks like to the user and are familiar with the basic features of the Windows Help application, youre ready to learn how to build a Help file. You have two options to create topic files for Windows Help: you can use Word for Windows and enter all the Help-specific coding and information manually, or you can use Microsoft Help Author and enter the information in dialog boxes. Although the first process certainly works, using Help Author as your primary authoring tool makes creating Help files simpler and easier.

The following sections show you how to build a simple Help file using both of these methods. The Help file you build in each case is exactly the same. If you are new to Help authoring, it is a good idea to be familiar with both authoring methods, but if you want to save time, and you know which method you want to use, you can complete just that section.

Using Help Author to Build a Simple Help File

This section explains how to create and build a simple Help file using Microsoft Help Author. The simple Help file consists of two topics, and one of the two topics includes a jump to the other topic.

To build this simple Help file, you will:

- Create a project subdirectory for the source files.
- Define a new Help project using the Help Project Editor.
- Create two topics using Word for Windows: enter context strings for the two topics, code a jump from one topic to the other, add a bitmap to one topic, and save the topic file as RTF.
- Build the Help file.
- Test the Help file by displaying it in Windows Help.

> Note: This section assumes you have already installed Help Author. If you have not, please go to Chapter 5, Using Help Author, and read the installation instructions before going any further.

Creating a Project Subdirectory

When you create a Help file using Help Author, you may want to create a separate project subdirectory for all the source files. (You might also create separate subdirectories for the bitmaps in your Help file.) For this example, create a temporary project subdirectory called SAMPLE in the Help Author directory.

To set up the SAMPLE subdirectory

- Enter the following commands at the MS-DOS prompt:

```
cd \helpath (or the directory name you specified during setup)
md sample
```

Starting the Help Project Editor

Help Author uses the Help Project Editor to control the Help creation process. So, to create a Help file, you first start the Help Project Editor. After starting the Help Project Editor, you can define the new project, add topic files, start builds, and view the completed Help file.

To open the Help Project Editor

1. Start Windows if it is not already running.
2. Open the Help Author group window in Program Manager, and double-click the Help Project Editor icon.

   When you first start the Help Project Editor, a blank, untitled Help Project Editor window opens (Figure 2.6).

Creating a New Help Project

If you have never used the Help Project Editor before, it will automatically start with a new, undefined project. But just to make sure, well create a new project for this sample walk-through.

To create a new Help project file

- From the File menu, choose New Project.

Defining the New Project

After you create a new project, you define project-level information, including the title of the Help project, the context ID for the Contents topic, and the version of the Help compiler to use for the build.

To define the new project

1. From the Edit menu, choose Project.
   The Project dialog box appears (Figure 2.7).
2. Type the following text in the Title box:
   ```
   Sample Help File
   ```
   This title will appear in the title bar of the built Help file.
3. Type the following text in the Contents box:
   ```
   topic1_ID
   ```
   We will use this context string to define both the Contents and the first topic in the Help file.
4. In the Help Version box, select the Help 3.1 option button.
   This indicates that you want to use Windows Help version 3.1.
5. Choose OK.

Adding a Topic File to the Help Project

When you create a Help project file, you add all the topic files that you want to include in the Help file. For this sample Help file, we will create just one topic file.

To add a new topic file

1. From the Edit menu, choose Add New Or Existing File.
   The Add New Or Existing File dialog box appears.
2. In the Directories box, double-click the Sample subdirectory.
   Or press the UP ARROW or DOWN ARROW key to select the directory, and then press ENTER.
3. Type the following filename:
   sample.rtf
4. Choose OK.
   Since this file is new, the Help Project Editor asks if you want to create it.
5. Choose OK.
   The Help Project Editor displays SAMPLE.RTF in the project window. Your screen should look like the one in Figure 2.8.

Creating the Topic File

To have a Help file, you must create topics. Topics contain the text and other information that appear in your Help file. You create topics using Word for Windows and the appropriate Help Authoring Template and then save the topics in a topic file. Once you add a topic file to the project, you can edit it without quitting the Help Project Editor.

To edit the sample topic file

- Double-click SAMPLE.RTF in the Project Editor window.
- Or select the file, and choose Edit File from the Edit menu.
  The Help Project Editor starts Word for Windows and opens the sample file. Your screen should look like the one in Figure 2.9.

  Note: All Word for Windows screens in this guide are shown as they appear with Hidden Text and Paragraph Marks selected from the View Preferences (version 1.1) or the Tools Options (version 2.0) dialog box.

Creating Individual Topics

First you add all the topics that you want included in the Help file.

To create a new topic

1. From the Insert menu, choose Topic.
   The Insert Topic dialog box appears (Figure 2.10).

   Note: The text boxes in the dialog box represent the topic footnotes supported by the Help compiler.

2. In the Title box, type the following text:
   ```
   Topic 1
   ```
   Each topic usually has a title. The title identifies the topic in the history list and in keyword searches the user performs.
3. In the Context String box, type the following text:
   ```
   topic1_ID
   ```
   Each topic usually has a context string. The context string identifies the topic within the Help file. For example, to jump to a topic you must provide the context string of the topic string to which you want to jump.
4. Skip Keywords, Browse Sequence, Build Tag, Entry Macro, and Comments.
5. Choose OK.
   Your screen should look like the one in Figure 2.11.
6. Repeat steps 1 through 5 for Topic 2. However, substitute the number 2 for the number 1 (Topic 2, topic2_ID).
   Your screen should now look like the one in Figure 2.12.

Inserting a Jump Hot Spot

Now that youve identified your two topics, you can insert a jump from Topic 1 to Topic 2.

To insert a jump

1. Create a new paragraph after the paragraph containing the words Topic 1 Title, and then type:
   ```
   Jump to Topic 2
   ```
2. Select the text you typed in step 1.
3. From the Insert menu, choose Jump or Pop-Up Hot Spot.
   The Insert Jump or Pop-Up Hot Spot dialog box appears (Figure 2.13).
4. Press TAB.
5. Type the following context string:
   ```
   topic2_ID
   ```
6. Choose OK.
   Your screen should look like the one in Figure 2.14.

Adding a Picture

The next step is to add a picture to one of your topics. Although you can add bitmaps directly to a topic, it is usually better to enter a bitmap reference that tells Windows Help the name of the bitmap file to display in the topic.

To insert a bitmap in the topic

1. Position the insertion point in the empty paragraph after the paragraph containing the words Topic 2 Title.
2. From the Insert menu, choose Graphic.
   The Insert Graphic dialog box appears (Figure 2.15).
3. In the File Name box, type the following text:

sample.bmp

Note: SAMPLE.BMP is one of the sample bitmaps provided with the Windows Help software.

4. Choose OK.
   Your screen should look like the one in Figure 2.16.

Saving the Topic File

Now that youve created the topic file, save it as RTF.

To save your topic file and close Word for Windows

1. From the File menu, choose Save As.
2. In the File Name box, type this filename:
   `sample.rtf`
3. Choose the options button.
4. From File Format, select RFT.
5. Choose OK to save the file.
6. From the File menu, choose Exit.
   If Word for Windows asks if you want to save changes to SAMPLE.RTF, choose No. You already saved your changes in RTF format in steps 4 and 5.

Saving the Project File

Now youre ready to save the Help project file.

To save the project file

1. From the File menu, choose Save Project As.
2. In the File Name box, type this filename:
   `sample.hpj`
3. Choose OK to save the file.

Building the Help File

After you have created topics for your Help file, you use the Help Project Editor to define the build options. Depending on the complexity of the Help file you are building, the Help project file may have very little information or quite a lot. For this sample Help file, you wont need to define any additional options. You are ready to start the build.

To start the build

- From the Compile menu, choose Start.
  The Compilation in Progress window appears, showing the progress of the build and any errors that occur.

Displaying the Help File

After you build a Help file, you use the Help Project Editor to display it in Windows Help to test it and to make sure the features you added work correctly.

To display the sample Help file

- From the File menu, choose Run Help On SAMPLE.HLP.
  The Help Project Editor displays the Help file in Windows Help (Figure 2.17).
  As you can see, Jump to Topic 2 appears in green, underlined text, indicating that it is a jump hot

spot.

To test the jump

1. Move the mouse pointer over the green text.
   Notice that the pointer changes to a hand shape with a pointing finger.
2. Click the left mouse button.
   Your second topic appears, as in Figure 2.18.
   The SAMPLE.BMP bitmap file appears below the Topic 2 text.

Now that you have your Help file working, you can quit Windows Help.

To quit Windows Help

- From the File menu, choose Exit.

Quitting Help Author

Now that you have successfully created a sample Help file using Help Author, you can close the program.

To quit Help Author

- From the File menu, choose Exit.
- Or double-click the Control-menu box.
  If you havent saved your changes, the Help Project Editor prompts you to save the project file.

Building a Simple Help File Without Help Author

This section explains how to create and build a simple Help file using Microsoft Word for Windows. The simple Help file consists of two topics, and one of the two topics includes a jump to the other topic.

To build this simple Help file, you will:

- Create a project directory for the source files.
- Create two topics using Word for Windows: enter context strings for the two topics, code a jump from one topic to the other, add a bitmap to one topic, and save the topic file as RTF.
- Create a Help project file with information for building the Help file.
- Build the Help file with the Help compiler.
- Test the Help file by displaying it in Windows Help.

Creating a Help Project Directory

When you develop a Help file, you ordinarily create a separate project directory on your hard disk drive to hold and organize all the source files. (You might also create separate subdirectories for the text files and bitmaps in your Help file.) For this example, create a temporary project directory called SAMPLE and copy the Windows Help software to that directory.

To set up the SAMPLE subdirectory

- Enter the following commands at the MS-DOS prompt:

```
cd \

md sample

copy [sourcepath]\hc31.exe c:\sample\hc.exe

copy [sourcepath]\hc31.err c:\sample\hc.err

copy [sourcepath]\winhelp.exe c:\sample
```

The sourcepath is the floppy drive letter or other drive letter or path that describes where the files are located.

Creating a Help Topic File

The first step in creating a Help file is to create the topic files. Topic files contain the text and other information that appears in your Help file.

Creating Individual Topics

The topic is the basic building block of a Help file. You create topics using Word for Windows (or other RTF editor) and then save them in a topic file.

To create a topic file

1. Start Word for Windows and open a new document.
2. Type the following text:

   ```
   Topic 1
   ```

3. Press CTRL+ENTER to insert a hard page break.
   Hard page breaks separate topics in a topic file.
4. Type the following text for the second topic:

   ```
   Topic 2
   ```

   Your screen should look like the one in Figure 2.19.

Note: All Word for Windows screens in this guide are shown as they appear with Hidden Text and Paragraph Marks selected from the View Preferences (version 1.1) or the Tools Options (version 2.0) dialog box.

Entering Context Strings

Each topic usually has a context string. The context string identifies the topic within the Help file. For example, to jump to a topic you must provide the context string of the topic string to which you want to jump. You insert the context string using the pound-sign (#) footnote in Word for Windows.

To enter a context string

1. Position the cursor at the beginning of the line containing the words Topic 1.
2. From the Insert menu, choose Footnote.
3. Type a pound sign (#) as the footnote character.
4. Type the following text as the footnote reference:

   ```
   topic1_ID
   ```

5. Repeat steps 1 through 4 for Topic 2. However, type topic2_ID as the footnote text.
   Your screen should look like the one in Figure 2.20.

Inserting a Jump

Now that youve identified your two topics, you can insert a jump from Topic 1 to Topic 2. To insert a jump, you first type the hot-spot textthat is, the text the user will click to make the jumpand then you identify the destination topic for the jump.

To insert a jump

1. Position the insertion point at the end of the first paragraph in Topic 1 and press ENTER. Then type:

   ```
   Jump to Topic 2
   ```

2. Select the text you typed in step 1.
3. From the Format menu, choose Character.
4. Select the Double Underline check box, and then choose OK.
   Formatting text as double underline tells the compiler to make the text Jump to Topic 2 into a hot spot.
5. Position the insertion point immediately after the Jump to Topic 2 text and type:

   ```
   topic2_ID
   ```

6. Select the text you typed in step 5.
   Be sure not to select any of the double-underlined text or the paragraph mark at the end of the line.
7. From the Format menu, choose Character.
8. Clear the Double Underline check box, select the Hidden check box, and then choose OK.
   Formatting text as hidden tells the compiler that topic2_ID is the context string of the destination topic for the jump.
   Your screen should look like the one in Figure 2.21.

Note: When you remove the selection highlight from the text, you will be able to see the hidden text formatting.

Adding a Picture

The next step is to add a picture to one of your topics. Although you can add bitmaps directly to a topic, it is usually better to enter a bitmap reference that tells Windows Help the name of the bitmap file to display in the topic.

To insert a bitmap in the topic

1. Create a new line after the line containing the words Topic 2.
2. On the new line, type the following:

   ```
   {bmc sample.bmp}
   ```

   Your screen should look like the one in Figure 2.22.

   The bmc command tells the Help compiler to position the SAMPLE.BMP bitmap below the text in your topic. (SAMPLE.BMP is one of the sample bitmaps provided with the Windows Help software.)

Saving the Topic File

After youve created the topic file, save it as RTF.

To save your topic file and close Word for Windows

1. From the File menu, choose Save As.
2. In the File Name box, type this filename:

   ```
   sample.rtf
   ```

3. Choose the Options button.
4. From File Format, select RTF.
5. Choose OK to save the file.
6. From the File menu, choose Exit.

   If word hen Word for Windows asks if you want to save changes to SAMPLE.RTF, choose No. You already saved your changes in RTF format in steps 4 and 5.

Saving the Project File

Now youre ready to save the Help project file.

To save the project file

1. From the File menu, choose Save Project As.
2. In the File Name box, type this filename:

   ```
   sample.hpj
   ```

3. Choose OK to save the file.

Creating a Help Project File

Now that youve created topics for your Help file, you create a Help project filea text file that tells the Help compiler exactly how to build the Help file. The Help project file can include different types of information. Depending on the complexity of the Help file you are building, the Help project file may have very little information or quite a lot. At minimum, the Help project file must include a list of the topic files used to build the Help file.

The structure of the Help project file is the same as .INI files in Windowsthe file is divided into sections, each of which contains information or settings for a particular aspect of an application. In the Help project file, the sections contain information about different aspects of the Help file. However, except for the [FILES] section, a section is required only if you want to include a specific feature that it contains.

The Help project file should have the same name as your Help file but with an .HPJ extension. You can use any text editor to create it, as long as you save it as unformatted text (standard ASCII format). For this exercise, save the Help project file in the SAMPLE directory.

Listing Topic Files

List the files that youre using to build your Help file in a section beginning with the following heading:

```
[FILES]
```

Under this heading, type the names of the RTF files to be included in the Help file. Because you have only one topic file, SAMPLE.RTF, the [FILES] section looks like this:

```
[FILES]

SAMPLE.RTF
```

Specifying Options

Although not required, you will usually include an [OPTIONS] section so that you can specify particular build options for the Help file. Enter these options under the following heading:

```
[OPTIONS]
```

To identify the project directory where the source files are stored, to indicate the window title you want displayed in the Help title bar, and to specify the level of warning messages to display during the build, type the following lines exactly as they are shown below:

```
[OPTIONS]
ROOT=C:\SAMPLE
TITLE=SAMPLE HELP
WARNING=3
```

> Note: You also included a bitmap in one of the topic files. But because the bitmap resides in the same project directory as the other files, you dont have to specify anything in the Help project file.

The Complete Help Project File

Once youve finished these steps, your Help project file should look like this:

```
[OPTIONS]
ROOT=C:\SAMPLE
TITLE=SAMPLE HELP
WARNING=3

[FILES]
sample.rtf
```

The [OPTIONS] section, if used, must appear first in the Help project file.

Save this file as SAMPLE.HPJ in the SAMPLE directory. Be sure you save the file as Text Only if you are using a word processor instead of an ASCII text editor.

Building Your Help File

The final step is the simplest.

To build your Help file

- Using the Windows Run command (or from the MS-DOS prompt), run the Help compiler from the SAMPLE directory, giving the name of your Help project file.

  For the sample you created in this chapter, type the following command:

  ```
  hc sample.hpj
  ```

  The compiler builds your Help file and displays messages during the build. When it is complete, you have a Help file named SAMPLE.HLP.

Testing Your Help File

After you build a Help file, open it in Windows Help to test it and to make sure the features you added work correctly.

To start Windows Help

1. From Program Manager, press F1 or use the Run command to start Windows Help.
2. After Windows Help is running, choose Open from the File menu.
3. Type the name (and path if necessary) of your Help file (SAMPLE), and then choose OK.
   Or use the Browse button to locate and select the SAMPLE.HLP file.
   When Windows Help first opens the Help file, it appears as in Figure 2.23.
   As you can see, Jump to Topic 2 appears in green, underlined text, indicating that it is a jump hot spot.

To test the jump

1. Move the mouse pointer over the green text.
   Notice that the pointer changes to a hand shape with a pointing finger.
2. Click the left mouse button.
   Your second topic appears, as in Figure 2.24.
   The SAMPLE.BMP bitmap file appears below the Topic 2 text.

Quitting Help

Now that you have your Help file working, you can quit Windows Help.

To quit Windows Help

- From the File menu, choose Exit.

Chapter 3 Designing Your Help System

The material in this chapter is a guide to only the first step in designing a Help system. It is an important step, but it may not take you as far as youd like to go. To further your understanding of online design issues, consult some of the many excellent books available on these subjects. For example, look at some books on user-interface design, screen design, graphics design, instructional design, hypertext, and online Help systems. These books go into much greater depth than we can go into here.

This chapter provides useful background information about designing online Help files. It offers general principles and guidelines you can follow to get started thinking about the design choices and decisions you will have to consider when creating your Help system. Specific guidelines for creating Help files consistent with Microsoft Windows are in Chapter 4, Help Authoring Guidelines.

Designing for People

Designing Help systems concerns designing for people, and people are reached by design not only as consumers but as coworkers who participate in the finished design. Every step must be acceptable, understandable, and convincing to enlist the users necessary cooperation. And the final result must be appealing, both rationally and emotionally. Therefore, the range of individual responses to your design must be taken into account.

One cannot have designed very many Help systems without developing a healthy respect for the uniqueness of individuals, for the complexity of human interaction, and for the usefulness of listening to users and watching their facial expressions while they use your system. Human factors such as usability and consumer advocacy are thus daily challenges in this work, not just industry jargon and academic abstractions.

What Help Authoring Requires

To create a well-designed Help system requires skill. The skill in Help authoring is the ability to adapt means to ends. Its working tools are words, pictures, and hypertext links, which are employed in endless ways to produce a variety of effects. Its audience is the users mind, which must not only be instructed but also kept interested and on track.

Help authoring has its basic skills; it also has its higher effects of style and of individual skill. This chapter can deal with some of the basic skills; the higher skills, however, Help authors can be trusted to discover on their own.

A Rhetorical Definition

In designing human interfacesand your Help system is after all, one kind of interfaceit is useful to remember that we are engaged in the business of convincing each other, that we are engaged in a rhetorical process. When the structure, the organization, the text, the graphics, the links, and the various details are skillfully used to produce an intended effect, we call that effect rhetorical. In its most general sense, rhetoric is simply a form of communication. In practice, almost every communication is rhetorical in that it uses some device to try to influence the thought and actions of an audience.

The Help author also has a rhetorical purpose, and rhetoric includes the ways in which the Help authors intended purpose and design are accomplished. For every author should work with a specific design in view. That design object may be merely to give users plain information; it may be to teach them important concepts; it may be to change users opinions about something; it may even be to amuse and entertain them. But whatever the specific design objective, your primary purpose is to make others see the subject as you see itwith the same clearness, the same fullness, the same understanding.

Because designing Help files is a rhetorical process, the Help designer must be clear about the Help files purpose right from the start. Otherwise, the result may be interesting or entertaining, but not necessarily effective or helpful to users. Therefore, your first task is to identify the objectives you hope to achieve by creating the Help file. A good deal of the early part of the design process should be spent gathering information about the product and target audience and in planning the system.

Choosing an Audience

Ever since Aristotle defined three kinds of rhetoric on the basis of the three kinds of audiences a speaker can address, the concept of audience has become a permanent part of rhetorical theory. Although we have come a long way from the Greek and Roman orators, we are not so far from their tradition. We still perform market surveys and usability studies to determine the correct audience for our software product, and we often begin our documentation plans with a statement about the intended audience for this Help system. But how close to the truth are we when we choose our audience?

Today, technical writing addresses a very diverse audience. We cannot assume that all our users will share the same cultural history, grow up in our social class, attend the same schools, or even share our native language. Even in an English-speaking market, users vary widely in age, experience, computer expertise, education, and interest. They also bring different vocabularies, expectations, cultural backgrounds, and habits to bear.

Your task as Help designers is to explain complex and technical procedures to a diverse audience. As our products reach into international markets, these problems multiply. Thus, the challenge to provide effective Help information to an increasingly diverse audience becomes greater each day.

Audience Expertise

Despite the difficulties of identifying a target audience, you must still make some assumptions about the people who will use your Help system. Designing Help files requires that you understand your intended audience and anticipate how they will use your information. User-interface designers have spent years analyzing the traits of users and classifying users into various categories. For example, one way to classify users is by identifying their computer skills.

Such schemes may work well for research and conceptual design, but they are hard to apply in practice. In the real world, these schemes may not hold as well because real users tend to overlap two or more categories that the researcher defines. However, this information is still useful because the users background often determines what kind of information you make available in your Help system and how you present the information.

The following table shows the most common ways to categorize user expertise.

User	Background
Novice	Knows little about computers, little or nothing about your application, and is new to Windows. Novice users are enthusiastic but are afraid to make mistakes. They have trouble getting started, not knowing what is important and what isnt. Novices also lack an understanding of basic skills and terminology. They usually try Help but may be intimidated by its features.
Intermediate	Has some knowledge of computers, has mastered the basics of an application or a specific area, and has a basic understanding of Windows. Intermediate users make errors but can overcome most through trial and error. They refer to documentation frequently to find answers but lack a complete understanding of most operations. They know where to find Help and are familiar with its basic features.
Expert	Completely familiar with computer operations, has a good understanding of MS-DOS, and has extensive experience with Windows. Expert, or power, users thoroughly understand how to use the product. They prefer shortcuts and alternative ways of doing things. They generally use Help only to look up specific information but are comfortable with Help when they do use it.

Keep in mind that these descriptions are very broad and serve only as a method to compare categories of real users. Also, a particular user is likely to have various levels of knowledge and expertise. For example, the expert in word processors may have no experience using spreadsheets, and the Windows novice may be an MS-DOS expert. When designing the information for your Help file, it may be helpful to work out a more precise description of the target audience.

Who Uses Help?

Current software products are complex, and complex products require complex Help systems. Even simple applications come with hundreds of pages of documentation, and Help systems, and online tutorials. Despite all this information, frequently they are left standing on the shelf or deleted from the hard disk because people dont read manuals or use Help. The reason they dont is that often the documents dont tell them what they want or need to know.

Because we cannot afford to produce information systems that nobody is expected to use, we must take care to design systems that provide useful information, that accommodate diversity, and that people actually use. That result is attained through effective design.

Why Do People Use Help?

As you design your Help file, determine why users will want to read the information you provide. Four common reasons why people use Help are:

- For reminders.
- Many users refer to Help to remind them of something they have forgotten. For example, users frequently look up specific tasks and keyboard shortcuts.
- To learn an application.
- Novices and other users often choose Help to learn a new application. Because Help is perceived as part of the application, it is the easiest to access.
- To find specific information.
- Many users request Help only when they have a specific question. Often, they have tried to do something on their own first and were unsuccessful.
- To explore.
- Users sometimes browse the Help file just to see whats there, following one topic to the next, gradually tracing a path through the information and building an understanding of it. Hypertext systems like Windows Help are especially compatible with this kind of use.

Why Don't People Use Help?

Almost all Windows-based applications offer Help. Despite its availability, many people have never used Help or have used it only occasionally. Usability studies show that users in test situations try Help once, and if they do not get the answer they are looking for, they do not try Help again. Why?

Three explanations are frequently offered for not using Help:

- Asking for Help often does not provide the right answer.
  Users have a low tolerance for unhelpful Help, and if they dont find what they are looking for, they frequently form a negative opinion of Help and never try it again.
- Asking for Help interrupts the users work flow.
  When requesting Help, users must remember their original question, find the answer to it somewhere in Help, and then remember the answer when they return to the application.
- Asking for Help is disorienting.
  The Help window changes the look of the screen and presents users with its own interface and feature set, which users must master to find information. Frequently, users forget the question that they had when they are forced to learn how to navigate through Help screens.

These three problems are more severe for less-experienced users. Experienced users are more comfortable with Help. They are less worried about getting back to where they were, they are more likely to remember what they were doing because it is more familiar, and their expectations are more realistic (or more pessimistic, depending on their previous experiences with Help).

So What Do Users Want?

Most of us are driven by motivations of one kind or another. Users are no different. Their primary motivation is finding answers to questions, solving problems, or learning new skills. Given that, our motivation as designers should be to make certain that users get what they want.

What each user wants varies, but all users seem to share some general expectations. They want:

- Help to function interactively.
- Context-sensitive Help.
- To keep their application in view while they are using Help.
- Help to explain the consequences of their choices.
- Information about Help itself.
- Summary information about the application (quick reference).
- Conceptual information about the application.

Most users want Help to answer very specific questions, but the question depends on what they are doing and what the computer is doing. Their questions are shaped by their immediate tasks. They want Help to be an expert that knows their situation and what information they need to continue.

Providing Context-Sensitive Help

Users want context-sensitive Help because it is convenient, saves time and effort, and is more likely to keep them from overlooking something important. Providing context-sensitive Help is one of the best design decisions you can make.

Users expect the computer to be user-friendly and to behave the way people do. When they ask for Help, they expect the computer to be aware of what has been going on, and they are therefore impatient with irrelevant Help. In fact, users believe that the computer does know but often refuses to help them.

Users do not expect context-sensitive information from books; they expect to find information in a book on their own. On the other hand, people expect Help to know their context, to interpret their request for Help in that context, and to understand their questions. Peoples impatience with Help and their unwillingness to try it again suggests that they expect Help to behave more intelligently than a book, and suggests that books are an inadequate model for Help because they lack context sensitivity.

Design Issues

Every craft has its design issues. Many issues are specific and limited in scope, but a few are fundamental and have the widest possible application. In Help, three design issues seem to underlie all the others:

- First, design is about making tradeoffs.

  You give up one thing to achieve something else. Design principles cannot be considered absolutes like the law of gravity, although frequently you may have to pretend that they are absolutes to achieve a certain effect. For example, if you want your Help file to have a consistent look and feel, you may have to restrict the types of formatting you use in your Help topics.

  To create a successful Help file, intelligently apply the principles in this chapter. That means you should use your own judgment about how these principles apply to the Help system you are creating. Naturally, these principles do not pretend to know about your product or your Help system. They merely discuss general principles about designing any Help system. Some principles will be relevant to your design and others wont. Ultimately the design is yours, and you will have to discover it on your own. These principles can help, but dont fall into the bad practice (which is so common today) of using a design simply because the technology provides the feature. Remember, successful systems test the skill of their designers, not of the technology or of the design principles.
- Second, design is about relating things to each other to achieve a purpose.

  To design with skill involves more than just choosing the correct elements. A topic design may be perfectly correct, perfectly logical, and yet for a particular subject and product it may be a very poor Help topic. In fixing the problem, we do not want to ask what is right or wrong; that question was already answered when the elements were put together. We ask rather what is better and what is not so good for our immediate purpose in this topic.

  That is the rhetorical challenge: to find the best means and employ them, to avoid what is feeble or vague or heavy-handed and choose what is strong and definite and balanced. What is constantly present in the designers mind is this question of figuring out the best way to produce certain effects. Questions of technology and features come in as something that defines limits and methods but that should never overshadow the fundamental rhetorical focus.
- Third, design is the designers responsibility, not the computers.

  Despite their usefulness, computers are machines and Help tools are tools. Help authors, not tools, are responsible for design. No matter how spiffy the tools become, they wont design the Help file for us. Our job is design; the computers is technology. But sometimes we forget this. We start to think that our tools will do our work. Tools may give us more choices, but they also demand more from us, because they cant tell us which choice to make.

  For example, a graphics program can give us 256 colors to choose from, but it cant tell us which color is appropriate for the picture we are creating. To use graphics software, we need to understand visual communication. Graphics programs can take some of the effort out of producing graphics, but we have to know something about graphics to communicate with these programs. And computers cant help us with that, or even make it easier.

A Word About Design Guidelines

It is worth stressing that design guidelines cannot provide an automatic solution to the design problem. They do not tell designers how to do exactly the right thing or exactly when to do it. Often they are either too specific or too general for a given situation because, to be genuinely useful, they are abstracted from any particular design problem. This reduces their power in any given context, making the guidelines dependent on their user for sensitive and intelligent interpretation. Guidelines are not a stand-alone tool, and they cannot substitute for effective evaluation and iterative refinement within a design. They can, however, provide helpful advice during the design process.

Guidelines emphasize the designers need to understand the intended audience and the tasks to be carried out, the need to adopt an iterative design process, the need to gather usability data on user performance, and the need to consider carefully how the guidelines can be applied in specific situations.

Designers cannot use design guidelines alone to achieve an effective design. A guideline cannot recommend one choice over another because each choice depends on many aspects of the situation. Guidelines are often based on informed opinion rather than on established principles. For that reason, guidelines should be viewed as an informal collection of suggestions rather than as an exact science. Designers will have to make many choices on their own and be prepared to test their decisions in the context of their design.

Nevertheless, the benefits gained from following design guidelines should not be underestimated. They provide valuable reference material to help with difficult decisions that crop up during the design process, and they are a springboard for ideas and a checklist for omissions. Used with the proper respect and in context, they are a valuable adjunct to relying on designer intuition alone to solve interface problems.

Fundamentals

It always helps to start somewhere, especially when the subject is as broad and unsettled as human interface design. While you prepare to design and build your first Help file or your next project, researchers and Help authors everywhere are collecting more data, testing current models, building new prototypes, and revising old theories. Out of this flurry of study and experimentation comes new insights and increased understanding of the problems we face in trying to create better information systems. The following list attempts to summarize the most important design principles to emerge from this collective experience.

Because these guidelines are general, they apply to the entire Help system:

- If you have input into the development of the application, place most of your design emphasis on making the applications interface clear and easy to understand.
- If you can improve the applications user interface, your task in Help will be much simpler, because it is easier to help users along as they encounter small problems than it is to explain a difficult interface. Help should never have to carry the burden of explaining the user interface, but it often does.
- If possible, design the application so that it uses context-sensitive Help.

  Most users dont want to interrupt what they are doing just to get help. They go to Help as a last resort when they cant figure out how to do something on their own. The more Help interrupts them and interferes with their getting work done, the more they will avoid Help. Context-sensitive Help provides a smoother bridge between the application and the Help information. Chapter 19, The WinHelp API, explains in detail how to program an application for context-sensitive Help.
- Whenever possible, simplify, simplify, simplify.

  Computer applications are becoming increasingly complex and difficult to use. If the Help system also becomes as complex and difficult to use as the application, users may not use it. Because Windows Help is a very rich, full-featured application, try to lighten the users load by making the Help information simple and easy to understand.
- Design your Help system for the online medium.

  Although you can buy programs that convert printed books into electronic documents, the resulting online documents are no substitute for a well-designed Help system. Online information is fundamentally different from printed material and offers advantages not available in books. Hypertext linking allows for more associative patterns of user interaction with the information. Converting a book to electronic form and then allowing users to page through it has no advantage over a book and the disadvantage of being harder to read and use.
- Provide different kinds of information to accommodate different learning styles and different individuals.

  Most users have two basic questions about software features: What is it? and What can I do with it? The first question can often be answered by providing introductory, conceptual information and help with terminology. The second question requires how-to, task-oriented information. Ideally, a Help system should address both these questions as well as provide information that falls into other information categories.
- Provide interactive screens of information that let users participate in learning.

  When using an application, users have some control over what they do. The application offers features, but the user chooses which features to use and when. Help should not take away that freedom of choice by limiting the users participation. If users are not actively engaged in the learning process, they may lose interest. Helps hypertext facility gives authors ample opportunity to create information that lets users browse and try things out on their own, without strict guidance.
- Group information in meaningful structures.

  One of the problems with online hypertext systems is that they have no physical concreteness. Users feel lost because they cant see the size and shape of the information. Presenting the structure and organization of your Help system clearly relieves users from having to figure it out. Grouping information in small, meaningful chunks also provides more ways for users to sort out and identify the parts they want.
- Accommodate different strategies for finding information.

Users require more than one path to the same information because they have different needs and levels of experience. Intermediate and expert users often rely on Search to find information because they feel comfortable with computers and know more or less what they are looking for. Novice users, on the other hand, prefer to browse until they find the information they want. Limiting your Help system to one or the other strategy limits its overall usefulness.

- Rely on the graphical user interfaces visual strength when creating your navigation screens.

  From about the age of five or six, the visual world becomes a more powerful influence on our understanding of the world than all the other senses. Character-based interfaces and text-only presentations require users to operate in a highly symbolic world, relying heavily on their memories to execute and repeat actions. Graphical interfaces can use visual cues to their advantage by offering users choices that require only recognition and not memory. Also, because everything in a graphical interface is visual, the Help system seems more natural and less intimidating if it is visual.
- When using graphics in your Help system, apply metaphors from the real world and use them consistently.

  The point of using metaphors is to make something abstract and difficult to grasp more concrete and comprehensible. Users already have a good understanding of average things from the real world. Build on that understanding by choosing metaphors that users can relate to easily. Once you establish a metaphor, dont undercut it by applying it inconsistently or randomly throughout the Help system.

Hypertext

Windows Help is a hypertext system in which users navigate through two-dimensional space on the computer screen. Hypertext simply means that information (the electronic pages) is linked in such a way that users can read the material nonsequentially. Because hypertext is possible only in the electronic medium, it has no corresponding equivalent in our familiar world of books and magazines. That makes it more difficult for people to catch on to, but more flexible once people do understand how to use it. Its primary advantage over printed materials is that it provides better ways to access information. But access alone is not communication; it isnt even information.

For hypertext to be successful, it must support a well-organized information structure. If the hypertext is well made, users should be able to choose links according to the associations they form when they read the information. Merely adding links willy-nilly to topics does not make a Help file usable. If you get carried away with the idea of hypertext, you will likely make a mess of things. On the other hand, if hypertext links are used skillfully, they can improve the informations organization and make it more accessible to users. They can provide new pathways through information without destroying familiar paths.

Hypertext is good for:

- Encouraging users to explore information.

  Because users are not forced to read the information in any particular order, hypertext allows much more freedom than printed documents. With hypertext, you can create the Help file to fit the information instead of forcing the information into conventional formats.
- Providing different ways to access information.

  In hypertext, users can choose different paths through the information. They can look at examples, read overviews, find keyboard shortcuts, or view related topics. They can also skip any information they arent interested in reading.
- Supporting associative thinking.

  Hypertext presents a model that is much closer to our natural way of thinking; it incorporates a three-dimensional network of topics and links that we can access by association. So, if designed properly, these networks should be superior to books for storing and accessing information.
- Presenting information in different forms.

  Because information exists electronically, it can be used and reused with ease and at virtually no cost. This makes hypertext ideal for accommodating different users and different learning styles. For example, you can organize some topics as a reference manual and another group of topics as step-by-step procedures.

Problems

Hypertext also has some well-known problems:

- Hypertext is disorienting.

  The most common problem that users have in hypertext systems is getting lost. This is especially true for first-time users who have trouble creating mental models of the information space.
- Hypertext lacks structure.

  Because users can choose their own order, writers must provide a structure and organization that makes exploration possible and yet does not become too disorienting or too limiting for the user.
- Hypertext hides information.

  Many hypertexts contain volumes of information, but almost all of it is hidden from view because it is displayed one screenful at a time. Getting users to the information they want quickly and directly is often difficult.
- Hypertext authors lack experience.

  People have been producing printed documents for centuries, but we have little experience with electronic documents and hypertext. And yet, creating a successful hypertext requires skill and experienceif you create meaningless links, you may well end up with an ineffective Help file.

Issues

When designing the hypertext structure of your Help file:

- Provide a home for your users.

  Because users get lost so easily, they need a place where they can return to recover their bearings. Windows Help provides a Contents button as a standard feature, which you can use as the home topic. Design this topic to introduce the document, show how it is organized, and provide convenient access to the rest of the information.
- Create the illusion of simplicity.

  If you create a simple interface and organization for your hypertext, you can provide large amounts of information without burdening the user. However, if your organization is complex and confusing, just a few topics may prove overwhelming.
- Follow the golden mean.

  Hypertext encourages excess. Once you get the hang of it, you are tempted to overuse it. But hypertext is only a way to organize information. If you provide too many links in your topics, users may think that the point of the information is not to read it or understand it, but to see where the next link takes them.
- Plan for the future.

  Hypertext takes shape as you create it. If it is very large, it may not be long before you have trouble comprehending all the details. Planning can help. Decide how you want to structure the Help file before you begin. Then create templates and placeholder topics to hold the information. As you proceed, create links and structure, even though they may lead to dead ends. By the time you finish the writing, you should have most of your structure in place.
- Practice makes expert.

  Hypertext is difficult to create, so it requires practice to become skilled. The best way to learn how to use hypertext is to create a few simple projects and keep trying new ideas until you understand the concepts. Then try creating something more difficult.

Navigation

If users are to develop an accurate model of your Help system, their position in the Help file must be clear. They should understand at all times where they are and where they can go. They should also understand the relationship between their present position and the location of other topics. And there should be a clearly defined route to related topics and an easy way for them to move around.

Developing a useful understanding of any system is essential to provide continuity in the Help file. This can be achieved by providing information relating the current screen to previous and subsequent screens. In other words, Help topics should indicate where users are, how they got there, and what they can do to continue. Without such information, users may find it difficult to keep track of how far along in their mental plan they are, making them less confident of their actions. This in turn may affect their efficiency and make them more prone to errors.

Information that preserves continuity should come at the beginning and end of each screen. You can facilitate navigation by developing a simple and logical structure that users can understand easily, and by providing location pointers and status information. These can take a variety of forms depending on the context. For example, the Help file could represent each category of information in the Help file with a different identifying icon.

You can facilitate navigation by placing orienting information in each topic that relates it to those preceding it and those following it. In this scheme:

- Orienting information at the top of the screen should relate the current screen to the preceding screen, identifying what the preceding screen was and why the current screen is being presented. For example, an icon identifying the topic category or domain could be positioned alongside the topic title.
- Orienting information at the bottom of the screen should relate it to the choice and consequences of actions available in the following screen. Placing cross-references at the bottom of a topic gives users choices for where they can go next.

Information Structure and Layout

Before designing a topic layout, you should analyze the information and decide what is the best method for presenting the information. To lay out screen elements effectively, you must understand the structure of the information. Screen design involves using visual properties such as space, position, and size to communicate structural relations.

There are three principal relations between information elements that you can convey by using visual organization:

- Elements may be grouped or associated together (or disassociated).
  For example, a caption associated with a specific picture.
- Elements may be sequenced or ordered.
  For example, a series of numbered steps.
- Elements may be assigned a relative importance.
  For example, a title in comparison with the rest of the topic.

To help you clarify these structural relationships when considering the layout of a Help topic, try the following exercise. Become a user, stand back from the screen, and ask questions such as:

- Which items are of the same kind?
- Which items are functionally related?
- Which comparisons are easy to make? Which are harder?
- Are there items that can be linked together?
- In what order should the various elements be presented?
- Does the order agree with the information presented?
- Which are the most important items?
- What information is least dominant? Most dominant?
- Is there a hierarchy of importance?
- What information stands out most clearly?
- Do the important structures come across clearly?
- Can I tell a simple story that summarizes the information?
- How could the display mislead users?

Make a list of the kinds of information on the screen, going from the visually most dominant to the least dominant. Now consider whether this order makes sense given the message being put across. If necessary, change the order or emphasis of elements in the topic. Then redesign the topic to convey this new structure.

Visual Grouping

One of the most important aspects of screen design is visual grouping. Visual grouping refers to factors that cause some parts of an image to be seen as related. In visual terms, users are more likely to see parts that are similar in some way as related to each other than parts that are dissimilar.

An example of how visual grouping is applied to topic design is in the use of white space. Generally you should include more white space around a group of related items rather than between each item. The white space surrounding the items will help clarify the group structure of the information.

Visual similarity includes these factors:

- Size
- Type style or shape
- Spacing
- Alignment
- Color
- Brightness

In contrast, visual separation is based on a dissimilarity of these factors.

Visual Order

Determining the order of information in a topic depends on the message you want to convey. Usually, this order is a straightforward sequence from top to bottom, but it may involve more complex movements. In fact, you can influence how users read the information by the way you order items in your topic layout. Guiding a users eye to the most important information is one of the key objectives you have in screen design.

When first looking at a topic, users generally scan the whole layout to get an overall impression. After that, users tend to focus on elements that have emphasis. The following list summarizes these tendencies. Keep in mind that these are tendencies and not absolutes:

- Users tend to scan to the left and upward.
- Users tend to scan in a clockwise direction.
- Users prefer horizontal scanning to vertical scanning.

Issues

- Instructions, cues, and important information should appear at crucial points in the order.
- When deciding where to place information in the order, keep in mind that screen areas have this order of importance: left is more significant than right, top is more significant than bottom.
- Tell the story visually.
- Place items in the topic so that the information tells the story visually. But take care not to let primary elements appear in too many places, or the topic may begin to look chaotic and confuse users.

Visual Hierarchy

Very little information exists in which all the elements have exactly the same importance, and there is no visual hierarchy. More commonly, information has a definite hierarchy of importance, in which some elements receive more emphasis than others. This is especially true in instructional materials like Help.

The effectiveness of the various means of emphasis derives from their perceptual qualities. Visual emphasis relies to a large degree on the effects of contrast. If one item is larger than the rest, it stands out. But if all but one of the elements are large, then the single small element is the most prominent.

There are many ways to create emphasis; the important thing is to set up a consistent visual hierarchy in your Help topics by determining which elements to emphasize and to what degree. These principles can help you do that:

- Visually emphasize the most important items in a topic.

  This will direct the attention of the user toward what matters most. For example, using a topic title in a larger type size and placing it at the top emphasizes it over the rest of the information in the topic.
- Capitalize on visual conventions and expectations.

  Emphasis is influenced by what is usual. For example, because most Western people read from left to right and from top to bottom, information placed at the top left receives more emphasis than information at the bottom or right part of the Help window.
- Make an element different from the elements surrounding it.

  The number of elements and each elements position in the topic affect how it is perceived in the visual hierarchy and what you must do to give it more emphasis. For example, a word buried in the middle of the topic requires more emphasis to bring it out than a heading at the beginning of a paragraph.
- Be careful not to overuse emphasis.

  Too much emphasisor too many words emphasized in too many wayscan create the opposite effect from the one intended. Instead of getting the message, the user may simply ignore it. Also, if you try to emphasize too much, you may make everything roughly equivalent and lose all emphasis.

Creating Emphasis

There are a number of techniques for making visual distinctions between elements and emphasizing those of particular importance:

- A different typeface or type style, such as bold or italic, is the most common way to highlight words and phrases.
- Type size can be used to distinguish and emphasize headingsthe larger the size, the greater the importance.
- Capitals can be used for special words and computer terminology, such as filenames, but should not be used for body text.
- Space can be used to separate itemsthe greater the distance between items, the greater the distinction.
- Indenting can be used to indicate subordinate information.
- Subheadings not only separate items but also summarize their content.
- Color is a very powerful means for distinguishing and highlighting elements; however, it should be used with care.
- Reverse video is very distinctive. However, because it is used in most applications to indicate items that have been selected, it may be confusing if it has a different meaning in the Help file.

Certain forms of emphasis are especially powerfulcolor, for exampleand should, therefore, be used with particular care. Try to establish a hierarchy of importance using these techniques, and then employ that hierarchy consistently throughout the Help file.

Consistency

Consistency is fundamental to effective screen design. It contributes to usability in a number of waysit facilitates learning, lessens the number of errors users make, and helps users develop an accurate model of the system. Of course, it is rarely possible to be completely consistent within a Help file, so you will often have to determine your priorities and make tradeoffs accordingly. For example, you should not be rigidly consistent if the decision results in an awkward design, when an inconsistent design would achieve the goal more efficiently.

The following guidelines can help you with consistency:

- If an element is functionally different, make it look different.

  For example, because they clearly have different functions in Help, jump and pop-up hot spots are differentiated by the type of underlining that they display in the hot-spot text.
- Use topic elements consistently throughout the Help file.

  The learning process is facilitated if the user can easily grasp the relationships between elements within the system. This can only occur if you use consistent terminology, provide consistent behavior for controls, and so on. For example, if users are to choose a blue circle that indicates that a tip is available, consistent use would mean that the blue circle always provides a tip and that no other cue would provide the same tip.
- When consistency isnt possible within the entire system, try to be consistent within a smaller domain.

  For example, if a certain information category, such as step-by-step procedures, requires you to vary from some principle established elsewhere in the Help file, at least be sure to use the feature consistently within all the topics of that category.
- Try to be consistent with features and expectations derived from other related systems and applications.
- If there are large inconsistencies between related systems, the users ability to generalize from one to another is reduced. It may also cause them to make more errors because they may assume that actions taken in one application have the same effect in other, related applications. Therefore, if your Help file deviates from Windows or Windows-based applications, users may find it particularly difficult to use and learn. For example, users expect to be able to click grey, 3D buttons, so dont include buttons in your Help file unless they have a similar function.

- If a topic or action is used inconsistently, provide some instruction or warning to users.

  For example, if screen shots of the interface are not normally hot, and you include one that is, tell users that this particular graphic includes hot spots.

Foreground and Background

In any image, some parts tend to be seen in the foreground and some in the background. In a graphical user interface, active and inactive windows create the illusion of foreground and background. For that reason, you can use windows to separate different kinds of information, especially where you want to create content that is somewhat independent from the main Help window.

When placing information in Help windows, use the following guidelines:

- Place subordinate information in pop-up windows.

  Because pop-up windows are displayed temporarily (until the user takes any action), they create less separation between the information in them and in the main Help window. That makes them ideal for displaying subordinate and supplementary information.
- Place independent information in secondary windows.

  Secondary windows share many of the same characteristics as the main Help window. For one thing, they can be displayed or dismissed independently of the main window. That makes them ideal for holding information that is tangential to the content in the main Help window. However, because secondary windows can exist with or without the main window, they should only be used when the information displayed in them is truly independent from the content in the main Help window.
- If you use secondary windows, consider carefully the size and relationship they have to the main Help window.

  Users will likely have other windows open besides Help windows. Too many windows on the screen at one time can confuse and disorient users. There is considerable usability data indicating that user performance deteriorates as the screen image becomes more complex. Users have a harder time finding the information they want, and they make more errors. In general, try to limit the amount of space that additional secondary windows occupy to less than 25 percent of the active screen area.

White Space

White space is the designers canvas. It is the area in which you display your message. It also creates the information boundaries that users rely on to recognize the message. White space includes both negative and positive valuesnegative space is the background, or the part we seem not to see; positive space is the foreground space that lies at the center of attention and dominates.

In online designs, space is two-dimensional, even though many elements simulate a three-dimensional space. Two dimensions have only length and width, but you can use certain techniques to create the impression of depth and volume.

Learning how to use white space is one of the most important aspects of screen design. The key to using white space depends on its ability to group and separate elements into recognizable symbols and images. An easy way to understand this principle is to consider spacing in type. Normal spacing in type allows our eye to recognize the symbols (letters) that form understandable words, as in this example:

GO OUT TONIGHT

If we remove the normal white space between letters, we cannot easily understand the symbols as a message:

GOOUTTONIGHT

In the same way, the proper use of white space in your topic layout will help users recognize and understand the message. To create a good lay out, make sure that all topic elementstopic title, text paragraphs, pictures, white spacecreate understandable images. A topic will look crowded and cluttered if too many elements occupy the space and prevent closure; it will look open if the space is balanced and generous.

Margins

Margins are the blank areas surrounding the information in a topic. In normal screen design, you have four margins to consider: top, bottom, left, and right. There isnt much you can do about the right and bottom margins in Help, so the left and top margins should be your primary concern:

- Left marginwith overlapping windows on the screen, text from one window can become confused with information in another window because there is only a thin dividing line (window border) separating the two. This can make the screen look unnecessarily cramped and cluttered. An ample left margin helps separate Help text and graphics from information in other windows.
- Top marginthe Help window has three horizontal layers above the information that also attract the users attention. To prevent Help menus and controls from overwhelming the display and distracting users, create a generous top margin.
- Balancetry to achieve a balance between margins. In other words, if you increase the left margin, increase the top margin proportionately. That way the information wont appear crowded into one area of the screen.
- Consistencywhen creating custom left and top margins, employ them consistently throughout the Help file. Otherwise, topic elements will appear to jump around on the screen as users move from one topic to the next.

Color

Color can enhance the Help file by adding interest and variety to the information. Unfortunately, color can also create problems that seriously limit its usefulness. Generally, color is good for grouping similar things and emphasizing elements but not good for conveying meaning. Because it is very effective, use it with care. Color can easily overpower the visual presentation of the Help topic and distract the user by drawing attention to itself and away from the other information. Color should complement a good layout rather than compensate for a bad one.

Color is good for:

- Showing relatedness between things.

  Users tend to think of all the red items as being related to each other, all the blue items related to each other, and so on.
- Creating emphasis and showing visual hierarchy.

  Color is a strong visual cue and an effective means of emphasis. Because it attracts the eye, a users eye will always move to the colored item in a black and white field. Bright colors attract the eye more than dull colors. That means colored elements automatically receive emphasis. Handled with good taste and judgment, color can help inform users and direct them to important information. However, when it is overused or used unwisely, color creates the wrong emphasis, and the topic may become difficult to read. Before you consider using color for an element, determine whether you want to give the item that much emphasis in the topic.
- Showing visual ordering, if you use the spectral order.

  In other words, users see red, orange, yellow, green, blue, purple as being in order.

Color is not good for conveying specific meanings. Users can easily differentiate colored items from noncolored items, but they cannot easily associate a color difference with a particular meaning. For example, if you use color to symbolize different functions in the Help topic, users may learn that symbolism very slowlygreen equals jump, red equals note, blue equals topic title, and so on. More than three text colors also make it difficult for users to keep track of the different meanings assigned to color.

Issues

- Use color sparingly, and only when it has a specific purpose in the topic.
- Because color emphasizes, that emphasis can either enhance the information or detract from it, depending on how it is used. If color is used to make the topic look pretty, chances are it will also detract from the information rather than enhance it. So never use colors just for their own sake.
- Do not combine too many colors in the same topic.

  Like other forms of emphasis, if color is overused it loses its impact. Two or three colors are often sufficient. More colors can be used if they help clarify the logical structure of the information, but too many colors destroy the unity of the display and decrease readability because the users eye jumps to the different colors. If black is used for body text and green for hot spots, that leaves only one additional color choice for text.
- Use color consistently.

  For example, if green is used for hot spots, it should represent hotness throughout the Help file, no matter where it occurs.
- Do not use colors in ways that contradict their conventional meanings.

  Green, for example, would not be a good choice for error messages. Remember, too, that meaning also depends on particular contexts: although green suggests ripe when talking about certain vegetables, it signifies untested when talking about a persons experience.
- Use brighter colors for more important information.

  Color has a luminance hierarchy that results in brighter colors appearing more dominant. For example, on a dark background a hierarchy of importance is white, yellow, cyan, green. If followed, the luminance hierarchy can complement typographical variations. If this hierarchy is ignored, the effects are very noticeable and potentially confusing. Distinctions between items that are not intended to imply an order of importance are best made by choosing colors close together in the hierarchy, for example, white and yellow.

- Choose a text/background color combination that maintains a high contrast between the characters and the background.

  Letters on a background of the same luminance are extremely difficult to read because the eye cannot bring an edge into focus. On the other hand, a high contrast facilitates focusing. However, take caution when using dark text on a bright background. The contrast may be good, but the brightness of the display can make reading unpleasant.
- Be sure your design is acceptable in black and white.

  Some users have monochrome monitors, so you should not rely exclusively on color for visual effects. In general, low-contrast colors do not transfer well to monochrome and grey-scale monitors. When displayed on grey-scale monitors, colors are converted to blacks, greys, and whites. That means you must choose colors so that the contrast between items is high enough to create contrast on a grey-scale screen. Otherwise, some screen elements may become invisible or very difficult to see.

Text

Typography is perhaps the most widely discussed area of graphic design. And not surprisingly. We have been using books for centuries, and all the while we have been improving and changing them. Much of the knowledge that has accrued in book design can be applied to the design of electronic text, but certainly not all of it. These days, Help designers must be as familiar with electronic text as book designers have been with paper and print.

Text is something that all Help systems use. In fact, because of limitationscost of goods, time, resources, display technology, and so ontext continues to be the dominant information element in Help files. For that reason, pay careful attention to the design issues that text raises. For example, how much text should you write for one screen? Which font should you use? Should you use color? Can tables and lists present information effectively online?

Text is good for:

- Conveying precise information.

  It is often impossible to judge visual parameters precisely, for example, the exact distance between two cities on a map. Consequently, where accuracy is required, numbers and words are more effective. (For example, maps translate their visual scale into numbers, one inch equals 100 miles, to be more precise.)
- Dealing with abstract notions.

  Pictures are not often used for depicting abstract ideas, such as the meaning of happiness, where traditionally words are much better suited.
- Expressing logical deductions.

  Pictures and diagrams can only roughly approximate logical relationships. Words and mathematical notation are usually superior. For example, Venn diagrams are a useful way to illustrate the principles of set theory, but they are not the primary means of expressing those ideas.

However, text can look dull and unappealing when used alone. A wall of text is especially disastrous online because users become easily overwhelmed by the information. Text-only Help files also may present problems for poor readers.

Legibility and Readability

For text, the most important factors are legibility and readability. Legibility describes how easily the individual letters and words can be differentiated from each other. Legibility involves many factors, but the most important factor is contrast. The higher the contrast, the more legible the text (if other factors are also considered). Most books achieve high contrast, and thus legibility, by printing black text on white paper. But printed materials also use inverse printing and color to gain high contrastwhite text on a black background or dark red text on a yellow background, for example. The same factors apply on computer screens, and so most word processors display black text on a white background. Colored text is also legible on computer screens if there is sufficient contrast.

Readability is similar to legibility but refers more to the ease and comfort with which the text can be read. Obviously the text has to be legible to be readable. But other factors affect readability. For example, the text may be perfectly legible but not very readable if the line lengths are too long or the spacing is too crowded. Another factor of readability is formatting. Users have more difficulty reading text if it has a lot of formatting changes. Too many colors or frequent formatting changes make the information less readable.

When reading any material, people organize the material by categorizing the various text elements according to their importance. They usually do not read all the words with equal attention and in the same order in which they are presented. You can make reading considerably easier if the text elements are visually distinct and consistent within the Help file. For instance, white space provides visual relief for users and helps them assimilate the information; it also signals divisions within the material.

The ease with which users can read text depends upon several factors:

- Its physical size

  A 10-point font is sufficient for most screen fonts.

- The space between lines (known as leading)

  Paragraph leading should facilitate reading by emphasizing each line of text. If lines are too close together, they can disrupt reading because of interference from the lines above and below the line being read. Adding sufficient space between lines improves the perceptual grouping of characters into rows.
- The length of the line

  How much leading you use depends on the type size and line length. For example, the longer the line, the more leading it requires because the users eye has a greater distance to travel when finding the start of each line.
- The users distance from the screen
- The angle at which the screen is viewed

Most of the time you can control only the first two factors, so pay particular attention to them to create the most readable text possible.

Issues

- Paragraphs should generally be indicated by white space between them rather than by indenting the first line or by using other print conventions.
- Left justification is the simplest and usually the best way to format the paragraphs because everything in the Help window is anchored to the left window border.
- Right justification and center-justified text can be effective for single paragraphs or other special-case text, but it is not advisable for body text because it produces a ragged left margin that disrupts reading.
- Different reading tasks demand different reading styles, and the presentation should be compatible with the required style.
- Is the user expected to read and remember the text, to locate particular items within it, or to react to some items without needing to remember them? For example, standard paragraph formatting is consistent with the users previous experience and is simpler to produce. However, if the user has to pick out one item from a set of items, the items should be displayed as a list.
- Whenever possible, avoid cramming too much information into a Help topic.
- Help text is a form of hypertext, or nonlinear writing. Users move forward and backward through a series of connected Help screens. Instead of flipping pages directly; they flip them indirectly, on the display. Hypertext is usually more effective if you present text in smaller amounts, rather than in long chapters and sections as in printed books.
- Generally, divide the information into small chunks, each topic occupying roughly the size of the Help window.
- This means a topic may consist of 100 words, 50 words, or whatever amount is required to create a complete idea. How much text you place in a single chunk depends on several factors: the content of the information, the size of the Help window, the font size, and the amount of white space. Always keep these factors in mind when considering the amount of text that you include in a single topic. Do not follow print-based conventions such as filling the page with type; these conventions usually have little or no meaning when creating Help files.

Lists

Lists organize information using vertical and horizontal alignment, much like tables. The alignment provides a recognizable structure that tells users the information is related. It also tells them to read vertically, starting from the top of the list.

Online lists arent significantly different from printed lists. You should consider the effects of using indents and nested lists, however. These techniques may work better on the printed page than they do in the Help window. Creating too many margins using indents may make the information more difficult to read. Usually white space is a better alternative than indenting for differentiating lists from the rest of the topic. Use the following guidelines when creating lists:

- If a number of items form an ordered group, place them in a list.

- Because lists define groups, use visual formatting that reinforces and clarifies the lists structure as a group.

  For example, use an equal amount of space between list items and include plenty of white space around the list. Do not introduce different formatting elements within the group, such as bold or color, that differentiate elements and weaken the groups internal structure. In other words, always treat all the items within the group the same.
- Avoid using lists within lists, or nested lists, because they may be ineffective online due to screen limitations.

  You have more control in print because the page is fixed and is larger than the average Help window. A long nested list in a small window may lose its structure and look like blocks of text stacked irregularly on top of each other.

Tables

Like lists, tables structure information into recognizable groups. However, tables provide more control over the information and allow greater flexibility in how the information is presented.

Generally, information within a table invites users to scan in both directionshorizontally and vertically. Therefore, how you design the table depends on how you want users to read the information. In other words, you must decide which orientationhorizontal or verticalbest suits the information. Vertical alignment in a table suggests that the items within the column are the same type (related by information category); horizontal alignment signifies that all items in the row have the same functional relationship to each other.

In printed documents, tables are always a fixed size. In Windows Help, however, you can create relative tables. A relative table is one that adjusts the width of its columns as the user resizes the window. Because the column widths resize dynamically, use a relative table only if you want the information to wrap. If the information in the table requires a more controlled presentation, create a nonwrapping table and have the user scroll to see hidden information when the window is too small to see the whole table. aardvark

Use the following guidelines when creating online tables:

- Take care to ensure that the table reads well horizontally.

  Although all tables have vertical columns, most tables are read horizontally.
- Use white space or thin rules to emphasize one orientation over another.
- When designing horizontally read tables, use ample space between rows but only enough space between columns to separate them. Dont spread the columns out to fill the screen.

  You can also emphasize a tables orientation by adding thin rules between columns or rows. Thin horizontal rules can help draw the users eye across the page. However, dont use vertical rules unless users should read the table vertically; the vertical lines draw the users eye down the screen and make horizontal reading very difficult.

  Using slightly different colored backgrounds for columns and rows also emphasizes the tables orientation but is generally a bad idea because the color may destroy the tables structure and interfere with readability.
- To distinguish the table heading from the table text, use a formatting attribute such as bold.

  Most tables use headings to show the tables orientation and to label the information in the columns or rows. If information in the table is read in both directions, you may want to use both row and column headings.

  Try reversing the order of the table columns or wrapping wider items to two lines if the information you put in tables varies in width, creating irregular spaces or very large gaps between columns.

These techniques should help make elements within a column more evenly matched.

- If a table is very long, you may want to group rows within the table by adding white space between the groups or by dividing the long table into two or more tables.
- However, this implies that the information within the table can be divided into logical groupings.
- If you must mix text and graphics in unusual ways in your Help file, consider using tables to create the effects you want instead of using indents, tabs, or other text-based formatting methods.

- Computers do not arrange text and graphics the way you might for a book or magazineby cutting and pasting things where you want them. Instead you have to rely on the word processor to position these elements in relation to each other. Tables provide a good solution to the problem by letting you place text and graphics precisely in a Help topic.

Graphics

There are many reasons to include pictures in your Help file. For one thing, they can contain a great deal of information in a small space. But pictures often pose the biggest problems for designers because they present so many possibilities from which to chooseshapes, sizes, colors, patterns, styles, and formats. And, of course, adding graphics requires artistic talent. Fortunately, you can hire talent. But you still have to answer the tough design questions.

Like text, a picture is only effective if it fulfills its intended purpose. Consequently, you should determine a pictures purpose before using it. The challenge is to provide graphics that not only look good but that inform and communicate as well. Of course, pictures can have many different functions, more than just the few mentioned here. In fact, a single picture is likely to fulfill several different functions at the same time.

Graphics are good for:

- Focusing attention and highlighting important points.

  In general, the visual sense is pre-eminent in information processing. Pictures catch and focus users attention. Consequently, users tend to assign particular importance to visual elements. Design your pictures so that they draw attention to essential information, especially if the information is unfamiliar to users, differs significantly from what users expect, or is likely to be missed.
- Stimulating ideas and adding interest.

  Pictures have an immediate visual impact. Unless they are extremely flat, they are more interesting than plain text. Most users would rather look at pictures, especially if the pictures are clearly displayed. Pictures also add interest to the information and relieve the visual monotony of text.
- Instructing and clarifying.

  Pictures communicate certain kinds of information more effectively than text and so are essential in many instructional materials. However, because pictures can dominate the information, it is important that they perform a specific function in the overall design of the Help file and that they do not prevent users from grasping other elements in the topic. Avoid using pictures purely as decoration, because decorative graphics distract users from the content.
- Showing complex interrelationships, patterns, trends, and shapes.

  Human vision is adept at recognizing and identifying visual patterns. For that reason, relationships that are confusing or obscure when expressed verbally frequently become straightforward and clear when expressed visually. Graphics, therefore, are useful where the information requires the user to understand complex relationships between items.
- Explaining and describing visual objects.

  Pictures express visual and spatial concepts better than text, especially when those concepts involve physical actions. By resembling the things they depict, pictures take advantage of our ability to recognize objects. Text, on the other hand, must describe visual objects and paint a picture in words. In many cases, a picture is more effective. For example, it is easier to show the honeybees dance than it is to describe it.
- Directing and improving actions.

  Pictures increase the speed at which users carry out actions because they dont have to understand complex processes. The Windows graphical user interface is a good example of how visual elements can make many tasks easier and faster to perform than the same actions in a nonvisual environment like MS-DOS.
- Remaining meaningful when reduced in scale.

  Usually, you can get better results when you reduce a graphic than you can with text. For example, the overall shape of a bar chart is largely unaffected as you make it smaller, whereas text quickly becomes unreadable as you decrease the type size.
- Providing access to other information (hypergraphics).

  When creating hot spots, pictures offer an interesting alternative to text. For example, you can create a graphical table of contents for the Help file, to show how the topics fit together and what links to what. Then, you can make the graphic hot and let users select items directly from the graphic.

- Persuading and influencing users.

  Political cartoons provide a good example of how pictures can be used to influence an audience toward accepting some idea or opinion.
- Conveying information across linguistic boundaries.

  People can understand the ideas and information in pictures much more easily than they can understand a foreign language. In fact, some pictures and images are almost universally understood. Although pictures can transcend language barriers, they may be interpreted in different ways by different cultures. Therefore, if you plan to distribute your Help file to non-English speaking countries, carefully consider international issues when creating and selecting your pictures.

Issues

Graphics take considerable time and effort to create and require much more disk space to store than text. They also display differently on monitors with different screen resolutions. If they are very large, the graphics may take a long time to appear on the users screen.

Of course, pictures can be used in many different ways for many different effects, and each use presents its own design problems and issues. Nevertheless, the next few sections provide a few guidelines to consider, including:

- Consider the overall quality of your graphics.

  When creating pictures, consider what users expect. Because the pictures appear on an electronic screen, users may expect the pictures and screen design to fall somewhere between the standards set by television and computer arcade games. It is important to try to meet users visual expectancy because poorly designed images can cause users to lose interest.
- Dont use too much detail in art that is shown at a reduced scale.

  Images with too much detail at a small scale may lose that detail when shown on lower-resolution monitors. Instead, try using a larger scale but only part of the picture to focus on the relevant information.
- Keep pictures relatively simple.

  Online pictures are displayed at a lower resolution and in a smaller area than printed graphics. So reduce the number of details and parts in each picture and avoid fine lines and tiny text (unless you want illegible text). And be careful not to confuse simple with badly drawn pictures. Simple pictures focus the users attention on important aspects. For example, the silhouette of an object may be sufficient for users to recognize it.
- Avoid distorted views and unusual perspectives.

  If possible, show a conventional three-dimensional view in your pictures. Realistic images are easier to recognize and understand than flat abstractions, unless the picture is a diagram or schematic.

Picture Types

Pictures include screen shots, icons, illustrations, line drawings, diagrams, graphs, cartoons, and photographs. Different picture types are used for different purposes. The type of picture you use depends on how you use it. Consider these two guidelines:

- Try to match the appropriate picture type to its function in the Help file.

  For example, if the purpose is to motivate, attract attention, excite, or amuse, colorful illustrations are likely to be more appropriate than screen shots.
- Before choosing a particular picture type, consider its strengths and weaknesses.

  For example, diagrams exploit the two-dimensional space of the screen, so they are suitable for portraying information that is already spatial, depicting physical and conceptual processes, and explaining structures or relationships. Therefore, diagrams would be a very good choice for a map, the human nervous system, or a family tree. Because diagrams rely on simplification for their effectiveness, they should not look realistic. The concepts matter most. Highly realistic images are likely to make the user think that the diagram represents an object when in fact it demonstrates a process or concept.

Contrast and Emphasis

In all communication, some elements have more emphasis than others. Designers use various devices to improve a pictures impact. One of the most important is contrast. In design, contrast is used to compare two elements to show differences. There are many ways to use contrast in your pictures to add emphasis. For example, you can create contrast using:

- Shape

  Shapes define the function and dimension of space. They can also create depth. Most designs use three basic shapes: four-sided, triangle, and circle. By exaggerating differences in shape, you can add emphasis. For example, using one irregular shape among a group of regular shapes improves contrast and makes the irregular shape more important.
- Size

  If there is low contrast in size (all the elements are roughly the same size), all the information will have the same emphasis. Making one of the elements larger or smaller than the rest improves the contrast and highlights the element that is different.
- Texture

  In the physical world, objects and surfaces have texture (rough or smooth). Our eyes can sense texture visually if the picture uses texture. For example, you can use simple shading techniques to create shadows and give a sense of dimension and texture to an otherwise flat drawing.
- Direction

  Lines and shapes create movement, show direction, and provide emphasis. Using vertical and horizontal movement in your layout can help guide the users eye to important information.
- Color and tone

  Color uses principles of tone to create contrast. Tone refers to the range of light valuesfrom white to black. Tones can be dull, like brown, or they can be bright, like yellow. To create emphasis, you can contrast bright colors and dull colors, or you can use complementary colors, such as red and green.

Combining Text and Graphics

Users learn better when information is presented in both words and pictures. Here are some guidelines:

- Place pictures as near as possible to the corresponding text.

  If you present the text and graphics on separate screens, provide a simple and clear method for displaying the picture. Because jumping to a new topic is more complicated for the user, it is usually better to place the picture in a pop-up or secondary window.
- Use visual grouping to indicate the relationship between the text and graphics.

  The ease with which users can associate a picture with text is dependent upon the visual grouping between them. Learning to see the visual structure requires you to suppress your natural tendency to identify, label, and read the information as a whole. To overcome this tendency, try to consider the display as merely a pattern of shapes. One method designers use for focusing on the overall design is to half shut their eyes, which filters out spatial detail.
- Try to position pictures consistently within a Help topic.

  Unless you have considerable design experience, it is better not to position pictures irregularly in a topic. A good solution is to place them in a consistent position (flush with the left margin, for example) in topics with similar content.

Explanatory Text

Including explanatory text can help users understand the importance of the pictures and see how they relate to other information. For example:

- Include captions and titles when the graphic is separated from the topic (when it appears in a pop-up window, for example).
- Use callouts to label parts of an illustration (callouts can be plain text or hot spots that further explain the part).

- Add comments to explain difficult graphics.

Picture Text

The text that appears in your pictures should be chosen in relation to the text in your Help file and should take screen legibility into account. In general:

- Use the same size text in pictures as body text if you want the entire picture to be legible.
- Use a smaller type size if you reduce the scale of the picture.
- Use greeked text when you dont want users to read it.
- Limit the number of different fonts in pictures.
- Avoid text at an angle.

Chapter 4 Help Authoring Guidelines

If you are creating Help for a Microsoft Windows-based application, you may want to follow these guidelines to ensure that your Help file looks and feels to the user the same as the Help they get in standard Windows-based applications such as Program Manager and Cardfile. These guidelines reflect the experience and thinking of current Help authors in all divisions at Microsoft and apply to Help systems using Windows Help version 3.0 or 3.1.

The Guidelines

Microsoft Help Guidelines are divided into two parts.

Guidelines	Description
Help Organization	Provides guidelines concerning the organization and structure of Help files.
Formatting and Style	Provides guidelines for controlling the appearance of Help topics.

Note: For writing issues such as punctuation, grammar, and terminology, refer to the Microsoft Publications Style Guide (MPSG), if you have a copy. Otherwise, consult another style manual.

Help Organization

Always consider the completed Help file when planning Help for your product. For example, develop a strategy for using secondary windows that can be consistently used throughout the Help file. When possible, work with an online designer to plan such elements as white space, color, leading, and other visual grammar.

If you plan to localize the Help file, work closely with your international team to determine the best localization strategies for your product.

Current Microsoft Help systems include: Help menu, main and sublevel Contents, Indexes, and individual Help topics. This section offers guidelines for each of these elements.

Help Menu

We recommend that Help authors adopt a simple and consistent model for Help menus and include the following standard items on all Help menus:

- Contents (called Index in Help 3.0)
- Search for Help on ...
- About <product> ...

The following items are optional and can appear on the Help menu between Search For Help On and About:

- Index (full alphabetic index)
- Keyboard Guide
- How to Use Help
- Tutorial (or other CBT components; for example, Cue Cards)
- <current context>
- <product-specific items> (for example, Lotus 1-2-3, Tools)

Do not use line separators between the initial navigation entries (Contents, Search for Help on..., and Index). Order optional items and insert line separators in whatever way works best for your product.

Contents Screens

The purpose of the main Contents screen is to present an overview of the information contained in Help and to provide a clear, logical path to information. The Help Contents functions similarly to a table of contents in a book and should follow a similar model. Users are encouraged to use Search, or a separate online index, when they want to look up information as they would in a book index.

Sublevel Contents screens provide a path from the main Contents to individual Help topics. Category entries on the main Contents screen jump to sublevel Contents screens that list either Help topics in that category or other sublevel Contents screens.

- Entries can be listed under separate category headings. For example, reference entries and procedural entries can be separated (Figure 4.1).
- Entries are presented by category, either as text jumps and/or hot bitmaps. List entries should be in learning-path order by frequency of use, or alphabetically (Figure 4.2 shows categories listed in learning-path order).
- The list of entries on the main Contents screen should be 10 items or fewer. Avoid having more than 15 entries if possible. If your list is longer than 15 entries, you should rethink your Help organization.
- Avoid deeply nested sublevels so the user doesnt have to jump more than two or three times to display a Help topic.
- On sublevel Contents screens with entries that jump to individual Help topics, avoid long scrolling lists.

Keyword Lists and Indexes

Keyword lists and indexes are used as navigational aids in conjunction with the Contents screens. The keyword list and the index should be similar to each other and follow the same model as a book index.

Work with a professional indexer to generate the keyword list and index.

Individual Help Topics

The following guidelines address organizational issues. See the next section, Formatting and Style, and the Microsoft Publications Style Guide for general writing style guidelines.

- Optimally, a Help topic is one, and at most two screens long. A Help screen is defined as approximately one half the width of a maximized window and 1520 lines long.
- You can minimize topic length by breaking information into smaller categories, by using multi-column lists of topics, and by creating pop-up and secondary windows for subordinate information.
- Avoid using too many pop-up hot spots on a single screen because they can make the screen hard to read. Each definition should pop up only once, the first time the word appears.

  An exception to this may occur in long topics, where an author includes a second definition for users who have scrolled past the first.
- Dont use jumps in running text that replace the topic in the main Help window. If you must use a jump in running text, use a secondary window so the original topic remains displayed in the main Help window.

  Other alternatives to embedding jumps in running text are:
  - Include an embedded text cross-reference and place the jump in the standard location at the end of the topic.
  - If the running text is a procedure, include enough information in the primary procedure for the user to complete the task.

Formatting and Style

These formatting and style guidelines address the actual look of the Help topics. If you follow these guidelines and use the style sheet in the Help Authoring Templates (WHAT30.DOT or WHAT31.DOT), your Help topics will look the same as standard Microsoft Help files. Because these guidelines address formatting and style issues only, they should be implemented in conjunction with the Help organization guidelines in the previous section.

These guidelines assume that you will be using Help Author and the Help Authoring Templates. The authoring templates include formatting styles that match the recommendations in the guidelines. Help authors may add additional formatting styles for their project as needed. Wherever possible, additional styles should be based on the existing styles in the authoring templates. A description of the styles can be found in Chapter 5, Using Help Author.

Guideline Organization

Each guideline is organized into two sections.

Principles	Contains the information you need to follow when authoring Help files.
Guidelines	Provides additional information to aid the authoring process.

Included in each section are examples that illustrate the principles and strategies.

Bulleted Lists and Numbered Lists

Bulleted lists are used for series of concepts, items, or options. Numbered lists are used for procedures or sequential lists.

Principles

- Use 4-by-4-pixel square bullets. These look best in all video configurations and are different enough from the procedure dingbat () to avoid confusion.
- Use a hanging indent for list text so that the second line of text wraps flush left with the first line of text.
- Use the same text margin (the distance from the left margin to the place where list text wraps) for both bulleted and numbered lists to align the text paragraphs of all lists in the Help file.

Follow the Microsoft Publications Style Guide for formatting numbered lists:

To create a numbered list

1. Use an infinitive phrase for the procedure subheading.
2. Make the subheading bold but do not use a colon.
3. Type the step number, press TAB, and then type the text for the step.\sgmlli2p Dont use a period after the number.
4. Repeat steps 13 for the other steps.

   Capitalize the first word of each bulleted list item.

   Use a period after each bulleted list entry if it completes an introductory phrase, but not after brief or one-word entries.

Strategies

If you want to include parenthetical information that supplements a procedure, use pop-up windows.

Color

Color has the following qualities that you need to consider when planning your Help system: color attracts the eye, adding emphasis; items that are the same color appear to be related, while items that are different colors appear to belong to separate groups; and colors have different associations for different people and cultures. In addition, much less detail and subtlety of color is available online than in print, and poor use of color may even cause user eyestrain.

Principles

- Add color to your Help files only with extreme care. Overuse of color actually makes information more difficult to process, because the user slows down to think about what the different color means. Avoid adding color to text, in particular, as it is difficult to read, and its meaning may be easily confused with Helps green hot spot text.

  If you absolutely must add color, use it to convey information about structure, such as grouping or hierarchy, rather than to imply a particular meaning, or as decoration.
- Always be consistent with the use of color. For example, since green is the defaul hot spot color, avoid using it for anything other than jumps or hot spots.
- Do not depend on color coding alone. The users monitor may not have color or may not correctly display colors. Some users may be color-disabled.
- Design for VGA colors as a best-case scenario. Avoid the brighter colors of the Windows 16-color palette because bright colors can cause after-images, complementary (opposite) colors appear to vibrate, and both can cause eyestrain.
- Dont rely on color to convey a particular meaning. Red, for example, implies warning in one culture, and happiness in another. Consult your localizer about color choices.

Strategies

- If you must add colors to your Help file, add as few as possible. Remember that there are already five colors in the Help interface, (black, white, light gray, dark gray, and blue) and green hot spots. Too many colors distract the eye. Consider designing illustrative or navigational graphics with a limited palette.
- If you must use colored text, use strong contrast. Do not put light text on a light background or bright text, like bright-green, on a bright background, like magenta.
- Its helpful to use colored text for the hidden text coding in the source files. It stands out and makes it easier to debug and localize the topic files.
- Help does not do any special color mapping for monochrome. It lets Windows do it.

Context Sensitivity

Context-sensitive Help provides the user with specific Help on actions and parts of the screen. Areas for which you can provide context-sensitive Help include menu commands, dialog boxes, error messages, language elements, command-line commands, fields, functions, screen regions, and tools.

The user receives context-sensitive Help by pressing F1 in an open menu or dialog box, or by choosing the Help button in an error message or dialog box. The user can also request it by pressing SHIFT+F1 or clicking the Help toolbar button and selecting a screen element, field, or menu item.

Pressing F1 or SHIFT+F1 provides the same context-sensitive Help but suggests different user motivation. By pressing F1, the user is asking for Help on the current context. By pressing SHIFT+F1, the user is choosing the context-sensitive mode in anticipation of requesting Help on a particular command or item.

Principles

- Provide specific, context-sensitive Help for menu commands, dialog boxes, fields, parts of the screen, tools, language elements, functions, and statements.
- Help on error messages should be context sensitive.
  If you choose to write Help topics for error messages, write a unique topic for each error message. Work with developers to eliminate redundant messages.
- Write a unique Help topic for each tool. Avoid long tables listing tools and their descriptions.

Strategies

- Negotiate with development and program management for greater depth of context sensitivity.
  Developers assign context identification numbers to parts of the product, so they determine how much of the product can be context sensitive. They are also responsible for the functionality of the Help tool and the placement of Help buttons in error messages and dialog boxes.
- Include a Help tool on the default toolbar for products that use toolbars.
- Include a Help button in error messages and dialog boxes as a visual reminder that context-sensitive Help is available.

Cross-References

Cross-references are jumps within Help, references to print-based documentation, or references to the online tutorial.

Principles

- Place cross-references at the end of a topic under the subheading See Also. This subheading replaces Related Topics. It may be either plain text or bold.

  When you make your decision about whether or not the See Also subheadings are bold in your Help file, consider whether the See Also information should be on the same level as other subheadings in the topic, or whether it is subordinate information. (One exception to placing cross-references at the end of a topic is making them accessible from the nonscrolling region if the cross-references are used for more of a navigational tool than for related information.)
- Cross-references are placed at the end of a topic so that the user can finish reading the topic before deciding which information to consider next. Avoid confusion by using the same wording for jump hot spots as the topics they refer to.

  It is not necessary to indent cross-reference lists without subheadings since they are introduced by See Also.
- Do not use a colon after the See Also subheading if it is bold.
- Align the See Also subheading, subsequent cross-reference subheadings (for example, Tutorial or Users Guide), and text flush left. Capitalize initial letters.

  If the cross-reference subheading and cross-references that appear below it are the same font, the latter may be indented to help differentiate it from the subheading.

  Some topics contain information that is relevant to several pieces of documentation. In these situations, use subheadings to group the different cross-references, preceding them with the standard See Also subheading. Align the cross-reference subheadings with the See Also subheading. To help differentiate the cross-reference text from the subheadings, indent the text, as in Figure 4.3.
- If you display cross-references in pop-up windows, either format the See Also subheading as a normal jump (green, underlined, with no bold) or place a hot bitmap to the left of the subheading.

  There are two ways to indicate that cross-references are contained in a pop-up window. You can format the words See Also as a normal jump hot spot, without bold formatting, or you can create a cross-reference bitmap and place it next to the See Also subheading, as in Figure 4.x.

  The user clicks the bitmap to see the cross-references (Figure 4.x).

Strategies

- Page numbers in cross-references can be difficult to track and update. Unless you have a system for tracking and updating page numbers, avoid putting them in cross-references.
- By adding cross-references in pop-up windows instead of the main topic, you further subordinate the cross-reference information and set it apart. The advantage is that you can create links to a topic without overburdening the visual presentation of the main topic. The disadvantage is that the user has to display the pop-up window to see the available cross-references.

Examples

Examples within topics are presented in various ways, depending on their number and type.

Principles

- Include short examples (one line or less) within the topic text where they best fit, without a subheading. You can introduce them with for example or for instance.

  List several examples within a topic below the subheading Examples. Align the subheading and examples flush left. Format them as either plain text or bold, depending on the amount of emphasis you want subheadings to have (Figure 4.x).
- Place subordinate examples (text or graphics) in pop-up windows.

  Do not include important examples in pop-up windows since this hides the information from the user and, and it cannot be copied.
- Place in a separate topic, examples that are too long to include within the topic or in a pop-up window. Use this solution for extreme cases only, because it creates added navigational difficulty.

Strategies

- Examples work well in secondary windows when you want to display the information along with the main Help window.
- Pop-up windows work well for examples that provide nonessential information.

Fonts

Fonts provide a unifying feature to Help files without sacrificing creativity and without interfering with the particular needs of individual products and users. Fonts affect readability and eye fatigue, and provide visual cues to the organization of information.

Principles

- Use the Normal style, 10-point plain MS Sans Serif, for body text in Help topics.

  This is the most readable screen font available. If you are shipping a product that will be run on Windows version 3.0, use 10-point Helv. Do not use Helvetica.
- If a bitmap is included within a paragraph, change the line spacing to Auto to avoid text overlap.
- Avoid mixing two font sizes in body text.

  You can use 8-point text sparingly, for example, in callouts to graphics, in labels, or as bold headings for nested tables.

  Shortcut keys and keycaps are an exception, as recommended in the Microsoft Publications Style Guide. Use 8-point CAPS to format shortcut keys (Figure 4.x).

  (Do not use the small kaps formatting attribute because the text may not wrap correctly.)
- Use a single font style for body text. It increases readablity and visual flow.
- If you plan to use color for text other than jumps, use it sparingly and for a specific purpose. Make sure that the colored items are the most important items on the screen.
- Avoid the use of italic except for variable arguments because it is difficult to read. (Italic for variable arguments is consistent with the industry standard.)
- Use plain text with initial caps for book titles.
- Use bold for topic titles, subheadings, and literals (such as programming statements or text you want the user to type). If you want subheadings to stand out, use bold. If you dont want them to stand out, use plain text.
- Use Letter Gothic, Courier, or System font for syntax statements.
- Avoid using TrueType® fonts. Not all users have True Type fonts installed.

Strategies

- If you do use TrueType fonts, consider what you are trying to accomplish, and the fonts accessibility to the user. Either ship the fonts with your product or make sure that users will have them.
- MS Sans Serif is strictly a screen font, so it doesnt appear in the Font list box in Word for Windows. To format text as MS Sans Serif, either apply the appropriate style from the authoring template style sheet or type the font name in the Font box.

Graphics

Graphics include line art, icons, screen shots of the interface, and pictures with hot spots (hypergraphics). Use graphic images to illustrate concepts and present information visually.

Because of the complex localization process, work closely with your international team to determine the best localization strategies for the graphics in your product.

Principles

- If you use a graphic next to a hot spot (in lists of jumps, for example), make both the graphic and the text hot.

  Users tend to click anything that looks like a picture, whether or not you intend it to be hot. Including graphic elements with text hot spots provides users some flexibility, or forgiveness, when choosing the hot spot.
- Since information in the Help window is anchored on the left, position bitmaps flush with the left margin, not centered, when they appear alone in a paragraph.
- If you include graphics within text paragraphs, carefully consider the size of the graphics because they may interfere with the leading.

  As a rule of thumb, graphics smaller or slightly larger than the cap-height will not cause layout problems; graphics larger than the cap-height will throw off the paragraph leading.
- When providing Help on an applications terminology, include graphics of the interface elements along with the definition.

  A great deal of users difficulty in learning new applications is their unfamiliarity with terminology. Many of these terms also have visual equivalents, scroll bars, tool bars, pointers, icons, and custom controls, for example. Showing a small graphic of the term being discussed can help users learn the names of visual elements they must know to use the application (Figure 4.x).
- Avoid confusion between the Help graphic and the application interface when presenting screen shots of the interface.

  The danger in using screen shots in Help is that they are easily confused with the real thing. One way to avoid confusion is to reduce scale, as in Figure 4.x:
- You can often achieve more emphatic results by presenting a partial screen shot. This method also allows you to focus on just that part of the interface you are illustrating (Figure 4.x).
- Or you can frame the screen shot with a colored background to avoid confusion (Figure 4.x).
- Another method is to make the graphics you use for screen shots appear more schematic-like, making them monochrome and simplifying the details, as in Figure 4.x.
- For visual presentation and readability, add white space to the left and right of a bitmap in text, or above and below a bitmap in its own paragraph (Figure 4.x).
- Consider carefully how graphics are combined with text in individual topics.

  Depending on the color, size, and number of graphics in a topic, they can easily dominate the text and draw the users eye to them. If the graphics function as subordinate elements in the topic, their dominance will make the information less readable.
- Use separate graphics for the active and inactive states of custom controls (buttons, for example) that you add to Help topics.

  In Windows, menu commands and dialog box options that are unavailable appear dimmed (grey instead of black), so users learn to rely on the application to control availability for them. Providing similar behavior in your Help file will make your controls more consistent with normal Windows behavior.
- Because of the complex localization process, avoid combining both text and graphics on icons and buttons.

  If you include text callouts or explanatory text in graphics, the entire graphic must be recreated in each language to localize the product. Text that is separate from the graphic can be translated without recreating the graphic. For example, in some cases you can create a table and place the graphic in

one column and the text in another column (Figure 4.x).

Strategies

- Develop a consistent strategy for using graphics that will enhance rather than detract from information.
- Dont burden users with unnecessary, decorative graphics. If the function of a graphic is to embellish the topic (make it interesting, beautiful, or colorful, for example), it may detract from the information presented in the topic and result in less effective communication.
- Be consistent with the visual grammar you create for your Help file by reinforcing the distinction between hot graphics and illustrative graphics.

  If a certain graphic is hot in one topic, it should be hot in other topics where it occurs. This is especially important when using icons because users match the visual appearance with certain behavior. Changing the function of a graphic without changing its appearance will be confusing. If you do not want the graphic to be hot in all cases, create two distinctly different versions of it.

  One way of making the differentiation clear is to use interface conventions for hot graphics, such as three-dimensional buttons (Figure 4.x).
- Or, you can use white space to separate the graphic from surrounding text and provide a well-positioned instructional phrase that tells users that the graphic is hot or not hot.
- Another method is to create graphics with text callouts that are formatted the same as hot spots in the Help interface:
- If the graphics are purely conceptual art, they can be created in a different style from other graphics:

  To facilitate the localization process, contact your international team to obtain the localized text that needs to be inserted in each bitmap and create these bitmaps according to the size of this localized text.

Headings

Headings should convey as much information as possible about a topic and help users locate information quickly.

Principles

- Align top-level topic headings flush left. Use 12-point bold type and initial capital letters. We recommend using the MS Sans Serif font. Use the Heading 1, template style.
- Align procedure subheadings flush left and use 10-point bold type. Use an infinitive phrase for procedure subheadings.

  Or use a dingbat () to indicate a procedure subheading and do not use bold type.
- Decide on plain text or bold for subheadings (depending on how much bold is used in the Help topics) and apply your decision consistently throughout the Help file. You may want to use different styles for different types of subheadings.
- Do not use a colon after bold headings or subheadings.
- Avoid using color in headings and subheadings, especially when the heading is bold. Colored headings can overpower the visual presentation of the Help file.
- Capitalize the first word of procedure headings.
- Topic titles should have white space around them. Heading styles in the authoring templates automatically add white space. If you must define additional heading styles, use the heading styles in the template as a guide to maintain the appropriate amount of white space.

Strategies

- Multiple visual cues for headings, such as bold, color, and indentation, can confuse users. Consider using only one visual cue to identify headings. For monochrome support, remember that you cannot rely on color as a visual cue.
- Icons or bitmaps in headings should complement the text. Consider the users response to color and size when designing icons.

Jumps

In Help, text or a bitmap in a topic can link to a topic in another window, display information in a pop-up window, or execute various actions through macros. It is important that users experience consistent functionality when they choose a jump, pop-up, or macro hot spot.

Principles

- Use 10-point MS Sans Serif for hot-spot text.
- Assign the J1 authoring template style to a list of hot-spot text to add points of leading between the lines and to wrap the hot spot text correctly when the Help window is resized.
- When creating lists of hot spots, align them with the preceding text (Figure 4.x).
  If you use color, a graphic, or some other visual indicator for a jump hot spot, dont use this visual marker for another purpose in the Help file.
- In running text, dont use jumps that replace the topic in the main window with another topic. Users can become disoriented when the main topic window changes before they have read all the text in the first topic.
- If you must use jumps in running text, use a secondary window so that the original topic remains visible.
- Dont use a pop-up hot spot within a pop-up window.
  This replaces the content of the first pop-up window and may cause user disorientation. A pop-up window should be used when you want to give additional information while retaining the content of the original message.
- When a graphic and text are used to indicate a jump or pop-up hot spot, make both hot (Figure 4.x).
- Users generally recall no more than five to nine chunks of information with any success. Dont put excessive demands on the user by having too many options to explore.
- When needed, use text by the hot spot to clarify for the user what will happen when they choose the hot spot.

Strategies

- Consider the overall design of all your Help files when determining how hot spots will appear and act. Provide the user with a consistent look and feel. For example, dont mix jump and pop-up hot spots, or green hot spots and graphical hot spots, in the same list or use the same visual marker for hot spots that result in different actions.
- Limit the number of hot spots in a navigational screen or list, such as Contents, Index, and See Also/Related Topics. If many hot spots are needed in this type of list, consider grouping the hot-spot topics under headings. The headings should not begin with gerunds or jump to a new topic. They should display subheadings for the hot spots (Figure 4.x).
- Include as few jump and pop-up hot spots as possible while still providing the needed information.
  When there are more than six headings in a navigational screen (such as Contents or an Index) or more than six subheading hot spots under a main heading, use a graphic and black text to indicate the hot spots. This helps prevent the green screen phenomenon (Figure 4.x).
- If colors other than the default green are used for hot spots, users must edit their WIN.INI file to change the hot-spot color. Consider including a procedure in the How To Use Help file to help them change the hot-spot color.
- Punctuation that follows hot text must also be formatted as hot to prevent wrapping problems.

Nonscrolling Region

In Help 3.1, you can set aside part of a topic as a nonscrolling region, which is a fixed area at the top of a window with a scrolling region below it. The nonscrolling region lets users keep in view information that has an important bearing on the main topic. You can include nonscrolling regions in both the main Help window and in secondary windows.

Principles

- Use color in the nonscrolling region only if the color supports a distinction between the information in the nonscrolling region and the rest of the topic.
- Be consistent when using nonscrolling regions in topics. Dont use the nonscrolling region for one topic but not for a topic at the same level.

  Limit the nonscrolling region to less than one-half the height of a standard Help window. Less than one-third is even better.
- Avoid using nonwrapping text in the nonscrolling region.
- A Help window does not display a horizontal scroll bar for the nonscrolling region. A user must enlarge the Help window if the nonwrapping text does not fit. There are certain exceptions to this principlefor example, when meaning is affected by wrapping text. Syntax lines often fall into this category.

Strategies

- Try using the nonscrolling region to display topic titles if the topic is lengthy. That way users dont have to remember what topic they are reading when scrolling.
- Keep one kind of information anchored at the beginning of the topic in the nonscrolling region. For example, you can display syntax statements in the nonscrolling region so that users can refer to them when reading parameter descriptions (Figure 4.x).
- The nonscrolling region is convenient for displaying custom controls (navigational buttons, CBT launch buttons, macros executed by buttons) that change with the content of each topic. Because the nonscrolling region is part of each topic, the controls you place in it can be specific to the information in the current topic.
- Try creating a graphic of the alphabet in the nonscrolling region for navigation through an index. Each letter can serve as a jump to a point within the scrolling region. In some cases, you will have no topics that fall under a letter. You can handle this situation in two ways: (1) gray out all letters not active but keep integrity of the alphabet, or (2) create a range of letters, for instance, A through F. Your material should determine your choice. But avoid mixing these two approaches in a graphic.
- The nonscrolling region can contain entire topics, especially if you want to use all the available window space without having scroll bars cover up part of the information. For example, you can create a topic that just displays a graphic in the nonscrolling region so that the window is the same size as the graphic (Figure 4.x).
- Put a Close button in the nonscrolling region of a secondary window. This reminds users that they must dismiss the window. A Close button in the nonscrolling region is always visible and accessible (Figure 4.x).

Notes, Comments, Hints, and Tips

Notes, comments, hints, and tips present information subordinate to the rest of the information in the topic. They are formatted for less visual emphasis within the topic. Whether you use a note, comment, hint, or tip depends on the type of information given.

Notes contain neutral or positive information that emphasizes or supplements important points. Notes can apply to a single paragraph or a whole topic. Comments contain supplemental information. Hints and Tips suggest alternative methods that may not be obvious, and Help users understand the benefits and capabilities of the product. Hints and Tips are not essential to the basic understanding of the text.

Principles

- Place the note, comment, hint, or tip in its own paragraph following the information it pertains to. Format the note, comment, hint, or tip appropriate to the contextnumbered list, table, body text, and so on (Figure 4.x).
- Avoid using color and bold together for the subheading.
- When using plain text for the subheading, add a colon with one space after it.
- When using bold, do not add a colon (Tip).
- Do not add a Comment subheading to comments.

Strategies

Use notes, comments, hints, and tips sparingly so that they retain their effectiveness.

Pop-Up Windows

Pop-up windows are nonmoveable, nonsizable windows that remain on the screen until the user dismisses them. Pop-up windows typically contain definitions of terms or other parenthetical information. Good uses for pop-up windows include illustrations, examples, notes, tips, and lists of keyboard shortcuts.

Principles

- Dont put pop-up hot spots within pop-up windows.
- Help dismisses the existing pop-up window to display the new pop-up window. This can disorient the user.
- When using a pop-up window to define a user interface item, you should include a graphic, if possible, that shows the item being defined.

Strategies

- Remember when deciding to use a pop-up window that the user cannot print the information in pop-up windows, that pop-up window titles dont appear in the history list, and that the user cannot return to a pop-up window using the Back button.
- Avoid placing too much text in a pop-up window since not all of it may fit (Figure 4.x).

Secondary Windows

A secondary window is an independent Help window that has a title bar and scroll bars but no menu bar. Secondary windows are resizable and moveable, and they remain open until closed by the user, whether or not the main Help window is open.

Principles

- Use the same formatting guidelines that apply to the main Help window. This includes use of nonscrolling regions and graphics.
- When deciding on a size for a secondary window, consider the amount of information that will appear in the window. Avoid forcing the reader to scroll horizontally.
- If you define more than one type of secondary window, make sure each type appears in a consistent location and in a consistent size unless the purpose of the window requires special placement or sizing.
- When varying size, it is better to vary the height of secondary windows than their width. Having secondary windows appear in too many locations or in too many sizes can create an unnecessary distraction for the user.
- The following example demonstrates how inconsistent sizing and placement of secondary windows are distracting (Figure 4.x).
- Dont define secondary windows to appear always on top since the user cannot override this author-defined setting.
- Be consistent when using jumps from secondary windows. If not handled carefully, jumps from secondary windows can disorient the user, contributing to the lost in hyperspace syndrome.
- Remember that secondary windows do not have a button barif you jump from one topic to another in a secondary window, consider adding a Back button (or a link to the previous topic) in the window.
- Use secondary windows to display information from jumps in running text. This retains the original context in the main Help window while giving the user access to the additional information in the secondary window.

Strategies

- Secondary windows dont automatically close when you close the main Help window. Its a good idea to add a CloseWindow macro (tied to a hot spot) to give users an easy way to close the secondary window.You can use a hypergraphic to simulate a Close button in a secondary window (Figure 4.x).
- Good uses for secondary windows are glossaries, procedures, examples, sample code, indexes, alphabetic lists, or other information the user may want to have available with or without the main Help window.
- Secondary windows do not have the ability to print or copy their contents to the Clipboard. You may want to add macros that perform these tasks, especially if you are using the secondary window for sample code or examples. You can add these macros by simulating buttons with graphics or with jump hot spots (Figure 4.x).

Shortcuts and Shortcut Keys

Shortcut keys and shortcuts are alternate methods for completing an action. A shortcut key consists of a letter, symbol, or function key, often pressed in combination with the CTRL, SHIFT, or ALT key. Shortcuts can be tools, buttons, or shortcut menus.

Principles

- Use 8-point uppercase for key names, but make the plus sign (+) 10 point (if there is one) so that it is readable online.
- Follow the Microsoft Publications Style Guide for formatting key combinations. Do not put a space between the plus sign and the key names. For example, CTRL+K. Do put a space between commas in a key sequence in which the user presses keys in sequence. For example, ALT, T, P.
- Use 10-point MS Sans Serif (the style that you are using for the list) for describing the shortcuts.
- Shortcuts that are tools or buttons should have the icon to the left of the name of the tool or button and should not be formatted as a jump (Figure 4.x).
- Do not format a list or table of shortcuts as jump hot spots. Instead, repeat them as jump hot spots in the See Also section.

Strategies

Shortcut keys/shortcuts can be presented in a variety of ways:

- In a paragraph within text
- As a list in text
- As a table
- In a pop-up window

Tables

In Help 3.1, you can create simple tables (tables without borders or shadows) using the Word for Windows table formatting features. Table design varies depending on content. The following sections discuss design principles and authoring strategies for working with tables.

> Note: Windows Help 3.0 does not support true tables. To create a table, you must use hanging indents or side-by-side paragraph formatting.

Principles

- Tables are difficult to read online, so dont overuse them.
- You can use bold or plain text for your table heads.
- Use bold for table heads when the table contains commands or options that the user must choose. The bold text directs the users eye to the options. Use 10-point bold (style Th).
- Use 8-point bold (style Th3) when the topic already contains a 10-point bold heading.
- Use a plain head with a rule for tables that contain lists of information (style Th2).
- You do not need to use a rule beneath the table heading.
- If you do use a rule, define either a broken or solid, single-line rule.
- If you dont use a rule, your heading must be bold and you must add 3 points of white space after the table heading.
- Use plain text for table text, and capitalize the first word in each column.
- Do not use a rule above the table heading.
- Align tables left with text that appears directly above the table. Tables with numbered or procedural lists should also be left aligned. Table heads should be aligned with table columns.
- In a table created using a hanging indent, avoid column headings that are much longer than the text within that column. If possible, the column heading should not be wider than the widest item in the column.
- The gutters between table columns should be narrow, based on the way the table is being used. Too small a gutter makes the text crowded and illegible; too large a gutter makes the table difficult to read horizontally. For example, a table designed to be read from left to right should have 1 to 2 picas of space between columns.
- A table whose information is grouped to be read vertically may need more space.
- A table used to position text and graphics precisely may need more space between columns.
- If possible, avoid nested tables (a table within a table). If necessary, follow the same guidelines as regular tables.
- If a topic has more than one table, format all the tables with the same column widths.
- Divide long tables into two or more smaller tables, whenever possible.

Strategies

- Tables provide a way to lay out text in more precise positions, for example, they are a good way to create a gutter between bitmaps and text.
- A table is designed to be read from left to right. The exception to this might be a table with graphics (such as icons or other bitmaps), where the user would scan a table vertically (read down or up). In this case, the graphics would still be placed in the left column.
- If you are creating a two-column list, you can use either the table feature in Word for Windows or hanging indents. However, for lists with three columns or more, use the table feature.
- Be aware of crowdedness in table text versus table columns. Localized versions of Help tend to increase by 30 percent or more.
- Relative tables should not be used if the precise layout of your information within the table is critical.

For example, if syntax statements should not wrap, do not use a relative table.

White Space: Margins, Indents, and Leading

When laying out visual elements, the more white space you place around an element, the more importance it has visually. Adding more white space between elements separates them in a visual organization. Less white space between elements groups them together.

Indents are less effective in online than in print documentation, and so are used less frequently. (When we refer to indents here, we mean variations from the left margin in the built Help file, not the paragraph formatting devices called Indentation that we use to create left margins, two-column lists, and text wrapping in numbered and bulleted lists.)

> Note: An 8-pixel left margin is built into the Help application. We recommend that Help authors add an additional 0.8 inches. (This 0.8" has been added to the styles in the Help Authoring Templates.)

Principles

- Use ample top and left margins to improve legibility in Help topics by separating Help from other images on the screen.
- Use a consistent left margin. It enhances the general legibility that is so important for online presentation and unifies a body of information.
- Avoid using indents online (except the indent necessary to wrap text in numbered and bulleted lists).
  Indenting works best when you can see a whole page (as in printed books). It visually structures the hierarchy of information. Online, indenting loses its context, especially in a small Help window. In addition, indents can give the impression that the left margin is constantly changing when you scroll through a long topic or jump to another topic.
- If its absolutely necessary to use an additional indent, align it with the left margin of text in a numbered or bulleted list. Do not use more than two indents.
- Use minimal white space between items you want to group. Use more white space between items you want separate.

Strategies

- Overcrowding Help topics compromises their legibility. Before reducing margins or removing white space to fit more text into a topic, consider reorganizing the information into smaller chunks or accepting more scrolling.
- Use solutions such as space between paragraphs for organizing topics visually and pop-up windows for subordinating information to compensate for the loss of indents as a layout option.
- If plain text follows a nonscrolling region, consider adding 6 points above the text to separate it visually from the nonscrolling region visually.

Chapter 5 Using Help Author

You have two options to create topic files for Windows Help: you can use Word for Windows and enter all the Help-specific coding and information manually, or you can use Microsoft Help Author and enter the information in dialog boxes. Help Author clearly makes creating Help files simpler and easier.

This chapter explains how to use the Help Project Editor to create Help project files and to build Help files. It also describes the Windows Help Authoring Templates and explains how to use them to create topic files.

> Note: The Help Author tools do not replace an understanding of the authoring process; they simply make it easier for you to perform many routine tasks used in creating Help files. To learn how to create a Help file using Help Author, see Chapter 2, Getting Started with Help Authoring. To learn how to use all the features in Help Author, keep reading this chapter, and refer to the other chapters in this authoring guide when you need to understand a particular Help feature.

About Help Author

Microsoft Help Author is an enhancement utility that makes it easier to create Help files for Windows versions 3.0 and 3.1. Help Author provides a way to do the most common Help tasks quickly and easily.

Help Author divides the authoring process into two feature sets: the Help Project Editor and the Help Authoring Templates.

The Help Project Editor makes it easy for you to:

- Create and edit Help project files.
- Add topic files to the project.
- Define and edit project file information.
- Compile Help files from Windows.
- Display error messages resulting during the build.
- View the built Help file.

The Help Authoring Templates modify Word for Windows so that you can easily:

- Create and edit Help topic files.
- Insert and edit topics, graphics, and hot spots.
- Format topic text and graphics.
- Save topic files in RTF format.
- View partial topic builds in Windows Help.

Installing Help Author

To install and use Help Author, you must have Windows version 3.0 or 3.1. Word for Windows version 1.1 or version 2.0 is highly recommended. If you do not have Word for Windows, please read the next section, Installing Help Author Without Word for Windows, to find out how to proceed.

To install Help Author on your hard disk drive

1. Insert the Microsoft Help Author Setup Disk into drive A.
2. From the File menu in Program Manager, choose Run.

   The Run dialog box prompts you for a path and filename.
3. Type a:\setup.exe and choose OK.

   The Welcome dialog box prompts you to select the files to install. The default selection is to install all the Help Author tools and the sample files.
4. Choose Continue. Or, if you do not want to install one of the tool sets or the sample files, clear its check box and then choose Continue.

   If you select the Windows Help Project Editor check box, you are prompted for the directory where Help Author will be installed. The default path and directory is C:\HELPAUTH.
5. Choose Continue to accept the default directory. Or, to install the Help Author files in a different directory, use the BACKSPACE key to erase the default directory and type a new directory and path in the text box. Then choose Continue.

   If you selected the Windows Help Authoring Templates check box in step 4, you are prompted to enter the path for Word for Windows. The path can be either a local path (hard disk drive) or a network path. If you dont have Word for Windows, you cannot install the Help Authoring Templates. In this case, please read Installing Help Author Without Word for Windows.

   Help Author supports Microsoft Word for Windows version 1.1x and 2.0. The setup program installs the correct templates, depending on which version of Word for Windows it finds in the directory you specify. The version 1.1 Word for Windows templates do not have the toolbar icons but do have the same keyboard equivalents as the version 2.0 templates. If you upgrade from Word for Windows 1.1 to 2.0, you must reinstall the Help Authoring Templates.
6. Ensure that the directory given in the text box is the one where you installed Word for Windows, and then choose Continue. If the given directory does not contain Word for Windows, use the BACKSPACE key to erase the current path. Type a new directory and path in the text box, and then choose Continue.

   The setup program copies files and then creates a group in Program Manager for Help Author. A dialog box informs you that the installation is complete. Or, if necessary, a dialog box prompts you to restart Windows to complete the installation. In that case, choose Exit to quit the setup program and restart Windows. After Windows restarts, you can begin using Help Author.

Installing Help Author Without Word for Windows

If you do not have Microsoft Word for Windows, you can still use Help Author with any other editor that supports rich text format (RTF). You may also use it with any text editor, such as Notepad, to edit your Help project file and topic files by hand.

To install Help Author without Word for Windows

1. Insert the Microsoft Help Author Setup Disk into drive A.
2. From the File menu in Program Manager, choose Run.
   The Run dialog box prompts you for a path and filename.
3. Type a:\setup.exe and choose OK.
   The Welcome dialog box prompts you to select the files to install. The default selection is to install all the Help Author tools and the sample files.
4. Clear the check box for the Windows Help Authoring Templates and choose Continue.
   You are prompted for the directory where Help Author will be installed. The default path and directory is C:\HELPAUTH.
5. Choose Continue to accept the default directory. Or to install the Help Author files in a different directory, use the BACKSPACE key to erase the default directory and type a new directory and path in the text box. Then choose Continue.
   The setup program copies files and then creates a group in Program Manager for Help Author. A dialog box informs you that the installation is complete. Or, if necessary, a dialog box prompts you to restart Windows to complete the installation. In that case, choose Exit to quit the setup program and restart Windows. After Windows restarts, you can begin using Help Author.
   Now, follow the steps in the Getting Started section below. In step 4, you will be prompted to give the program you want to use to edit your RTF files. The default editor is NOTEPAD.EXE. Choose OK to edit your RTF files with Notepad. Or use the BACKSPACE key to erase the default editor and type a new editor in the text box. Or choose the Browse button to select an editor from a list box.

Getting Started

After you install the files on your hard disk drive, you can begin using Help Author. To use Help Author you simply choose menu commands and complete the dialog box options. Each option corresponds to a specific part of the authoring process. Although there is no fixed method for using Help Author, you can follow these general steps to get started.

1. Open the Help Author group in Program Manager and double-click the Help Project Editor icon.
2. Enter project information (Edit Project command).
3. Add topic files to the project (Add New Or Existing File command).
4. Edit the topic files (add text, graphics, and so on.) to create the content for the Help file (Edit File command).

   Note Because Windows Help versions 3.0 and 3.1 differ considerably, Microsoft provides a separate template for each version. If you plan to create a Help system that uses Help version 3.1 features, use the 3.1 template (WHAT31.DOT). Otherwise, use the 3.0 template (WHAT30.DOT) as the authoring template. If you use the 3.1 template, your Help file will not work in Help version 3.0. When creating topic files, Help Author assigns the appropriate template based on your selection in the Project dialog box.
5. Save the topic files and close your editor. The Help Authoring Templates always save your documents in rich text format.
6. Define the Help build options (Edit menu commands).
7. Save the Help project file (Save or Save As command).
8. Build the Help file (Start command).
9. View the built Help file (Run Help On command).
10. Fix errors and make revisions, then rebuild the Help file.

Getting Help

The rest of this chapter explains both the Help Project Editor and the Help Authoring Templates in detail. If you have questions, refer to the appropriate section of this chapter. You can also complete the walkthrough in the following section to learn how to use Help Author. However, if you would rather get started using the software, both the Help Project Editor and the Help Authoring Templates have Help files that explain how to use their features. You can refer to the Help files on your computer rather than looking for the information in this guide.

To get Help on the Help Project Editor

1. Choose the menu command you want to learn about.
2. When the dialog box appears, choose the Help button.
   You can also choose Contents from the Help menu in the Project Editor window (or press F1) and choose a topic from the Contents.

To get Help on the authoring templates in Word for Windows 1.1

1. Open the Help Author group window in Program Manager.
2. Double-click the Help On WHAT icon.
3. Choose a topic from Contents.

To get Help on the authoring templates in Word for Windows 2.0

1. Open or create a topic file based on an authoring template.
2. From the Help menu in Word for Windows, choose the Help On WHAT command.
3. Choose a topic from Contents.

Windows Help Project Editor

Windows Help Project Editor (WHPE) is the part of Microsoft Help Author that you use to create and edit Help project files and to compile Help files from within Windows. The Help Project Editor makes creating Help files easier for Help authors.

When you create a Help project file, you generally follow these steps:

- Enter project information
- Add the topic files to the project
- Specify Help project options
- Save your work
- Compile the Help file
- View the built Help file

> **Note:** Although the Help Project Editor makes it much easier to create and edit Help project files, it does not directly support every option available in Windows Help version 3.1. The following sections describe the features that it does support. If you want to use a feature not supported directly by the Help Project Editor, you should use Word for Windows or another text editor to open the Help Project file and add the features you want. You can still use the Help Project Editor after that, and it will not change any of the features you add. (For detailed information on all the 3.1 Help Project file features, refer to Chapter 16, The Help Project File.)

The Help Project Editor Window

The Help Project Editors main window displays a list of topic files (.RTF) for the current project. Topic files are listed relative to the directory where the Help project file resides. Double-clicking a file in the list starts your RTF editor and opens that topic file so you can edit it.

When you first start the Help Project Editor, a blank, untitled Help Project Editor window opens. You create and edit the Help project file in the workspace in this window (Figure 5.1).

The Help Project Editor Menus

The Help Project Editor has three menus: File, Edit, and Compile.

The File Menu

The File menu contains the following commands.

Command	Description
New Project	Creates a new, untitled project in the Help Project Editor window. The Help Project Editor lets you save changes to the current Help project file before opening a new project.
Open Help Project	Opens an existing Help project file. When you choose this command, the Help Project Editor lets you save changes to the current Help project file before opening a new project. If opening the Help project file creates any warnings or errors, the Help Project Editor displays the errors and warnings in the View Errors window. **Note** If the Help project file was created and saved using the Help Project Editor, it will not produce any errors.
Save Project	Saves changes to the current Help project file. If the Help project file is untitled, the Help Project Editor prompts you for a name.
Save Project As	Saves the current Help project file under a new name.
Run Help On filename	Starts Windows Help and opens the latest build of the current Help project file. If the Help file did not build correctly, Windows Help displays an error message: either File not found or Help file damaged. Rerun setup. If the current project is a new, untitled Help project file, this command is dimmed.
Exit	Exits the Help Project Editor and closes the current Help project file. If a project is currently open and has been edited, the Help Project Editor prompts you to save changes before closing the project.

The Edit Menu

The Edit menu contains the following commands.

Command	Description
Add New Or Existing File	Adds topic files to the current Help project.
Remove File	Removes from the Help project file the currently selected file in the project window.
Edit File	Starts your RTF editor, if it is not already running, and opens the file currently selected in the project window.
Project	Displays a dialog box where you can edit project-level information, including the title of the file, the context ID for the

	Contents topic, and the Help compiler (HC30.EXE or HC31.EXE) to use for the build.
Comments	Displays a dialog box where you can enter comments that will appear at the beginning of the Help project file.
Compression	Displays a dialog box where you can set the level of compression to use during the build.
Application Contexts	Displays a dialog box where you can map context IDs used for context-sensitive Help.
Graphics	Displays a dialog box where you can enter a bitmap directory or list bitmaps to include in the build.
Window Definitions	Displays a dialog box where you can enter main and secondary window definitions. This command is available only if you are using the 3.1 Help compiler for the build.

The Compile Menu

The Compile menu contains the following commands.

Command	Description
Start	Starts compiling the current Help project and shows the progress of the build in an MS-DOS box. If the current Help project file includes unsaved changes, the Help Project Editor prompts you to save the changes before it begins compiling.
View Errors	Displays or hides the View Errors dialog box. A check mark appears next to this command when the View Errors dialog box is visible. If you choose the View Errors command when you are not compiling, the Help Project Editor displays the errors from the most recent build of the current Help project, even if the build occurred in a previous session.

Creating a New Help Project File

To create a new Help project file

From the File menu, choose New Project.

Opening an Existing Help Project

You can open any Help project file, whether you used the Help Project Editor to create it or not.

To open a Help project file

1. From the File menu, choose Open Help Project.
   The Open Help Project dialog box appears.
2. If the file is on a different drive, select the drive you want from the Drives box.
   If the drive to which you want to change does not appear in the Drives box, it may be a network drive to which you are not connected. Connect to the drive, and then use this dialog box option.
3. In the Directories box, double-click the directory you want.
   Or press the UP ARROW or DOWN ARROW key to select the directory, and then press ENTER.
   The current directory initially is the same directory as the currently selected file in the Help Project Editor window.
4. In the File Name box, double-click the name of the file you want to open.
   Or select the file, and then choose OK. (The default extension is .HPJ, so initially only files of that type will appear in the list box.)
   Or type the filename in the box, and then choose OK. You can enter a full path in this box, including drive and directories.

Editing Project Information

You can edit project-level information, including the title of the Help project file, the context ID for the Contents topic, and the version of the compiler to use for the build.

To edit project information

1. From the Edit menu, choose Project.
   The Project dialog box appears.
2. Type the title you want to give to the Help file.
   This title appears in the title bar of the Help window when this Help file is opened.
   Note The Title box defines the TITLE option for the Help project file.
3. Type the context string of the topic you want to designate as the Contents for your Help file.
   Note The Contents box defines the CONTENTS option for the Help project file.
4. To create a Help file for Windows Help version 3.1, select the 3.1 option button.
   If you do not select this option button, the Help Project Editor assumes you are building a version 3.0 Help file. This selection also determines which authoring template Help Author will use to create and edit your topic files.
5. Choose OK.

Adding Topic Files to the Help Project

When you create a Help project file, you add all the topic files that you want included in the build of the Help file.

To add a topic file to the Help project file

1. From the Edit menu, choose Add New Or Existing File.
   The Add New Or Existing File dialog box appears.
   Note The Add New Or Existing File command defines the [FILES] section of the Help project file.
2. If the file is on a different drive, select the drive you want from the Drives box.
   If the drive to which you want to change does not appear in the Drives box, it may be a network drive to which you are not connected. Connect to the drive, and then use this dialog box option.
3. In the Directories box, double-click the directory you want.
   Or press the UP ARROW or DOWN ARROW key to select the directory, and then press ENTER.
   The current directory initially is the same directory as the currently selected file in the Help Project Editor window.
4. In the File Name box, double-click the name of the file you want to add.
   Or select the file, and then choose OK. (The default extension is .RTF, so initially only files of that type will appear in the list box.)
   Or type the filename in the box, and then choose OK. You can enter a full path in this box, including drive and directories.
   The new file is added at the end of the list of files in the Help Project Editor window. If the selected file does not exist, the Help Project Editor asks you if you want to create a new, untitled RTF topic file based on the currently selected authoring template. Choose OK to create the file, or Cancel if you dont want to create the file.

Choosing an RTF Editor

When you install the Help Project Editor, it installs Word for Windows or the word processor of your choice as the RTF editor. The editor you choose is saved as an entry in the WIN.INI file. The Help Project Editor uses this entry to open your RTF editor when you choose the Edit File command.

To specify an RTF editor

1. From the Edit menu, choose Edit File.
   Or double-click the file you want to edit.
   The Specify RTF Editor dialog box appears if the Help Project Editor cannot find Word for Windows on your path.
2. Type the name and path (including drive and directory) of the RTF editor you want to use.
   Or choose the Browse button to find and select an RTF editor.
   If you do not have Word for Windows, the Help Project Editor recommends using Notepad, the accessory text editor that comes with Windows.
3. Choose OK.

The Help Project Editor saves your choice in the rtf= line in the [EXTENSIONS] section of the WIN.INI file.

Changing Your RTF Editor

If you want to change the RTF editor you are using after you have installed Help Author, you can do so using one of two methods:

- Use the Change button in the Help Project Editor Help file.
- Edit your WIN.INI file by hand and remove the current editor.

Using a Custom Macro to Change Your RTF Editor

Ordinarily, to change your default RTF editor you must edit the WIN.INI file and restart Windows. To save you that trouble, the Help Project Editors Help file contains a custom macro that does it for you.

To change your RTF editor using the custom Help macro

1. Open the Help Project Editor.
2. From the Help menu, choose Contents to open the Project Editors Help file.
   Or press F1 to open the Help file.
3. Choose the topic Changing Your Default RTF Editor from the Help Contents screen.
4. Follow the instructions in the Help topic.

Changing Your RTF Editor by Hand

To change your default RTF editor by hand, you must edit the WIN.INI file.

To change your RTF editor by hand

1. Open the WIN.INI file in Notepad or some other text editor.
2. Go to the [EXTENSIONS] section.
3. Delete the filename following the rtf= option and type the new name (and full path if necessary) of the RTF editor you want to use.
   Or delete the entire rtf= line from the WIN.INI file.
4. Save the WIN.INI file, and then exit the text editor.
5. Restart Windows.
6. After restarting Windows, restart the Help Project Editor.
   If you typed a new editor in the WIN.INI file, your editor is ready to use. If you deleted the rtf= entry, you will be prompted to specify a new RTF editor when you choose the Edit File command.

Editing Topic Files

You can edit any topic file without quitting the Help Project Editor.

To edit a topic file

1. Select the file you want to edit.
2. From the Edit menu, choose Edit File.
   Or double-click the file.
   The Help Project Editor starts your RTF editor and opens the file you selected.

Removing Topic Files from the Help Project

If you no longer want to include a topic file in a Help project, you can remove it from the list of files.

To remove a topic file from the Help project file

1. Select the file you want to remove.
2. From the Edit menu, choose Remove File.
   Or press DEL or SHIFT+DEL.

Specifying a Location for Bitmap Files

If you use graphics in your Help file, you can use the Help Project Editor to provide a list of bitmaps to include in the build or to define the directory where the Help compiler can find bitmap files referenced in the Help topic files.

To include bitmaps in the Help file

1. From the Edit menu, choose Graphics.
   The Graphics dialog box appears.
2. In the Directory box, type the name and full path (if necessary) of the directory that contains the bitmap files.
   Note The Directory box defines the BMROOT option for the Help project file (version 3.1 only).
3. In the File Names box, type the name of each bitmap file, including the drive and path if necessary.
   If you are using the 3.1 Help compiler for the build, you need only list the filenames of bitmaps not residing in the bitmap directory.
   Note The File Names box is the equivalent of the [BITMAPS] section of the Help project file.
4. Choose OK.

Cutting or Copying Bitmap Filenames

You can copy the bitmap names listed in this dialog box and paste them into another document or application.

To cut or copy text in the Graphics dialog box

1. Select the text in the Graphics dialog box that you want to cut or copy.
2. Choose the Cut button to cut text.
   Or choose the Copy button to copy the text to the Clipboard.

Pasting Bitmap Filenames into the Graphics Dialog Box

If you have a text file that lists all the bitmaps in your Help project, you can paste them into this dialog box using the Paste button.

To paste text from the Clipboard into the Graphics dialog box

1. Place in the Clipboard the text that you want to paste into the Graphics dialog box.
2. Choose the Paste button.

Specifying the Level of Compression

To reduce the size of the built Help file, you can specify the level of compression, or you can use no compression during the build.

To specify the level of compression

1. From the Edit menu, choose Compression.
   The Compression dialog box appears.
2. Select the level of compression you want.

Select	To specify
None	No compression
Medium	Medium compression (approximately 40 percent)
High	High compression (approximately 50 percent) If you are using the 3.0 Help compiler to build the file, the Medium compression option is dimmed. **Note** The Compression option buttons define the COMPRESS option for the Help project file.

3. If you want the Help compiler to use the keyphrase table from a previous build of the Help file, select the Use Old Keyphrase File check box.
   Use an existing keyphrase file to speed up the build process; to achieve maximum compression, however, dont reuse the old keyphrase file.
   The check box for using the old keyphrase file is dimmed unless the compression level is set to High.
   Note This check box defines the OLDKEYPHRASE option for the Help project file.
4. Choose OK.

Mapping Application Context IDs

If you are including context-sensitive Help as a feature of your Help system, you can use the Help Project Editor to map context IDs in the application to context IDs in your topic files.

The Application Contexts dialog box displays all the context ID numbers, Help context strings, and application context strings entered for the current Help project file.

Note The Application Contexts dialog box defines the [MAP] and [ALIAS] sections of the Help project file.

To map context IDs

1. From the Edit menu, choose Application Contexts.
   The Application Contexts dialog box appears.
2. Define the context ID mappings for this Help file (see the following section).
   Or enter the context ID mappings from an existing text file.
3. If you are using hexadecimal ID numbers, select the Display Numbers In Hexadecimal check box.
4. To save the changes without closing the dialog box, choose the Define button.
   Or choose OK to save the changes and close the dialog box.

Defining New Context IDs

To define a new context ID

1. From the Edit menu, choose Application Contexts.
   The Application Contexts dialog box appears.
2. From the list box, select New Context.
3. In the Context Number box, type a context number.
   If you are using hexadecimal ID numbers, select the Display Numbers In Hexadecimal check box.
4. In the Help Context String box, type a context string.
5. Choose the Define button.

You can add new context IDs to the Help project file by hand using the Define button. However, context IDs entered by hand do not have an application context string. Help context strings and context ID numbers entered by hand are saved in this format:

```
[MAP]
Help_Context                    0xFF
```

Help context strings entered by hand do not use #define.

Updating Context IDs from a File

If you or the application developers create a file containing the application context IDs and ID numbers, you can read this information into the Help project file rather than entering it by hand. It is usually preferable to use this option to enter context IDs if the IDs are generated by developers and maintained in one of the applications resource files (an .H file, for example). Also, context IDs entered from a file have an application context string.

Context IDs are entered or updated from a file in this way:

- If the application context string does not appear in the [MAP] section of the Help project file, the Help context string matches the application context string.
- If the application context string appears in the [MAP] section of the Help project file, only the context ID number is updated.
- If the [MAP] section of the Help project file contains application context strings that do not appear in the resource file, they are deleted along with their associated Help context strings.

To enter or update context IDs from a file

1. From the Edit menu, choose Application Contexts.
   The Application Contexts dialog box appears.
2. Choose the Update button.
   The Update Contents From Application File dialog box appears.
3. If the file is on a different drive, select the drive you want from the Drives box.
   If the drive to which you want to change does not appear in the Drives box, it may be a network drive to which you are not connected. Connect to the drive, and then use this dialog box option.
4. In the Directories box, double-click the directory you want.
   Or press the UP ARROW or DOWN ARROW key to select the directory, and then press ENTER.
   Windows displays the names of all files in that directory that are the type selected in the List Files Of Type box. To display a different type of file, select the type you want from the List Files Of Type box.
5. In the File Name box, double-click the name of the file that has the context IDs.
   Or select the file, and then choose OK.
   Or type the filename in the text box, and then choose OK. You can enter a full path in this box, including drive and directories.
   The context IDs are displayed in the list box.

Application context strings, Help context strings, and context ID numbers entered or updated from a file are saved in this format:

```
[ALIAS]
APPLICATION_CONTEXT=Help_Context
[MAP]
#define APPLICATION_CONTEXT      0xFF
```

If the Help context string is the same as the application context string, the entry in the [ALIAS] section is omitted. The #define directive appears with the application context ID in the [MAP] section because the ID was entered from a file.

Updating Context IDs from More Than One File

If the context IDs you are mapping to the Help file are contained in more than one file, you can use the #include directive to update from all the files.

To enter or update context IDs from more than one file

1. Create a file that contains all the files you want to use in the update.
   You can use the #include directive to include the extra files.
2. From the Edit menu, choose Application Contexts.
   The Application Contexts dialog box appears.
3. Choose the Update button.
4. Follow steps 3 through 5 above to open the resource file.

Editing Context IDs

You can edit context ID numbers and Help context strings. The Help Project Editor also displays application strings so that you can correctly map your Help context strings to the appropriate application context strings. You cannot, however, edit or modify an application context string.

To edit a context ID number or Help context string

1. From the Edit menu, choose Application Contexts.
   The Application Contexts dialog box appears.
2. Select the context ID you want to edit.
   Or select New Context to define a new context ID for the Help project file.
3. Type a new context ID number or edit the current context number shown in this box.
   If you are using hexadecimal ID numbers, select the Display Numbers In Hexadecimal check box.

4. Type a new context string or edit the current context string shown in this box.
5. Choose the Define button.

Deleting Context IDs

You can delete any context IDs from the Help project file when they are no longer necessary or valid.

To delete a context ID

1. From the Edit menu, choose Application Contexts.
   The Application Contexts dialog box appears.
2. Select the context ID you want to delete.
3. Choose the Delete button.
   The Help Project Editor will display a dialog box to confirm the deletion if the context ID is not part of a resource file.

Specifying Window Definitions

To create a secondary window for your Help file or to modify the way the main Help window appears, use the Help Project Editor to define the window characteristics for your Help file.

The Window Definitions dialog box includes all the attributes that you can specify for the main window or for any secondary windows that you create.

Note The Window Definitions dialog box defines the [WINDOWS] section of the Help project file.

Defining the Main Window

To define the main Help window

1. From the Edit menu, choose Window Definitions.
   The Window Definitions dialog box appears only if you are building a 3.1 Help file.
2. In the Window Names box, select main.
   The name main is dimmed in the Name box because you cannot change the name of the primary Help window.
3. In the Caption box, type the caption you want to appear in the main windows title bar.
   If you entered a title in the Project dialog box, that title appears in this dialog box as the window caption. You can edit the caption here or in the Project dialog box.
4. Specify a background color for the topic and nonscrolling areas, if desired.
5. To change the position of the main Help window, select the Set Window Position check box.
   The window position options for the main window are not available unless you select this check box.
6. Specify the size and position of the main window, if desired.
7. To save the changes without closing the dialog box, choose the Define button.
   Or choose OK to save the changes and close the dialog box.

Defining a Secondary Window

To define a secondary window

1. From the Edit menu, choose Window Definitions.
   The Window Definitions dialog box appears.
2. In the Window Names box, select New Window.
3. In the Name box, type the name you want to give to the secondary window.
   The default name new appears in the Name box when you select a New Window class. You can edit this name or type over it.
4. To create the window definition, choose the Define button.
5. In the Caption box, type the caption you want to appear in the windows title bar.
   The default caption New Window appears in the Caption box when you select a New Window class. You can edit this name or type over it.
6. If you want the secondary window to stay on top of other windows while the user has this Help window open, select the Stay On Top check box.
7. Specify a background color for the topic and nonscrolling areas, if desired.
8. Specify the size and position of the secondary window.
9. To save the changes without closing the dialog box, choose the Define button.
   Or choose OK to save the changes and close the dialog box.

Specifying Background Colors

You can specify the background color for the main topic area and the nonscrolling region of a window.

To specify background colors

1. From the Edit menu, choose Window Definitions.
   The Window Definitions dialog box appears.
2. In the Window Names box, select a window name.
3. From the Background Colors drop-down box, choose the color you want.
   Or choose Other to pick a custom color from the Windows Color dialog box.
4. From the Nonscrolling Region drop-down box, choose the color you want.
   Or choose Other to pick a custom color from the Windows Color dialog box.

An example of your selection for each background color appears in the display box to the left of the color box.

The default selection means that Windows Help will match the users default screen color.

Setting the Window's Position

You can indicate the exact size and location of any Help window.

To set a windows size and position

1. From the Edit menu, choose Window Definitions.
   The Window Definitions dialog box appears.
2. In the Window Names box, select a window name.
3. To manually set the size and position of the main window, select the Set Window Position check box, and then use the simulated window. (For more information, see the following sections.)
   If you are setting the position of a secondary window, the Set Window Position check box is automatically chosen for you.
   Or define the windows size and placement by typing the coordinates in the four boxes to the left of the Window Position control area. The simulated window adjusts dynamically as you type.

Using the Window Sizer

The Window Position control area consists of two rectangles: the large rectangle represents your total screen area, and the inside rectangle represents the window you are defining. The sizing handles let you change the window size directly by using the mouse.

Moving the Window

To move the simulated window

1. Position the mouse pointer inside the window, and then press and hold down the mouse button.
2. Drag the window where you want it.
   As you move the simulated window or change its size, the coordinates to the left show the exact coordinates of the window placement.

Resizing the Window

To resize the simulated window

- Drag a sizing handle in the direction you want to change the window.
- As you move the simulated window or change its size, the coordinates to the left show the exact coordinates of the window placement.

Deleting Window Definitions

You can delete any secondary window definition that you create.

To delete a window definition

1. From the Edit menu, choose Window Definitions.
   The Window Definitions dialog box appears.
2. In the Window Names box, select the name of the window you want to delete.
   The Delete button will dim if you select the main or the New Window class because you cannot delete those definitions.
3. Choose the Delete button.

Adding Comments to the Help Project File

You can add your own comments at the beginning of the Help project file. These comments are for your use only; during the build, the Help compiler ignores any comments you enter.

Note The Comments dialog box has the same purpose as adding comments by hand to the Help project file using semicolons (;) or C-style comments (/* and */).

To add comments to the Help project file

1. From the Edit menu, choose Comments.
   The Comments dialog box appears.
2. Type your comments.
   If you want more space to view and enter your comments, you can change the size of the Comments dialog box by dragging its window borders. You can also maximize it. Any changes you make to the size or position of this dialog box are saved from session to session.
3. When you finish typing your comments, choose OK.

Cutting or Copying Comment Text

You can copy text from this dialog box and paste it into another document or application.

To cut or copy text in the Comments dialog box

1. Select the text in the Comments dialog box that you want to cut or copy.
2. Choose the Cut button to cut text.
   Or choose the Copy button to copy the text to the Clipboard.

Pasting Text into the Comments Box

If you have a text file that has some text you want to include as comments in the Help project file, you can paste it into this dialog box using the Paste button.

To paste text from the Clipboard into the Comments edit box

1. Place in the Clipboard the text that you want to paste into the Comments dialog box.
2. Choose the Paste button.

Saving Help Project Files

Use the Save Project command to save changes to an existing Help project file, and use the Save Project As command to save a new (untitled) Help project file or to rename the current file.

To save changes to an existing Help project file

From the File menu, choose Save Project.

To save a new Help project file or save the current file under a different name

1. From the File menu, choose Save Project As.
   The Save Project As dialog box appears.
2. To save the file on a different drive, select the drive you want from the Drives box.
   If the drive to which you want to change does not appear in the Drives box, it may be a network drive to which you are not connected. Connect to the drive, and then use this dialog box option.
3. In the Directories box, double-click the directory in which you want to save the file.
   Or press the UP ARROW or DOWN ARROW key to select the directory, and then press ENTER.
4. In the File Name box, type a name for the file.
   The .HPJ extension is automatically assigned to the filename if you dont specify an extension. If you type an extension, be sure that it is .HPJ.
5. Choose OK.

Building the Help File

You can use the Help Project Editor to start the Help compiler and build Help files from within Windows. During the build, the Compilation in Progress window displays the progress of the build as it occurs in MS-DOS.

The Help compiler automatically turns on the REPORT option and sets the WARNING level to full-reporting. (These options are normally entered by hand in the [OPTIONS] section of the Help project file.)

If you are using the 3.1 Help compiler for the build, the Help Project Editor uses the ERRORLOG option to save to a file the errors displayed on the screen. If you are using the 3.0 Help compiler for the build, the error output must be directed to a file; you will not see the error output during the build.

To start a build

From the Compile menu, choose Start.

The Compilation in Progress window appears, showing the progress of the build and any errors that occur.

Viewing Error Messages

The View Errors window displays the error messages that occur during a build or upon opening a Help project file, if the Help Project Editor detects errors. The displayed errors are also written to a file with the same base name as the Help project file but with an .ERR extension. You can show or hide this window by using the View Errors command on the Compile menu.

To show the View Errors window

From the Compile menu, choose View Errors.

The View Errors window appears and displays the error messages from the most recent build of the Help file. A check mark appears to the left of the command on the menu to indicate that the window is open.

Once displayed, you can change the size and position of the View Errors window. The Help Project Editor saves your changes between sessions.

To hide the View Errors window

From the Compile menu, choose View Errors.

The View Errors window closes.

Displaying a Project's Help File

After you compile a Help file, you can use the Help Project Editor to start Windows Help and open the Help file of the current Help project.

To display the current projects Help file

From the File menu, choose Run Help On.

The Help Project Editor displays the Help file in Windows Help.

If the Help file did not build correctly, Windows Help displays an error message: either File not found or Help file damaged. Rerun setup. If that happens, check the View Errors window to see what error messages the compiler encountered during the build. When you are ready, try building the Help file again.

Quitting the Help Project Editor

To quit the Help Project Editor

From the File menu, choose Exit.

Or choose Close from the Control menu.

Or double-click the Control-menu box.

If you are working on a Help project file and havent saved your changes, the Help Project Editor prompts you to save the file.

The Windows Help Authoring Templates

The Windows Help Authoring Templates (WHAT30.DOT and WHAT31.DOT) are document templates that modify Word for Windows. You can use the Help Authoring Templates to create and edit Help topic files. The Help Authoring Templates make creating Help files easier by providing dialog boxes to add Help features that would normally be created using specialized Help coding and word-processing functions. For example, instead of entering footnotes for a topics context string and title, you can simply type the information in a dialog box. (The process for entering topic-level information by hand is discussed throughout this book but primarily in Chapters 6 through 10.)

> Note: The information in this section is based on Word for Windows version 2.0, but the authoring templates also work with version 1.1. If you are using Word for Windows version 1.1, some of the commands and authoring steps may vary slightly from whats in this section.

How the Help Authoring Templates Work

The Help Authoring Templates are similar to other Word for Windows templates: they contain a set of specialized macros and styles that become available when you open a document based on the template. The templates modify existing commands and add new commands to the Word for Windows menus. The result is a version of Word for Windows that has all the original word-processing functionality and yet is optimized for creating Help files.

Because Windows Help version 3.1 offers more features than version 3.0, Microsoft provides a separate template for each version of Windows Help. If you plan to create a Help system that runs under Windows Help version 3.1, use the 3.1 template (WHAT31.DOT). Help version 3.0 files, on the other hand, use WHAT30.DOT as the authoring template.

If you are using the Help Project Editor along with the Help Authoring Templates, the document template applied to your topic files is determined by two things: the version of Word for Windows you have installed on your hard disk drive and the version of Help you are using to build this Help file. Based on those choices, the Help Project Editor automatically applies the correct template to your topic files when you edit them using the Help Project Editor.

> Note: If you start a Help project using one version and then switch to another version (from version 3.0 to 3.1, for example), you will have to open the RTF files in the project and change the templates manually in Word for Windows (the Format Document command in version 1.1 or the File Template command in version 2.0).

The Template Macros

The Help Authoring Templates include macros that modify the functionality of Word for Windows to improve its use as a Help authoring tool. To make the macros easy to use, they have been attached to menu commands. The primary objective in creating the macros is to provide new Help authors and software developers a faster and better way to create Help files.

However, the macros included represent the minimum set necessary for creating Help files. They do not provide a shortcut method for authoring every available feature, nor do they replace an understanding of Windows Help functionality and how to author without using the templates. For that reason, Help authors are encouraged to modify the existing macros and create new macros to meet their specific authoring requirements.

To use the macros, create a new Help topic file and base it on WHAT30.DOT or WHAT31.DOT. The macros are then available as menu commands. When using the macros, the templates attempt to restore Word for Windows to its state before the macro executed. If the window was split, for example, the template attempts to restore the split panes after running the macro.

The following table shows the complete list of macros included in the templates and the menu commands that activate them.

Macro function	Attached to command
Create new topic file	File New
Open topic file	File Open
Save topic file	File Save
Save topic file with new name	File Save As
Edit topic footnotes	Edit Topic
Edit hot-spot information	Edit Hot Spot
Edit bitmap reference	Edit Graphic
Build current topic	View Topic
Build topic file	View File
Build current selection	View Selection
Add new topic	Insert Topic
Insert jump or pop-up	Insert Jump or Pop-Up Hot Spot
Insert Help macro	Insert Macro Hot Spot
Insert bitmap reference	Insert Graphic

The Template Style Sheet

The style sheet included with the Help Authoring Templates contains the standard styles Microsoft recommends for creating Help files. They are provided to enable Help authors to format all the text and graphic elements of a Help file without having to go through the time and effort of creating a style sheet from scratch. Of course, the existing styles can be modified and additional styles can be created if your Help system requires those changes. But these styles should provide a good starting place for any style sheet, no matter how sophisticated.

The following table describes each style included in the style sheet.

Style	Definition	Used for
bitmap	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 6pt Flush left, Line Spacing Auto, Space Before 6 pt After 6 pt, Next style: Normal	Bitmaps that occur alone in a paragraph
Ex	Font: Courier (or Letter Gothic) 8 Point, Indent: Left 6pt Flush left, Line Spacing: Exactly 12 pt, Space Before 3pt, Keep Lines Together, Tab stops: 26pt; 46pt; 66pt; 86pt; 106pt, Next style: Ex	Programming code examples and ASCII text
heading 1	Font: MS Sans Serif (or Helv) 12 Point, Bold, Indent: Left 6pt Flush left, Line Spacing: Exactly 16pt, Space Before 14pt, After 6pt, Next style: Normal 2	Topic titles
heading 2	Font: MS Sans Serif (or Helv) 10 Point, Bold, Indent: Left 6pt Flush left, Line Spacing: Exactly 12 pt, Space Before 6pt After 3pt, Next style: Normal	Bold subheadings (procedures, notes, comments, hints, tips, and examples)
heading 3	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 6pt Flush left, Line Spacing: Exactly 12 pt, Space Before 6pt After 3pt, Next style: Normal	Plain subheadings (procedures, notes, comments, hints, tips, and examples)
heading 4	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 18pt First -12.25 pt Flush left, Line Spacing: Exactly 12 pt, Space Before 6pt After 3pt, Next style: Normal	Plain subheadings
Jl	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 18pt First -12 pt Flush left, Line Spacing: Exactly 12 pt, Space Before 3pt, Next style: Jl	Lists of jumps
Jli	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 30pt First -12pt Flush left, Line Spacing: Exactly 12 pt, Space Before 3pt, Next style: Jli	Indented lists of jumps
Lb1	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 18pt First -12pt Flush left, Line Spacing: Exactly 12 pt, Tab stops: 18pt, Next style: Lb2	The first item in a bulleted or numbered list
Lb2	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 18pt First -12pt Flush left, Line Spacing: Exactly 12 pt, Space Before 3pt, Tab stops: 18pt, Next style: Lb2	Succeeding items in a bulleted or numbered list

Lp1	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 18pt Flush left, Line Spacing: Exactly 12 pt, Space Before 3pt, Next style: Lb2	Paragraphs within a bulleted or numbered list
Normal	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 6pt Flush left, Line Spacing: Exactly 12 pt, Space Before 4pt, Next style: Normal	Body text
Normal 2	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 6pt Flush left, Line Spacing: Exactly 12 pt, Next style: Normal	Paragraphs following topic titles
Normal3	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 5.75pt Flush left, Line Spacing: Exactly 12 pt, Space Before 10pt, Next style: Normal	Table text
Normal Indent	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 18pt Flush left, Line Spacing: Exactly 12 pt, Space Before 4pt, Next style: Normal	Indented paragraphs
Sa1	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 6pt Flush left, Line Spacing: Exactly 12 pt, Space Before 8pt, Next style: Jl	See Also subheadings
Sa2	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 6pt Flush left, Line Spacing: Exactly 12 pt, Space Before 6pt, Next style: Jl	Subheadings below See Also subheading
Th	Font: MS Sans Serif (or Helv) 10 Point, Bold, Indent: Left 5.75pt Flush left, Line Spacing: Exactly 12 pt, Space Before 6pt After 2pt, Border: Bottom (Single), Next style: Tp	Table heading
Th2	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 5.75pt Flush left, Line Spacing: Exactly 12 pt, Space Before 6pt After 2pt, Border: Bottom (Single), Next style: Tp	Table heading, not bold
Th3	Font: MS Sans Serif (or Helv) 8 Point, Bold, Indent: Left 5.75pt Flush left, Line Spacing: Exactly 12 pt, Space Before 6pt After 2pt, Border: Bottom (Single), Next style: Tp	Table heading, 8-pt font
Tp	Font: MS Sans Serif (or Helv) 10 Point, Indent: Left 5.75pt Flush left, Line Spacing: Exactly 12 pt, Space Before 3pt, Next style: Tp	Paragraphs within two-column lists and hanging-indent tables

The Help Authoring Template Commands

The Help Authoring Templates do not have their own menus. Instead, they make use of the existing menus in Word for Windows by either modifying the functionality of a standard command or by adding a new command to a standard menu. The Help Authoring Templates modify four menus: the File menu, the Edit menu, the View menu, and the Insert menu.

The File Menu

The authoring templates modify four commands on the standard Word for Windows File menu: New, Open, Save, and Save As.

Command	Description
New	Creates, new, untitled topic file or template. When you create a new topic file, it is based on one of the Help Authoring Templates instead of the Word for Windows NORMAL.DOT default template.
Open	Opens an existing Help topic file or template. When you choose this command, the Help Authoring Templates assume the file type is .RTF rather than the Word for Windows standard .DOC format.
Save	Saves changes to the current Help topic file. If the Help topic file is untitled, the Help Authoring Templates prompt you for a name. **Note** The Help Authoring Templates save topic files in .RTF format rather than the Word for Windows standard .DOC format.
Save As	Saves the current Help topic file under a new name. **Note** The Help Authoring Templates use the .RTF extension as the default extension rather than the Word for Windows standard .DOC extension.

The Edit Menu

The authoring templates add three commands to the standard Word for Windows Edit menu: Topic, Hotspot, and Graphic.

Command	Description
Topic	Displays Help topic information in a dialog box so you can edit it. You can use this command whether the topic actually contains footnotes or if you used another editor to add the footnotes to the topic. **Note** The Help Authoring Templates ignore any footnotes not at the beginning of the topic or not supported by the templates, including all MULTIKEY footnotes.

Hot Spot	Displays information about a hot spot in a dialog box so you can edit it. If you have not selected a hot spot before choosing this command, you will see an error message.
Graphic	Displays information about a bitmap reference so you can edit it. You can select all or part of the bitmap reference before you choose the command, but do not select any other text or you will create an error.

The View Menu

The authoring templates add three commands to the standard Word for Windows View menu: Topic, File, and Selection.

Command	Description
Topic	Builds the current topic as a Help file and then displays the resulting file in Windows Help.
File	Builds the entire topic file as a Help file and then displays the resulting file in Windows Help.
Selection	Builds the text and graphics you selected and then displays the resulting Help file in Windows Help.

The Insert Menu

The authoring templates add four commands to the standard Word for Windows Insert menu: Topic, Jump or Popup Hotspot, Macro Hotspot, and Graphic.

Command	Description
Topic	Adds a new Help topic to the topic file. The authoring templates add new topics at the end of the topic file, or immediately before the current topic. (The current topic is the topic where the insertion point is located, even if the insertion point is at the beginning of the file.) If you want to insert the new topic in a specific place, be sure the insertion point is positioned properly before you choose this command. If this is the first topic you are adding to the file, no page break is inserted after the topic information.
Jump or Pop-Up Hot Spot	Inserts a jump or pop-up hot spot in the Help topic. **Note** In WHAT30.DOT this command is simply Insert Hot Spot.
Macro Hot Spot	Inserts a macro hot spot in the Help topic.

Note This command is only in WHAT31.DOT.

Graphic	Inserts a bitmap reference in the Help topic.

Creating New Help Topic Files

To create a new Help topic file

1. From the File menu, choose New.
2. In the Use Template box, select the authoring template to use for this topic file.
   Note The authoring template should correspond to the version of Help you want to build.
3. In the New box, select Document as the type of file.
4. Choose OK.

Opening Existing Help Topic Files

You can open any Help topic file, whether or not you have used the Help Authoring Templates to create it.

To open a Help topic file

1. From the File menu, choose Open.
   The Open dialog box appears.
2. If the file is on a different drive, select the drive you want from the Drives box.
   If the drive you want to change to does not appear in the Drives list box, it may be a network drive to which you are not connected. First connect to the drive and then use this dialog box option.
3. In the Directories box, double-click the directory you want.
   Or press the UP ARROW or DOWN ARROW key to select the directory, and then press ENTER.
   Windows displays the names of all files in that directory that are the type selected in the List Files Of Type box. To select a different file type, use the List Files Of Type list box.
4. In the File Name box, double-click the name of the file you want to open.
   Or select the file and choose OK. (The default extension is .RTF, so initially only files of that type will appear in the list box.)
   The current directory is represented by the open folder icon.
   Instead of using the File Name and Directories list boxes to add a file, you can type the complete path in the File Name box and click OK.

Adding New Topics

When you create a Help topic file, you add all the topics that you want included in the build of the Help file.

To add a topic to the Help topic file

1. From the Insert menu, choose Topic.
   The Insert Topic dialog box appears. The text boxes in the dialog box represent the topic footnotes supported by the Help compiler.
   For an explanation of each of the entries in this dialog box, see Chapter 6, Creating Topics.
2. In the Title box, type the title that you want to give this topic.
3. In the Context String box, type the context string that you want to assign to this topic.
4. In the Keywords box, type the keywords that you want users to use to find this topic in a keyword search.
   Keywords must be entered in this text box exactly as they would be entered in a footnote window. You must separate each keyword with a semicolon.
5. In the Browse Sequence box, type the list name and/or sequence number if this topic is to be included in a browse sequence.
6. In the Build Tag box, type the build tags you want to assign to this topic, if you are using build tags. You must separate each build tag with a semicolon.
7. In the Entry Macro box, type the macro string that you want Windows Help to execute when the user jumps to this topic.
8. In the Comment box, type any optional comments that you want to include with this topic.
   These comments are ignored by the Help compiler.
9. Choose OK.
   Or if you want to add the new topic before the current topic, clear the Place New Topic At End Of File check box, and then choose OK.
   Note If you clear this check box, the template adds the new topic immediately before the current topic. (The current topic is the topic where the insertion point is located, even if the insertion point is at the beginning of the file.) If this is the first topic you are adding to the file, no page break is inserted after the topic information.

Editing Topic Information

You can edit any topic in the file whether or not you created it using the Help Authoring Templates.

To edit a topics footnote information

1. Place the insertion point anywhere inside the boundaries of the topic you want to edit.

   Note Because Help topics are separated by page breaks, the current topic is defined by everything that appears between two page breaks.
2. From the Edit menu, choose Topic.

   The Edit Topic dialog box appears; any information previously entered in the topic is displayed in the appropriate text boxes.

   Note Information in footnotes not at the beginning of the topic or not supported by the authoring templates do not appear in the dialog box.
3. Edit the information displayed in the Title, Context String, Keywords, Browse Sequence, Build Tag, Entry Macro, and Comment boxes.

   Or type the appropriate information if the box is empty and you are adding the feature.

   Note Keywords must be entered in this text box exactly as they would be entered in a footnote window. You must separate each keyword with a semicolon.
4. Choose OK.

Adding Jumps and Pop-Up Windows to Help Topics

You can add hot spots to text or graphics in your Help topics.

To create a jump or pop-up hot spot

1. Select the text or graphic that you want to use to create the hot spot.
   Be careful not to select any part of the topic that you do not want to include in the hot spot.
2. From the Insert menu, choose Jump Or Pop-Up Hot Spot.
   The Insert Jump Or Pop-Up Hot Spot dialog box appears.
   For a complete explanation of each of the entries in this dialog box, see Chapter 8, Creating Links and Hot Spots.
3. Edit the text in the Text box, if necessary.
   If you select text or a bitmap reference before choosing the command, the selected text appears in the Text box when the dialog box opens. If you select a picture that has been pasted into the topic, Picture Hot Spot replaces the Text box since the picture has no text and cannot be edited.
   Note Although this entry is required, the template does not check the text or bitmap name for correctness.
4. Type the context string of the destination topic.
   Note Although this entry is required, the template does not check for the existence of the context string or for correct syntax. It does check for illegal characters, however.
5. Select the type of hot spot you want to create.
6. Select the Unformatted check box if you want the hot spot to appear as authored (no underline, custom color) rather than the standard underlined-green text.
7. If the jump is to a topic in a separate Help file, type the name of the Help file in the File Name box.
   Note These jumps do not include jumps to topics in any topic file included in the current Help project file.
8. If the jump is to a topic that you want displayed in a secondary window, type the name of the secondary window in the Window Name box.
   Note A pop-up window cannot be displayed in a secondary window. If you choose Pop-Up as the hot-spot type and try to enter a name here, a dialog box appears and prompts you to choose between a pop-up or secondary window.
9. Choose OK.

Adding Macro Hot Spots to Help Topics

Windows Help version 3.1 offers the option of adding macro hot spots to Help topics, in addition to the standard jump or pop-up hot spots.

To create a macro hot spot

1. Select the text or graphic that you want to use to create the hot spot.
   Be careful not to select any part of the topic that you do not want to include in the hot spot.
2. From the Insert menu, choose Macro Hot Spot.
   The Insert Macro Hot Spot dialog box appears.
   Note For a complete explanation of each of the entries in this dialog box, see Chapter 8, Creating Links and Hot Spots.
3. Edit the text in the Text box, if necessary.
   If you select text or a bitmap reference before choosing the command, the selected text appears in the Text box when the dialog box opens. If you select a picture that has been pasted into the topic, Picture Hot Spot replaces the Text box since the picture has no text and cannot be edited.
4. Select the macro from the list box that you want Windows Help to execute when the user chooses the hot spot.
   Or type the macro name in the Macro box.
   Note The list box includes all valid macro names, including abbreviations.
5. Type the appropriate parameters for the macro you selected.
   Note If you want to enter a complex macro string for this hot spot, you must type the additional macros and parameters by hand.
6. Select the Unformatted check box if you want the macro hot spot to appear as authored (no underline, custom color) rather than the standard underlined-green text.
7. Choose OK.

Editing Hot-Spot Information

You can edit hot-spot information for jumps, pop-up windows, and macros. Because this information is significantly different depending on the type of hot spot, the Help Authoring Templates use two dialog boxes: one for jump and pop-up hot spots and one for macro hot spots.

To edit a jump or pop-up hot spot

1. Select the hot spot that you want to edit.

   You can select all or part of the hot spot, or you can simply place the insertion point within the hot spot. But do not select any text outside the hot spot or you will create an error.
2. From the Edit menu, choose Hot Spot.

   The Edit Jump Hot Spot or Edit Pop-Up Hot Spot dialog box appears.

   Note The Help Authoring Templates determine which caption to display in the dialog box by the formatting of the text or graphic you selected. If the hot spot has single underlining, the template displays Edit Pop-Up Hot Spot; if it has double underlining, the template displays Edit Jump Hot Spot.
3. Edit the text in the Text box, if necessary.

   If you select text or a bitmap reference before choosing the command, the selected text appears in the Text box when the dialog box opens. If you select a picture that has been pasted into the topic, Picture Hot Spot replaces the Text box since the picture has no text and cannot be edited.

   Note Although this entry is required, the template does not check the text or bitmap name for correctness.
4. Edit the context string of the destination topic.

   Note Although this entry is required, the template does not check for the existence of the context string or for correct syntax. It does check for illegal characters, however.
5. Select the type of hot spot you want.
6. Select the Unformatted check box if you want the hot spot to appear as authored (no underline, custom color) rather than the standard underlined-green text.
7. If the jump is to a topic in a separate Help file, type the name of the Help file in the File Name box.
8. If the jump is to a topic that you want displayed in a secondary window, type the name of the secondary window in the Window Name box.

   Note A pop-up window cannot be displayed in a secondary window. If you select Pop-Up as the hot-spot type and try to enter a name here, a dialog box appears and prompts you to choose between a pop-up or secondary window.
9. Choose OK.

To edit a macro hot spot

1. Select the macro hot spot that you want to edit.

   You can select all or part of the macro hot spot, or you can simply place the insertion point within the macro hot spot. But do not select any text outside the hot spot or you will create an error.
2. From the Edit menu, choose Hot Spot.

   The Edit Macro Hot Spot dialog box appears.

   Note The Help Authoring Templates display this dialog box if the hidden text in the hot spot begins with an exclamation point (!), the symbol used to indicate a macro hot spot.
3. Edit the text in the Text box, if necessary.

   If you select text or a bitmap reference before choosing the command, the selected text appears in the Text box when the dialog box opens. If you select a picture that has been pasted into the topic, the message Picture Hot Spot replaces the Text box since the picture has no text and cannot be edited.

   Note Although this entry is required, the template does not check the text for correctness.
4. Edit the macro string in the Macro box.

   Or select a different macro from the list box.

   Note Although this entry is required, the template does not check for the existence of the macro

string or for correct syntax. It does check for illegal characters, however.

Note If you want to enter a complex macro string for this hot spot, you must type the additional macros and parameters by hand.

5. Select the Unformatted check box if you want the hot spot to appear as authored (no underline, custom color) rather than the standard underlined-green text.
6. Choose OK.

Adding Graphics to Help Topics

To use graphics in your Help file, you can paste the graphics directly into the topic or use a bitmap reference that tells the Help compiler which graphic to display and how to align it in the topic. The Help Authoring Templates make it easy to add graphics as bitmap references.

To insert a picture

1. Place the insertion point in the topic where you want to include the picture.
2. From the Insert menu, choose Graphic.
   The Insert Graphic dialog box appears.
   For a complete explanation of each of the entries in this dialog box, see Chapter 10, Adding Graphics.
3. Type the name of the bitmap file in the File Name box.
   Note Although this entry is required, the template does not check for the existence of the bitmap name or for correct syntax.
4. Choose the type of alignment you want to use for the graphic.
5. If you want this graphic to function as a hot spot, choose the hot-spot type.
6. Choose OK.
   If you choose any hot-spot type other than Not A Hot Spot, you will see the appropriate hot-spot dialog box.

Editing Graphic Information

If you added graphics to your Help topics using bitmap references, you can edit this information.

To edit a bitmap reference

1. Select the bitmap reference that you want to edit.
   You can select all or part of the bitmap reference, or you can simply place the insertion point within the bitmap reference. But do not select any other text or you will create an error.
2. From the Edit menu, choose Graphic.
   The Edit Graphic dialog box appears.
3. Edit the name of the bitmap file in the File Name box, if necessary.
   The text of the bitmap reference you selected before choosing this command appears in the File Name box when the dialog box opens.
   Note Although this entry is required, the template does not check for the existence of the bitmap name or for correct syntax.
4. Select the type of alignment you want to use for the graphic.
5. If you want this graphic to function as a hot spot, select the hot-spot type.
6. Choose OK.
   If you change the graphic from not hot to hot or from one hot-spot type to another, the template displays the appropriate hot-spot dialog box so you can enter hot-spot information. If you change the graphic from hot to not hot, the hot-spot formatting is removed from the bitmap reference.

Saving Topic Files

Use the Save command to save changes to an existing topic file. Use the Save As command to save a new (untitled) topic file or to rename the current file under a new name.

To save changes to an existing topic file

From the File menu, choose Save.

Note The templates save topic files in .RTF format rather than the Word for Windows standard .DOC format.

To save a new topic file or save the current file under a different name

1. From the File menu, choose Save As.
   The Save As dialog box appears.
2. If you want to save the file on a different drive, select the drive you want from the Drives box.
3. In the Directories box, double-click the directory in which you want to save the file.
   Or press the UP ARROW or DOWN ARROW key to select the directory, and then press ENTER.
   The current directory is represented by the open folder icon.
4. In the File Name box, type a name for the file.
   Instead of using the File Name and Directories list boxes to save a file, you can type the complete path in the File Name box and click OK.
   The .RTF extension is automatically assigned to the filename if you dont specify an extension. If you specify an extension, it overrides any automatic extension.
5. Choose OK.

Displaying Topics in Windows Help

If you want to check the progress of your work, you can use the Help Authoring Templates to create a Help file from the topics you are working on and display the built file in Windows Help. You can create a temporary Help file of the entire topic file, a portion of the topic file, a single topic, or a selection within a topic.

To build the file, the template creates the temporary files TEMP.RTF and TEMP.HPJ. After the file is built, the template starts Windows Help and opens TEMP.HLP, which contains the topic file, topic, or selection.

To view a compiled version of the current topic file

From the View menu, choose File.

If the topic file includes graphics, you may be prompted to give the bitmap directory.

To view a compiled version of the current topic

1. Place the cursor within the topic you want to view.
2. From the View menu, choose Topic.

   If the topic includes graphics, you may be prompted to give the bitmap directory.

To view a compiled version of a partial topic file

1. Select the portion of the topic file you want to view.
2. From the View menu, choose Selection.

   If the selection includes graphics, you may be prompted to give the bitmap directory.

Help Authoring Template Keys

Use the following keys in the Help Authoring Templates.

To	Press
Open an existing topic file	CTRL+F12
Save a topic file	SHIFT+F12
Save a topic file under a new name	F12

Chapter 6 Creating Topics

Chapter 2 showed you how to create a simple Help file with just two topics. This chapter goes into more depth about how to create and edit Help topic files. It also describes how to insert Help-specific codes that communicate information to the Help compiler about the particular structure and organization of the Help file. For example, topic codes are used to identify topics, to create keyword indexes and browse orders, and to run Help macros.

What Are Topics?

The topic is the basic unit of information in a Help file; all information is displayed in topics. A topic can include text, graphics, hypertext links, and custom controls. Because their use varies, topics can range in size from short examples containing a single picture or word to lengthy explanations involving several graphics and hundreds of words. Figure 6.1 shows a topic from the sample Help file.

Topics are stored in topic files, which contain one or more topics. How you organize the topics within topic files depends on your own work habits and the type of information contained in the topics. At the extreme, you can place all the topics in one large topic file, or you can create a separate topic file for each topic in the Help project. For most projects, it is probably best to avoid either extreme and organize the topics so that they reflect the overall organization of your Help file. For example, if you create a simple Help system with three kinds of information, you might want to create three topic files, one for each information category.

Just remember that the way you organize topics within the topic file does not affect how a user views the topics. The organization that the user sees when using the Help file is something you build into the Help file using authoring features. For example, you can use any of the following Help features to organize the presentation of topics in your Help file.

Help feature	Description
Hypertext links	Let you connect any two topics together so that users can jump from topic to topic following a particular line of interest. This is the primary method you use to organize the Help topics in a Help file.
Browse sequences	Let you organize a number of topics into a linear sequence that users can browse the same way they flip through pages in a printed book.
Keyword search	Lets you assign keywords to topics so that users can search for a particular item and access information directly without following a hierarchical path.

Getting Started

Before you start creating topics for the Help file, you need to make at least two preliminary decisions. You must:

- Choose a text editor that you will use for writing and saving the topic information.
- Make a project directory on the hard disk drive where you can store the topic files that the compiler will use to build the Help file.

Choosing a Text Editor

The first thing you have to decide before you can create any Help topics is which word processor or text editor you are going to use. The primary requirement is that it must support rich-text format (RTF). RTF is the standard Word document exchange format that Microsoft Word applications use to convert formatted documents into a set of instructions that other applications can read and interpret. Most commonly, RTF is used to transfer Word documents between Macintosh and PC platforms or across communications networks. RTF file format consists of text intermixed with RTF statements that control document, paragraph, and character formatting. For a full description of the RTF file format accepted by th Help computer, see Appendix B, Help RTF Statements.

However, because the Windows Help compiler only supports topic files that have been saved in RTF, choosing the right word processor is critical to your online Help project. Although you can use any word processor or text editor that has an option to save files as RTF or that has some other means for converting its native format to RTF, the files it produces may not be fully compatible with the Help compiler. That is because some word processors and conversion utilities produce RTF that can be interpreted correctly by other word processors but cannot be interpreted correctly by the Help compiler because they use nonstandard techniques. For that reason, we strongly recommend that you use Word for Windows version 1.1 or 2.0.

> Note: If you use Microsoft Word for the Macintosh to create topic files, we recommend using version 4.0b or later.

Making a Help Directory

After you choose an RTF editor, you should make a project directory and decide how to organize the directory structure that will hold all the project files. How you organize your specific Help project depends on the type of Help file you are creating. If the Help project is small or uncomplicated, you may want to place all the project files in one directory. If the Help project is large or complex, you may want to make a directory tree that reflects the organization of the Help project. For example, one method is to make a directory for each subcategory within the Help file and then create an \ART and \RTF subdirectory for the pictures and text used in each subcategory. Figure 6.2 shows an example of this kind of directory structure.

Creating New Topics

The first step in creating a Help file is to prepare the text. There are several ways to create text for the new topics. You can:

- Type the topic text in Word for Windows (or other compatible editor).
- While you are creating new material, you may want to keep the file in normal .DOC format until you are ready to compile the topics. Working in Words normal format is easier and saves time. When you are ready to build the Help file, you can save the topic text as RTF.
- Import one or more topics from an existing RTF topic file into a new topic file.
- Word for Windows includes an RTF filter that can import or export RTF text. If you have files that are already in RTF or Word format, you can import them directly into Word and save them as new RTF topic files. To import just a portion of an RTF file, you can open the RTF file and copy the text you want to the Windows Clipboard and then paste it into the new topic file.
- Import text from any electronic document by converting its file format to a format readable by Word for Windows.
- Word for Windows can import many types of word-processing files. If you have files that are not in RTF or Word format, you can import them into Word and save them as RTF files. Also, if your information has a consistent format, you can write automated conversion programs or Word macros to add the necessary RTF statements to the text document. Because the compiler expects valid RTF, if you import or generate invalid RTF, the topic files might fail to produce the results you expect.
- Use an optical character recognition (OCR) scanner to convert printed text to electronic form and import the resulting file into Word for Windows (or other compatible word processor).

  Because printed manuals often serve as good source material for online documentation, you can scan the material if it does not exist in electronic form or if it cannot be converted to a compatible format. When saving the scanned file, ASCII (text-only format) is sufficient to import the file into Word for Windows.

Dividing the Text into Topics

Whether you type the text from scratch or work with existing text documents, you must divide the text into discrete chunks of information, or topics. This is perhaps the most difficult aspect to writing online Help: deciding how to divide the body of information into separate topics. To do that you must decide how large to make the topics and where to make the breaks in the content. A general guideline is to keep topics short and present information in small, relatively equal-sized chunks that users can scan easily. However, longer, scrolling topics may be appropriate if the target audience is more expert or if the content suffers from fragmentation when divided into too many chunks.

Exactly how you divide the information will vary depending on the Help project and the specific content. In all cases, you must balance conflicting needs. Small topics provide more flexible ways to organize the information; longer topics simplify the design and help users understand relationships among ideas. In general, try to present one concept or idea per topic and limit each topic to roughly one screen of information. Using this method, most topics should be from one to seven paragraphs.

Identifying Topics

To identify a unit of text as a topic, you insert a hard page break in the topic file. The Help compiler treats all the information between page breaks as one topic.

To identify a new topic

Insert a hard page break (CTRL+ENTER in Word for Windows) at the end of the text block that you want to become a topic.

The page defines the topic boundaries.

Figure 6.3 shows two sample topics as they appear in Word for Windows: Topic A and Topic B.

In addition to the page break, each topic also includes a context string identifier, and most topics have a title.

Topic footnote	Description
Context string	Unique identifier for the topic that the compiler uses as a reference. The context string is used to identify individual topics and to create links between topics in the Help file. The context string is required.
Title	Descriptive name for the topic. The topic title provides an identifying link between the name displayed in Help dialog boxes, such as Search and History, and the actual topic heading displayed in the Help window.

For complete information about these and other topic footnotes, see the Understanding Topic Footnotes section and following sections in this chapter.

Editing Topics

While creating and editing the topic text, you work in Word for Windows so you can use Words powerful editing features to perform the following tasks:

- Edit text, perform search and replace, or run the spell checker.
- Apply character, paragraph, and table formatting.
- Add Help features like hot spots and pictures.

> Note: Some Word formatting attributes are ignored by the Help compiler. For a complete discussion of the formatting attributes supported by Help, see Chapter 7, Formatting Topics. The list of unsupported features is small, so you have considerable freedom in how you format the topic text.

After creating and editing the topics, you save the file as RTF and exit Word or, if you want to leave Word running, just close the open document.

Understanding Topic Footnotes

Windows Help redefines a number of Word for Windows features so that they have a different meaning in Help topic files than they do in a normal word-processing document. For example, Windows Help uses the footnote feature in Word for Windows to pass important topic information to the Help compiler. Topic footnotes have very specific meanings in Help, as described in the following table.

Footnote	Name	Purpose
Asterisk (*)	Build tag	Defines a tag that specifies topics the compiler conditionally builds into the system. Build tags must be the first footnote in a topic when they are used.
At sign (@)	Comment	Includes an author-defined comment about the Help topic.
Dollar sign ($)	Title	Defines the title of a topic.
Exclamation Point (!)	Macro	Defines a macro that Help executes when the user enters the topic.
Letter K	Keyword	Defines a keyword the user uses to search for a topic.
Other letters	Multikey keyword	Defines an alternate keyword table that the user uses to search for a topic.
Plus sign (+)	Browse sequence	Defines a sequence that determines the order in which the user can browse through topics.
Pound sign (#)	Context string	Defines a context string that uniquely identifies a topic. Because hypertext relies on links provided by context strings, topics without context strings can only be accessed using keywords or browse sequences.

Note: If you are using build tags, footnote them at the very beginning of the topic. Place other footnotes in any order you want. For information about using each of these specific footnotes, see the following sections in this chapter.

Assigning Context Strings

After creating individual topics by inserting page breaks, you then identify each topic by giving it a unique context string. Context strings identify each topic in the Help file. Each context string must be uniqueit can be assigned to only one topic within the Help project.

Windows Help uses the context string to find a specific topic when users choose a hot spot that references that topic. Although not technically required, it is a good idea to assign a unique context-string identifier to all topics within the Help system. Topics without context strings cannot be accessed by hot spots (jumps, pop-ups, or macros), although they can be accessed through browse sequences or keyword searches. If writers include topics without context strings, it is their responsibility to ensure that users can access that information.

See the Assigning Keywords and Specifying Browse Sequences sections in this chapter for more information. Also, see Chapter 8, Creating Links and Hot Spots, for information about using context strings in hot spots.

Inserting Context-String Footnotes as Topic Identifiers

To enter a context string in the topic, you use a pound sign (#) footnote. A context-string footnote must be the first footnote in the topic unless you use build tags to specify individual topics for the build. If you use build tags, the build-tag footnote must precede the context string. (For more information about build tags, see Assigning Build Tags, later in this chapter.)

To insert a context string

1. Position the insertion point at the beginning (extreme left) of the first line in the topic.
2. From the Insert menu, choose Footnote.
   The Footnote dialog box appears.
3. Type the pound sign as the custom footnote mark, and then choose OK.
   A superscript pound sign (#) appears in the text window, and the insertion point moves to the footnote window.
4. Type the context string to the right of the pound sign in the footnote window.
   Use only a single space between the pound sign and the text string. (The context string itself shouldnt contain any spaces.) For example, you might type the following context string to identify the Saving Files topic in Cardfile Help:

   ```
   # savefile_card
   ```

   Figure 6.4 shows this footnote window.

   Note: After inserting a footnote, you can close the footnote window or leave it open. If you are going to code additional footnotes, you may want to leave it open. In Word for Windows version 1.1, you close the footnote window by double-clicking the split bar (the black rectangle between the text window and the footnote windows vertical scroll bars). In Word for Windows version 2.0, you can just click the Close button.

Inserting Spot References

In addition to the context string that identifies the topic, you can also include additional context strings to identify particular locations within the topic. These secondary context strings are called spot references because they reference specific locations, or spots, within a topic.

When executing a jump, Windows Help uses the location of the context string as a reference point. Windows Help displays topics with only one context string identifier from the top down, starting with the first line in the topic. If you want users to be able to jump to a specific location within the topic, you can insert a context string at that location. When executing a jump to the spot reference, Windows Help displays the topic as if that location were the top of the topic. For example, to create a jump to the middle of a topic, you insert a context string at that location.

> Note: You cannot insert a title footnote in the middle of a topic, so any spot references that you define cannot have a subtitle.

To insert a spot reference context string

1. Position the insertion point at the location where you want to insert a spot reference.
   A context string inserted at the beginning of the topic is used to identify the topic. A context string inserted anywhere else in the topic is treated as a spot reference.
2. From the Insert menu, choose Footnote.
   The Footnote dialog box appears.
3. Type the pound sign as the custom footnote mark, and then choose OK.
   A superscript pound sign (#) appears in the text window, and the insertion point moves to the footnote window.
4. Type the context string to the right of the pound sign in the footnote window.
   Use only a single space between the pound sign and the text string. (The context string itself shouldnt contain any spaces.) For example, you might type the following context string to identify the Save As Optiontopic within the Saving Files topic of Cardfile Help:

   ```
   # saveas_card
   ```

   Figure 6.5 shows this footnote window.

Context String Guidelines

When creating context strings, follow these guidelines:

- Every topic should have at least one context string, and that string must be unique within the Help file.
- Context strings can contain the alphabetic characters AZ (uppercase or lowercase), the numeric characters 09, and the period (.) or the underscore (_) character. Even though Windows Help doesnt distinguish between uppercase and lowercase letters, you might want to mix the case of characters to make the string more readable, as in this example:

  # Saving_RTF_Files
- Context strings can have as many as 255 characters. However, shorter strings are easier to remember and easier to type when referencing them in jump, pop-up, and macro hot spots.

Inserting Topic Titles

Enter a topic title after the context string. Topic titles help users identify individual topics. For example, Windows Help displays the topic title you enter in the Bookmark dialog box, in the History window, and in the Go To list of the Search dialog box. You use a dollar sign ($) footnote to identify the topic title.

To insert a topic title

1. Position the insertion point at the beginning of the topic (to the right of the pound-sign footnote).
2. From the Insert menu, choose Footnote.
   The Footnote dialog box appears.
3. Type the dollar sign as the custom footnote mark, and then choose OK.
   A superscript dollar sign ($) appears in the text window, and the insertion point moves to the footnote window.
4. Type the topic title to the right of the dollar sign in the footnote window.
   Use only a single space between the dollar sign and the first word of the title. For example, you might type the following title for the Saving Files topic in Cardfile Help:

   ```
   $ Saving Files
   ```

   Figure 4.6 shows this footnote window.

Note that you can include spaces in topic titles.

Topic Title Guidelines

When creating topic titles, follow these guidelines:

- You can define only one title footnote per topic, and it must be at the beginning of the topic. You cannot place title footnotes at spot references.
- Since you only need topic titles to identify topics that users can jump to or that can be displayed in Help dialog boxes, you might consider not assigning topic titles to topics that are displayed only in pop-up windows.
- Enter the title string exactly as it appears in the title heading of the topic text because it is the string Windows Help displays in dialog boxes and windows for the user. In the previous example, the topic title Saving Files exactly matches the bold heading Saving Files in the topic text.
- Titles can have as many as 127 standard characters, including accented characters. Any embedded spaces in the title count as characters. Windows Help truncates titles longer than 127 characters.

  Windows Help follows these rules when sorting titles alphabetically for the Search facility:
  - Short titles come before long titles.
  - Punctuation characters come before numeric characters.
  - Numeric characters come before alphabetic characters.

Assigning Keywords

In printed books, a good index provides an outline of the books contents and helps readers quickly find specific information. Indexes also associate related information in a book; when several pages are listed under a common index entry, readers know that the information on those pages is related in some way.

In Windows Help, the Search feature provides a type of electronic index, similar to a book index, in which users look up words in the index to find specific information quickly. By typing a keyword or choosing one from the keyword list in the Search dialog box, users can see all the topics associated with a keyword and go directly to one of the associated topics. Because keyword search is a fast way for users to find information, you should consider assigning keywords to most topics in your Help file. Likewise, if you do not want users to access certain topics through the Search facility, you should not assign any keywords to those topics.

Help makes it easy to add keywords to topics. However, much of the work in creating a good keyword index involves analysis and planning. When you create an index for a book, you need to plan the index entries carefully, eliminate duplicate or ambiguous entries, and apply the entries consistently to all pages of the book. The same issues apply to Help keyword indexes.

How Keywords Work in Windows Help

Keyword indexes are similar to book indexesthe keywords are analogous to first-level entries, and the topics containing the keywords are analogous to second-level index entries. A keyword index consists of a list of index entries, or keywords. In a book index, each entry includes a page reference, which tells the reader where to find the indexed information. In a keyword index, each keyword includes a list of topics (the electronic equivalent of page references), which tells users where to find the information associated with that particular keyword. Just as you can have more than one reference following a single index entry in a book, you can assign the same keyword to more than one topic in the Help file.

When a user chooses the Search button, a dialog box appears that lists all the keywords associated with each topic in the upper half of the dialog box. The user types a word or selects one from the list and then chooses the Show Topics button. Help lists all the topics containing the selected keyword in the lower half of the dialog box. (Remember, topics are identified by the $ footnote.) Figure 6.7 shows how the sample application has the keyword open in two topics: PLACEHOLDER and PLACEHOLDER.

The user can double-click a topic in the list box to display that topic in the main Help window.

Defining Keywords for a Topic

The keywords that appear in the Search dialog box come from the keywords you add to individual topics in your Help file. A topic can have many keywords, and the keywords can be from a single index or from multiple indexes (see Creating Multiple Keyword Indexes, later in this section).

When you create a new topic, you can define the keywords you want to associate with that topic. The topic must have at least one keyword to be included in the Search dialog box (if you dont want an index entry for the topic, dont define any keywords). You use an uppercase K footnote to create keyword entries for a topic.

To assign a keyword to a topic

1. Position the cursor at the beginning of the topic text.
2. From the Insert menu, choose Footnote.
3. Type an uppercase K as the custom footnote mark, and then choose OK.
   A superscript K (K) appears next to the heading, and the insertion point moves to the footnote window.
4. Type the topic keyword(s) to the right of the K in the footnote window.
   Use only a single space between the K and the first keyword. Separate multiple keywords with a semicolon (;). For example, you might type the following keywords for the Cardfile topic:

   ```
   K Cardfile keyword go here PLACEHOLDER
   ```

   Figure 6.8 shows this footnote window.

Placing Keywords in the Topic

When the user goes to a topic by choosing a keyword from the Search dialog box, Help displays the selected topic in the main window, starting from the beginning of the topic. If the information related to the keyword is located in the middle or toward the end of the topic, the user may not be able to see the relevant information without scrolling the topic. This result may not be what you want.

If you want users to be able to go directly to relevant information within a topic (and see it without scrolling), you can place additional keywords with the information you want users to find. Keywords placed within a topic function as spot references (similar to context-string spot references) because they index specific locations, or spots, within the topic. To access the spot-referenced material, users choose the keyword from the Search dialog box.

In the Search dialog box, all keywords appear the same. The user cannot tell the difference between keywords placed at the beginning of the topic and those placed elsewhere in the topic. However, when the user goes to the topic, Windows Help uses the location of the keyword footnote as a reference point. If the keyword footnote is located in the middle of the topic, Help displays the topic as if the middle location were the top of the topic.

> Note: Because you cannot insert a title footnote in the middle of a topic, any keywords that you place in the middle of the topic use the main topic title in the Search dialog box.

To define a keyword in the middle of the topic

1. Place the insertion point where you want to define the keyword.
   A keyword inserted anywhere except the beginning of the topic is treated as a spot reference.
2. From the Insert menu, choose Footnote.
   The Footnote dialog box appears.
3. Type an uppercase K as the custom footnote mark, and then choose OK.
   A superscript K (K) appears in the text window, and the insertion point moves to the footnote window.
4. Type the keyword(s) to the right of the K in the footnote window.
   Use only a single space between the K and the first keyword. Separate multiple keywords with a semicolon (;). For example, you might type the following keywords to index a procedure within the Cardfile topic:

   ```
   K switching to List view
   ```

   Figure 6.9 shows this footnote window.

Creating Multiple Keyword Indexes

Just as some books include multiple indexes, Help supports multiple keyword indexes. The additional keyword indexes enable an application to look up information that is defined in alternate, custom keyword indexes. For example, you can create a general index for all subjects in the Help file and create specialized indexes for more narrowly defined areas. If your application includes embedded functions, you can use a second keyword index to enable users to get context-sensitive Help on the function calls included in the parent application. Or, if your application competes in the marketplace with another popular program, you can use an additional keyword index for the competitors product. Users familiar with keywords from the competitors application can find matching keywords in your applications Help file.

Creating additional keyword indexes is a two-part process. First, the MULTIKEY option must be placed in the [OPTIONS] section of the Help project file, as in this example:

```
[OPTIONS]
MULTIKEY=L
```

> Note: Be sure to limit your multikey index footnotes to one case, usually uppercase. In this example, topics with the footnote L would have their keywords incorporated into the additional keyword index, whereas those assigned the lowercase letter l would not. We recommend using uppercase letters over lowercase letters. (See the MULTIKEY option in Chapter 16, The Help Project File.)

Second, the topics to be associated with the additional keyword index must be written and assigned footnotes using the letter specified in the MULTIKEY option. Multikey footnotes can be placed either at the beginning of the topic or elsewhere if you want to create multikey spot references.

To assign a multikey keyword to a topic

1. Place the insertion point at the beginning of the topic text or wherever you want to define the keyword.
   A keyword inserted anywhere except the beginning of the topic is treated as a spot reference.
2. From the Insert menu, choose Footnote.
3. Type the uppercase letter used in the MULTIKEY option as the custom footnote mark, and then choose OK.
   A superscript letter appears in the topic, and the insertion point moves to the footnote window.
4. Type the keyword(s) to the right of the letter in the footnote window.
   Use only a single space between the footnote character and the first keyword. Separate multiple keywords with a semicolon (;). Figure 6.10 shows a sample multikey footnote entry.

When assigning multikey keywords to topics, be sure to associate only one topic with a multikey keyword. Help does not display the standard Search dialog box for a multikey keyword search. Instead, it displays the first topic found with the specified keyword. If you want the topics in your additional keyword index to appear in the Search dialog box, you must specify a K footnote for the topics.

If Help cannot find the multikey keyword in the alternate keyword index, Help displays a default topic that contains the keyword Default Topic, or it displays the Help topic not found error message if it cannot find the default topic.

An application can display a Help topic that contains a multikey footnote. For information on the parameters passed by the application to Help, see the Searching for Help with Keywords section in Chapter 19, The WinHelp API.

Keyword Guidelines

When adding keywords, follow these guidelines:

- A topic should have a title footnote assigned before you assign it keywords.
- Index keywords can include any ANSI character, including punctuation (except semicolons) and accented characters. You can use spaces between words to create a keyword phrase. For example, you could make PLACEHOLDER a single index keyword as follows:

  ```
  K keyword phrase; keyword; keyword
  ```

- Help ignores any formatting assigned to keywords (such as bold or underline).
- The keyword search handles uppercase and lowercase letters the samefor example, PLACEHOLDER is the same as placeholder.
- A keyword footnote can have as many as 1023 characters. If your keyword list exceeds that limit, you can use additional K footnotes within the same topic. For example, you might have three keyword footnotes in a topic:

  ```
  K keyword phrase; keyword;keyword
  K keyword phrase; keyword;keyword
  K keyword phrase; keyword;keyword
  ```

Figure 6.11 shows this footnote window.

Specifying Topic-Entry Macros

To customize the way Windows Help works with your Help file, you may want to execute a macro when the user enters a topic. Windows Help includes a variety of macros, any of which can be included as a footnote macro. In general, Help macros can be used to:

- Modify Help menus and menu functionality.
- Modify Help buttons and button functionality.
- Modify Help window behavior.
- Create hypertext links.
- Create and manipulate text markers.
- Assign a Help macro to a keyboard access key (or key combination).
- Start an application.
- Register a function within a DLL as a Help macro.

The complete list of macros that Windows Help version 3.1 supports is found in Chapter 15, Help Macro Reference. Refer to that chapter for full details about each macro.

How Entry Macros Work

If a Help macro is included in a topic footnote, Windows Help executes that macro when the user first enters that topic. Users can display topics by:

- Clicking on a jump to that topic.
- Clicking the Back button or a browse button.
- Selecting a topic from the history list or keyword search list.

Windows Help does not execute the macro a second time if the user performs any operation within the original topic, such as displaying a pop-up window or jumping to a spot reference.

> Note: Windows Help executes any entry macros after it has laid out the text and graphics in the topic.

Inserting Macro Footnotes

You use an exclamation point (!) as the footnote character to execute topic-entry macros.

To insert a topic-entry macro

1. Position the insertion point at the beginning of the topic (to the right of the other footnotes).
2. From the Insert menu, choose Footnote.
   The Footnote dialog box appears.
3. Type the exclamation point as the custom footnote mark, and then choose OK.
   A superscript exclamation point (!) appears in the text window, and the insertion point moves to the footnote window.
4. Type the macro to the right of the exclamation point in the footnote window.
   Use only a single space between the exclamation point and the first word of the title. For example, you might type the following macro:

   ```
   ! ChangeButttonBinding("btn_contents", "JumpID(`hgcd.hlp', `acc_idx_hg')")
   ```

   Figure 6.12 shows this footnote window.

Note that you can include spaces in macros.

Entry Macro Guidelines

When creating topic entry macros, follow these guidelines:

- You can execute any macro listed in Chapter 15, Help Macro Reference, within a topic footnote; however, if the topic is to be displayed in a secondary window, the macro may be ignored or cause undesirable behavior.
- The macro text can have as many as 512 characters, including any embedded spaces.
- You can specify more than one macro to be executed by separating them with a semicolon or colon, as in the following example:

```
! ChangeButttonBinding("btn_contents", "JumpID(`hgcd.hlp',
`acc_idx_hg')");EnableButton("btn_up");ChangeButttonBinding("btn_up",
"JumpID(`cdcd.hlp', `hlpidx_idx_card')")
```

Specifying Browse Sequences

Invariably, certain topics relate to other topics. For the information in your Help file to be clear and understandable, you might need to display certain groups of topics in a specific sequence. Windows Help lets you group specific topics and structure them in units called browse sequences. A browse sequence is simply a series of topics that appear in a sequence when the user chooses the << and >> buttonslike turning pages in a book. These buttons are optional in Windows Help version 3.1. To use browse buttons in a Help file, you must include the BrowseButtons macro in your Help project file. For more information, see the [CONFIG] section in Chapter 16, The Help Project File, and the BrowseButtons section in Chapter 15, Help Macro Reference.

When you design your Help file, you must first decide if you want to use browse sequences. If you do, you must decide which topics to group together into browse sequences. You might decide to place the topics within a browse sequence in separate files, with the first topic in the sequence at the beginning of the file and the last topic in the sequence at the end of the file. In the sample Help file, each category of information is grouped in a separate browse sequence.

Once these sequences become part of a Help file, the << and >> buttons on the Windows Help button bar let users move back and forth between topics within the browse sequence. The browse buttons become active whenever Windows Help displays topics coded with a browse sequence. At the beginning of the sequence, only the >> button is active. At the end of the sequence, only the << button is active. In the middle of the sequence, both buttons are active.

How Browse Sequences Work

You define browse sequences for your topics by assigning a plus sign (+) footnote to the topics in each browse sequence. A browse sequence footnote has a sequence name, followed by a colon, followed by a sequence number. A sequence name identifies the sequence to which the topic belongs. The sequence number specifies the location in the sequence where the topic is displayed.

A Single Browse Sequence

If your Help file has only one browse sequence, you dont need to assign a sequence name to browse sequence footnotes. You simply specify the sequence number in each footnote. Windows Help then displays the topics in the order of these sequence numbers.

You might want to skip one or more numbers in a sequence so you can add new topics later. For example, use increments of five or ten. The Windows Help compiler ignores skipped numbers; only their order is significant. (See the Sorting Order section, later in this chapter.)

You arent required to use sequence numbers in footnotes. If you dont, Windows Help displays topics in their order of appearance in the topic file.

Multiple Browse Sequences

To set up more than one browse sequence in a Help file, use different sequence names to identify the different browse sequences. In the browse sequence footnote for each topic, assign a sequence name and a sequence number separated by a colon, as shown below:

```
sequence-name:sequence-number
```

For example, heres how the command topics within the FILENAME.RTF file, and the procedure topics within the FILENAME.RTF file, are arranged in a browse sequence:

FILENAME.RTF	FILENAME.RTF
cmd:0005 cmd:0010 cmd:0015 cmd:0020 cmd:0025 . . .	how:0005 how:0010 how:0015 how:0020 how:0025 . . .

This creates two different browse sequences: one for PLACEHOLDER and one for PLACEHOLDER.

If you dont want to keep track of sequence numbers for each sequence and you want to display topics in their order of appearance, use a constant value for the browse-sequence number in each footnote. For example, to display the topics in the PLACEHOLDER and PLACEHOLDER sequences in their order of appearance within the file, assign the sequence names and a constant number as follows:

FILENAME.RTF	FILENAME.RTF
cmd:0000 cmd:0000 cmd:0000 cmd:0000 cmd:0000 . . .	how:0000 how:0000 how:0000 how:0000 how:0000 . . .

This still creates the two browse sequences, but the topics in each sequence are displayed in their order of appearance in the topic file.

Sorting Order

Sequence numbers can include both numbers and alphabetic characters. During the compiling process, theyre sorted using an alphabetic sort, not a numeric sort. Always use the same number of characters for each number in a sequence (dont use a two-character number for one topic and a three-character number for another). Otherwise, in certain cases, a higher sequence number could appear before a lower one.

For example, even though the number 100 is numerically higher than 99, the topic with sequence number 100 will appear before topic 99 because Windows Help is comparing a string beginning with 1 to a string beginning with 9. To keep the topics in their correct numeric order, you must make 99 a three-digit string (099).

Assigning Browse Sequence Footnotes

Once you determine the number and names of browse sequences and the order of topics in each sequence, you can assign sequence names and numbers to the topics.

To assign a browse sequence

1. Position the cursor at the beginning of the topic text.
2. From the Insert menu, choose Footnote.
3. Type a plus sign as the custom footnote mark, and then choose OK.

   A superscript plus sign (+) appears next to the heading, and the cursor moves to the footnote window.
4. Type the sequence name, a colon, and the sequence number to the right of the plus sign.

   Use only a single space between the plus sign and the sequence name. For example:

   ```
   + how:000
   ```

   Figure 6.13 shows this footnote window.

Assigning Build Tags

Build tags are labels (strings) that you assign to a topic to exclude that topic from a build under certain conditions. Build tags are optional and not necessary for most Help projects. However, they do provide a means of supporting different versions of a Help system without having to create different source files for each version. Topics without build tags are always included in a build, as are all topics not expressly excluded from the build in the BUILD expression of the Help project file.

How Build Tags Work

Including a build tag footnote with a topic is equivalent to setting the tag to True when compared to the value set in the Help project file. The Help compiler assumes all other build tags to be False for that topic. After setting the value of the buildtag footnotes to True, the Help compiler evaluates the BUILD expression in the [OPTIONS] section of the Help project file.

> Note: All build tags must be declared in the Help project file, regardless of whether the tags are declared conditionally. If the evaluation results in a True state, the Help compiler includes the topic in the build. Otherwise, the Help compiler omits the topic.

The Help compiler includes topics that do not have a build tag footnote in all builds, regardless of the build tag expressions defined in the Help project file. For this reason, you may want to use build tags primarily to exclude specific topics from certain builds. If the Help compiler finds any build tags not declared in the Help project file, it displays an error message.

By allowing conditional exclusion of specific topics, you can create multiple builds using the same topic files. This saves the Help writing team time and also enables you to maintain a higher level of consistency across your product lines.

Assigning Build Tag Footnotes

You use the asterisk (*) to insert build tags as footnotes. When you assign a build tag footnote to a topic, the Help compiler includes or excludes the topic according to information specified in the BUILD option and [BUILDTAGS] section of the Help project file. For more information about the BUILD option and the [BUILDTAGS] section, see Chapter 16, The Help Project File.

To assign a build tag to a topic

1. Position the cursor at the beginning of the topic text, so that it appears before all other footnotes for that topic.
2. From the Insert menu, choose Footnote.
3. Type an asterisk as the custom footnote mark, and then choose OK.
   A superscript asterisk (*) appears next to the heading, and the cursor moves to the footnote window.
4. Type the build tag name to the right of the asterisk in the footnote window.
   Use only a single space between the asterisk and the build tag name. For example:

   ```
   * test_build
   ```

   Figure 6.14 shows this footnote window.

Build Tag Guidelines

When adding build tags, follow these guidelines:

- A build tag must be the first footnote in a topic.
- Build tags can be made up of any alphanumeric characters and are not case sensitive. For example, TEST_BUILD is the same as test_build.
- A build tag can have as many as 32 characters.
- Build tags may not contain spaces. Instead of spaces use underscores to separate parts of the build tag, as in this example:

  ```
  * beta2_build
  ```

- You can specify multiple build tags by separating them with a semicolon, as in the following example:

  ```
  * procedure_topics; beta2_build; test_build
  ```

Inserting Topic Comments

You can include comments with any topic in your Help file. Comments are purely optional statements that you create for your own use. They have no meaning to users or to the Help compiler, but they may be useful in your process. For example, you could use a comment footnote that tells writers and editors the topic's status. Whether you use comments and how you use them is strictly up to you. You use an at sign (@) footnote to identify a comment.

To insert a comment into a topic

1. Position the insertion point at the beginning of the topic (to the right of the other footnotes).
2. From the Insert menu, choose Footnote.
   The Footnote dialog box appears.
3. Type the at sign as the custom footnote mark, and then choose OK.
   A superscript at sign (@) appears in the text window, and the insertion point moves to the footnote window.
4. Type the comment to the right of the at sign in the footnote window.
   Use only a single space between the at sign and the first word of the comment. For example, you might type the following comment for the Saving Files topic in Cardfile Help:

   ```
   @ This topic has been edited but not proofed yet.
   ```

   Figure 6.15 shows this footnote window.

Comment Guidelines

Because the Help compiler ignores comment footnotes, comments can have as many standard characters as you want, including accented characters and spaces.

Chapter 7 Formatting Topics

Help supports most of the character and paragraph formatting features supported in Word for Windows. However, because Help topics are displayed online in windows whose sizes vary dynamically when the user changes the windows size, Helps display differs from the printed page produced by Word for Windows. Help also creates a number of formatting anomalies when translating Words paper orientation to the online medium.

This chapter explains how to format the text and graphics you place in Help topics and describes the character and paragraph formatting features Help supports. If a formatting feature in Word for Windows is not discussed in this chapter, it isnt supported.

Understanding Help Formatting

Windows Help redefines a number of Word for Windows features so that they have a different meaning in Help topic files than they do in a normal word-processing document. Learning to create Help topics is in part a matter of learning to apply standard word-processing conventions in some unconventional ways. The remainder of this chapter provides some quick-reference information that you can refer to as you are learning how to create topic files.

Character Formatting

Help topic files can include most types of Word for Windows character formatting. The following sections describe the character formatting properties that Help supports.

Fonts

Help supports multiple fonts and font sizes. Help attempts to match the typeface, type family, and size that the author specifies in the Help file. Font information is passed to the Windows font manager without special processing.

Topic layout within the Help window is based on screen fonts rather than printer fonts. Because most word processors perform their screen layout based on the installed printer, the layout in the word processor may not match exactly the layout in Help.

The following screen fonts are available on all Windows version 3.1 systems:

Courier 10,12,15
Modern
MS Sans Serif 8,10,12,14,18,24
MS Serif 8,10,12,14,18,24
Roman
Script
Small
Symbol 8,10,12,14,18,24

Windows version 3.1 also includes the following TrueType™ fonts:

Arial®
Arial Bold
Arial Bold Italic
Arial Italic
Courier
Courier Bold
Courier Bold Italic
Courier Italic
Times New Roman®
Times New Roman Bold
Times New Roman Bold Italic
Times New Roman Italic
Symbol

In many cases, the TrueType fonts provide better scaling and presentation quality than the raster or vector fonts used with Windows. If you have users with Windows version 3.0, however, they will not have any TrueType fonts on their system.

In addition to the standard Windows fonts, you can define custom screen fonts to ship with your Help file. If you dont ship all your fonts with the Help file, theres a chance that the fonts on the users system wont match the fonts available in your Help font. When a requested font is unavailable, Help lets Windows determine a compatible font to use. The Windows font-mapping function usually produces a good match, but in some cases fonts might look quite different. To avoid font mismatches, you should ship any custom fonts with your Help file or use just the standard Windows fonts.

> Hint: If you are using Word for Windows, you should cancel the Display As Printed option (version 1.1) or the Line Breaks And Fonts As Printed option (version 2.0) because Help doesnt use printer fonts. This option causes Word to display fonts as they will appear on the printed page. If you clear this option, your Help topics in Word will more closely match the way they appear in Help.

Special Fonts

Help does not accept characters from special or nonstandard fonts because RTF does not really identify them well. Windows Help assumes that text has priority and that all fonts use the ANSI character set. The symbol font is special-cased as a SYMBOL font, but there are no other exceptions. The Zapf Dingbat font is supported because it is treated as a symbol font. Nonstandard characters are not allowed.

Special Characters

Many TrueType fonts, such as Times New Roman, have special characters for basic typesetting symbols, and Help can display these symbols. The problem is that when Word saves a document in RTF format, it converts selected symbols to special RTF tags:

opening single quotation mark	\lquote
closing single quotation mark	\rquote
opening double quotation mark	\ldblquote
closing double quotation mark	\rdblquote
bullet	\bullet
em dash	\emdash
en dash	\endash

Help ignores Words RTF tags and displays nothing in their place. For example, special characters such as Words bullets dont show up in compiled Help files.

Other special characters, such as the trademark symbol, do not have special RTF tags; instead, Word saves them as \hh codes, which specify a hexadecimal character number. These are displayed correctly in Help. So if you want to use special characters in your Help file, you can edit the RTF topic files before compiling and replace the Word RTF tags with the appropriate \hh codes:

\lquote	==>	\'91
\rquote	==>	\'92
\ldblquote	==>	\'93
\rdblquote	==>	\'94
\bullet	==>	\'95
\endash	==>	\'96
\emdash	==>	\'97

Each of the search targets has a trailing space, so be sure to include in your search, but dont include a space after the replacement string.

> Hint: Dont use Words Insert Symbol command to insert special characters: it inserts symbol fields, which the Help compiler doesnt recognize. Instead, use the Windows Character Map application to copy the character to the Clipboard. Then paste the character into your topic.

> Hint: Although Help doesnt support many special characters, it does support the nonbreaking space (CTRL+SHIFT+SPACE).

Bold text

Help uses the Windows default boldness, which is achieved by doubling the pixels that make up the font. In most cases, this creates a bold that is darker and higher in contrast than the bold in printed text.

Italic text

Italic text is supported, but it is generally difficult to read online. Help adds white space on the right edge of italic text to achieve a smoother combination of italic and nonitalic text. If you mix both italic and nonitalic text in hot spots (single underlines or double underlines), the added white space may cause broken underlines.

Small Caps

Although Help supports small Caps, the font you choose may limit its usefulness. When rendering small caps in Help, the small caps ascender reaches the baseline of normal text, making the text slightly smaller than the small-caps text displayed in Word for Windows.

Colored text

You can change the text color to any valid RGB value. However, Help may modify the color if the users display does not support the authored color or if the users background color conflicts with the authored color.

Unsupported Character Formatting

Windows Help does not support the following character attributes:

- Superscript and subscript text. If you want to use superscript or subscript letters in the Help file, you can define a special font whose characters sit above or below the baseline. You can also use bitmaps of superscript/subscript characters.
- Expanded and condensed letter spacing.
- All Caps text. To use capital letters in the Help file, just type the text in all capitals using the SHIFT key instead of formatting the text as All Caps.

If you use an unsupported attribute in your topic, Help will fail to display the attribute. For example, superscripted text appears as normal text.

Windows Help uses certain character formatting attributes to create hot spots; therefore, Help does not support the standard meaning of the attributes. The character attributes that Help supports differently from Word for Windows are described in the following table:

Attribute	Purpose
Double-underlined or strikethrough text	Specifies the hot-spot text that the user will see displayed in the Help window. The user can then choose this text to make a jump to another topic or to execute a macro.
Single-underlined text	Specifies the hot-spot text that the user will see displayed in the Help window. The user can then choose this text to cause a pop-up window to appear.
Hidden text	Specifies the context string for the topic that will be displayed or the macro(s) that will execute when the user chooses the hot-spot text.

Note: For information about using each of these specific formatting properties, see Chapter 8, Creating Links and Hot Spots.

Using Strikethrough Text in Word for Windows

In Word for Windows version 1.1, deleted text with revision marks is marked with the strikethrough property, which in RTF is indicated with the \strike statement.

In Word for Windows version 2.0, \strike is used to mark revisions and \deleted was added to indicate deleted text with revision marking. So if you save a version 1.1 file as RTF, strikethrough text loses its revision markings but looks the same on screenstrikethrough. Conversely, if you save a version 2.0 RTF file that has strikethrough text and open it in Version 1.1, the strikethrough text acquires the revised \ deleted property.

Therefore, if you use strikethrough text to indicate hot spots in your topic files, you should not edit the RTF files in both Word for Windows versions (1.1 and 2.0). Instead, convert all your Version 1.1 files to Version 2.0 by opening them and then saving them as RTF. Or, if you must use both versions of Word for Windows, do not use strikethrough or deleted text attributes.

Inserting Superscript and Subscript Characters

Even though Help does not support superscript and subscript character formatting attributes, you can insert these characters into your Help files using either of two workarounds. You can:

- Use a font editor to define a custom superscript/subscript font whose characters sit above or below the baseline.
- The custom font should be a larger point size than your normal body text so that you can place the superscripted/subscripted characters in the high or low portion of the font character.
- Create bitmaps of the superscript/subscript characters and place them in the Help file using bmc references.
- Make sure that you save the bitmaps without extra white space around them so that they display the same as regular text on-line.

Paragraph Formatting

Help supports most of the major paragraph formatting features of Word for Windows, and in most cases paragraphs displayed in the Word document look the same in Help. The main difference between a Word document and a Help topic is the display medium: a document is printed on a piece of paper that has a fixed size, but a topic is displayed in a window. Window sizes can vary, and in the case of main and secondary windows the user can change the size of the window after the topic has been displayed. Many of the differences between the look of a topic in Word and the look of a topic in Help stem from Helps support for sizeable windows.

To support sizeable windows, Help uses a word-wrapping mechanism to ensure that paragraph text is not clipped at the right margin. You can disable the word wrapping by following the instructions in Disabling Word Wrapping in Paragraphs, later in this section.

The following sections discuss the paragraph formatting features supported by Help. For information on table formatting, see Table Formatting, later in this chapter.

Alignment

Help supports the following paragraph alignment attributes:

- Left alignment
- Right alignment
- Center alignment

These attributes have the same meaning as in Word for Windows, except that left-justified and right-justified text is justified to the Help windows internal margin of 8 pixels on the left.

Help does not support justified text.

Indents

Help supports the following paragraph indentation attributes:

- Left indents, including positive and negative values.
- Left indents are relative to the left window border. Negative values position the left margin of the paragraph to the left of the left window border.
- Right indents, positive values only.
- Right indents are relative to the right window border (or the vertical scroll bar, when present).
- First line indents, including positive and negative values.
- Word defines an implied tab stop at the position of the left indent, but Help does not provide this tab stop. To tab to the left indent position, define a left tab at the same position as the left indent.

Indents have the same meaning as in Word for Windows, but they may have different measurement values. Figure 7.1 shows how Word indents are used in paragraphs displayed in Help.

<Help paragraph. Callouts to show left, right, and first-line indent values>

The Help windows 8-pixel margin on the left prevents text from bumping against the left window border. You can specify a negative indent to push text or graphics closer to the left edge; however, if your negative indent is too large, you may push the text off the screen where the user cant scroll to see it. A negative indent of 7 points will move the text to the edge without pushing it beyond the window border.

With a negative left indent, you can bleed pictures off the left window border. For example, the following topic uses a negative left indent of 3 picas to crop the left edge of a picture (Figure 7.2).

<Help bleed>

Paragraph Spacing

Help supports all Word paragraph spacing attributes.

The Space Before and Space After attributes add white space before and after a paragraph, which is the same function they have in the word processor, but they may have different values in Help.

Text displayed in the Help window includes a 3-pixel top margin to prevent it from bumping against the button bar. However, bitmaps do not include a margin. So if you use a bitmap in the first paragraph of a Help topic, you should include space before if you want the picture to have white space around it. Otherwise, the picture will display immediately below the button bar with no intervening white space. This is a helpful technique to use when you want to fill the Help window with a picture.

Line Spacing

Help supports two kinds of line spacing: Auto and Absolute (or exact). Automatic line spacing (marked with Auto, Single, 1.5, or Double line-spacing attributes) creates a variable line spacing that is determined partly on the height of the largest character (or bitmap treated as a character) within the line. Auto line spacing ensures a safe spacing that adjusts to the content of the paragraph. However, Helps method for determining the appropriate spacing for an automatically spaced line doesnt take into account line length, color, and other attributes that can affect the legibility of a paragraph.

Absolute or exact line spacing is determined by the author and is set to the absolute value specified. Absolute line spacing is used when the author wants more control over the way paragraphs appear in Help. It can be used to add white space and create more openness, or it can be used to reduce the white space between lines and tighten up the paragraph. If you want to condense or expand the line spacing in a paragraph, use the Exactly line spacing attribute, and specify a point size for the line. Or, if you are using Word for Windows 1.1, specify the exact line spacing that you want, but use a negative value to indicate that the spacing is absolute or exact.

Tabs

Help supports the following types of tab stops:

- Left-aligned tabs
- Right-aligned tabs
- Center-aligned tabs
- Default tab stops (up to 32)

Although Help supports these tab stops, very complex combinations of alignment and tabs may not appear the same in Help as they do in Word for Windows.

Help does not support leader characters or decimal tabs. Decimal tab stops are treated as left tab stops.

In paragraphs with first-line indents, Word defines an implied tab stop at the position of the left indent. Help does not provide this tab stop. To tab to the left indent position, define a left-aligned tab at the same position as the left indent.

If you include embedded windows in your Help file, do not use right-aligned or center-aligned tabs to position the embedded window. Use indents instead.

In many cases, Word tables are more flexible and easier to use than tabbed paragraphs. For information on using tables, see Creating Tables, later in this chapter.

Using Paragraph Borders

Help supports the following paragraph borders:

- Any combination of left, right, top, and bottom borders, including boxed paragraphs (all four borders)
- Thin single borders (1 pixel wide)
- Thin double borders (each line 1 pixel wide)
- Thick single borders (2 pixels wide)

Figure 7.3 shows which line-width choices in the Word for Windows version 2.0 Border Paragraphs dialog box translate to the supported border types (thin single, thin double, and thick single).

<Word Border Paragraphs, circles/callouts identifying which line values produce which border types>

Help does not support the following border attributes:

- Colored borders (Help draws all borders in black)
- Shadowed borders
- Hairline borders
- Adjustable distances from text
- Paragraph shading or background coloring
- Table-cell borders (set in the Border Cells dialog box)

Paragraphs inside tables can have borders, but the table itself cannot have borders. For more information, see Table Formatting, later in this chapter.

Word combines border attributes in consecutive paragraphs. For example, if two consecutive paragraphs have a boxed border, Word draws a single box around both paragraphs rather than drawing boxes around the individual paragraphs. Help does not combine border attributes of consecutive paragraphs; for consecutive boxed paragraphs, Help draws individual boxes around each paragraph.

Disabling Word Wrapping in Paragraphs

By setting the Keep Lines Togetherparagraph attribute, you can prevent Help from word-wrapping a paragraph. When the window border is sized inside the right boundary of the paragraph, Help adds a horizontal scroll bar and clips the right edge of the paragraph. If the paragraph is several lines long, youll need to place forced line breaks where you want the lines to break.

To prevent a paragraph from wrapping

1. Place the insertion point anywhere in the paragraph.
2. From the Format menu, choose Paragraph.
   The Paragraph dialog box appears.
3. In the Keep Paragraph box (Word 1.1), select Together.
   Or, in the Pagination box (Word 2.0), select Keep Lines Together.
4. Choose OK.

To create a line break in a nonwrapping paragraph

1. Place the insertion point where you want the line to break.
2. Press SHIFT+ENTER.

> Note: Windows Help does not support the standard meaning of the Keep With Next paragraph attribute. In Help, this property specifies that the paragraph should be included in the nonscrolling region of the Help window. For information about using this formatting property, see Chapter 9, Defining Topic Windows.

Unwanted Line Breaks

Because Windows Help treats any font or style change as a word break, punctuation characters may wrap to a new line and become orphaned from the sentence they belong to if they follow a hot spot or a bitmap reference. In these cases, the punctuation character follows Help codes that require format changes (from double-underline to hidden text to normal text, for example). Because of the format change, Help treats the transition as a word break and wraps the trailing punctuation when necessary (depending on how the user resizes the Help window).

To prevent this from occurring in your Help file:

- If the punctuation follows a hot spot, include the punctuation mark in the hot spot (before the hidden text).
- If the punctuation follows a bitmap reference, use a nonbreaking space between the bitmap reference and the punctuation mark.

Table Formatting

Tables are flexible and powerful formatting tools for Help topics and are the easiest way to create complex arrangements of text and graphics. They are particularly useful for positioning embedded windows in ways not supported by the basic Help alignment attributes.

Help lets you create left-aligned tables using any combination of rows and columns. You can place any type of topic information, including text and embedded windows, in table cells. Help wraps paragraph text inside tables, but it doesnt change the word wrapping of table paragraphs when the window is resized. The table is a fixed object, and its width remains constant as the window is sized. If the right window border clips the right side of the table, Help displays a horizontal scroll bar to let the user scroll the hidden information into view.

Help supports the following table formatting attributes:

- Left-aligned tables
- Variable column width and inter-column spacing
- Left indent for the table
- Inter-row spacing (using paragraph spacing attributes)
- Merged cells

Paragraphs inside tables can use any of the paragraph formatting attributes described in Formatting Paragraphs, earlier in this chapter.

Help does not support the following Word table features; the Help compiler displays a Table formatting too complex message if it encounters unsupported table features in a topic:

- Cell borders (although paragraphs inside tables can use borders)
- Custom row heights
- Right-aligned and center-aligned tables

> Note: If you apply a border to a paragraph contained in a table, Help applies the border to the table cell, not the paragraph. To use paragraph borders inside tables, you must either apply the border to the paragraph style or apply the border to the paragraph before moving it into the table.

Creating Tables

In Windows Help version 3.1, you can create simple tables (tables without borders or shadows) using the table formatting features in Word for Windows. In Word for Windows, a table is simply a collection of paragraphs that have been divided into individual cells. The Help compiler interprets these tables as a type of side-by-side paragraph formatting and displays them in the Help window.

You can author a table either as fixed width or relative.

Creating Fixed-Width Tables

A fixed-width table is one that appears in Help exactly the way you create it. In other words, the table retains the original width when displayed in Help, no matter how the user resizes the window.

To create a fixed-width table

1. Place the insertion point where you want to insert the table.
2. From the Insert menu, choose Table (Word 1.1).
   Or, from the Table menu, choose Insert Table (Word 2.0).
   The Insert Table dialog box appears.
3. Type the number of columns and rows you want.
   Or, if youre not sure what your requirements are, you can accept the default settings and make the appropriate additions and changes later.
4. Choose OK.
5. Type the information in each of the cells.
   Figure 7.4 shows a two-column table in a topic file.

Creating Relative Tables

Help also supports relative or autosized tables. Unlike fixed-width tables, the columns in relative tables wrap dynamically when the user resizes the Help window. However, the columns in relative tables will only maintain their relative widths down to a minimum size. After that, they will not wrap. The minimum size is determined by the author who creates the table.

Relative or autosized tables are created by making the table rows centered. However, the procedure is different depending on which version of Word for Windows you are using. In Word for Windows version 1.1, you create centered tables by using the Format Table command.

To create a relative table using Word for Windows 1.1

1. Create a standard table.
2. From the Format menu, choose Table.
   The Format Table dialog box appears.
3. Select the Align Rows Center option button.
4. Select Apply To Whole Table, and then choose OK.

Word for Windows version 2.0 does not have a Format Table command, so if you create a centered table using a frame and the Format Frame command, Word does not generate the correct RTF for the Help compiler to create an autosized table. To create the table in Word 2.0, use the Row Height command.

To create a relative table using Word for Windows 2.0

1. Create a standard table and select all the rows.
2. From the Table menu, choose Row Height.
   The Row Height dialog box appears.
3. Under Alignment, select the Center option button.
4. Choose OK.

Setting the Minimum Width

The authored column widths are the minimum widths that will be displayed. Windows Help will make the table larger to fit a larger window, but it will not wrap it smaller. If the window is sized smaller than the authored table, the window displays horizontal scroll bars. Therefore, you must author the table at the smallest size that you want it to take in the window. For example, if you want it to wrap small, you should create the table at its smallest size, like this:

This	This
text	text
is in	is in
the	the
first	second
column.	column.

Windows Help sizes each column proportionally, but Help will not break words within the table to create nice text blocks. Windows Help does not understand optional hyphens or word breaking at all. So you are responsible for creating the right margin you want in this small size.

Adjusting the Table Columns

When you size table columns in Word for Windows, you can use three different methods. Using methods two and three allow you to reset the right column border to its former position after sizing the table:

- If you size by dragging a column border, columns to the right are shifted horizontally, but their width remains the same. The result is that the right boundary of the table changes position.
- If you hold down the SHIFT key before dragging the column border, the column immediately to the

right is sized, but its right border stays the same.

- If you hold down CTRL before dragging the column border, all columns to the right are sized proportionally, and the right boundary of the table stays the same.

Creating Tables Without Using Word's Table Feature

Because Windows Help version 3.0 does not support tables, you must use alternative methods to create tables. There are three ways to create tables without using the table feature in Word for Windows:

- Use a hanging indent to place short pieces of text in the beginning column(s) and longer text pieces in the final column.
- As the user resizes the window, the text in the outside column wraps to accommodate the new window size.
- Use Word for MS-DOS or Word for the Macintosh to create side-by-side paragraphs.
  Using this method, you can create tables with only two columns.
- Format the table manually using tab stops, and then format the entire table as nonwrapping to keep the table intact when the user resizes the Help window.

Hanging Indent Tables

The easiest way to create a simple table without the table feature is to use a hanging indent. The hanging indent creates the first column and forces the text in the second column to wrap to the indent.

First column	The hanging indent creates the first column. Text in the second column starts at the tab stop and wraps to fit the current window size, regardless of how wide it is in the topic file.

You can also create hanging indent tables with more than two columns as long as the information in the first columns is short.

First column	Second column	The hanging indent and tabs create the first and second columns. Text in the third column starts at the second tab stop and wraps to fit the current window size, regardless of how wide it is in the topic file.

Hanging indent tables have limited uses, but they work best when there are only two or three columns and the text in the first columns is short enough to fit on one line without wrapping. To have a hanging indent table appear correctly in Help, you must use a tab stop at the beginning of the tables left margin (where the wrapping column begins). You can use hanging indent tables in Help versions 3.0 and 3.1.

To create a table using a hanging indent

1. Type the text for the first column, press TAB, type the text for the second column, and then press ENTER.
2. Repeat step 1 until you have created all the rows in the table, including the table heading.
3. Select all the paragraphs in the table.
4. Hold down the SHIFT key while dragging the left (bottom) indent marker to the position you want for the left indent.
5. Insert a tab stop at the same location where you placed the left indent marker.

   Note If you want more precision, you can choose the Format Paragraph command nd type in the measurements for the left indent, first line indent, and tab stop.

   Note The style sheet in the WHAT30.DOT and WHAT31.DOT Help Authoring Templates includes two styles, Th and Tp, that you can use to create hanging indent paragraphs automatically.

   Figure 7.5 shows a two-column table created with a hanging indent.

Side-By-Side Paragraphs

To create tables with similar amounts of text in each column, you can use Word for MS-DOS or Word for the Macintosh version 3.x. Refer to the documentation accompanying these products to learn how to create side-by-side paragraphs.

Note: Windows Help version 3.1 no longer supports side-by-side paragraph formatting. You can display a Help file that contains side-by-side paragraphs only if the file was built with the Help 3.0 compiler. To build the file using the Help 3.1 compiler, you must convert any information that uses that format into a Word table or paragraphs with a hanging indent. If you open the old topic files in Word for Windows, Word converts the side-by-side paragraphs into table format automatically.

Manually Formatted Tables

To create more complicated tables than hanging indents allows, or if you cannot create side-by-side paragraphs because you dont have Word for MS-DOS or Word for the Macintosh, you can format the tables manually using tab stops. The tab stops align the text into columns as shown below:

First column text	Second column	Text in the third column cannot wrap
is built up line by	text is kept to	because each line in the table is a
line with a strict	a strict column	separate paragraph.
column width.	width also.	

Each column in a manually formatted table is a separate paragraph, so the entire table should be formatted as Keep together to keep the table rows from wrapping when users resize the Help window. If portions of the table are hidden, Help displays a horizontal scroll bar so that users can view the hidden information.

To create a table manually using tab stops

1. Type the text for the first column, press TAB, type the text for the second column, press TAB, type the text for the third column, and then press ENTER.
2. Repeat step 1 until you have created all the rows in the table, including the table heading.
3. Select all the paragraphs in the table.
4. From the Format menu, choose Paragraph.
5. In the Keep Paragraph box (Word 1.1), select Together.
   Or, in the Pagination box (Word 2.0), select Keep Lines Together.
6. Choose OK.
   Figure 7.6 shows a two-column table created with tabs.

Chapter 8 Creating Links and Hot Spots

So far we have been talking about creating individual topics. Thats certainly a necessary first step, but it isnt sufficient to create a Help file because unless you create links between those topics, users have no way to access the information. Unlike books, Help files dont have physical pages users can turn. They rely instead on links, the structural and navigational connections between topics. Links determine which topics are connected to each other and the relationship between connected topics. For example, a link may determine that Topic B is subordinate to Topic A or that Topic C is cross-referenced from Topic A.

Linking information together electronically is what makes normal text into hypertext. Normal text is what we read in most printed materials: newspapers, magazines, novels, brochures, catalogs, instruction manuals, and so on. It is structured sequentially so that people read one page after another. Hypertext simply means that you can create links between the electronic pages (topics) in such a way that users can read the material nonsequentially, or in any order they choose. If the hypertext is well-made, they should be able to choose links according to the associations they form when they read the information. In Windows Help, this notion of hypertext is the basis for all Help files.

This chapter explains how to create links between topics and topic files. It also explains how to create hot spots that run Help macros and activate pop-up windows.

Linking Topics Using Jumps

Help provides many ways to link topics, but most methods are implicit: they connect topics by category (browse sequences) or by subject matter (keywords). These methods dont provide a concrete connection for the user: click something and a topic appears. For that reason, the most common method for linking topics is to use jumps. Jumps are one-way links between topics: when the user chooses a jump, Help displays the topic connected by the link (usually referred to as the destination topic). Because jumps are one-directional, a link from Topic A to Topic B does not guarantee a link in the opposite direction. Jumps can occur automatically when a topic is displayed, or they can occur when the user clicks a hot spot, which is a special topic region within a Help topic. A jump hot spot is a word or a graphic that the user chooses to move to another topic in the Help file. Hot spot text appears as green, underlined text in the Help window. Graphics used as hot spots do not change appearance.

Jumps are a flexible navigational tool. You can create many links between topics and display topics in a variety of windows. By effectively linking information in your Help file, you can make it easy for users to find specific information. You can link related topics using cross-references, inviting your users to explore related information. You can create logical paths through the information so users can browse without getting lost.

Creating Tables of Contents

Users can only see the information you have created if you provide them with jumps to that information. If you dont provide a link to a particular topic, users will not be able to choose that information. (Of course, you can have topics that are accessed only by keyword search, but for the moment we are assuming that users are making their choices based on what is visible in the main Help window.) For that reason, it is a good idea to create a Help Contents topic as a minimal device for providing access to every topic in your Help system. That way users will have at least one jump to all primary topics.

Jumps provide a good way to create structure and organization within a Help file. For example, you can use jumps to create a hierarchical browsing order for a collection of topics. Consider the following graph, which represents the organizational structure of a Help file as defined by the topic jumps.

Figure 8.1. Using Links to Create an Information Hierarchy

Here, the topic links create a clear information hierarchy. The four table of contents topics (C0 through C3) contain lists of hot spots that connect to information topics (Inf1 through Inf11). The contents topics let the user move from general to detailed lists of information, quickly arriving at the appropriate topic. For example, the Figure 8.2 shows two contents topics.

With a three-level hierarchy as shown here, users could access any topic using only two mouse clicks. Add a third level of contents topics, and you provide access to all topics using only three mouse clicks.

Using Jumps to Reference Related Information

In addition to creating overall structure in the Help file, you can use jumps to create links between topics that contain related information. For example, the following graph shows a few cross-reference links between topics in our sample Help file.

Figure 8.3. Adding Cross-Reference Links

The cross-reference links let users quickly find related information without having to return to the Contents topic (C0). Cross-reference jumps can be an effective tool for providing multiple levels of detail in the same topic. A topic might contain detailed information thats interesting to some users, but not all. Using cross-reference jumps, you can hide the more detailed information in a separate topic, as shown in Figure 8.4:

Jump Destinations

The topic displayed as a result of a jump is called the destination topic. You can create jumps to any topic in the current Help file. You can also create jumps to a specific location within a topic.

> Note: Using Help macros and other techniques, you can create jumps to topics in other Help files. For more information, see Creating Links Between Help Files, later in this chapter.

You can display the destination topic in the main Help window, in a secondary window, or in a pop-up window (Figure 8.5).

For a complete discussion of Help windows, see Chapter 9, Defining Topic Windows.

Defining Mid-Topic Jump Destinations

Normally, Help displays the first line of the topic at the top of the Help window. However, you can define jump destinations anywhere within a topic. That way when Help displays the topic, the mid-topic destination appears at the top of the Help window. Mid-topic jumps are especially useful if you have longer topics, because you can create jumps that take users directly to the relevant information within the topic.

To insert a mid-topic destination

1. Position the insertion point where you want the topic to appear in the Help window.
2. From the Insert menu, choose Footnote.
   The Footnote dialog box appears.
3. Type the pound sign (#) as the custom footnote mark, and then choose OK.
4. Type a context string for the mid-topic destination.

The mid-topic destination appears at the top of the Help window, as in Figure 8.6.

Using Jump Macros

Help provides several macros that also perform topic jumps. You might consider using these macros to create a specialized user interface for displaying topics. For information on using macros, see Chapter 14, Help Macros.

Activating Jumps

You can author a jump to be activated automatically when the topic is displayed or when the user chooses a hot spot.

Entry Macros

You can place entry macros in any topic and in the Help project file. Entry macros let you run one or more Help macros with the display of the Help file or a topic. Entry macros linked to topics can be activated by jump hot spots. For example, if users jump to a destination topic with an associated entry macro, the macro string for that topic is run automatically.

Entry macros are a powerful tool for configuring Help to display special topics or to provide helpful navigational feedback. Chapter 14, Help Macros, discusses the different ways a Help file can run macros and suggests how to use them in your Help file.

Using Entry Macros to Create Automatic Jumps

Entry macros have many uses, one of which is to create automatic jumps when a topic is displayed. For example, you can use an entry macro to display a group of topicsin the main Help window and in one or more secondary windowswhen the user enters a particular topic.

When displaying groups of topics together, you should identify a master topic to display in the main window. By placing an entry macro in the master topic, Help can display the whole group of topics when users jump to the master topic. After displaying the master topic, Help processes the automatic jumps contained in the master topic and displays the appropriate topics in the other windows, as in Figure 8.7:

Creating Hot Spots

A hot spot is a region within a Help topic that users can interact with. Help changes the mouse pointer to a hand when the pointer passes over a hot spot. The user can click the mouse to initiate the action associated with the hot spot.

You can format the following elements as hot spots.

Element	Description
Text	A contiguous series of characters.
Picture	A single picture.
Hypergraphic	A rectangular region defined on top of a special picture called a hypergraphic.

How Jump Hot Spots Appear in Help

After you build the Help file, users can identify hot spots in two ways:

- By default, the text in jump hot spots (if they are not formatted as invisible) appear as green, underlined text.
- When the mouse pointer moves over a jump hot spot, the pointer changes from an arrow to a hand. This change occurs whether or not the hot spot is visible.

Figure 8.8 shows a Help topic that has all three types of hot spots. Notice the hot areas created by each type of hot spot. To choose a hot spot, users position the mouse pointer over a hot spot and click the left mouse button.

When the user clicks a hot spot, Help processes the associated action immediately. The action occurs at the moment the mouse button is depressed.

Text Hot Spots

A text hot spot uses a word or phrase as the hot region. The characters in the text hot spot must be contiguous. Help can word-wrap text within a hot spot, so the hot region might extend to two lines or more. By default, Help displays text hot spots as dark green text with a single underline. To override the default formatting, you can use the method described in the Changing the Standard Appearance of Hot Spots section, later in this chapter. Help then uses whatever Word formatting you apply to the text.

If you change the default formatting, keep in mind the following considerations:

- Use some visual or semantic cues to identify the hot spots.If you make the hot spot look exactly like surrounding text, users will have to move the mouse pointer around to find hot spots. This can frustrate users or prevent them from finding important information youve included in your Help file.
- The single underline used with the default hot-spot formatting helps users identify hot spots on monochrome displays. Also, the underline lets color-blind users distinguish hot spots from surrounding text.
- Usability tests with Windows Help files have shown that some users confuse italic text with hot spots.

Bitmap Hot Spots

You can create graphical hot spots using bitmaps. With a bitmap hot spot, the user clicks anywhere on the bitmap to activate the hot spot. Bitmap hot spots use the same display mechanism as regular pictures, so you can use your 16-color bitmaps and metafiles as hot spots.

Bitmap hot spots use the same positioning options as regular pictures (except for inline bitmaps, which dont support text wrapping). For information on the alignment options for Help graphics, see Chapter 11, Creating Hypergraphics.

Hypergraphic Hot Spots

You can create pictures with multiple hot spots so that the user can interact with different parts of the picture. Because each hot spot can initiate a different action, you can use hypergraphics to extend the usefullness of graphics in your Help file. To create hypergraphic hot spots, you use a separate application called Hot Spot Editor, which is described in Chapter 11, Creating Hypergraphics.

Types of Hot Spots

Help lets you define several types of hot spots.

Hot spot type	Action
Jump	When the user clicks the hot spot, Help displays the destination topic in the specified window, either main or secondary.
Interfile jump	When the user clicks the hot spot, Help opens a new Help file and displays the destination topic in the specified window.
Pop-up	When the user clicks the hot spot, Help displays the topic in a pop-up window.
Macro	When the user clicks the hot spot, Help executes the specified macro.
Invisible	The hot spot has no visible formatting to indicate it is a hot spot. However, the pointer changes to a hand when moved over the hot spot.

Special Hot-Spot Characters

When creating hot spots in Help topics, Windows Help assigns a specific meaning to certain standard characters. These special Help characters are described below.

Character	Name	Purpose
Asterisk (*)	Underlined hot spot	Specifies that the hot spot should be underlined only. The default green color is removed from the hot spot.
At sign (@)	Interfile jump	Specifies that the jump is to a topic located in a different .HLP file from the current file.
Exclamation point (!)	Macro hot spot	Specifies that the hot spot contains a Help macro to be executed.
Percent sign (%)	Invisible hot spot	Specifies that the hot spot should be invisible. The default green color and underlining are removed.
Right angle bracket (>)	Window name	Specifies that the topic is to be displayed in the specified window, either the main Help window or a secondary window.

For information about using each of these special characters, see the following sections in this chapter.

Inserting Standard Jump Hot Spots

The coding for a standard jump hot spot has two parts:

- A reference word or phrase formatted as double-underlined text (the jump hot-spot text) that the user chooses to make the jump
- The context string, formatted as hidden text, that identifies the destination topic

> Note: In Word for Windows, select the Hidden Text check box in the View Preferences (version 1.1) or Tools Options (version 2.0) dialog box to display hidden text. You can also assign a macro key to make selecting this option more convenient.

To format a word or phrase as a jump hot spot

1. Select the jump hot-spot text.
2. From the Format menu, choose Character.
3. Select the Double Underline check box, and then choose OK.
4. Position the insertion point immediately after the last letter in the double-underlined jump hot-spot text.
5. From the Format menu, choose Character again.
6. Clear the Double Underline check box, select the Hidden check box, and then choose OK.
7. Type the context string assigned to the topic that is the target of the jump.
8. From the Format menu, choose Character again, clear the Hidden check box, and then choose OK.

Figure 8.9 shows a correctly formatted jump hot spot in a topic file.

Jump Hot Spot Guidelines

When creating jump hot spots, follow these guidelines:

- You can have spaces within the hot spot text, but be sure no spaces are between the double-underlined jump text and the hidden text context string. For example, the following is correct.

  [missing graphic]

  The following is incorrect.

  [missing graphic]
- If you place the context string at the end of a paragraph or at the end of a line with a soft carriage return (SHIFT+ENTER), be careful not to format the paragraph or carriage-return mark as hidden text.

Creating Links Between Help Files

If you build a Help system that has more than one Help file, you can link the information together by creating jumps from one Help file to another, or interfile jumps. An interfile jump is simply a jump from a topic in one Help file to a topic in a different Help file. (Jumps like the ones weve discussed so far occur within a single Help file. Remember, Help file does not refer to the individual RTF topic files that contain the topics, but to the compiled, binary .HLP file.)

> Note: Chapter 16, The Help Project File, contains information on building a multifile Help system. Here, it is assumed that youve compiled the individual Help files and want to create a jump from one file to another.

You can use several methods to create interfile jump hot spots:

- Create an interfile jump hot spot
- Create a macro hot spot and use one of the Jump macros
- Have an application or DLL send an API message to Help requesting a new Help file and topic

How Interfile Jump Hot Spots Appear in Help

In the built Help file, interfile jumps appear exactly the same as standard jumps. For our example, the interfile jump hot spot appears as the underlined text Jump Text (see Figure 8.x). To make the jump, users position the mouse pointer over the interfile jump hot spot text and click the left mouse button.

Windows Help finds and displays the topic referenced by the interfile jump hot spot, as in Figure 8.x.

Creating Standard Interfile Jump Hot Spots

The simplest way to create an interfile jump is to create a standard jump hot spot and add the name of the Help file:

```
context-string@.HLP-filename
```

Context-string is the context string for the topic you are jumping to, and HLP-filename is the name of the .HLP file that contains the topic.

To create an interfile jump hot spot

1. Follow the steps to create a standard jump hot spot.
2. Insert an at sign (@) immediately after the last character of the context string.
3. Type the name of the file that contains the context string assigned to the topic that is the target of the interfile jump.

   Note Both the at sign and Help filename must be formatted as hidden text.

   Note You can also specify a path along with the Help filename. Generally, this is not recommended because the path is likely to vary from computer to computer and also because the Help file may change its location later. If you have circumstances that diminish these risks, you may want to specify a path, as in these examples:

   ```
   ctx_string@\\server\share\file.hlp
   ctx_string@c:\dir1\dir2\file.hlp
   ```

   Figure 8.x shows a correctly formatted interfile jump hot spot in a topic file.

For more information about creating interfile jumps using macro hot spots, see the Creating Macro Hot Spots section, later in this chapter. For more information about the Jump macros, see Chapter 15, Help Macro Reference. For an explanation of the WinHelp API, see Chapter 19, The WinHelp API.

How Windows Help Locates Help Files

To perform an interfile jump, Help must be able to find the destination Help file on the users system. When the user chooses an interfile jump hot spot, Help looks for the Help file in the following locations:

1. The directory of the Help file that initiated the interfile jump (Helps current directory).
2. The MS-DOS current directory.
3. The users Windows directory.
4. The Windows SYSTEM directory.
5. The directory containing WINHELP.EXE.
6. The directories listed in the users PATH environment variable.
7. The directories specified in WINHELP.INI.

If Help cannot find the Help file after searching in all these locations, it displays a Help file not found error message.

> Note: Help also uses this search order when you execute Help from the MS-DOS command line or when an application sends an API call to Help.

Creating a WINHELP.INI File for Search Paths

If you are distributing your application and Help files on removable media such as CD-ROM, you can have your setup program define the drives and directories where Help should look for Help files. The paths you define will be added to Helps regular search paths when performing interfile jumps.

The syntax for the WINHELP.INI entry is:

```
[FILES]HLP-filename=drive:\directory-string, message
```

Parameter	Description
HLP-filename	The name of the Help file for which you want to define a search path.
drive	The letter of the drive that contains the Help file. The drive letter must be an alpha character. You can specify as many as 26 drives.
directory-string	The directories where Help should look for the Help file. The directory string can have as many as 256 characters.
message	The message that Help should display in a dialog box, prompting the user to insert the correct disk into the designated drive. The message can have as many as 50 characters.

If the application or setup program creates an invalid entry, Help will display the standard Help file not found error message. So, Help authors are encouraged to test the validity of each WINHELP.INI entry.

Example

The following entry tells Help to look in the \PRODUCT\FILES directory of the P drive for the Cardfile Help file. If necessary Help will prompt the user to insert the CD into the drive:

```
[FILES]
CARDFILE.HLP=P:\PRODUCT\FILES, Please insert your Windows version 3.1 CD
into drive P:.
```

When creating entries in the WINHELP.INI file, you must also observe the following guidelines:

- Each entry can be no longer than 512 characters.
- The first application to create the WINHELP.INI file must also create the [FILES] section heading. You can use use the Windows function WritePrivateProfileString to create entries in the WINHELP.INI file. For more information on this function, see the Windows version 3.1 SDK.

Creating Links to Version 3.0 Help Files

In some circumstances you may want to create a jump hot spot in your 3.1 Help file that jumps to a topic in a 3.0 Help file. If you create a standard interfile jump, Help will display the Help topic does not exist error message when the user chooses the jump. That is because Windows Help version 3.1 does not allow you to jump to any arbitrary topic in a 3.0 Help file based on the context string. However, you can use macros to create links to topics in a 3.0 Help file.

From a 3.1 Help file, you can:

- Jump to the Contents topic in a 3.0 Help file by using the JumpContents macro, as in this example:

  ```
  !JumpContents("30file.hlp")
  ```

- Jump to a topic with a particular keyword in a 3.0 Help file by using the JumpKeyword macro. This method will work only if the topic has the keyword defined for it and that topic appears first in the list of topics that have that keyword defined.

  ```
  !JumpKeyword("30file.hlp", "keyword")
  ```

- Jump to a specifc topic in a 3.0 Help file that has a context number defined in the [MAP] section of the Help project file by using the JumpContext macro.

  ```
  !JumpContext("30file.hlp", number)
  ```

- Jump to any topic within a browse sequence in a 3.0 Help file by combining one of the above Jump macros with the Next and Prev macros. This method will work only if the topic you want to access is on a browse sequence, and you know the context number of another topic on the same browse sequence. When you use this method, Help will not display any of the intermediate topics in the sequence.

  ```
  !JumpContext("30file.hlp", 801);Next();Next();Next()
  ```

For information on how to create a macro hot spot, see the Creating Macro Hot Spots section, later in this chapter.

Interfile Jump Hot Spot Guidelines

When creating interfile jump hot spots, follow these guidelines:

- You can have spaces within the hot spot text, but be sure no spaces are between the double-underlined jump text and the hidden text context string, as in Figure 8.x:

  Figure 8.x shows an incorrect example:
- If you place the context string at the end of a paragraph or at the end of a line with a soft carriage return (SHIFT+ENTER), format the paragraph or carriage-return mark as normal text (dont format it as hidden text).

Creating Pop-Up Hot Spots

Windows Help lets you display information in a pop-up window that appears whenever the user chooses a hot spot such as a word or a phrase or a graphic (see Chapter 10, Adding Graphics). A pop-up window is a window that appears on top of the main Help window and contains a topic.

You can use pop-up windows to display hints, notes, or any other type of supplementary information. A pop-up window can contain text, graphics, or hot spots (jump hot spots, pop-up hot spots, or macro hot spots). However, if you have a hot spot within a pop-up window, the first pop-up window closes when the user chooses the hot spot. In other words, you can only display one pop-up window at a time on the screen.

> Note: The procedures for creating pop-up hot spots are exactly the same as those for creating jump hot spots with one exception: the text for pop-up hot spots is formatted with a single underline and jump hot spot text is formatted as double-underline. Therefore, if you are familiar with the material in the previous section, you may want to skim the information in this section.

How Pop-Up Hot Spots Appear in Help

After you build the Help file, users can identify pop-up hot spots in two ways:

- By default, pop-up hot spots (if they are not invisible) appear as green, dotted-underlined text.
- When the mouse pointer moves over a pop-up hot spot, the pointer changes from an arrow to a hand. This change occurs whether or not the hot spot is visible.

In the sample pop-up window in Figure 8.x, the first paragraph includes a pop-up hot spot that displays a pop-up window when chosen. The words Pop-up text appear in the pop-up window.

To see the pop-up window, the user clicks the hot spot, which causes the window to appear directly on top of the current topic, as in Figure 8.x.

The pop-up window includes a shadow to make it more distinguishable from the main window, and it remains open until the user clicks the left mouse button again or presses any key (which includes choosing a hot spot within the pop-up window).

Creating Standard Pop-Up Hot Spots

The coding for a standard pop-up hot spot has two parts:

- A word or phrase, formatted as underlined text (the hot-spot text), that the user chooses to display the pop-up window
- The context string, formatted as hidden text, that identifies the pop-up topic (what the user sees in the pop-up window)

To format a word or phrase as a pop-up hot spot

1. Create the content of the pop-up topic.
2. Define a context string footnote for the pop-up topic.
   For example, the following context string identifies the pop-up window topic for Topic A:
   ```
   # ctx_string_topic_a
   ```
3. Go to the topic from which you want to display the pop-up window.
4. Select the hot spot text that will display the pop-up window.
5. From the Format menu, choose Character.
6. Select the Underline check box, and then choose OK.
7. Position the insertion point immediately after the last letter in the underlined hot spot text.
8. From the Format menu, choose Character again.
9. Clear the Underline check box, select the Hidden check box, and then choose OK.
10. Type the context string assigned to the pop-up topic.
11. From the Format menu, choose Character again, clear the Hidden check box, and then choose OK.
    Figure 8.x shows a correctly coded pop-up hot spot in a topic file.

Pop-Up Hot Spot Guidelines

The same guidelines for creating jump hot spots apply to creating pop-up hot spots:

- Be sure no spaces are between the underlined pop-up hot spot text and the hidden text context string, as in Figure 8.x:

  Figure 8.x shows an incorrect example:
- If you place the context string at the end of a paragraph or at the end of a line with a soft carriage return (SHIFT+ENTER), format the paragraph or carriage-return mark as normal text (dont format it as hidden text).

Creating Macro Hot Spots

In addition to jump hot spots and pop-up hot spots, you can also create macro hot spots, a word or a phrase or a graphic (see Chapter 10, Adding Graphics) that the user chooses to execute a Help macro. Text formatted as macro hot spots appear as green, underlined text in the Help windowexactly the same as jump hot spots.

You can use macro hot spots to customize Windows Help or individual Help topics in many ways. For example, you could have macro hot spots that:

- Start another application
- Create a new button or menu item
- Change what happens when a user chooses an existing button or menu item
- Display a topic in a secondary window
- Change the size and location of a Help window
- Print the current topic
- Display a custom Help file in a second Help window
- And perform many other actions

Macros provide a convenient way to create many interesting effects and refine the presentation of your Help file. How you use them is really up to you.

> Note: The procedures for creating macro hot spots are exactly the same as those for creating jump hot spots with one exception: the hidden text following the hot-spot text contains the macro string(s) to be executed instead of a topic context string. Therefore, if you are familiar with the material in the Creating Links Between Topics section, you may want to skim the information in this section.

How Macro Hot Spots Appear in Help

In the built Help file, macro hot spots appear exactly the same as standard jump hot spots. For our example, the macro hot spot appears at the underlined text Macro hot spot Text (see Figure 8.x). To choose the macro hot spot, users position the mouse pointer over the macro hot-spot text and click the left mouse button.

Windows Help executes the macro referenced by the macro hot spot, as in Figure 8.x.

Creating Standard Macro Hot Spots

The coding for a standard macro hot spot has two parts:

- A reference word or phrase formatted as double-underlined text (the macro hot spot text) that the user chooses to make the macro hot spot
- The macro string, formatted as hidden text, that identifies the macro that Help executes when the user chooses the hot spot

> Note: In Word for Windows, select the Hidden Text check box in the View Preferences (version 1.1) or Tools Options (version 2.0) dialog box to display hidden text. You can also assign a macro key to make selecting this option more convenient.

To format text as a macro hot spot

1. Select the macro hot-spot text.
2. From the Format menu, choose Character.
3. Select the Double Underline check box, and then choose OK.
4. Position the insertion point immediately after the last letter in the double-underlined macro hot spot text.
5. From the Format menu, choose Character again.
6. Clear the Double Underline check box, select the Hidden check box, and then choose OK.
7. Insert an exclamation point (!) as the first character of the macro string.
   Note The exclamation point must be formatted as hidden text.
8. Type the macro string that you want Help to execute when the user chooses this hot spot text.
9. From the Format menu, choose Character again, clear the Hidden check box, and then choose OK.

Figure 8.x shows a correctly coded macro hot spot in a topic file.

Macro Hot Spot Guidelines

When creating macro hot spots, follow these guidelines:

- The macro hot-spot text can have as many as 512 characters, including any embedded spaces.
- Be sure no spaces exist between the double-underlined macro hot spot text and the hidden text macro string. For example, the following is correct:

[missing graphic]

The following is incorrect:

[missing graphic]

- You can specify more than one macro to be executed by separating them with a semicolon or colon, as in the following example:

```
!ChangeButttonBinding("btn_contents", "JumpID(`hgcd.hlp',
`acc_idx_hg')");EnableButton("btn_up");ChangeButttonBinding("btn_up",
"JumpID(`cdcd.hlp', `hlpidx_idx_card')")
```

- If you place the macro string at the end of a paragraph or at the end of a line with a soft carriage return (SHIFT+ENTER), format the paragraph or carriage-return mark as normal text (dont format it as hidden text).
- You can execute any macro listed in Chapter 15, Help Macro Reference, from a macro hot spot; however, if the topic is to be displayed in a secondary window, the macro may be ignored or cause undesirable behavior.

Creating Links to Online Tutorials

Help version 3.1 includes limited support for creating links between an applications Help file and an online tutorial. This means that you can create hot spots within the Help file that link to specific tutorial lessons. When the user chooses a tutorial hot spot, Help starts the tutorial application and informs it that the user has requested a tutorial lesson. Help then turns control over to the tutorial application so that it can run the lesson.

Creating tutorial links within a Help file requires cooperation from all sources:

- The Help author is responsible for creating the hot spots that the user chooses to see a tutorial lesson.
- Windows Help is responsible for sending a message to the tutorial application and informing it that the user has requested a particular lesson. To send this message, Help may also have to start the tutorial application.
- The tutorial application is responsible for starting and running the tutorial lesson and for returning control to Help when the user has finished using the tutorial.

Creating the Hot Spots

If your application includes an online tutorial, you can use the ExecProgram macro to create hot spots within the Help file that link to tutorial lessons.

> Note: For detailed information about the ExecProgram macro, see Chapter 15, Help Macro Reference.

To create a macro hot spot that links to a tutorial lesson

1. Follow the steps to create a standard macro hot spot.
2. Type the tutorial information for this hot spot in the ExecProgram macro string.

   Note The exclamation point (!) and macro string must be formatted as hidden text.

   For example, the following macro would start the tutorial application TUTOR.EXE, which is located in the LESSONS subdirectory, and display a lesson on the applications Toolbox in a maximized window:

   ```
   !ExecProgram("c:\\lessons\\tutor.exe toolbox.cbt", 1)
   ```

   Note If your tutorial application supports command-line options, you can also include them when starting the lesson. For example, a command-line option may allow you to start the lesson in a specific location. Refer to the documentation that accompanies your tutorial application for information about the command-line options that it supports.

   Figure 8.x shows a correctly formatted macro hot spot to a tutorial lesson.

Tutorials on Windows Help

Because Windows Help is included in so many different software packages, users often have their Help system replaced by the install program they are using to install a new software program. This makes it virtually impossible for any single application to know which version of Windows Help users have installed on their system. For that reason, Help authors at Microsoft have decided not to create online tutorial lessons about Windows Help. Therefore, Windows Help 3.1 has removed the CBT aware functionality that was required to support tutorials about Help.

Changing the Standard Appearance of Hot Spots

A standard hot spot in Windows Help means that the hot spot appears as green, solid- or dotted-underlined text to the user. You can change the default appearance of hot spots by:

- Removing both the green color and underline (invisible hot spots).
- Removing just the green color (underlined hot spots).
- Changing the green color and removing the underline (custom hot spots).

No matter how you change the appearance of hot spots, the Windows Help cursor changes to a hand pointer when it is over the hot-spot text. You may want to change the default appearance of hot spots to create a custom look for your Help file.

> Note: Changing the standard appearance of hot spots may or may not be a good idea. It depends on why you are creating the Help file. If you are creating Help for a standard Windows application, you may not want to have your hot spots look different from the standard that Windows users recognize. On the other hand, if you are creating a Help file for your personal use, changing the hot spots may be essential to getting the look you want. For more discussion about these issues, see Chapter 4, Help Authoring Guidelines..

> Note: Changing the appearance of jump hot spots, pop-up hot spots and macro hot spots involves essentially the same procedures.

Creating Invisible Hot Spots

You can make some or all of your hot spots invisible. Invisible hot spots appear as normal text to the user, but the Windows Help cursor changes to a hand pointer when it is over an invisible hot spot. Invisible hot spots are a useful way to create hot spots that look like regular text. For example, you may want to use invisible jump hot spots in a Contents topic to have better control over the appearance of the list. Or, you may want to use invisible pop-up hot spots in a Master Glossary topic where all the hot spots display a pop-up window with a definition.

> Note: Because invisible hot spots do not have any visible formatting to inform users, it is a good idea to tell them that there are hot spots on the screen.

To create an invisible hot spot

1. Follow the steps to create a standard jump hot spot, pop-up hot spot, or macro hot spot.
2. Insert a percent sign (%) as the first character of the context string (just before the exclamation point if it is a macro hot spot).

   Note The percent sign must be formatted as hidden text.

   Figure 8.x shows a correctly formatted invisible hot spot in a topic file.

Creating Underlined Hot Spots Without Color

You can create underlined hot spots in your Help topics. Underlined hot-spots appear to the user as normal, solid- or dotted-underlined text without the standard green hot spot color. However, the Windows Help cursor changes to a hand pointer when it is over the hot spot. Underlined hot spots are a useful way to remove the default green color but retain the underlining as the hot-spot indicator.

To create an underlined hot spot

1. Follow the steps to create a standard jump hot spot, pop-up hot spot, or macro hot spot.
2. Insert an asterisk (*) as the first character of the context string (just before the exclamation point if it is a macro hot spot).

   Note The asterisk must be formatted as hidden text.

   Figure 8.x shows a correctly formatted underlined hot spot in a topic file.

Creating Hot Spots with Custom Colors

You can create custom-color hot spots in your Help topics by changing the default color of the hot spots. Custom-color hot spots are a useful way to create a unique look for your Help file. However, you should be aware of the issues surrounding the decision to change Helps standard hot spot color.

For guidelines on using color in Help files, see Chapter 3, Designing Your Help System.

Issues to Consider Before Changing Hot Spot Colors

When deciding whether to use colored text in your Help file, consider the following issues:

- Author-defined colors may conflict with the users default screen colors.
- Most Microsoft applications do not override the users default colors, and users expect Help to be consistent with that behavior.
- In some circumstances (displaying on gray-scale monitors, for example), colored text may not be visible on the users machine.
- Help cannot guarantee that colors the user hasnt chosen will have enough contrast to be visible.
- You cannot rely on color to make important distinctions of meaning.
- If the product is localized for an international market, the colors will have no intrinsic meanings in the various countries and cultures. Moreover, on monochrome monitors the color disappears, so the Help file loses any meaning associated with color.

You can solve all these problems only if users are in complete control of their screen colors.

The Windows Color Pallette

The Windows pallette provides 16 colors, many of which are not good choices as text colors. Windows will only display text in solid colors, so you cant use any dithered colors. If we assume that the background color of the Help window is white (not necessarily a good assumption), this eliminates using any bright colors such as yellow or cyan. The dark colors, like dark green and dark red, are all readable, but some, like dark blue, have less contrast with black. Dark green is readable and not visually distracting, but brighter than dark blue. If you want to give the text more emphasis, you might choose dark green.

On the other hand, if you want to give the colored text less emphasis but have some differentiation from black, you might choose dark blue. The effect of the blue text would be more subtle than the dark green. However, blue text is less recognizable than black text because we have fewer receptors in our eyes that can detect blue. Therefore, blue fades to the background and does not stand out. Blue is high-contrast, but isnt easy to distinguish from black for everyone. So generally, blue text should be used sparingly as a secondary color.

Conflicts with the User's Screen Colors

The Windows default color scheme includes black text on a white background. However, the Windows Control Panel lets users change their default screen colors, including the text and background colors. Because the Help application does not interfere with a users selection, Help authors cannot assume that Help will always display black text on a white background.

Depending on a users choice, colored text may be difficult or impossible to see. For example, if a user chooses yellow text on a black background, and the Help author uses dark blue text in the Help topics, the colored text may be very difficult to read. Individual computers may also have hardware or other display-related problems, such as an inability to resolve contrast between some colorslight grey and dark grey, for example.

Problems with Color on Different Displays

Help cannot predict how the text colors are actually displayed on different monitors. For example, on

some gas-plasma displays that use VGA color drivers, Help displays colors as various shades of red (gray-scale red).

Help can predict the readability of a given text color on a given background color only under two circumstances:

- The text color is the users default text color.
  In this circumstance, Help relies on the user to make the right choice. If the text isnt readable, the user must change the color in Control Panel.
- The author defines a text color that is exactly the same as the background color.
  In this circumstance, Help uses the default text color and displays it as normal text.

Problems with Assigning Meaning to Color

In addition to the technical aspects of color, color also affects a users emotions. In fact a big part of human color perception is psychological and emotional. For example, dark green and dark blue are fairly bland colors but appeal to a wide audience. Dark red is brighter and hence more attractive to the eye, but it is often associated with anger or strong emotion.

These emotional responses are also related to cultural associationsred means stop, warning, passion; green equals go, ripe, money. In American culture, red text might be good for warnings, but it can be a very bad color for Help text that contains neutral information. Moreover, you cannot use color to create absolute meaning because color associations vary within a culture, and from culture to culture. If youre writing Help for an international product, the English version will be used by many non-Americans who will interpret your color associations differently.

Overriding Custom Hot Spot Colors

If your Help file contains a custom color for hot spots, the user cannot use the standard WIN.INI entriesJumpColor, PopUpColor, MacroColor, IFJumpColor, IFPopUpColorto override your custom colors. To override an author-defined hot spot color users must type the following entry in the WIN.INI file:

```
[Windows Help]
Colors=NONE
```

When Help finds this entry in the WIN.INI file, it uses the system default colors for foreground and background colors, regardless of how the Help file was authored.

If you change the standard color of hot spots in your Help file, it is your responsibility to instruct users how to override your custom colors.

Creating Custom Hot Spots

There are four ways to create a custom-color hot spot. You can:

- Create an underlined hot spot and change the color of the hot spot text to one of the Windows standard 16 colors (Windows color, standard solid- or dotted-underline).
- Create an underlined hot spot and change the color of the hot spot text to a custom RGB color (custom RGB color, standard solid- or dotted-underline).
- Create an invisible hot spot and change the color of the hot spot text to one of the Windows standard 16 colors (Windows color, no solid- or dotted-underline).
- Create an invisible hot spot and change the color of the hot spot text to a custom RGB color (custom RGB color, standard solid- or dotted-underline).

To create a hot spot with a Windows color and standard underline

1. Follow the steps to create a standard jump hot spot, interfile jump hot spot, pop-up hot spot, or macro hot spot.
2. Insert an asterisk (*) as the first character of the context string (just before the exclamation point if it is a macro hot spot).

Note The asterisk must be formatted as hidden text.

3. Select the hot spot text.
4. From the Format menu, choose Character.
5. In the Color box, choose the color that you want for the hot spot text.
6. Choose OK.

   Figure 8.x shows a correctly formatted custom underlined hot spot in a topic file.

To create a hot spot with a custom RGB color and standard underline

1. Follow the steps to create a standard jump hot spot, interfile jump hot spot, pop-up hot spot, or macro hot spot.
2. Insert an asterisk (*) as the first character of the context string (just before the exclamation point if it is a macro hot spot).

   Note The asterisk must be formatted as hidden text.
3. Select the hot spot text.
4. From the Format menu, choose Character.
5. In the Color box, choose the color that approximates the custom color you want for the hot spot text.

   For example, if you want to create a custom blue, choose one of the blues from the Color box.
6. Choose OK.
7. Save the file as RTF.
8. Open the RTF file as Text; do not convert it to .DOC format.
9. In the \colortbl section of the RTF header, change the RGB value of the hot spot color.

   For example, if your hot spots are blue (\red0\green0\blue255), change the RGB value of blue in the \ colortbl to your custom color, \red0\green0\blue64, for example.

   Note Help uses \red0\green128\blue0 for the standard hot spot color.

   Figure 8.x shows a correctly formatted custom hot spot in a topic file.
10. Save the RTF file as text.

    When you build the Help file, the hot spots will appear in the custom color.

To create a hot spot with a Windows color and no underline

1. Follow the steps to create a standard jump hot spot, interfile jump hot spot, pop-up hot spot, or macro hot spot.
2. Insert a percent sign (%) as the first character of the context string (just before the exclamation point if it is a macro hot spot).

   Note The percent sign must be formatted as hidden text.
3. Select the hot spot text.
4. From the Format menu, choose Character.
5. In the Color box, choose the color that you want for the hot spot text.
6. Choose OK.

   Figure 8.x shows a correctly formatted custom hot spot in a topic file.

To create a hot spot with a custom RGB color and no underline

1. Follow the steps to create a standard jump hot spot, interfile jump hot spot, pop-up hot spot, or macro hot spot.
2. Insert a percent sign (%) as the first character of the context string (just before the exclamation point if it is a macro hot spot).

   Note The percent sign must be formatted as hidden text.
3. Select the hot spot text.
4. From the Format menu, choose Character.
5. In the Color box, choose the color that approximates the custom color you want for the hot spot text.

   For example, if you want to create a custom red, choose one of the reds from the Color box
6. Choose OK.
7. Save the file as RTF.

8. Open the RTF file as Text; do not convert it to .DOC format.
9. In the \colortbl section of the RTF header, change the RGB value of the hot spot color.
   For example, if your hot spots are red (\red255\green0\blue0), change the RGB value of red in the \ colortbl to your custom color, \red64\green0\blue0, for example.
   Note Help uses \red0\green128\blue0 for the standard hot spot color.
   Figure 8.x shows a correctly formatted custom hot spot in a topic file.
10. Save the RTF file as text.
    When you build the Help file, the hot spots will appear in the custom color.

Chapter 9 Defining Topic Windows

After the Help file is built, topics appear in the Help window. In addition to the main Help window, you can also display Help topics in other kinds of windows: secondary windows, pop-up windows, and embedded windows. By using different windows for the Help information, you can create topics that communicate information effectively.

This chapter discusses how to define Help windows for your topic.

About Help Windows

There are four kinds of Help windows.

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

Help files can display topics in the main window, one secondary window, and one pop-up window at the same time. Although you can display only one secondary window and one pop-up window at a time, you can use them throughout your Help file. Figure 9.1 shows a Help file that uses a main window, a secondary window, and a pop-up window.

Inside topics, Help also displays embedded windows, which are rectangular regions that display graphics, multimedia data, and other specialized information. Embedded windows are formatted with the topic text, and they move when the topic is scrolled or when the window is sized.

Figure 9.2 shows a topic in the main Help window that includes an animation displayed in an embedded window.

To display topics in windows, you use jumps. When you create a jump hot spot, you choose a topic to display and a window in which to display the topic. Jumps are discussed in Chapter 8, Creating Links and Hot Spots.

The Main Help Window

Although it is possible to create a Help file that does not use the main Help window, most Help files do have a main window. Help defines the basic characteristics of the default main Help window, but you can change these attributes if you want. You can define a total of eight attributes for the main window. Main window attributes include:

- The title that appears in the main windows title bar.
- The position of the windows upper left corner.
- The windows size.
- The windows state (maximized or normal).
- The background color of the scrolling and nonscrolling regions.

The main window attributes are determined by the entry in the [WINDOWS] section of the Help project file and are used when the Help file is displayed. After the file is built, the main window attributes cannot be changed while the Help file is open. The window attributes do not apply to any other Help file that might be opened during a users session.

A main window definition has the following form:

```
main="caption", (x-coord, y-coord, width, height), window-state,
(scrolling-RGB), (nonscrolling-RGB)
```

The following example shows a main window definition:

```
[WINDOWS]
main="Sample Help", (50, 50, 800, 900), 0, , (192,192,192)
```

This definition defines a main window that displays Sample Help in the title bar. The window is positioned in the upper-left corner of the screen and is 800-by-900 (in Help coordinates). The window appears in its normal state with a light grey background in the nonscrolling region.

The following sections describe each attribute in a main window definition.

> Note: The information here is a summary of the window attributes. For a complete description of each attribute, refer to the [WINDOWS] section in Chapter 16, The Help Project File.

Window Name

The window-name must be main for the main Help window and cannot be changed.

Caption

The caption is the text that appears in the main windows title bar and in the icon label when the window is minimized. In the sample definition above, the window caption is Sample Help.

X-Coord and Y-Coord

The x-coord and y-coord attributes are numbers that define where the window appears on the users computer screen. The numbers represent the horizontal and vertical location of the upper-left corner of the window as defined in terms of Helps 1024-by-1024 coordinate system. For example, the coordinates for the sample main window are (50, 50), or the upper-left corner of the screen.

When Help first displays the window, it uses the position values youve specified. Users can move the window after it has been displayed.

Width and Height

The width and height attributes are numbers that define the normal width and height of the window, again in terms of the Windows Help coordinate system. In the example definition, the window is 800-by-900.

When Help first displays the window, it uses the size youve specified. Users can change the window size after it has been displayed.

Window State

The window-state is a number (0 or 1) that indicates whether the window is displayed at normal size or maximized when Help first opens it. In the example, the main window is displayed at normal size.

The following table shows the effect of the window-state attribute.

State	Initial window position
Normal	Window appears in the location and size defined by your position and size values.
Maximized	Window fills the entire screen. If the user restores the window to its normal state, it occupies the part of the screen defined by your position and size values.

Scrolling and Nonscrolling Region Background Colors

The scrolling-RGB and nonscrolling-RGB attributes are number sequences that define the background color of the windows scrolling and nonscrolling regions. The numbers represent the red, green, and blue (RGB) components of the color. If these numbers are not given, the scrolling and nonscrolling areas use the users default colors as defined by the Control Panel Colors setting. Help supports any solid or dithered colors available from the Windows system colors. In the example, the main window uses light gray [(192,192,192)] as the background color for the nonscrolling region.

Use the Color setting in Control Panel to see the colors resulting from different RGB values. To change the color of text in the window, use your word processor. Format the text in the color that you want it to appear. Just remember not to create conflicts between the text color and any custom background color you might have specified.

The Windows Help Coordinate System

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

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

x * (horiz/1024)

The y-coordinate of the upper-left corner is at the pixel given by the following formula:

y * (vert/1024)

The windows width and height are also specified in Helps coordinate system. For example, a window may specify a width of 511 and a height of 511. A standard VGA card has a resolution of 640-by-480 pixels, so the width is actually the following number of pixels:

511 * (640/1024) = 319

The height is actually the following number:

```
511 * (480/1024) = 240
```

In other words, the window would take up approximately one quarter of the area of the Windows Help screen (half the width multiplied by half the height).

To convert from pixels to Windows Help coordinates, you invert the ratio between Windows Helps resolution and the video resolution. Again, assuming the resolution of your video card is horiz by vert pixels, and the horizontal and vertical locations (or dimensions) you want are x and y pixels, the x-coordinate (or width), in Windows Help coordinates, is as follows:

x * (1024/horiz)

The y-coordinate (or height), in Windows Help coordinates, is:

y * (1024/vert)

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

Secondary Windows

You can create secondary windows for your Help file. A secondary window is an independent Help window that complements and supports the main Help window and displays similar information. Secondary windows have many uses but are very effective when you want to display new information without having users lose information that is already displayed in the main Help window. In other words, secondary windows let you have two Help windows open at the same time. Here are just a few possibilities.

You can use secondary windows to:

- Display the Help Contents or alphabetic index.

  Placing the contents or index topics in a secondary window that has jumps to the main Help window lets users browse through the information without losing the list of choices.
- Create a separate content window for a specifc type of Help information.

  For example, you can create a secondary window that displays step-by-step procedures in it. That way users can close the main window and just use the secondary window with their application.
- Create a window that displays pictures, examples, and other information related to the main topic.

  Because secondary windows function independently of the main Help window, you can use them to show users extra material at the same time that they are looking at the main topic. For example, in Windows version 3.1, the Help files include a master glossary that is displayed in a secondary window.
- Gain complete control over the Help interface.

  You cannot remove the standard menus and buttons from the main Help window. However, if you want to create a Help system with custom controls (menus only or buttons only, for example), you can display all the Help topics in a secondary window. To provide functionality to users, you can add a nonscrolling region and create custom controls using Help macros that simulate menus and buttons.

Whether you use secondary windows and how you use them depends entirely on how you have designed the information model for your Help system.

How Secondary Windows Work

A secondary window has the same characteristics as a standard window. It includes a Control menu, a title bar, minimize and maximize buttons, and window borders. Users can move, resize, minimize, or maximize secondary windows. If necessary, secondary windows can display horizontal and vertical scroll bars that users can use to view information displayed in the secondary window. Secondary windows do not include a menu bar or button bar or any other Help controls. Any custom controls or functionality in secondary windows must be defined by individual Help authors for each Help file.

A secondary window is completely independent of the main Help window It can be open even if the main Help window is closed. When open, a secondary windows title appears in the Windows Task List. If a user minimizes a secondary window, it displays its own icon. Help uses the standard Windows Help question-mark icon unless the Help author specifies a custom icon using the ICON option in the [OPTIONS] section of the Help project file. (See Chapter 16, The Help Project File, for details.)

Any topic that you can display in the main Help window you can also display in a secondary window. Secondary windows can include any standard Windows Help feature, including text, graphics, jump hot spots, pop-up windows, macro hot spots, nonscrolling areas, or embedded windows.

Secondary windows appear on demand, usually when the user chooses a menu item, Help button, or hot spot. A macro in the Help project file or a WinHelp API call can also be used to display a secondary window. For information about using Help macros to display a secondary window, see Chapter 14, Help Macros. For information about how an application can send an API call to Help in order to display a secondary window, see Chapter 19, The WinHelp API.

Secondary window attributes are specified when a Help file is opened and remain in effect until the Help file is closed.

Limitations

Despite their usefulness, secondary windows have the following limitations:

- Windows Help can display only one secondary window at a time. If the user completes an action that requires Help to display a secondary window while one is already open, the new secondary window replaces the current one.
- A single Help file can define only up to five secondary window types, which limits the number of different ways you can use secondary windows. For example, you cannot create a design that customizes the secondary window for every topic displayed in it; instead, you should design secondary windows for categories of information.
- You cannot use all of the Help macros in secondary windows. Some macros are ignored when executed from a secondary window, and others are not recommended because they may create conflicts if executed from a secondary window. To find out whether a macro can be executed from a secondary window, check the macros Comments section in Chapter 15, Help Macro Reference.
- Secondary windows remain open if the user closes the main Windows Help window using the Control menu. The user must then close the secondary window using the secondary windows Control menu or a custom control that you provide.
- Secondary window topics dont appear in the history list nor are they recorded in the back list. So, if you place important information in a secondary window, the user will not be able to use the History or Back button to return to a previously viewed topic.
- The keyword search feature in Windows Help displays all topics in the main Help window, even if that topic is normally displayed in a secondary window. You can assign keywords to secondary window topics, but when accessed from the Search dialog box, they will not appear in the secondary window.
- You cannot use Help macros to change the on-top state of a secondary window without affecting the main Help window.

Creating a Secondary Window

Help creates a secondary window only if one is defined in the [WINDOWS] section of the Help project file for that Help file. Adding a secondary window to your Help file requires the following steps:

- Creating the topic that will appear in the secondary window
- Defining the attributes of the secondary window in the [WINDOWS] section of the Help project file
- Creating any jumps in the main topic file to the topic displayed in the secondary window

Defining Secondary Window Attributes

You can define a total of ten attributes for each secondary window that you create. The attributes are the same as those for the main window, except that a secondary window includes a unique name and may be authored to stay on top of other windows.

Secondary window attributes include:

- The name of the secondary window.
- The title that appears in the secondary windows title bar.
- The position of the windows upper left corner.
- The windows size.
- The windows state (maximized or normal).
- The background color of the scrolling and nonscrolling regions.
- The windows on-top state (on top or normal).

You can define as many as five secondary windows for a single Help file. The window attributes are determined by the entry in the [WINDOWS] section of the Help project file and are set when the Help file is opened. Once, the window attributes cannot be changed while the Help file is open. The window attributes do not apply to any other Help file that might be opened during a users session.

A secondary window definition has the following form:

```
window-name="caption", (x-coord, y-coord, width, height), window-state,
(scrolling-RGB), (nonscrolling-RGB), ontop-state
```

The following example shows a sample secondary window definition:

```
[WINDOWS]
graph="Charts", (0,0,600,723), 0, , (192,192,192), 0
```

This definition creates a secondary window named graph that displays Charts in the title bar. The window is positioned in the extreme upper-left corner of the screen and is 600-by-723 (in Help coordinates). The window appears in its normal state with a light grey background in the nonscrolling region. The window does not stay on top of other application windows.

> Note: The following sections describe each attribute in a secondary window definition. If you have already read about Help window attributes in The Main Window section, you may want to skim this section and just read about the window-name and ontop-state attributes.

[?] \sgmlnitp

The information here is a summary of the secondary window attributes. For a complete description of each attribute, refer to the [WINDOWS]section in Chapter 16, The Help Project File.Window Name

The window-name identifies a secondary window and gives it a name that you can then use when creating jumps and Help macros. In the sample definition above, the window name is graph.

Whenever you include this name in a jump hot spot, Help displays the topic in a secondary window. (For an example of a jump to a secondary window, see Creating Jumps to Secondary Windows, later in this chapter.)

Caption

The caption is the text that appears in the secondary windows title bar and in the icon label when the window is minimized. In the sample definition above, the window caption is Charts.

X-Coord and Y-Coord

The x-coord and y-coord attributes are numbers that define where the secondary window appears on the users computer screen. The numbers represent the horizontal and vertical location of the upper-left

corner of the window as defined in terms of Helps 1024-by-1024 coordinate system. For example, the upper-left corner coordinates for the sample secondary window graph are (0,0), the extreme upper-left corner of the screen.

When Help first displays the window, it uses the position values youve specified. Users can move the window after it has been displayed.

Width and Height

The width and height attributes are numbers that define the normal width and height of the window, again in terms of the Windows Help coordinate system. In the example definition, the window is 600-by-723.

When Help first displays the window, it uses the size values youve specified. Users can change the windows size after it has been displayed.

Window State

The window-state is a number (0 or 1) that indicates whether the window is displayed at normal size or maximized when Help first opens it. In the example, the graph window is displayed at normal size.

The following table shows the effect of the window-state attribute.

State	Initial window position
Normal	Window appears in the location and size defined by your position and size values.
Maximized	Window fills the entire screen. If the user restores the window to its normal state, it occupies the part of the screen defined by your position and size values.

Scrolling and Nonscrolling Region Background Colors

The scrolling-RGB and nonscrolling-RGB are number sequences that define the background color of the windows scrolling and nonscrolling regions. The numbers represent the red, green, and blue (RGB) components of the color. If these numbers are not given, the scrolling and nonscrolling areas use the users default colors as defined by the Control Panel Colors setting. Help supports any solid or dithered colors available from the Windows system colors. In the example, the secondary window uses light gray [(192,192,192)] as the background color for the nonscrolling region.

Use the Color setting in Control Panel to see the colors resulting from different RGB values. To change the color of text in the window, use your word processor. Format the text in the color that you want it to appear. Just remember not to create conflicts between the text color and any custom background color you might have specified.

On Top State

The ontop state specifies whether the secondary window stays on top of other windows. If the author defines the window as on top, the user cannot change the window behavior using the Always On Top command in Help. The on-top value is either 0 for normal behavior or 1 for on-top behavior. The main Help window cannot be authored as a topmost window. The example window graph is not defined as an on-top window.

Creating Jumps to Secondary Windows

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

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

Creating Standard Jumps to Secondary Windows

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

```
context-string>window-name
```

Context-string is the context string for the topic you are jumping to, and window-name is the name of the secondary window where you want the topic to appear.

To create a jump to a secondary window

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

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

Figure 9.x shows a correctly formatted hot spot for a secondary window jump.

> Note: As with other hot spots, you can change the basic appearance of a hot spot that references a secondary window. Because the secondary window information is completely contained within the hidden text, the procedures are exactly the same as those for regular jump hot spots. Refer to the Changing the Standard Appearance of Jump Hot Spotssection in Chapter 8, Creating Links and Hot Spots, for details.

Creating Jumps to the Main Help Window

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

```
context-string>main
```

The window name main is reserved for the main Help window and must be used when specifying jumps to the main window.

To create a jump from a secondary window to the main Help window

1. Follow the steps to create a standard jump hot spot.
2. Insert a right angle bracket (>) immediately after the last character of the context string.
3. Type main as the window-name.

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

Figure 9.x shows a correctly formatted jump hot spot for the main window.

To help you coordinate the links in your Help file so that the correct topics are displayed in the appropriate window, the following table presents a complete summary of the possible links you can create between the main Help window and a secondary window.

Jump location	Type of jump	Result of the jump
Main window	standard	The topic is displayed in the main Help

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

Creating Interfile Jumps to Secondary Windows

If the destination topic is in another Help file, you can also include the name of the Help file before the window-name, as shown here:

context-string[@filename]>window-name

The filename is the name of the Help file that contains the topic you want to display in the secondary window.

To create an interfile jump to a secondary window

1. Follow the steps to create a standard jump hot spot.
2. Insert an at sign (@) immediately after the last character of the context string.
3. Type the name of the file that contains the context string assigned to the topic that is the target of the interfile jump.

   Note The at sign and Help filename must be formatted as hidden text.
4. Insert a right angle bracket (>) immediately after the last character of the Help filename.
5. Type the name of the secondary window where you want the topic to appear.

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

   Figure 9.x shows a correctly formatted interfile jump hot spot in a topic file.

Closing Secondary Windows

Because secondary windows are independent windows, they do not close by themselves. The user has the greatest responsibility for closing a secondary window, but applications can also close a secondary window. When dealing with Help requests from different applications, Help automatically closes secondary windows to avoid situations in which the content of the main Help window and the secondary window are totally unrelated.

The following list describes all the possible ways to close a secondary window:

- The user can choose the Close command from the Control menu (or double-click the Control-menu box).
- The user can choose a custom control provided by the Help author that executes the CloseWindow macro.
- The user can open a new Help file.
- The user can exit Help.
- The application can send a HELP_QUIT message to Help and close Windows Help.
- The application can send a CloseWindow macro to Help and close a specific secondary window.
- Another application, different from the one currently displaying the secondary window, can request Help. This request will cause Help to close any open secondary windows and display the requested Help file in the main window.

> Note: If a different application from the one displaying Help topics requests Help and requests Help to display a topic in a secondary window, Help will close the main window before opening the new Help file and displaying the topic in the secondary window.

Defining Nonscrolling Regions

The main Help window and secondary windows display a scroll bar when the topic is too long or too wide to fit in the windows current size. By using the scroll bar, users can scroll the topic to see the out-of-view information. In addition to defining the Help window itself, you can also set aside part of the window as a nonscrolling region, which displays text or graphics in a fixed location immediately below the button bar. Text or graphics in the nonscrolling region remain in place even when users scroll the topic. Because they dont scroll with the rest of the topic, nonscrolling regions can have a number of different uses. Here are just a few possibilities.

You can use the nonscrolling region to:

- Display topic titles.

  That way users dont have to remember what topic they are reading when they scroll.
- Keep one kind of information anchored at the beginning of the topic.

  For example, you can display syntax statements in the nonscrolling region so that users can refer to them as they read the parameter descriptions.
- Display custom controls that change with the content of each topic.

  Because the nonscrolling region is part of each topic, the controls you place in it can be limited to helping the user with the information in the current topic.
- Display information that you never want to scroll.

  The nonscrolling region can contain entire topics, especially if you want to use all the available window space without having scroll bars cover up part of the information. For example, you can create a topic that just contains a graphic, and you can display that topic in the nonscrolling region of a secondary window that is the same size as the graphic.
- Create two different views of the same information.

  Help can display either the nonscrolling or scrolling region of a topic in a pop-up window. Help displays the region that contains the context-string footnote (# footnote) in the pop-up window. If you include the context-string footnote in the scrolling region, you can have some hot spots in the Help file that display the nonscrolling region in a pop-up window and some hot spots that display the entire topic in the main window.

How the Nonscrolling Region Works

To differentiate the nonscrolling region from the scrolling region of a topic, Help places a thin line between the two regions. If the scrolling region contains a vertical scroll bar, the scroll bar does not extend beyond the line into the nonscrolling region. The width of the nonscrolling region is the same as the window in which it appears. Therefore, changing the width of the Help window also changes the width of the nonscrolling region.

Any text that appears in the nonscrolling region wraps to fit in the visible portion of the window unless the text has been formatted as nonwrapping. If the nonwrapping text does not fit in the current Help window size, users must make the Help window larger to see it,because the horizontal scroll bar does not scroll text in the nonscrolling region.

The height of the nonscrolling region is determined by the amount of information you include in the nonscrolling region. A nonscrolling region can include any Windows Help feature that appears in the scrolling region, including text, bitmaps, and hot spots. Help includes hot spots from both regions in the tabbing order, the hot spots in the nonscrolling region coming first in the order.

Users can copy or print the information in a nonscrolling region. The information in the nonscrolling region appears first when copied or printed. The line that separates the nonscrolling and scrolling regions is not copied to the Clipboard or printed with the topic.

The background color of the nonscrolling region is determined either by a predefined setting that the author specifies in the [WINDOWS] section of the Help project file or by the users default system colors. If the nonscrolling region includes pop-up window hot spots, the background color in the pop-up windows will match the scrolling region rather than the colors in the nonscrolling region.

Creating a Nonscrolling Region

Help displays topic information in a nonscrolling region only if a Help author creates one by applying the Keep with Next formatting attribute to at least the first paragraph in a topic. If the first paragraph is not formatted as Keep With Next, Help does not create a nonscrolling region. Also, Help ignores any random paragraphs within a topic that might be formatted as Keep With Next.

To create a nonscrolling region

1. Be sure the text and graphics you want to place in the nonscrolling region are the first paragraphs of the topic.
2. Select the paragraph (or paragraphs) containing the information.
3. From the Format menu, choose Paragraph.
4. Select the Keep With Next check box, and then choose OK.
   The formatted paragraph(s) will be placed in a nonscrolling region when you build the Help file.

Setting the Background Color

To make the nonscrolling region visually distinct from the scrolling region, you can change the background color. You specify a background color for a nonscrolling region in the [WINDOWS] section of the Help project file. For example, the following entry in the Help project file tells Windows Help to use a light gray background for the nonscrolling region in the main Help window:

```
[WINDOWS]
main="Command Reference", (0,0,1023,1023), , , (192,192,192)
```

The numbers (192, 192, 192) are the only numbers in this entry you need to change to set the background color. They represent the RGB components of the background color.

> Hint: Use the Color application in the Windows Control Panel to see the colors resulting from different RGB values. Choose Color Palette, and then choose Define Custom Colors.

To change the foreground color (in other words, the text), simply format the text using the appropriate Color option in Word for Windows. (For more information, see the [WINDOWS] section in Chapter 16, The Help Project File.)

Pop-Up Windows

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

You can use pop-up windows to:

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

How Pop-Up Windows Work

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

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

Pop-up windows appear on demand, usually when the user chooses a menu item, button, or hot spot. A macro in the Help project file or a WinHelp API call can also be used to display a pop-up window. For information about using Help macros to display a pop-up window, see Chapter 15, Help Macro Reference. For information about how an application can send an API call to Help in order to display a pop-up window, see Chapter 19, The WinHelp API.

Pop-up windows remain open until the user clicks the mouse button or presses any key (which includes choosing a hot spot within the pop-up window).

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

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

Creating a Pop-Up Window

Creating pop-up windows involves two steps:

- Creating the topic information that will be displayed in the pop-up window
- Creating the hot spot or custom control in the main topic that will access the pop-up topic

Creating Pop-Up Topics

To create a pop-up window, you must first create the topic contents that will be displayed in the pop-up window. Pop-up topics can reside in the same file as regular topics, or they can be placed in a separate topic file. Like regular topics, pop-up topics are referenced by their unique context strings.

Creating Pop-Up Window Hot Spots

After creating pop-up topics, you create pop-up hot spots for displaying the pop-up window. The pop-up hot spot can be text, a bitmap, a hypergraphic hot spot, a menu item, a Help button, a custom control, or a macro hot spot. The coding for a pop-up window hot spot has two parts:

- A word or phrase, formatted as underlined text (the hot spot text), that the user chooses to display the pop-up window
- The context string, formatted as hidden text, that identifies the pop-up topic (what the user sees in the pop-up window)

To create a pop-up hot spot using text or graphics

1. Follow the steps to create a standard pop-up hot spot.
   For more information, see Chapter 8, Creating Links and Hot Spots.

To create a pop-up hot spot using a hypergraphic

1. Define a hot spot and then select Pop-up as the hot spot type in the Attributes dialog box of Hotspot Editor.
   For more information, see Chapter 11, Creating Hypergraphics.

To create a pop-up hot spot using a Help menu item or button

1. Use the ChangeButtonBinding macro to assign a pop-up macro to the standard menu item or button you want to use to display the pop-up window.
   Or create your own custom menu item or button and define a pop-up macro for it.
   For more information, see Chapter 13, Customizing the Help File, and Chapter 15, Help Macro Reference.

To create a pop-up hot spot using a Help macro

1. Follow the steps to create a standard macro hot spot and define a pop-up macro as the macro to execute when the user chooses the hot spot.
   For more information, see Chapter 13, Customizing the Help File, and Chapter 15, Help Macro Reference.

Embedded Windows

If you want to extend the functionality of Windows Help, you can create embedded windows for your Help file. An embedded window is a rectangular region within a topic that uses a dynamic-link library (DLL) to display information. Embedded windows can display many different kinds of information but are especially useful for displaying information that will maintain a consistent size within a topic. Here are a few possibilities.

You can use embedded windows to:

- Display 256-color bitmaps.
- Display multimedia 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 text file or database.
- Display screen images, such as interface elements and dialog boxes, stored in the applications resource files to save disk space in the compiled Help file.
- Display any other information that has special display requirements or that cannot be included using Helps standard feature set.

Whether you use embedded windows and how you use them depends entirely on your willingness and ability to create a dynamic-link library to control the contents within the embedded window. Help does not provide any custom DLLs for you.

How Embedded Windows Work in Help

An embedded window is simply a window (child window) within the topic window. Embedded windows do not have any standard window elements; they are just rectangular regions. Users cannot minimize, maximize, or resize an embedded window, and Help authors cannot use embedded windows as hot spots. However, you can include hot spots in an embedded window that are controlled by the DLL, but users will not be able to use the keyboard equivalents to access the hot spots. That is because embedded windows cannot receive the input focus which means they cannot process keystrokes. So, you should not place anything in an embedded window that requires keyboard input from users.

Windows Help positions the embedded window using the justification character (left, right, or character) specified by the author in the embedded window reference. The embedded window DLL is expected to display the information in the window and resize the window appropriately. Help expects the window size to remain fixed as long as the topic is displayed. The window element and DLL determine the size and content of the embedded window, and Help arranges the other elements of the topic around the embedded window.

Embedded windows are supported by DLLs that have specific components required by the embedded window interface. Developers familiar with DLL programming can write DLLs to support new user-interface or display elements not available in the standard Windows Help feature set. For example, developers can create a DLL to display 256-color bitmaps in an embedded window. For a discussion of the embedded window interface and for a description of a sample DLL, see Chapter 20, Writing DLLs for Windows Help..

Windows Help displays embedded windows only when necessarywhile the topic containing the embedded window is being displayed. However, an embedded window may exist while it is not being displayed (if the topic scrolls past the embedded window, for example). Because it is part of a specific topic, an embedded window goes away when the user displays a different topic.

Creating an Embedded Window

To create an embedded window, you insert an embedded window reference in the RTF topic file where you want the window to appear. This reference has the following general form:

```
{ewx DLL-name, window-class, author-data}
```

Parameter	Description
x	A character specifying the alignment of the window: l for a left-justified window, r for a right-justified window, or c for a character-aligned window.
DLL-name	The name of the DLL that controls the embedded window. The filename should not include an extension or be fully qualified, but it can include a relative path. Windows Help assumes .DLL or .EXE to be the default extension.
window-class	The name of the embedded window class as defined in the source code for the DLL.
author-data	An arbitrary string, which Windows Help passes to the embedded window when it creates the window. This string can be one or more substrings separated by any punctuation mark except a comma. The DLL is responsible for parsing this string. The string is terminated by the closing brace (})

Note: You may need to consult the developer of the DLL to obtain the information in the embedded window reference.

The following example shows a valid embedded window reference:

[missing graphic]

Positioning Embedded Windows

Embedded windows are displayed within a topic paragraph. Often, the topic paragraph contains text to be positioned in relation to the embedded window element. Help supports three different positions for embedded windowscharacter-aligned, left-justified, or right-justified.

The following table describes the position attributes that control the placement of an embedded window in relation to the paragraph text.

Attribute	Description
Character	The embedded window is displayed at the exact position in which it was inserted. Help adjusts the height of the line containing the embedded window to allow space for the window.
Left	The left edge of the embedded window aligns with the left paragraph margin, and the paragraph text wraps to the right of the embedded window.
Right	The right edge of the embedded window aligns with the right paragraph margin, and the paragraph text wraps to the left of the embedded window.

Each position defines how the embedded window is positioned with respect to the topic text, and each position requires a slightly different command reference in the topic file:

character	{ewc DLL-name, window-class, author-data}
left-aligned	{ewl DLL-name, window-class, author-data}
right-aligned	{ewr DLL-name, window-class, author-data}

Character-Aligned Embedded Windows (ewc)

The ewc (embedded window character) command causes Windows Help to treat the window as a text character. It aligns the window on the baseline of the type in exactly the same place in the paragraph where the command occurs. Because the window is treated as text, paragraph formatting properties assigned to the paragraph also apply to the window. Text coming before or after the window does not wrap around the window. Help adjusts the height of the line containing the embedded window to allow enough space for the embedded window.

Figure 9.x shows how Word for Windows displays the embedded window reference in the topic file.

After you compile your topic file, Windows Help displays the embedded window, as in Figure 9.x.

You can display multiple character-aligned embedded windows on the same line. Help adjusts the line height to allow for the tallest embedded window.

> Note: Dont specify absolute line spacing (also called negative line spacing or exact line spacing) for a paragraph that has an ewc reference. If you do, the embedded window might appear on top of the succeeding paragraph when Windows Help displays the topic.

Left- and Right-Justified Embedded Windows (ewl, ewr)

The ewl (embedded window left) command and ewr (embedded window right) command cause Windows Help to place the embedded window along the left or right paragraph margin and wrap the paragraph text around the edge of the embedded window. Text is aligned with the upper-right or upper-left corner of the window.

You normally place ewl and ewr references at the beginning of a paragraph to ensure proper wrapping of text around the right edge of the window. For example, typical ewl and ewr references are formatted as follows:

Paragraph text follows the embedded window reference...

Paragraph text follows the embedded window reference...

Figure 9.x shows how ewl and ewr references appear in the Word for Windows topic file.

> Note: Do not put any space characters between the ewl or ewr reference and the paragraph text unless you want the first line of text indented from the rest of the text that wraps along the right edge of the window (ewl) or from the left topic margin (ewr). To ensure that text wraps correctly around an embedded window, insert a soft carriage return at the end of each line of text.

After you compile the topic file, Windows Help displays the embedded window to the left or right of the text, as in Figure 9.x.

You can also put an ewl or ewr reference at the end of a paragraph, as in this example:

Paragraph text precedes the embedded window reference.

Paragraph text precedes the embedded window reference.

Figure 9.x shows how this type of embedded window reference appears in the Word for Windows topic file.

When you include an ewl or ewr reference this way, Windows Help wraps the text to the paragraph end and then displays the embedded window. After you compile the file, the embedded window appears under the text and to the left or right, as in Figure 9.x.

Left and Right Margins

The paragraph indent determines the distance between the window border and the left or right edge of the embedded window.

> Note: Within tables, the paragraph indent determines the distance between the cell border and the left or right edge of the embedded window.

Help reformats paragraphs when the user changes the horizontal width of the window. The resulting changes in word wrapping can change the position of embedded windows within the paragraph.

For example, in Figure 9.x, the right margin setting determines the distance of the right-aligned embedded window from the window border.

In Figure 9.x, the user has reduced the width of the window, and the embedded window has changed position:

> Note: The Word Keep Lines Together attribute, used to prevent Help from word-wrapping a paragraph, does not prevent Help from moving a left- or right-aligned embedded window in relation to the window border. The window will move when the window is sized, but the paragraph text wont move.

Vertical Positioning

The vertical position of a left- or right-aligned embedded window depends on the insertion point of the element within the topic paragraph. Help uses the following rules to determine the vertical position:

- If the element is the first item in the paragraph, the embedded window is displayed in the first line of the paragraph.
- If the element is not the first item in the paragraph, the embedded window is displayed in the line immediately below the insertion point.

Left- and right-aligned windows are displayed with their top edge aligned with the tallest character in the line. The top edge of the embedded window is aligned with the top edge of the line, and the rest of the embedded window extends down into the paragraph. Figure 9.x shows a left- or right-aligned embedded

window is positioned in a line.

> Note: Because character-aligned embedded windows are treated as large characters, they can increase the height of a line. This will affect the positioning of left- or right-aligned embedded windows displayed in the same line.

Figure 9.x shows three topic paragraphs displayed in Word and in Help. Each paragraph has identical content and Word formatting, but the insertion point of the embedded window changes from one to the next. In this example, the three embedded windows are left-aligned.

<three paragraphs in Word and Help, with elements in three different positions: first char, after first word, and last char>

In Figure 9.x, the three embedded windows are right-aligned.

<three paragraphs in Word and Help, with elements in three different positions: first char, after first word, and last char>

Help cannot display multiple left- or right-aligned embedded windows in the same vertical position. If you have multiple left- or right-aligned embedded windows inserted in the same position within the paragraph, Help displays the windows in different vertical positions within the paragraph. For example, Figure 9.x shows a topic paragraph displayed in Word and in Help.

<Word paragraph. Callouts identifying icons as left- and right-aligned bitmaps>

Using Tables to Position Embedded Windows

If you need to display multiple embedded windows in the same vertical position, you can use tables to position the embedded windows. Table formatting lets you more closely control the position of embedded windows within a paragraph. For example, you can position two or more left- or right-aligned windows within the same vertical plane. Or you can place an embedded window in the center of a paragraph. Paragraphs within tables are not reformatted when the window is sized.

Left- and right-aligned embedded windows are aligned with the cell borders. The word-wrapping of paragraphs in tables is not affected by changes in window size position, so if you want to prevent an embedded window from moving with the window border, you must place the containing paragraph in a table cell.

Figure 9.x shows a Help topic that uses tables to create some sophisticated paragraph layouts.

How Help Locates .DLLs

When executing custom DLLs, Windows Help loads the DLL only when it is needed by the Help file. To load a DLL, Help must be able to find it on the users system. When preparing to use a DLL, Help looks in the following locations:

1. Helps current directory.
2. The MS-DOS current directory.
3. The users Windows directory.
4. The Windows SYSTEM directory.
5. The directory containing WINHELP.EXE.
6. The directories listed in the users PATH environment variable.
7. The directories specified in WINHELP.INI.

If Help cannot find the DLL after searching in all these locations, it displays an error message.

To increase the likelihood that Help will locate the DLL quickly, you should also observe the following guidelines:

- Use unique names for all DLLs accessed by the Help file.
- When installing your application on a users hard disk drive, your setup program should copy all custom DLLs to the directory where Help is located.
- If your product is distributed on CD-ROM, copy WINHELP.EXE and any custom DLLs to the users hard disk drive.
- Define a WINHELP.INI entry for each custom DLL that your Help file is using so that Help knows where to locate them.

For an explanation of the WINHELP.INI file, see the Creating Links Between Help Files section in Chapter 8, Creating Links and Hot Spots.

Chapter 10 Adding Graphics

When building a Help file, you dont have to create text-only filesyou can add interest and clarity by using pictures. This chapter describes different ways to include pictures in your Help files and discusses issues you should consider when obtaining and preparing pictures for a Help file.

Creating Pictures

You can create pictures for your Help file using whatever tools you want, including many popular graphics programs for Microsoft Windows or the Apple Macintosh. The main requirement is that Windows Help must be able to display the image.

Sources

In order to create good quality pictures, you should consider different source materials. The most obvious source is the application, which contains the interface elements most Help files document. However, magazines, illustrated encyclopaedias and catalogs are also good sources of visual material. Another source might be physical objects, which can be photographed, scanned (if they are fairly flat), measured, or drawn.

Professional designers continually use visual references for their work. For example, if producing a design that features airplanes, the designer will collect as many photographs and detailed drawings as possible. By referring to a collection of pictures, the designer can produce an original picture which is convincing and accurate.

When determining the style of a picture it may be worth studying the work of professional illustrators in magazines, advertisements, books, and so on. Care should be taken not to confuse simple with badly drawn images. Simple designs are often very successful since they focus the users attention on the key aspects of the picture. For instance, a silhouette of an object is often quite sufficient for it to be recognized.

If you use a picture that has been published without permission and without changing it in any way, then you may be infringing on someones copyright. In this case you should either seek permission from the publishers or adapt the picture for your own use. The original picture, when appropriate, should be acknowledged in your documentation.

Picture Quality

When creating pictures, you should consider the overall quality that users expect. Since the pictures appear on an electronic screen, users may expect the pictures and screen design to fall somewhere between the standards set by television and computer arcade games. It is important to try to meet users visual expectancy since the impact of poorly designed images can cause users to lose interest.

An important difference between printed images and online images is their resolution. The resolution of paper-based images is generally much higher than images on the computer screen.

Dont use images with too much detail at a small scale since the detail will likely be lost on the screen. Conversely, an enlarged image which is only partly visible can add another dimension to screen presentation.

Watch out for staircasing on lower resolution monitors. This is the stepped effect caused when displaying sloping lines. It can be avoided by using only vertical and horizontal lines and alleviated with lines at 45 degrees.

Tools and Methods

You can use a number of methods to create pictures for your Help file. Each method will influence the visual appearance of the picture as well as your approach to design. If possible, try to match the strengths and weaknesses of the method used to create the picture to the kind of image for which it is best suited.

The resolution and accuracy of any device will affect the quality of the final image. Also, many methods require some ability to draw. Employing a professional designer for this work (unless there is one in the team) should be consideredthey will probably save time and produce better results.

For example, you can use any of the following methods to create pictures:

- Use the Windows screen capture facility to take a screen shot and then edit the bitmap in Windows Paintbrush.
- Use a graphics application to create the picture from scratch.
- Use a scanner to capture existing illustrations or clip art.
- Use a digitizing tablet to trace an image.
- Import a picture created using a Macintosh graphics application.

Each picture that you include in the Help file must be saved as a separate file. One way to manage pictures for a project is to create a subdirectory where all pictures are stored, such as an \ART subdirectory in the project directory.

When building the Help file, the Help compiler can compress bitmap files (including .SHG files) if you specify Medium or High using the COMPRESS option in the [OPTIONS] section of the Help project file. On the average, you can expect 25 percent to 50 percent reduction in size using compression. (For details, see Chapter 16, The Help Project File.)

Picture Formats

Pictures that appear in Help files must be in one of the following standard Windows formats.

Format	Description
.DIB	Windows version 3.x device-independent bitmap
.BMP	Windows version 2.x bitmap
.WMF	Windows metafile
.SHG	Windows Help hypergraphic bitmap
.MRB	Windows Help multiresolution bitmap

Several applications, such as Microsoft Word for Windows, MicrografxÒ Designer, CorelÒ Draw, and AldusÒ PageMakerÒ and FreeHandÒ, and others can save files or export them as metafiles. However, some applications, Aldus FreeHand, for example, use a metafile format that differs somewhat from the standard metafile format for Windows. For that reason, you cant use their files directly. You must convert them to the standard metafile format for Windows or convert them to bitmap files.

> Note: If Help cannot display the file because it is in an unsupported format, you can take a screen shot of it, paste it into Windows Paintbrush, and then save it as a Windows-based bitmap. If you cant display the picture in Windows, you must use a conversion utility to convert the file to a supported format.

> Note: Because Windows-based metafiles are not binary compatible with the Macintosh, you should use only bitmaps if you want to ensure cross-platform compatibility.

Creating Bitmaps from Screen Shots

Another easy way to create art for your Help file is to use the Windows screen capture facility. Using screen shots, you can create a bitmap from anything that you can display in Windowsa window, dialog

box, or icon, for example. You can also use this method to convert unsupported picture formats into Windows-based bitmap format. For example, you may want to use a full-featured draw program to create art for your Help file but save the final pictures as bitmaps. You can do that just by previewing the finished art in Windows. When the image displays in Windows, you can copy it to the Clipboard and paste it into a graphics editor such as Windows Paintbrush. Then you use the graphics editor to clean up the image as needed, and save the image in an appropriate format.

To create a bitmap from a screen shot

1. Display on the Windows desktop the element you want to make into a picture.
   For some shots, this may require a special setupstarting the application, opening a document, scrolling to a specific location, and so on.
2. Press ALT+PRINTSCREEN to copy a picture of the active window to the Clipboard.
3. Open Paintbrush.
4. From the View menu, choose Zoom Out.
   Zooming out ensures that Paintbrush will capture the entire screen dump.
5. From the Edit menu, choose Paste.
6. Select any Paintbrush tool.
   Paintbrush displays a zoomed out version of the screen dump.
7. From the View menu, choose Zoom In.
   The screen shot appears full size.
8. Make any changes you want to this piece of art.
9. Use the Pick tool to select the portion of the drawing you want to save as the final bitmap.
10. 10. From the Edit menu, choose Copy To.
11. 11. Save the file as a Windows bitmap.

Creating Pictures from Scratch

Both paint and draw programs are now widely available. Paint programs are well-suited to freehand-type illustrations. They include many features such as facilities for brushes, lines and shapes, filling areas with colors and patterns, image manipulation, and so on. Draw programs are better suited for creating mechanical or structured images. Generally, they take longer to learn to use than paint programs, but they allow greater flexibility in editing the details of the drawing. Unlike pictures created by paint programs, pictures created in draw programs can contain multiple layers, each of which can be manipulated independently.

Painting and drawing programs are based on two very different ways of describing a picture. Painting software uses an array of color pixels (or bit maps) to represent the image. Drafting or object orientated software describes pictures as a list of graphic primitives such as line or circle which have a number of attributes such as position and size.

Creating graphics from scratch may pose a large problem for the inexperienced Help author because their use seems to require the ability to draw. Frequently there appears to be a bewildering variety of shapes, styles, colors, patterns, and specialized tools to choose from. However, creating effective graphics often depends to a large extent on applying sound principles, rather than on pure artisitc skill. The documentation that comes with your graphics editor should provide some direction and guidelines for designing and creating graphics. And if you want to provide a more professional look to your pictures, you can have them created by a competent graphic designer or illustrator.

Microsoft Windows does not include a draw program, but it does offer a basic paint programWindows Paintbrush. You can use Paintbrush to create freehand illustrations and prepare screen images of your application or the Windows environment. The procedure for creating a simple drawing in Windows Paintbrush involves the following steps.

To create a simple drawing in Windows Paintbrush

1. Start Paintbrush.
2. Determine the size of the drawing area.

You define the drawing area by choosing the Image Attributes command from the Options menu. Frequently, it is a good strategy to use the default drawing area (equivalent to the entire screen) while you are creating the drawing so that you have more room to work. When you are ready to save the drawing, you can define the size of the final art.

3. Select a background and foreground color.

   You can change from a color palette to a black-and-white palette (or vice versa) by choosing the Image Attributes command from the Options menu. However, after you begin a drawing, you must stay with the palette you selected.
4. Select a drawing tool.

   In most cases, you will use several tools to create a picture.
5. Draw the picture.

   If you make a mistake while drawing, you can use the Undo command on the Edit menu and the Eraser tool.
6. Make any changes or edits to the picture.

   If you need to see the drawing in greater detail to make fine adjustments, you can use the Zoom In command on the View menu. The Zoom In command magnifies a portion of the drawing so you can change one pixel at a time.
7. 7 When the picture looks the way you want it to, save it as a Windows 16-color bitmap.

   Make sure that you save just that portion of the drawing that you want to appear in Help and not the entire drawing area (if the drawing area is larger than the picture you are creating).

For complete instructions on how to use Paintbrush to create pictures, see the Microsoft Windows Users Guide, version 3.0 or 3.1. If you are using a different graphics editor to prepare your pictures, see the users guide for the graphics application you are using to learn how to create the pictures.

Creating Realistic 3D Graphics

When an object is displayed in two dimensions, as in a drawing, depth relationships, such as whether one line is in front of or behind another, are often ambiguous. On the other hand, when perceiving an actual solid object or scene, the human eye uses certain information, sometimes referred to as depth cues, in order to deduce the spatial structure.

You can include some of these cues, such as occlusion (closer objects overlap those that are further away) and shading, in order to make your pictures more realistic. There are various ways of showing depth with three-dimensional images. You can:

- Remove lines that would be obscured by opaque surfaces. This helps clarify the location of lines and objects.
- Make lines that are further away less visible in the picture, by using dotted or fainter lines.
- Paint the faces of an object in different colors in order to indicate the existence or absence of surfaces.
- Shade the faces of an object to different intensities according to the direction of the light. This method improves the realism. In Windows the light source is always assumed to be above and to the left of an object, as you can see by looking at the shading on any command button.

Creating Pictures from Scanned Images

Scanning images is one of the easiest ways to include graphics in the Help file because the most difficult taskcreating the imagehas already been done. You only have to work the scanning software and then perform final cleanup and editing in Windows Paintbrush or some other graphics application.

The problem with most scanned color images, however, is that they need at least 256 colors to look realistic when they are displayed on a computer screen. Because Windows Help does not support 256-color pictures, and also because 256-color pictures often do not display adequately on systems with standard VGA (16-color) video adapters, you are limited in the kind of images you can scan for use in Help files.

Sixteen-color pictures are suitable for diagrams, line drawings, cartoons, and simple drawings, most of which can be scanned successfully. Also, there is an increasing amount of public clip art available that would be suitable to scan. The goal is to offer the highest possible image quality to users with low-end video hardware but avoid short-changing users with more powerful video capabilities. If you can use a scanner to obtain pictures that meet Helps basic requirements, then scanning is a good way to create graphics for your Help file.

Creating Bitmaps with a Digitizing Tablet

A digitizing tablet is the best way to trace the outline of an existing image and ensures that the proportions are correct. A mouse or a tablet can be used to copy a picture by eye, but this requires more skill than tracing.

Tablets are based on an absolute coordinate system, while mice use relative coordinates: this affects the physical drawing action and is the principal reason why mice are not good for tracing.

Digitizing tablets can be fitted with a stylus (like a pen) which is good for freehand drawing or a transparent cross-hair cursor which is good for accurate tracing and for positioning elements on screen. Mice are also good for positioning elements.

Creating Pictures on the Macintosh

Many graphics artists use the Apple Macintosh computer as their primary tool for creating art. Although the file formats most commonly used by Macintosh applications are not supported by Windows Help, you can use the Macintosh computer to create art for your Help file. Basically, you have two options:

- You can use a Macintosh draw or paint program to create the art and convert the files to a standard Windows graphics format.
- You can insert art in a Word for the Macintosh document and then import the entire document to Word for Windows and capture the art in screen shots.

Importing Art from Macintosh Graphics Applications

Most draw and paint programs that run on the Apple Macintosh have the ability to save files in PICT format. Since there are several ultilities that convert graphics files from PICT to DIB format and from DIB to PICT format, you can create art for your Help file using your favorite Macintosh graphics application and then transfer the art to the PC so it can be built into the Help file. You can also move original art and screen shots Windows saved in DIB or BMP format to the Macintosh application and edit them there.

To move art back and forth between the Macintosh and PC, you need the following tools:

- A conversion utility that can convert the file formats you are using. Alchemy is one example.
- Network hardware and software that connects the PC and Macintosh computers or software like DOS Mounter that enables Macintosh computers to read and write to MS-DOS 3.5-inch floppy disks formatted on the PC.
- A palette file in the Macintosh application that matches the standard Windows 16-colors.

Some Conversion Issues

When importing art created by Macintosh applications, you should be aware of the following issues:

- When transferring files from Windows to the PC, the Macintosh graphics application may not be able to see the files in its Open list box.

  If this happens, it is probably because the converted file has the wrong TYPE and CREATOR set. You can change these settings manually with Macintosh utilities such as DiskTop; however, DOS Mounter and some network software (Microsoft LANManager 2.1, for example) perform automatic extension mapping, which assigns the proper TYPE and CREATOR to a file with a particular MS-DOS file extension whenever a file is moved to the Macintosh.
- Because Windows Help only supports bitmaps with 16 colors, you should create a custom Windows

16-color palette for your graphics application.

That will ensure that the art you create and edit on the Macintosh will display with the correct colors when transferred back to Windows. If you want to use dithered colors that are found in Windows Paintbrush, you may need to create those as well, for example as custom brush patterns in Studio/8.

- Some graphics applications on the Macintosh save files with a minimum 256-color depth (8-bit color).

  If this is true, make sure that the conversion utility reduces the file to 16 colors (4-bit color) when it converts it to a bitmap. Or open the 256-color bitmap in Paintbrush and save the file as a 16-color bitmap. Storing bitmaps on the PC in 16 colors will also keep file sizes to a minimum and conserve disk space.
- MS-DOS only accepts filenames that have 8 characters plus a 3-character extension.

  Because the Macintosh does not have this limitation, you should follow the MS-DOS 8-character naming convention when naming the art you create on the Macintosh, and use the .PIC extension. Otherwise, the filenames of graphic files you transfer may be difficult to interpret.

Using Batch Files to Control the Conversion Process

If the conversion utility you are using runs in MS-DOS, you may want to create a batch file to control the conversion process. Batch processing ensures consistency in the conversion routine and also allows you to convert many files at the same time. The following example shows one way to set up a batch-file process for converting art between the Macintosh and PC.

The first batch file converts Windows BMP files to Macintosh PICT format:

```
@echo off
break on
if "%1" == "" goto ERROR
echo.
echo Converting BMP file to PICT format...
echo.
:RESTART
for %%p in (%1) do c:\tadpole\alchemy -mo %%p
shift
if not "%1" == "" goto RESTART
goto MOVEFILE
:ERROR
cls
echo This batch file converts Windows BMP files to Mac PICT format.
echo Wildcards and multiple sets are allowed.
echo Syntax is:
echo.
echo %0 file(s).BMP
echo.
echo where file(s) = separate file names and/or wildcards.
echo.
goto EXIT
:MOVEFILE
echo.
echo Conversion complete.
echo Now moving file to EDITS\PICT folder...
echo.
copy *.pic ..\edits\pict
del *.pic
:EXIT
echo.
echo All done!
```

The second batch file converts Macintosh PICT files to Windows BMP format:

```
@echo off
break on
if "%1" == "" goto BIGERROR
set loc=%1
echo.
echo ---------------
echo !!! WARNING !!!
echo.
echo If you are processing art for existing SLM
echo art files (that have already been 'addfile'd), you MUST first
echo check out the file(s). Otherwise the edits and changes will be lost!
echo.
echo Press the CTRL+C keys to stop this batch file, or pause
direxist \archive\%loc%
if errorlevel 1 goto ERROR1
goto PICT2ARC
:ERROR1
echo.
echo     \\TOAD\MAC!ARCHIVE\%loc% does not exist.
echo     Please check the spelling of the directory name
echo     or confirm you are starting from the net drive.
echo.
goto EXIT
:PICT2ARC
echo.
echo     Placing copy of PICT files in \\TOAD\MAC!ARCHIVE\%loc%...
echo.
copy *.pic \archive\%loc%
direxist c:\tadpole\%loc%\art
if errorlevel 1 goto ERROR2
goto PICT2C
:ERROR2
echo.
echo     C:\TADPOLE\%loc%\ART does not exist.
echo     Please check the spelling of the directory name.
echo.
goto EXIT
:PICT2C
echo.
echo     Moving files to C:\TADPOLE\%LOC%\ART...
echo.
copy *.pic c:\tadpole\%loc%\art
del *.pic
c:
cd \tadpole\%loc%\art
:CONVERT
echo.
echo     Converting PICT files to BMP...
echo.
for %%p in (*.pic) do c:\tadpole\alchemy %%p -wo -D -f c:\tadpole\bmp16.pal
del *.pic
echo.
echo     Now, addfile or check in the .BMP files...
echo.
goto EXIT
:BIGERROR
cls
```

```
echo This conversion starts from the server drive, copies ALL the
echo PICT files to the net archive directory, then moves those files
echo to your C: drive in the appropriate \TADPOLE SLM project
echo directory and performs the conversion. Part of the conversion
echo limits the colors of the PICT files to the basic Windows 16 colors.
echo.
echo Syntax is:
echo.
echo       PREP4PC (location)
echo.
echo (location) is the name of the application subdirectory.
echo For example, the syntax:
echo.
echo       PREP4PC calc
echo.
echo would place the final converted file(s) in
echo the C:\TADPOLE\CALC\ART directory.
echo.
:EXIT
set loc=
break off
```

Importing Art From Microsoft Word for the Macintosh

If you have art that is stored in Word for the Macintosh files, you can import that art into Word for Windows and use it in your Help file. To convert the art, you import the Word for the Macintosh document to Windows and then take a screen shot of the art and save it in Paintbrush as a Windows bitmap.

To import art from a Word for the Macintosh file

1. Save the Word for the Macintosh file as RTF.
2. Open the RTF file in Word for Windows.
3. Select the picture you want to use in your Help file.

   Note If you are using Word for Windows version 2.0, do not double-click the piece of art, or Word will convert the picture into an embedded Draw graphic. If that happens, you will not be able to paste the picture into Paintbrush.
4. Copy the picture to the Clipboard.
5. Open Paintbrush.
6. From the Edit menu, choose Paste.
7. Make any changes you want to the picture.
8. When the art looks the way you want it to, save it as a Windows 16-color bitmap.

Preparing Pictures for Different Displays

One important issue to consider when creating graphic images is how the pictures will appear on different video displays. Help can display graphics on computers with many different kinds of video hardware, including the most common display typesmonochrome, color, EGA, VGA, and 8514. When you prepare pictures for Help, you should analyze the target audience for the Help file and determine what percentage of that audience has monochrome or color displays, and what percentage has each type of video adapter so that you can create pictures that will look good on the end-users machine.

Because Windows Help is device independent, it stores text and graphics by point size rather than by their width and height in pixels. When displaying bitmaps, Windows Help tries to maintain a bitmaps logical size across all displays (EGA, VGA, or 8514, for example). When displaying bitmaps, Help sizes text and graphics proportionately by preserving the authored point size of the text or bitmap file. Because screens have different point sizes based on their resolution, bitmaps take up different proportions of the screen. This means that a picture will distort unless the authors display and the users display have the same resolution.

Therefore, to make certain that the pictures you create display correctly on the users display, you should create pictures in the same screen resolution (EGA, VGA, VGA+, or 8514) that you want Windows Help to use when it displays the topics on the users computer. For example, if most of your users have VGA adapters, you should create the pictures on a machine with standard VGA resolution. (For more information about supporting multiple display resolutions in Help, see Chapter 12, Creating Graphics for Different Displays.)

When creating color pictures for Help files, Windows Help assumes that your pictures have 16 or fewer colors. If you dont use Windows Paintbrush to create the pictures, the color palette in your paint program should have no more than 16 colors. Since Windows Help supports only 16-color pictures, you dont need to worry about palette conflictsall pictures use the standard Windows system palette. However, you may have to make palette adjustments in the graphics application, especially if the graphics program has 256-color support. If you create a monochrome bitmap using Word for Windows, the monochrome bitmap uses the windows foreground and background colors.

The descriptions in the remaining sections of this chapter assume that youve identified and prepared the pictures you want to include. If you need more information about preparing pictures, see the users guide for the graphics application you are using.

Placing Graphics in Topic Files

After you create and prepare the pictures for your Help file, you can place them in a topic file in two ways.

- You can paste the pictures directly in the topic, if your word processor supports Windows version 2.1 or later graphics.

  Pictures placed using this method are referred to as inline bitmaps.
- You can save each bitmap file in your graphics application and, in the Help topic file, specify the file by name where you want it to appear.

  Pictures placed using this method are referred to as bitmaps by reference, because the topic contains only a reference, or pointer, to the bitmap file, not the actual picture itself.

Each method has advantages and disadvantages. Use the information in the following sections to help you decide which method to use.

For detailed information about topic layout, including the effects of margins and paragraph leading, see Chapter 3, Designing Your Help System.

Placing Pictures Directly in the Topic File

Pictures placed directly in a topic appear like pictures on a page in a book. To place the picture in a topic, you import the bitmap directly from the Clipboard into Word for Windows, and then paste the picture where you want it in the topic. You can format the topic text so that it is positioned below or alongside the bitmap. When you save the topic file as RTF, the pasted-in bitmap is converted and included in the Help file.

The advantages to using this method are:

- You can see the picture whenever you work in the topic file.
- You dont have to tell the compiler where to find the bitmap files when building the Help file.

The disadvantages are:

- Using the same picture more than once will increase the size of the Help file.
- Bitmaps pasted directly into Word for Windows cannot exceed 64K. If they do, the Help topic will cause an out-of-memory condition during the build, and the compiler will abort the build.
- You can use only inline bitmaps (those treated the same as a character) in a Word for Windows document, which limits how you can display text and graphics in your Help file. For example, you cannot wrap text around an inline bitmap.
- Pictures inserted directly into a Word for Windows document affect performance, especially when scrolling or saving documents.
- You cannot change the picture inside Word for Windows. Instead, you must modify the picture in your graphics application and then re-import it into Word for Windows.
- You cannot use graphics with multiple hot spots in your topic files. For more information, see Chapter 11, Creating Hypergraphics.
- Only Word for Windows lets you include bitmaps directly in the topic file. So, you cannot use a Macintosh word processor, such as Microsoft Word for the Macintosh, to create any of your RTF topics.

Placing Bitmaps in Word for Windows

Word for Windows treats bitmaps as a single, very large character. How a bitmap displays on the screen, however, is determined in part by the printer you select. The bitmap sizes shown in the Format Picture dialog box are based on the pixel size Word for Windows associates with the printer and its printing resolution. Generally, if you insert a bitmap and select Display As Printed (version 1.1) in View Preferences (or Line Breaks and Fonts As Printed in version 2.0), the actual ruler size of the bitmap is the same as that indicated by the dialog box.

If you cancel Display As Printed, change printers, or change the display resolution (from EGA to VGA, for example), the size in the Format Picture dialog box changes. When Display As Printed is canceled, Word for Windows assumes the pixels are screen pixels. Distortions in the displayed bitmap may result from the difference in size between screen pixels and printer pixels.

In Word for Windows, every bitmap has a frame around it. The frame is invisible, however, unless you select a border from the Format Picture dialog box.

A bitmap that is visible in the topic file often appears as a blank box while you are typing. Similarly, as you move through the topic file, picture frames are blank to speed scrolling and typing. When you pause, Word for Windows fills in the picture(s).

Importing Bitmaps into Word for Windows

After creating your bitmaps, you can copy them to the Clipboard and then paste them into your topic file using Word for Windows.

To insert a bitmap into a topic file

1. Copy the bitmap from your graphics program to the Clipboard.

2. Position the insertion point where you want the bitmap to appear in the topic file.
3. From the Edit menu, choose Paste.
   Or press SHIFT+INS.

Selecting Inline Bitmaps with the Mouse

Before you can do anything with the bitmap, you have to select it. To select a pasted-in bitmap, click inside the picture frame. When you select a bitmap, eight sizing handles appear on the picture frame: one on each side and one on each corner, as in Figure 10.x.

Sizing Pasted-In Bitmaps

Once you select a bitmap, you can use the sizing handles to change its size by scaling it. The handles move both toward and away from the center of the bitmap.

To scale a pasted-in bitmap

1. Select the bitmap.
2. Hold down the SHIFT key while you drag the sizing handles to change the scaling of the image.
   When you drag the sizing handles, scaling information appears in the status bar. Measurements are in percentages of the original image size.

Formatting Inline Bitmaps as Hot Spots

You can do more with bitmaps in your Help file than simply display them. Windows Help lets you use bitmaps as hot spots. You can create graphics, such as icons or buttons, and use them as jumps to particular topics or as hot spots for pop-up windows or Help macros.

To format an inline bitmap as a hot spot

1. Select the bitmap in the topic file that you want to format as a hot spot.
2. From the Format menu, choose Character.
3. Select the Double Underline check box, and then choose OK.
   Or select the Underline check box if you want to create a pop-up hot spot.
4. Position the insertion point to the immediate right of the bitmap.
   Leave no spaces or characters between the bitmap and the insertion point.
5. From the Format menu, choose Character.
6. Clear the Double Underline check box, select the Hidden check box, and then choose OK.
   Or clear the Underline check box if this is a pop-up hot spot.
7. Type the context string of the destination topic for the jump.
   Or type the context string assigned to the pop-up topic.
   Or type an exclamation point (!) followed by the macro and its parameters if you want Windows Help to execute a macro when the user chooses this hot spot.
8. From the Format menu, choose Character.
9. Clear the Hidden check box, and then choose OK.

Figure 10.x shows three bitmaps formatted as hot spots.

Placing Bitmaps by Reference in the Topic File

To achieve finer control over the placement of a picture, you can place a bitmap reference in the topic. The bitmap reference is like a coded message that tells the compiler the name of the bitmap file and how to position the picture with respect to the topic text. When Windows Help encounters a bitmap reference, it inserts a reference to a central location in the Help file where the picture is stored. The Windows Help application uses this referenced information provided by the Help compiler to display the picture.

You can use bitmap references to display a variety of pictures in Help, from tiny bullet characters to full-screen illustrative art. The advantages to using bitmap references are:

- You have the widest range of options for positioning the picture with text. For example, you can treat the picture as a text character, or you can wrap text around the left or right edge of the picture.
- You should not notice any change in the performance of Word for Windows; bitmaps included by reference do not affect it any more than text does.
- You can change the graphic as often and as much as you like without changing the reference in the topic file. The references are simply pointers to the actual bitmap files.
- You can use hypergraphics (pictures with hot spots) in your topic files. In fact, hypergraphics must be included by reference. For more information, see Chapter 11, Creating Hypergraphics.
- You can use any text editor that produces RTF files to create your Help topics, including Word for the Macintosh.

The disadvantages are:

- You cant see the graphic when you are working in the word processor.
- You must tell the compiler where to find the actual bitmap files referenced in the topic files when you build the Help file. Although this is a simple task, mistakes in this process will cause errors when you compile the Help file.

Including Bitmaps by Reference

To include bitmaps by reference in a topic file, you must make the following changes to the topic and Help project file:

- Enter a reference command in the topic file where you want the picture to appear.
- Make an entry in the Help project file that tells the compiler where to find the bitmap files referenced in the topic files.

Help supports three different positions for bitmaps included by referencecharacter, left-aligned, or right-aligned. Each position defines how the bitmap is placed with respect to text, and each position requires a slightly different command reference in the topic file:

character	{bmc bitmap-filename}
left-aligned	{bml bitmap-filename}
right-aligned	{bmr bitmap-filename}

Positioning Character-Aligned Bitmaps (bmc)

Windows Help treats a bmc bitmap as a text character. It aligns the bitmap on the baseline of the type in exactly the same place in the paragraph where the reference command occurs. Because the bitmap is treated as text, any paragraph formatting properties assigned to the paragraph also apply to the bitmap. Text coming before or after the bitmap does not wrap around the bitmap, but it may appear slightly higher on the line if any white space is saved at the bottom of the bitmap image.

Help files use bmc bitmaps in many ways. For example, you might include a bitmap in the explanation of an interface element. Figure 10.x shows how Word for Windows displays the bitmap reference in the topic file.

> Note: Dont specify negative line spacing for a paragraph that has a bmc bitmap reference. If you do, the bitmap might appear on top of the succeeding paragraph when

Windows Help displays the topic.

After you compile your topic file, Windows Help displays the referenced bitmaps, as in Figure 10.x.

Positioning Left-Aligned Bitmaps (bml)

Windows Help places a bml bitmap along the left margin, with text wrapping along the right edge of the image. Text is aligned with the upper-right corner of the bitmap, so any white space saved at the top of the image affects the way text appears in relation to the bitmap.

You normally place bml references at the beginning of a paragraph to ensure proper wrapping of text around the right edge of the bitmap. For example, a typical bml reference is formatted as follows:

Paragraph text follows the bitmap reference...

Figure 10.x shows how a bml reference appears in the Word for Windows topic file.

> Note: Do not put any space characters between the bml reference and the paragraph text unless you want the first line of text indented from the rest of the text that wraps along the right edge of the bitmap.

After you compile the topic file, Windows Help displays the referenced bitmap to the left of the text, as in Figure 10.x.

You can also put a bml reference at the end of a paragraph, as in this example:

...Paragraph text precedes the bitmap reference.

Figure 10.x shows how this type of bml reference appears in the Word for Windows topic file.

When you include a bml reference this way, Windows Help wraps the text to the paragraph end and then displays the bitmap. After you compile the file, the bitmap appears under the text and to the left, as in Figure 10.x.

Positioning Right-Aligned Bitmaps (bmr)

Windows Help places a bmr bitmap along the right margin, with text wrapping along the left edge of the image. Text is aligned with the upper-left corner of the bitmap, so any white space saved at the top of the image affects the way text appears in relation to the bitmap.

You normally place bmr references at the beginning of a paragraph to ensure proper wrapping of text around the left edge of the bitmap. For example, a typical bmr reference is formatted as follows:

Paragraph text follows the bitmap reference...

Figure 10.x shows how a bmr reference appears in the Word for Windows topic file.

> Note: Do not put any space characters between the bmr reference and the paragraph text unless you want the first line of text indented from the left topic margin.

After you compile the topic file, Windows Help displays the referenced bitmap to the right of the text, as in Figure 10.x.

You can also put a bmr reference at the end of a paragraph, as in this example:

Paragraph text precedes the bitmap reference.

Figure 10.x shows how this type of bmr reference appears in the Word for Windows topic file.

When you code a bmr reference this way, Windows Help wraps the text to the paragraph end and then displays the bitmap. After you compile the file, the bitmap appears under the text and to the right, as in Figure 10.x.

Storing Bitmaps with Data

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

When you use the bmc, bml, or bmr command, the bitmap data is stored separately from the Help file text. A single copy of the bitmap is used for all bmc, bml, or bmr commands that display the same bitmap. Using these commands can therefore reduce the size of your Help file if you reference large bitmaps more than once in a file. The disadvantage is that Windows Help must retrieve the bitmap from the disk before displaying it in the topic, so these bitmaps may display more slowly than those referenced with the wd commands, especially if the Help file is stored on a CD-ROM disc.

Benefits

When you use the bmcwd, bmlwd, or bmrwd command, the bitmap data is stored in the same location in the Help file as the text. The bitmap data becomes part of the Help file, and this data appears in the location given by the command. When the bitmap is stored inline with the text, Windows Help may display it faster than if it were positioned using the bmc, bml, or bmr command because it doesnt have to retrieve information from different locations on the disk. Reading data from different locations on a CD-ROM is very time consuming, and it can also be slow on magnetic media.

Moreover, storing bitmaps with data maximizes the benefits of using the OPTCDROM option in the Help project file. Help keeps up to 12K of compressed topic text in memory. Since bitmaps with data are kept with the topic text, they take advantage of this buffering. So, generally, topics less than 12K after compression do not need to be re-read from disk when the window is resized or redrawn.

Limitations

The primary limitation of the wd commands is that each bmcwd, bmlwd, or bmrwd command stores a separate copy of the bitmap. That means your Help file will be larger if you use bitmaps stored with data than if you use the commands without wd, especially if you reference larger bitmaps more than once. Large bitmaps that are referenced using wd may also cause out-of-memory error messages when compiled under MS-DOS.

Another limitation is that bitmaps stored with data cannot be more than 64K after compression. If the compression reduces the size of the bitmap to less than 64K, then you can store it with data. How much compression is achieved depends on the bitmap. That means you can have two different bitmaps that are 80K before compression, and one may compress to less than 64K and the other may not. Therefore, Help claims a 64K limit for bitmaps stored with data, even though the actual limit may vary slightly.

Because the Windows bitmap format does not include any compression, all bitmaps are compressed during the build by the Help compiler. If you want a quick way to determine how large the bitmaps will be when they are put into Help, you can use Hotspot Editor or Multi-Resolution Bitmap Compiler, since they compress bitmaps using the same mechanism as the Help compiler. If the bitmap is still too large, you may have to reduce its size or complexity.

Guidelines

Use the following guidelines to decide whether to use the normal or wd versions of these commands:

- Use bmc, bml, and bmr for all bitmaps that appear frequently in your Help file.

  If a bitmap is used frequently, not placing it with data may be more appropriate, unless the performance benefit outweighs the size factor.
- Use bmcwd, bmlwd, and bmrwd for small bitmaps (less than 64K) that appear only once or infrequently in your Help file.

  Bitmaps with data use extra disk space only when the same bitmap is used more than once in the Help file.
- Use bmc, bml, and bmr for large bitmaps (greater than 64K).

Large bitmaps stored with data may worsen performance if the topic is drawn twice to remove unnecessary scroll bars. If the topic is over 12K, it will be read, laid out, and then read a second time as it is laid out without the scroll bars. Because bitmaps cause most topics to exceed 12K, this problem is likely to occur if you store bitmaps with data. Bitmaps not stored with data, on the other hand, are kept in a different location from the text, so they are read from disk only once if the topic text is less than 12K.

- Use bmc, bml, and bmr for bitmaps that appear in two or more topics normally displayed in sequence (for example, in a browse sequence).

  Help caches in memory the 50 most recently accessed bitmaps that are not stored with data. This means that a bitmap may only have to be read from disk once if two Help topics reference it successively. Unlike bitmaps stored with data, these cached bitmaps have to be re-read only if theyre moved out of the cache by 50 new bitmaps, or by low memory conditions.

  Cached bitmaps not stored with data degrade performance if they are displayed only once. Since they are stored in a different location from the topic text, Help requires an additional seek to locate them. This cost, if incurred, is generally negligible on magentic disks but high on CD-ROM. Nonetheless, this performance cost may be less than the cost of reading the topic twice if the topic is larger than 12K.

Creating Bitmap Hot Spots

You can do more with pictures in your Help file than simply display them. You can also create graphics, such as icons or buttons, and format them as hot spots. With a bitmap hot spot, the user can click any part of the graphic to activate the hot spot: jump, pop-up, or macro. Bitmap hot spots use the same formatting and positioning characterisitics as text hot spots, so you can use any bitmap or metafile as a hot spot.

To format a bitmap as a jump hot spot

1. Type the bitmap reference (bmc, bml, bmr) in the topic file where you want to create the jump.
2. Select the entire bitmap reference, including the bitmap filename and the opening and closing braces.
3. From the Format menu, choose Character.
4. Select the Double Underline check box, and then choose OK.
5. Position the insertion point immediately after the closing brace of the bitmap reference.
   Leave no spaces or characters between the brace of the bitmap reference and the insertion point.
6. From the Format menu, choose Character.
7. Clear the Double Underline check box, select the Hidden check box, and then choose OK.
8. Type the context string assigned to the topic that you want users to jump to when they choose this hot spot.
9. From the Format menu, choose Character.
10. Clear the Hidden check box, and then choose OK.

Figure 10.x shows the GLOSS.BMP bitmap correctly formatted to jump to the Contents topic of the Glossary Help file when it is chosen.

To format a bitmap as a pop-up hot spot

1. Type the bitmap reference (bmc, bml, bmr) in the topic from which you want to display the pop-up window.
   Be sure you have created the topic (and its context string) that will appear in the pop-up window.
2. Select the entire bitmap reference, including the bitmap filename and the opening and closing braces.
3. From the Format menu, choose Character.
4. Select the Underline check box, and then choose OK.
5. Position the insertion point to the immediate right of the closing brace of the bitmap reference.
   Leave no spaces or characters between the brace of the bitmap reference and the insertion point.
6. From the Format menu, choose Character.
7. Clear the Underline check box, select the Hidden check box, and then choose OK.
8. Type the context string assigned to the pop-up window topic.
9. From the Format menu, choose Character.
10. Clear the Hidden check box, and then choose OK.

Figure 10.x shows the RT.BMP bitmap coded to display a pop-up window (with the rt05_how_pm context string) containing all cross-references to the current topic when it is chosen.

To format a bitmap as a macro hot spot

1. Type the bitmap reference (bmc, bml, bmr) in the topic file where you want the macro.
2. Select the entire bitmap reference, including the bitmap filename and the opening and closing braces.
3. From the Format menu, choose Character.
4. Select the Double Underline check box, and then choose OK.
5. Position the insertion point to the immediate right of the closing brace of the bitmap reference.
   Leave no spaces or characters between the brace of the bitmap reference and the insertion point.
6. From the Format menu, choose Character.
7. Clear the Double Underline check box, select the Hidden check box, and then choose OK.
8. Type an exclamation point (!) followed by the macro and its parameters that you want Windows Help

to execute when the user chooses this hot spot.

9. From the Format menu, choose Character.
10. Clear the Hidden check box, and then choose OK.

Figure 10.x shows how to format the WINCLK.BMP bitmap so that Windows Help executes a macro starting the Clock application when it is chosen.

Assigning Bitmap Locations in the Help Project File

Once you have the bitmaps properly situated in the topic file, you must be sure the Help project file contains the information necessary to access and use them. This section describes the entries in a Help project file that are required to display bitmaps.

When you use bitmap-reference commands to position bitmaps, you must tell the Help compiler where to find the bitmaps. The easiest way to do that is to include a BMROOT option in the [OPTIONS] section of the project file.

> Note: Another way to identify bitmaps is to list the filenames individually in the [BITMAPS] section of your Help project file. This method is not recommended unless you are creating a Help file for Windows Help version 3.0. For more information about the [BITMAPS] section, see Chapter 16, The Help Project File.

The BMROOT option lists the paths where you want Windows Help to look for bitmaps positioned by the bmc, bml, and bmr references. You can give absolute paths or paths relative to the directory containing the Help project file. This option has the following syntax:

BMROOT=pathnames

For example, you can create a subdirectory off the main project directory where you keep all the bitmaps:

```
[OPTIONS]
    .
    .
    .
BMROOT=.\BMP
    .
    .
    .
```

Assuming the Help project file for the Help file is in the WIN31\HELP directory, this option tells Windows Help to look for bitmaps in the WIN31\HELP\BMP directory.

You can assign more than one path to the BMROOT option; Windows Help searches all the paths, in the order given, for the specified bitmaps. Separate multiple paths with commas. For example:

```
[OPTIONS]
BMROOT=.\BMP, .\WMF, .\SHG
```

Chapter 11 Creating Hypergraphics

Using convetional graphics, you can define only one hot spot per graphic. Although that is useful, it limits the way in which you can link other kinds of information to pictures. Fortunately, Windows Help provides a tool that lets you define many hot spots using a single picture. This chapter describes how to add multiple hot spots to graphics so that they can link to other information in the Help file.

What Are Hypergraphics?

A hypergraphic is a picture containing one or more hot spots. Like bitmap hot spots, hypergraphics let you use graphics instead of text to create links to other topics, display information in pop-up windows, and execute Help macros. The difference between bitmap hot spots and hypergraphics is that, using hypergraphics, you can define multiple hot spots within the same picture to perform all these operations. In this way, creating hypergraphics extends the usefulness of graphics in your Help file.

Conceptually, a hypergraphic consists of two layers:

- A Windows bitmap or metafile
- One or more rectangular hot spots superimposed on the bitmap

The bitmap or metafile creates the picture that the user sees in the Help window, and the hot spots define regions within the picture that the user can interact with.

Here are just a few examples of how you might use hypergraphics in a Help file:

- Take a screen shot of a dialog box in your application and define hot spots for each field so that a user can click the hot spots and learn how to use the dialog box options.
- Create an illustration and add hot spots over different parts of the illustration that the user can click to see a description of the corresponding part.
- Create a Contents screen or Help menu using a picture so that you can control the placement of text and graphics more precisely than with paragraph formatting. Define hot spots over the appropriate icons or jumps to give users access to the Help topics.
- Create a visual map and add hot spots that jump to topics on the different regions shown in the map.
- Create buttons for each letter of the alphabet and arrange them in groups (as a typewriter keyboard does, for example). Use the button picture as a hypergraphic in an online index or glossary so that the user can click a letter and see the entries for that letter of the alphabet.
- Create a series of pictures that depict different states of an action or simulation. Make each picture in the series a hypergraphic so that when the user clicks a hot spot the picture changes to show the new state (like a door opening and closing, for example).

The rest of this chapter explains how to create and add hypergraphics to your Help file.

Hotspot Editor

To create hypergraphics you use Hotspot Editor, an application that lets you open any standard Windows bitmap or metafile and add hot spots to it. You can then save the bitmap in hypergraphic format, with the picture and hot spots combined. Using Hotspot Editor, you can define hot spots that link to other Help topics and graphics, that execute a Help macro, or that access multimedia events (if provided for by external DLLs). After you create a hypergraphic and save it with Hotspot Editor, you can add it to Help topics exactly like any other bitmap. The only restriction is that hypergraphic files produced by Hotspot Editor must be included by reference using the bmc, bmr, or bml format. (For an explanation of these formats, see Chapter 10, Adding Graphics.)

Although you can create similar effects in the Help file by carefully creating and positioning several individual bitmaps next to each other and formatting each of them as hot spots, Hotspot Editor is more efficient and much easier to use. Using several bitmaps to simulate a single-image hypergraphic also has the drawback of requiring Help to perform multiple locating operations when displaying the topic. Usually this means Help must read the disk several times, slowing overall performance, especially on CD-ROM.

If you are creating a 3.0 Help file, you cannot use Hotspot Editor. In that case, see Creating Hypergraphics Without Hotspot Editor, later in this chapter.

Starting Hotspot Editor

Hotspot Editor is an independent application that runs in the Windows operating system, so you can start it the same way you start other applications in Windows. If you selected the option to install the graphics tools when you installed Help Author, the setup program automatically creates a program item icon for Hotspot Editor in the Help Author group window of Program Manager.

To start Hotspot Editor

1. Open the Help Author group in Program Manager and double-click the Hotspot Editor icon.

If you did not use the setup program to install Hotspot Editor, you can add a program item icon for it in a Program Manager group.

To create a program item for Hotspot Editor

1. Open the group in the Program Manager window where you want to add the program item.
2. From the File menu, choose New.
3. Select the Program Item option, and then choose OK.

   The Program Item Properties dialog box appears.

   Tip To open this dialog box quickly, hold down the ALT key and double-click in an empty part of the group window.
4. Type Hotspot Editor in the Description box.
5. In the Command Line box, type SHED.EXE and include a path if the application is not in your Windows directory.

   For example, you might type c:\hlptools\shed.exe

   You can also use the Browse button and select the application filename from the directory where it is stored .
6. Fill in the remaining options in the Program Item Properties dialog box if you want to, and then choose the OK button.

   For help with the Program Item Properties dialog box, choose the Help button or press F1 while using the dialog box.

Importing Image Files

To create a hypergraphic, you must first import the graphic that you want to use to define hot spots. Because Hotspot Editor is not a draw or paint program (its an editor only), you must create the actual graphic image outside Hotspot Editor. Hotspot Editor creates a hypergraphic (.SHG) file when the imported graphic image is saved.

Hotspot Editor can open and save any standard Windows bitmap (.BMP) or device-independent bitmap (.DIB) created by a Windows draw or paint program, such as Windows Paintbrush. If you are using Windows Paintbrush, you should save the image file as a monochrome bitmap or 16-color bitmap. Hotspot Editor is also compatible with standard Windows metafile (.WMF) files.

> Note: If Hotspot Editor cannot open the file because it is in an unsupported format, you can take a screen shot of it, paste it into Windows Paintbrush, and then save it as a Windows bitmap. If you cant display the picture in Windows, you must use a conversion utility to convert the file to a supported format.

> Note: If you change an image file after you have edited the graphic in Hotspot Editor, you can re-import the modified image file in Hotspot Editor without losing any hot spot information. For more information, see Editing and Replacing Images, later in this chapter.

Hotspot Editor Window

Hotspot Editor has a standard multiple document interface (MDI) window that lets you edit multiple graphic images at the same time. Each image that you open in Hotspot Editor is displayed in a separate document window. Switching from one document window to another does not affect previously selected windows. Likewise, actions performed on a document window affect only that window. Document windows can be minimized; Hotspot Editor displays, as accurately as possible, a reduced version of the image in the minimized window icon. The status bar at the bottom of the Hotspot Editor window displays a brief description of the currently selected menu or command and attribute information for the currently selected hot spot.

Hotspot Editor Menus

Hotspot Editor has four menus: File, Edit, Window, and Help. The commands for these menus are described in the following lists.

The File Menu

Commands on the File menu let you open or save graphics files.

Command	Description
Open	Opens existing 16-color bitmaps (.DIB or .BMP), metafiles (.WMF), or hypergraphics (.SHG). More than one graphic file can be open at one time.
Close	Closes the currently selected document window.
Save	Saves changes to a hypergraphic. Hotspot Editor displays a warning if you are overwriting the original bitmap.
Save As	Saves a hypergraphic under a different name. The default extension .SHG is used if one is not specified.
Exit	Exits Hotspot Editor.
File Names	Opens a previously edited image file or hypergraphic. Hotspot Editor lists the last four graphics you edited on the File menu.

The Edit Menu

Commands on the Edit menu let you edit hot spots and define their attributes.

Command	Description
Undo	Undoes the last action you performed.
Attributes	Displays the attributes for each hot spot. Attributes include the binding string, binding type, binding attribute, hot-spot identifier, and the bounding box coordinates. (For more information about hot-spot attributes, see Adding Hot Spots to Bitmaps, later in this chapter.)
Select	Displays a dialog box listing all the defined hot-spot IDs for the bitmap. You choose an item from the list to select a particular hot spot.
Delete	Removes the currently selected hot spot.
Cut	Removes the currently selected hot spot from the hypergraphic and places it in the Clipboard.
Copy	Copies the currently selected hot spot and places it in the Clipboard.
Paste	Pastes a graphic or hot spot from the Clipboard. If the Clipboard contains a graphic, it is pasted into Hotspot Editor. If the Clipboard contains a hot spot, it is pasted in the upper-left corner of the graphic. You can then move the hot spot to the desired location.
Replace	Replaces the underlying bitmap in a hypergraphic without losing any hot spot information that has already been defined for the hypergraphic.
Preferences	Displays a dialog box that lets you set default values for the binding string, binding type, binding attribute, and hot-spot

identifier options of the Attributes command.

The Window Menu

Commands on the Window menu let you arrange and select Hotspot Editor windows and icons.

Command	Description
Cascade	Arranges and resizes the open graphic windows so that each title bar shows.
Tile	Arranges and resizes the open graphic windows so that they all fit in the workspace without overlapping.
Arrange Icons	Aligns the graphic window icons evenly along the bottom of the main Hotspot Editor window.
Window Names	Lists all the currently opened hypergraphic windows. To select a window and display its graphic, choose the window name from the list.

The Help Menu

Commands on the Help menu display Help and copyright information for Hotspot Editor.

Command	Description
Contents	Displays the Contents topic for the Hotspot Editor Help file.
About	Displays information about the Hotspot Editor application, including copyright notice and version number.

If you are opening menus, the gray status bar at the bottom of the Hotspot Editor window displays information about the currently selected command. It also displays information about commands and hot spots.

Once you select a hot spot, the status bar displays the following information for that hot spot:

- Context string
- Binding type
- Binding attribute
- Hot-spot identifier
- Bounding box coordinates

Opening Image Files

Before you can add hot spots to a bitmap with Hotspot Editor, you must first create an image file and then open it in Hotspot Editor to edit. To edit more than one image file at a time, open multiple files. Each file appears in its own document window in Hotspot Editor.

Opening New Files

To open a new image file

1. From the File menu, choose Open.
   The Open dialog box appears.
2. If the file is in the current directory, select the name of the file in the Files box.
   If the file is not in the current directory, double-click the directory you want in the Directories box, and then select the name of the file in the Files box.
   Hotspot Editor displays in the Files box only those files whose extensions match the supported file formats.
   The current directory is shown below the File Name box. Double-clicking the two periods [..] at the top of the Directories box shows the files and directories located one level closer to the root directory.
3. Choose OK.
   Instead of using the Files and Directories boxes to open a file, you can type the complete path in the File Name box, and then click OK.

Opening Previously Edited Files

Hotspot Editor records the last four files you opened. Their paths and filenames are displayed at the bottom of the File menu in Hotspot Editor.

You can open a previously edited file in two ways: either choose the file you want to open from the File menu or type the number that appears to the left of the file you want to open.

Adding Hot Spots to Images

After opening an image file in Hotspot Editor, you can create hot spots that link to text, graphics, or multimedia events. You insert hot spots to the areas of the picture that you want to be hot, just as you determine what text you want to be hot in the topic files. For example, you might want to make a button in a screen shot a hot spot that creates a pop-up window when the user chooses the hot spot.

For each hot-spot you add to an image, you follow the same two steps.

To add a hot spot

1. Draw the hot spot rectangle on top of the image.
   This defines the region that the user will click to activate the hot spot.
2. Define the properties for the hot spot.
   This determines what action Help will take when the user choose the hot spot.

Drawing Hot Spots

A hot spot can be any rectangular area of the graphic image. Defining a hot spot is similar to drawing a rectangle in a draw or paint program.

To draw a hot spot on an image

1. Position the mouse pointer on the bitmap where you want to define a hot spot and press the left mouse button.
   This creates the anchor for one corner of the hot-spot rectangle.
2. While holding down the left mouse button, drag the mouse until the rectangle encloses the area you want to define as the hot spot.
   A flexible box stretches from the anchor point to the position of the mouse pointer, expanding and contracting as you move the mouse.
3. When you are satisfied with the size of the hot-spot rectangle, release the mouse button.
   After you release the mouse button, the hot-spot rectangle displays eight sizing handles, indicating that it is the currently selected hot spot. You can use the sizing handles to resize the rectangle.
   Your hot spot should look like the one in Figure 11.1.

You can draw hot spots so that they overlap each other; however, in the built Help file, only the top most hot spot is active in a stack of overlapped hot spots.

Creating a Tabbing Order

In the built Help file, users can select the hot spots either with the mouse or with the keyboard. When using a keyboard, users press the TAB key to move from one hot spot to the next. This is called the tabbing order.

Hotspot Editor creates the tabbing order in the same order that you create the hot spots within the graphic.

To create a specific tabbing order

1. Define each hot spot in the same order you want users to follow in the built Help file.

Viewing the Tabbing Order

After you have created one or more hot spots, you can view the tabbing order by using the Select command on the Edit menu.

To view the tabbing order in a hypergraphic

1. From the Edit menu, choose Select.

   The Select dialog box appears, listing all the defined hot-spots in the hypergraphic. The hot spots are displayed in the order in which they were created, with the first hot spot at the top of the list box.

Changing the Tabbing Order

If you have already created hot spots in a hypergraphic, and later you want to change the tabbing order, you can use commands on the Edit menu to do so.

To change the tabbing order

1. From the Edit menu, choose Select.

   The Select dialog box appears with the hot spots listed in the current tabbing order, as shown in Figure 11.x.
2. Select the hot spot that you want to move to the top of the tabbing order.
3. Choose the Select button.

   The hot-spot rectangle displays eight sizing handles, indicating that it is the currently selected hot spot.
4. From the Edit menu, choose Delete.

   Or press the DEL key.

   The hot spot is deleted.
5. From the Edit menu, choose Undo.

   The hot spot is restored to the hypergraphic, and it is moved to the top of the tabbing order. To view the new tabbing order, use the Select command.
6. From the Edit menu, choose Select to view the new tabbing order.

   The selected hot spot is at the top of the tabbing order, as shown in Figure 11.1.

   Note If you did not assign unique identifiers to each hot spot, it may be impossible to tell that the tabbing order has changed.
7. Repeat steps 15 to move another hot spot.

Defining Hot-Spot Attributes

After creating a hot spot, you must define its attributes in the Attributes dialog box. Hot-spot attributes include binding information for the hot spot, the hot-spot name, and the bounding box coordinates.

The following table describes the hypergraphic attributes.

Field	Description
Context String	Specifies binding information for the hot spot, in the form of a context string or a macro.
Type	Indicates the type of action to be taken when the user chooses the hot spot. The four binding types are pop-up, jump, macro, and searchable.
Attribute	Specifies whether the hot spot is to be visible or invisible when displayed to the user in the Help window. Hot spots are always visible in Hotspot Editor.
Hotspot ID	Specifies a unique identifier for the hot spot. The hot-spot name is used internally by Hotspot Editor to make it easier to identify hot spots in the Select dialog box. Hotspot Editor assigns an incremental number to the hot-spot name; however, you can type your own unique name.
Bounding Box Coordinates	Displays the coordinates for the hot-spot rectangle: left, right, top, and bottom. The coordinates are measured in pixels and are restricted to the size of the graphic image.

To define attributes for a hot spot

1. Select the hot spot and choose Attributes from the Edit menu.
   Or click the hot spot with the right mouse button.
   Or double-click the hot spot with the left mouse button.
   Or select the hot spot and press ENTER.
   The Attributes dialog box appears (Figure 11.2).

   Note: If you use the right mouse button to define a hot-spot rectangle, Hotspot Editor displays the Attributes dialog box when you release the mouse button.

1. Complete the Attributes dialog box and choose OK.
   The attributes defined for the hot spot appear in the Hotspot Editor status bar.
   For more information about each attribute, see the preceding attributes table.
2. Repeat this process for each hot-spot region that you want to define.

To define a jump hot spot

1. From the Type list, choose Jump.
2. In the Context String box, type the context string of the topic Help jumps to when the user chooses the hot spot.
3. Choose OK.

To define a pop-up hot spot

1. From the Type list, choose Pop-up.
2. In the Context String box, type the context string of the topic that Help displays in the pop-up window when the user chooses the hot spot.
3. Choose OK.

To define a macro hot spot

1. From the Type list, choose Macro.
2. In the Context String box, type the macro (or macros) that Help executes when the user chooses the

hot spot.

For a Help macro, type the macro. (For more information, see Chapter 15, Help Macros.)

3. Choose OK.

To define a searchable hot spot

1. From the Type list, choose Searchable.
2. In the Context String box, type the keyword (or keywords) that you want to associate with this hypergraphic.

   Separate each keyword with a semicolon. (For more information, see Chapter 6, Creating Topics.)
3. Choose OK.

To define the appearance of the hot spot

1. Choose Visible or Invisible for the binding attribute.

   Visible hot spots are outlined in black or are displayed in inverse video.
2. Choose OK.

To change the default name assigned to the hot spot

1. In the Hotspot ID box, type the name you want to assign to the hot spot.

   This name helps you identify the hot spots in the graphic if you use the Select command (described in the next section). If you dont enter a hot-spot name, Hotspot Editor appends a number at the end of the name Hotspot.
2. Choose OK.

To change the size or location of the hot spot

1. Edit the bounding box values.

   The numbers used in the bounding boxes represent pixels.
2. Choose OK.

Setting Preferences

When you define a hot spot, Hotspot Editor uses the binding information and hot-spot identifier stored in the Preferences dialog box. To change the default values, edit them. Setting your own preferences for hot-spot attributes saves time and effort when creating hypergraphics that require settings different from Hotspot Editors internal defaults. Hotspot Editor assigns the attributes you enter to subsequent hot spots created with these preferences.

The following table shows the default setting for each attribute.

Field	Default entry
Context String	(Empty)
Type	Jump
Attribute	Invisible
Hotspot ID	Hotspot

To change Hotspot Editor preferences

1. From the Edit menu, choose Preferences.
2. In the Context String box, type a context string or macro.
3. Select the binding type.
4. Select the binding attribute.
5. Type a hot-spot name.

   This entry affects only the hot-spot name. Hotspot Editor continues to assign numbers incrementally to each new hot spot.
6. Choose OK.

Editing Hot Spots

After you create a hot spot, you can select it for editing and then cut, copy, paste, delete, resize, or move it. When editing hot spots, all attributes defined for the hot spot remain with the hot spot.

Selecting Hot Spots

A selected hot spot contains eight sizing handles that you can use to resize the hot-spot rectangle.

To select a hot spot

Click the hot spot with the left mouse button.

Or press TAB or SHIFT+TAB until you select the hot spot you want.

To select a hot spot using the Select command

1. From the Edit menu, choose Select.
   The Select dialog box appears. All hot spots defined for the hypergraphic appear in the Hotspots box. By default, Hotspot Editor does not select a hot spot in the list unless one was selected before choosing the Select command. In that case, Hotspot Editor highlights the currently selected hot spot.
2. In the Hotspots box, select a hot-spot name.
   Hotspot Editor displays the binding information for the selected hot-spot name.
3. Choose the Select button.
   The hot spot shows eight sizing handles to indicate that the hot spot is selected.

Deleting Hot Spots

To delete a hot spot

1. Select the hot spot.
2. From the Edit menu, choose Delete.
   Or press the DEL key.

Cutting or Copying Hot Spots

Hotspot Editor lets you transfer hot spots to and from the Clipboard using the Cut, Copy, and Paste commands on the Edit menu. Transferring hot spots is useful to:

- Create multiple versions of the same hot spot with the same attributes.
- Move a hot spot from one image file to another.
- Save time when repeatedly defining hot spots in a hypergraphic.

To cut or copy a hot spot to the Clipboard

1. Select the hot spot.
2. From the Edit menu, choose Cut or Copy.
   Hotspot Editor transfers the selected hot spot to the Clipboard using Hotspot Editors proprietary clipboard format.

Pasting Hot Spots

You can paste images or hot spots from the Clipboard into Hotspot Editor. Pasting an image is the same as opening the file. If you paste a hot spot, Hotspot Editor pastes it in the upper-left corner. You can paste a hot spot into the same image file or another image file.

To paste a hot spot

From the Edit menu, choose Paste.

The hot spot is pasted in the upper-left corner of the hypergraphic being edited.

Moving Hot Spots

After you create a hot spot, you can move it anywhere within the current image area.

To move a hot spot

1. Select the hot spot.
2. Position the mouse pointer in the center of the selected hot spot.
   The pointer changes to a hand, indicating that you can move the hot spot.
3. Drag the hot spot to the new location and release the mouse button when the hot spot is where you want it.
   The hot-spot sizing handles disappear while dragging and reappear when you release the mouse button.

   Note: If the Hotspot Editor window is larger than the graphic displayed in the image area, Hotspot Editor does not let you move the hot spot outside the image area. If the window is smaller than the image area (that is, if the window shows scroll bars), Hotspot Editor scrolls in the direction of the attempted move if a hot spot crosses the edge of the scroll bar.

Resizing Hot Spots

The eight sizing handles on a hot spot let you change its size the same way you change the size of a window.

To resize a hot spot

1. Select the hot spot.
2. Position the mouse pointer on the sizing handle on the border or corner that you want to change.
   The pointer changes to a two-headed arrow.
3. Drag the sizing handle until the rectangle is the size you want.
   If you drag a side handle, the rectangle changes size on only one side, the side of the border you drag. If you drag a corner, the two adjoining sides that form the corner change size at the same time.
4. Release the mouse button.

Saving the Hypergraphic

After you have defined all the hot spots and made all your changes to the image, you save it as a hypergraphic.

To save a hypergraphic

From the File menu, choose Save As to save the file with hot spots using the .SHG filename extension provided by Hotspot Editor.

Hotspot Editor saves your graphic with the appropriate hot-spot coding. You can now include this graphic in your Help file using a bmc, bml, or bmr reference.

Editing and Replacing Images

Sometimes you will create a hypergraphic and then discover that you need to change the original picture used in the hypergraphic. Since you cannot use Hotspot Editor to change the graphic, you must use the graphics application that created the original image.

However, after an image has been saved as a hypergraphic (.SHG file), it cannot be opened again in the original graphics program. Because Hotspot Editor does not alter the original bitmap or metafile when creating the hypergraphic, you can edit the original bitmap (.BMP) or metafile (.WMF) in the graphics program and then re-import it into Hotspot Editor after you have made your changes.

To edit or replace the bitmap in a hypergraphic

1. Open the original bitmap or metafile in the graphics application you used to create the image.
   Or create a new bitmap for the hypergraphic.
2. Make your changes to the image.
3. Save the edited image in the graphics application.
4. Copy the image to the Clipboard.
5. Start Hotspot Editor and open the hypergraphic that contains the graphic you want to replace.
6. From the Edit menu, choose Replace.
   The new or edited graphic replaces the original bitmap.
   Note If the new image is smaller than the original graphic included in the hypergraphic, Hotspot Editor may move the hot spots to ensure that they are still on top of the image.
7. Save the changes to the hypergraphic.

Preparing Hypergraphics for Different Displays

Normally, when it saves a bitmap in .SHG format, Hotspot Editor records the display resolution (EGA, VGA, or 8514) in the .SHG file based on the resoultion of the computer its running on and ignores the resolution information stored in the original bitmap. This can cause problems if you want to display your hypergraphics on different display resolutions. For example, if you have three bitmaps, created on EGA, VGA, and 8514 devices, and you edit them with Hotspot Editor on a VGA system, Hotspot Editor marks all the resulting .SHG files as VGA.

To correct this problem, you can either can save the hypergraphic with the correct resolution information, or you can compile these files with Helps Multi-Resolution Bitmap Compiler (MRBC). For information about MRBC, see Chapter 12, Creating Graphics for Different Displays.

Saving Resolution Information

When creating hypergraphics, you can save the resolution information of the target monitor (not just the monitor on which they are created). This is especially important if you want to create hypergraphics for the Macintosh, because you cannot use Hotspot Editor on a Macintosh computer. Also, if you use this feature, you do not have to compile hypergraphics with MRBC to make them appear correctly on different displays, including Macintosh displays.

To save resolution information in the hypergraphic, you must add a line to your SHED.INI file. The entry tells Hotspot Editor whether you want to save .SHG files with the resolution information of a monitor other than the one on which they were authored. The monitor type actually saved depends upon the first letter of the extension, just as when using MRBC.

To save resolution information, add the following line to your SHED.INI file:

```
[Hotspot Editor]
ResBasedOnExt=1
```

If the value of ResBasedOnExt equals 1, Hotspot Editor sets the appropriate resolution based on the first letter of the file extension. The following table shows how Hotspot Editor interprets the extensions.

First character	Example	Resolution
C	bitmap.cbm	CGA
E	bitmap.ebm	EGA
V	bitmap.vbm	VGA (default)
8	bitmap.8bm	8514
M	bitmap.mbm	Macintosh
other	bitmap.bmp	VGA

For example, if you create a hypergraphic to be displayed on a Macintosh, you can specify the MAC extension M and the hypergraphic will look good on a Macintosh display, without using MRBC.

Creating Hypergraphics Without Hotspot Editor

If you are using Windows Help version 3.0, or if you dont have Hotspot Editor, you can create hypergraphics in the Help file by carefully positioning several individual bitmaps next to each other and formatting each of them as hot spots.

This technique might be called the jigsaw puzzle method because it is similar to taking a whole picture and cutting it into pieces and then pasting it back together, and formatting the pieces that you want to be hot as bitmap hot spots. For example, you could take a screen shot of a menu that has been dropped down and make a separate bitmap of each command, as in Figure 11.3.

In the topic file, you place the bitmap references in separate paragraphs and format them so that there is no space between the paragraphs. Otherwise, the individual bitmaps will display with white space or gaps between them. Also, you should format all the bitmap paragraphs as Keep Together so that the individual bitmaps dont wrap when the user resizes the Help window. When you have the bitmaps positioned, format each bitmap reference as a hot spot, as in Figure 11.4.

The resulting graphics will look like a single hypergraphic when built into the Help file, as in Fugure 11.5.

If done carefully, users will not be able to tell the difference between this imitation hypergraphic and a true hypergraphic created with Hotspot Editor. However, using several bitmaps to simulate a single-image hypergraphic has the disadvantage of requiring Help to perform multiple locating operations when displaying the topic. Usually this means Help must read the disk several times, slowing overall performance, especially on CD-ROM. The decreased performance may or may not be significant, depending on the size and number of bitmaps being used.

Chapter 12 Creating Graphics for Different Displays

One important issue to consider when creating graphics images is how the pictures will appear on different video displays. Help can display graphics on machines with many different kinds of video hardware, including the most common display typesmonochrome, color, EGA, VGA, and 8514. When you prepare pictures for Help, you should analyze the target audience for the Help file and determine what percentage of that audience has monochrome or color displays, and what percentage has each type of video adapter so that you can create pictures that will look good on the end-users machine.

This chapter describes how to prepare pictures for displaying on different video resolutions.

How Help Displays Images

Because Windows Help is device independent, it stores text and graphics by point size rather than by their width and height in pixels. When displaying bitmaps, Windows Help tries to maintain a bitmaps logical size across all displays (EGA, VGA, or 8514, for example). To do that, Help determines whether a bitmap includes information about its resolution. If it does, Windows Help either displays the bitmap as authored or modifies it (by stretching or shrinking it) if the display resolution is different from the authored resolution. If no resolution is stored with the bitmap, Windows Help uses the default values for VGA.

Because of stretching or guessing, Help bitmaps sometimes do not appear the same as they were authoredunless the authoring monitor and the display monitor have the same resolution. For example, a bitmap created on a VGA display becomes distorted when displayed on an 8514 display. That is because Help uses the logical size (pixels per inch) of the display driver to determine how to size the picture. VGA displays have 96 pixels per inch, and 8514 displays have 120 pixels per inch. In both cases, Help keeps pictures and text proportionate. So, if you create all your bitmaps in VGA resolution, Help will stretch the bitmaps when displaying them on an 8514 monitor.

> Note: Help considers an 8514 display to be any monitor that has 120 pixels per inch. Because some high-resolution monitors deliver 96 pixels per inch to get more inches on the screen, Help uses the VGA bitmap on these devices.

Multi-Resolution Bitmap Compiler

To compensate for differences in the resolutions of authored and displayed bitmaps, you can use the Multi-Resolution Bitmap Compiler (MRBC). MRBC provides a way to prevent Help from stretching bitmaps inappropriately when displaying them on different resolutions.

MRBC is a utility program that lets you create bitmaps or metafiles with different resolutions (CGA, EGA, VGA, or 8514) and combine them into a single graphic that you can compile into a Help file. When Windows Help version 3.1 encounters a multiresolution bitmap or an .MRB file created by MRBC, it cycles through the stored resolutions and selects the bitmap that most closely matches your display.

If you want the pictures in your Help file to display correctly on various monitors, you must create multiple copies of the same bitmapa different bitmap for each monitor or which the Help file will be displayed. You cannot create just the VGA version, for example, and expect it to convert properly to lower resolution monitors. There may be tools that automate bitmap-stretching, but often the results are visually inferior.

> Note: You cannot edit or view the .MRB output file using any existing graphics program, and, therefore, it has no value outside the Windows Help environment.

Including .MRB Files in a Help File

To include .MRB files in a Help file, reference them in the RTF source file. The syntax for an .MRB bitmap reference is the same as for other bitmaps.

Bitmap type	Syntax in topic file
inline .MRB bitmap	
left-aligned .MRB bitmap	
right-aligned .MRB bitmap	

Using MRBC

Because MRBC is a bound application, you can run it with MS-DOS or OS/2Ò. MRBC supports CGA, EGA, VGA, and 8514 resolutions.

You can run MRBC from the command line in interactive or silent mode. Use interactive mode when you want or require input. Use silent mode for batch processing and automated builds.

MRBC has the following syntax:

MRBC [/s] name1.bmp [name2.bmp, ...namen.bmp]

Parameter	Description
/s	Specifies silent mode. If the /s switch is used, MRBC determines the resolution of each bitmap based on the first character of the bitmap extension. If the /s switch is not used, MRBC prompts the user for the resolution of each bitmap.
name1.bmp, name2..bmp, ...namen.bmp	Specifies the name(s) of the bitmap(s) to be included in the .MRB file.

Bitmap paths can be relative or absolute. Relative paths are relative to the users current directory.

The output file MRBC creates has the same filename as the first bitmap on the command line (name1.bmp) but with an .MRB extension. All output files are written into the current directory.

MRBC Preprocessing Checks

Before starting the conversion process, MRBC performs two preprocessing checks. First it checks the existence and format of each bitmap listed on the command line. If any bitmap listed is not found or is in an invalid format, MRBC stops processing and displays an error message. Then MRBC checks for the existence of the output file. If a name1.mrb file already exists, MRBC will overwrite it.

Starting the Conversion Process

If the command-line entry passes the two preprocessing checks without errors, the conversion process begins. If the /s switch is not used, MRBC prompts you to give the monitor type for each bitmap. In the prompt message, the bitmap names are displayed as typed on the command line, either with no path, with a relative path, or with an absolute path.

For example, if you type:

```
MRBC c:\bitmaps\vga\menu1.bmp \ega\menu2.bmp
```

and the preprocessing checks pass successfully, MRBC displays the following messages:

```
Please enter the monitor type for the bitmap c:\bitmaps\vga\menu1.bmp
```

and

```
Please enter the monitor type for the bitmap ..\ega\menu2.bmp
```

You supply the appropriate response (VGA and EGA in the above examples), and then MRBC creates the output file and writes it to the current directory.

The valid case-insensitive responses to the request for monitor type are CGA, EGA, VGA, and 8514. If you enter a monitor-type option that is not recognized, MRBC displays a warning message reminding you of valid entries.

The number of bitmaps specified on the command line is not limited; however, MRBC supports only the four monitor types mentioned above. At run time, Windows Help version 3.1 selects the best possible bitmap for your monitor type.

In addition to resolution, Windows Help uses color as a selection criterion. Therefore, it is possible, and recommended, to include a color version and a monochrome version of each bitmap in the same .MRB file to ensure compatibility with both monitor types.

> Note: MRBC does not check for valid matches between bitmap resolution and the entered option. For example, if you enter VGA for an EGA graphic, MRBC accepts it and marks it as a VGA bitmap. When the bitmap is displayed on a VGA monitor, however, it will not appear as authored. Check .MRB bitmaps on the target monitors to ensure that they display correctly.

If you use the /s switch, the conversion process occurs without your input. Instead, MRBC uses the first character of the input file extension to determine the bitmap resolution. The following table shows how the extensions are mapped.

First character	Example	MRBC interpretation
C	(bitmap.cbm)	CGA bitmap
E	(bitmap.ebm)	EGA bitmap
V	(bitmap.vbm)	VGA bitmap
8	(bitmap.8bm)	8514 bitmap
other	(bitmap.bmp)	VGA bitmap

For example, if you type:

```
MRBC /s house.vga house.ega house.cga house.8xx
```

MRBC creates a HOUSE.MRB file that consists of VGA, EGA, CGA, and 8514 bitmaps.

> Note: As in interactive mode, MRBC does not check for valid matches between bitmap resolution and the entered option. You must match the correct extensions with the correct display resolutions.

In both interactive and silent modes, MRBC accepts * and ? as wildcards for the filenames.

In some cases, the drawing tool that creates bitmaps stores the resolution in the bitmap file itself. MRBC

checks all bitmap files to determine whether the resolution is specified. If MRBC finds the resolution specified in the bitmap files, it uses the stored ratio instead of the one you type and displays an informative warning message. MRBC gives this warning message whether the conversion is performed in interactive or silent mode.

Compressing MRB Files

If you are not using the extended version of the Help compiler (HCP.EXE), you may run into memory problems when creating Help files with MRBC. For example, if the compiled Help file seems unusually large compared to the size of the MRB files included in the build, it may mean the compiler did not have enough memory to compress the MRB bitmaps.

For example, the following shows a case in which the total size of the individual bitmaps included in the MRB is larger than the MRB file itself.

Bitmaps

EGA 29258

VGA 35210

8514 48166

MRB 33494

Help file

without the MRB 4172

With the MRB 72883

The MRB is smaller than each of the bitmaps it contains because the bitmaps in the MRB are compressed. However, when the MRB is built into the Help file, the compiler did not have enough memory to compress it completely, so the size of the Help file grew by more than the size of the MRB. (During the build, the compiler decompresses the MRB file in order to check for SHG hot spots and then recompresses the MRB file).

If the COMPRESS option is set to Medium or High, the compiler displays a warning that there is not enough memory to compress the bitmap. When compression is Off, the compiler does not display this message, because it doesnt make sense. Therefore, you should turn compression on when building Help files with MRB file so that you can see the warning message. Then, if the compiler displays the warning that there is not enough memory to compress the bitmaps, you should either compile in OS/2 or use the extended version of the compiler (HCP.EXE).

MRBC Error Messages

MRBC version 1.1 displays the following errors and warnings. All preprocessing and run-time errors cause MRBC to stop the conversion process. Run-time warnings are for information only and do not affect the .MRB file.

Preprocessing Errors

Bitmap or directory %s is not found.

PROBLEM: MRBC cannot find the specified bitmap or directory.

RESULT: MRBC aborts the operation and does not create the bitmap file.

SOLUTION: Check the path and bitmap filename. Make any necessary corrections, and then rerun MRBC.

File %s is not a valid bitmap.

PROBLEM: MRBC found the file but does not recognize the bitmaps format.

RESULT: MRBC aborts the operation and does not create the bitmap file.

SOLUTION: Make sure that the bitmap file is saved correctly. Make any necessary corrections, and then rerun MRBC.

Run-time Errors

Out of memory.

PROBLEM: There is not enough memory to run MRBC.

RESULT: MRBC aborts the operation and does not create the bitmap file.

SOLUTION: Try compiling the Help file under OS/2. Or, free some memory or reduce the size of the bitmaps, and then rerun MRBC.

Out of disk space.

PROBLEM: There is not enough free disk space for MRBC to create the bitmap file.

RESULT: MRBC aborts the operation and does not create the bitmap file.

SOLUTION: Free some disk space, and then rerun MRBC.

Run-time Warnings

The supported monitor types are CGA, EGA, VGA, and 8514.

PROBLEM: The command line specifies an invalid monitor type. This message appears only in interactive mode.

RESULT: MRBC prompts you to enter the monitor type for the specified bitmap. If you enter a valid display type, MRBC continues processing. Otherwise, MRBC aborts the operation.

SOLUTION: Enter one of the supported display types when prompted.

Chapter 13 Customizing the Help File

When creating your Help file, you might want to add new capabilities or change the way Help works with your information. The following list shows some of the ways to customize Help. You can:

- Add new menus and buttons to Help.
- Assign keyboard equivalents to custom Help features.
- Run other applications from the Help file.
- Change the title that appears in the Help window.
- Create a custom icon for the Help file.
- Create a custom How To Use Help file.

This chapter explains how to customize your Help file.

Defining Menus and Buttons

Windows Help uses menus to organize commands and buttons to provide navigation controls for the Help file. To simplify the interface for your Help file, you might want to use menus and buttons to provide quick access to Help file features. Menus are used in almost all applications for Windows, and button bars or tool bars are commonly used as well. If your users have any experience with Windows, its likely they already understand how to use menus and button bars.

In addition to the standard menus and buttons, you can add custom menus, menu items, and buttons, which can run standard Help macros or external commands that you register with the Help file. You can design your Help file to change the menu and button bar when the user opens the Help file or displays certain topics during the Help session. Customizing the Help menu bar and button bar requires you to use macros.

This section introduces Help menu and button-bar customization and discusses how to use specific Help macros to make those changes. It assumes some familiarity with Help macro syntax and usage, even though this information is covered in a later chapter. Use the material in this section to get an idea of the kinds of changes you can make to Help menus and buttons. Then refer to Chapter 14, Help Macros, and Chapter 15, Help Macro Reference, for more information about using specific macros.

Menu and Button Macros

Help displays a menu bar and a button bar in its main window. The menu and button bars are not displayed in secondary windows. You can use the standard menu and button bars as is, or you can define new menus and buttons to add to the standard items. To define the menus, menu items, and buttons, you use Help macros. Your Help file can change the menus, menu items, and buttons at any time during a Help session.

You can reconfigure the menu or button bar at different times. For example, you might:

- Display a new menu when the user opens the Help file.
- Change the menus to reflect a certain type of information contained in a group of related topics.
- Change menu items to reflect options chosen by the user.

The menu and button bars are displayed only in the main window, and menu and button macros must be run from the main window. For example, a topic might have a macro that adds a button to the button bar. The button macro works only if the topic is displayed in the main window. It wont work if the topic is displayed in a secondary window.

Properties of Menu Items and Buttons

Menu items and buttons have similar characteristics. They both run macros when the user clicks them or presses a key combination, or mnemonic. Both buttons and menu items can be disabled (made unavailable) and enabled (made available) during the Help session.

Buttons and menu items share the following properties.

Property	Description
ID	Specifies an internal name for the button or menu item that identifies the button or menu item in Help macros; for example, to disable a button, you specify the button ID with the DisableButton macro.
Text	Specifies the text displayed on the button or menu item.
Mnemonic	Specifies a keyboard alternative for the button or menu item. The mnemonic is one of the characters in the button or menu text and is identified by an ampersand (&) placed before the mnemonic character.Menu-item mnemonics are available only when the menu is displayed, but button mnemonics are available whenever the input focus is on the main window.
Macro	Specifies the Help macro or macros to run when the user chooses the button or menu item.
Position	Specifies the position of the menu item within a menu, or the position of a button within the button bar. Position values are used only when a menu item or button is displayed. All further references to the menu item or button use the ID.

Suggested Uses for Menu Items and Buttons

Buttons are well-suited for displaying a limited number of frequently used functions. Buttons let the user access a feature using a single key combination or mouse click.

Menus are good for displaying larger collections of functions, especially when those functions can be organized into logical groups. For example, a menu bar might contain 25 menu items, but by dividing those items into six separate menus, the application can avoid overwhelming the user with too much information.

Help can place a check mark next to a menu item, but it cannot place a check mark on a button. This makes menu items more suitable for setting and displaying the status of user options with an on or off state.

Defining Menus

Help can display multiple drop-down menus containing several menu items. Space considerations might preclude using more than eight or nine menus, and on a 640 by 480 VGA screen, a drop-down menu can display about 20 menu items. However, if you only consider the physical limitations, you may introduce serious usability problems. For that reason, when designing custom menus take into account both the physical and usability factors.

Help provides a series of macros for configuring the menu bar. Using these macros, you can do the following:

- Add menus to the menu bar
- Add menu items to menus, at the end of the menu or in a specific position
- Remove menu items
- Change menu item macros
- Disable or enable a menu item
- Add and remove check marks from menu items
- Assign accelerator keys to menu functions

The following sections describe how to use Help macros to perform each of these tasks. For comprehensive information about using Help macros, see Chapter 14, Help Macros. For information about how to assign accelerator keys to menu and button functions, see Defining Accelerator Keys, later in this chapter.

> Note: Some menu macros dont work when run from a topic displayed in a secondary window. In those cases, you must run the macros from the Help project file or from a topic displayed in the main window.

Using the Standard Menus

Windows Help provides a standard menu bar for use with Help files; this menu bar displays this menu bar whenever you start Help. You must use the standard menus in your Help file. The standard menu bar is shown in figure 13.1.

You can access the standard menu functionality, as long as you use the specified menu IDs. The standard Help menu has the following menus, menu items, IDs, and macro assignments.

Menu / Item	ID	Macro	Description
File menu	mnu_file	None1	None1
File Open	mnu_open	FileOpen	Displays the Open dialog box.
File Print	mnu_print	Print	Prints the topic displayed in the main window.
File Print Setup	mnu_psetup	PrinterSetup	Displays the Print Setup dialog box.
File Exit	mnu_exit	Exit	Closes Help.
Edit	mnu_edit	None1	None1
Edit Copy	mnu_copy	CopyDialog	Displays the Copy dialog box.
Edit Annotate	mnu_annotate	Annotate	Displays the Annotate dialog box.
Bookmark	mnu_bookmark	None1	None1
Bookmark Define	mnu_bkdefine	BookmarkDefine	Displays the Bookmark Define dialog box.
Bookmark list	None2	None2	Lists the first nine bookmarks defined in the Help file. These items are displayed only if bookmarks are defined.
Bookmark More	mnu_bkmore	BookmarkMore	Displays the Bookmark dialog box for Help files that have more than nine bookmarks defined. This menu item is displayed only if ten or more bookmarks are defined.
Help	mnu_help	None1	None1
Help How To Use Help	mnu_helpon	HelpOn	Displays the How To Use Help file in a new Help window.
Help Always On Top	mnu_ontop	HelpOnTop	Displays all Help windows on top of other application windows.
Help About	mnu_about	About	Displays the About dialog box.

1 Macros cannot be run directly from the menu bar, only from menu items displayed on drop-down menus.

2 The bookmark list is controlled directly by Help and cannot be changed using macros.

Adding Menus

In Windows Help, menus dont run macros; they just display a list of menu items (or commands). Before defining menu items, you must define the menus. To add a menu to the menu bar, you use the InsertMenu macro. This macro has the following syntax:

```
InsertMenu("menu_id", "menu_text", menu_position)
```

For example, to insert a menu with the ID mnu_options and the text &Options (the ampersand turns the letter O into the acceleration key), at the third position on the menu bar, youd use the following macro:

```
InsertMenu("mnu_options", "&Options", 2)
```

The menu_position value starts with zero as the first menu, so the third menu has the value 2. When you insert a menu, all menus to the right of the insertion position are moved to the right. Because the Bookmark menu is in the third position before the insertion, the Options menu would now come after the Edit menu but before the Bookmark menu (Figure 13.2).

Adding and Removing Menu Items

Menu items are displayed on menus, which appear when the user selects a menu name on the menu bar. Menu items are associated with Help macros, so when the user chooses a menu item, the Help macro is run.

Adding Menu Items

Help provides two macros for adding menu items, AppendItem and InsertItem. As their names imply, the first macro adds a menu item to the end of a menu, and the second macro inserts a menu item at a specific place on the menu.

Inserting Items

To add a menu item in a specific position, you use the InsertItem macro, which has the following syntax:

```
InsertItem("menu_id", "item_id", "item_text", "item_macro", item_position)
```

For example, the following macro adds a Show Index Window menu item as the first item on an Options menu:

```
InsertItem("mnu_options","item_showindex","Show &Index Window...", "JI(`index.hlp', `idx_main')", 0)
```

By placing an ampersand (&) before the word Index of the menu item text, the macro assigns the mnemonic character I to the menu item. Also notice the use of the ellipses following the menu item text; this is a Windows convention that indicates the menu item displays a dialog box.

In the next example, a more complex macro is added to an Options menu:

```
InsertItem("mnu_options","item_showindex","Show &Index Window",
"IfThenElse(IsMark(`show_index'),
`DeleteMark(`show_index')',
`SaveMark(`show_index')')",0)
```

When the user chooses the resulting menu item, the macro saves or deletes a marker, depending on whether the marker is already set in the Help file.

> Note: You cannot insert menu items between or after the standard items on the Bookmark menu (with the menu ID mnu_bookmark). Help appends the standard items to any custom ones youve inserted.

Appending Items

The AppendItem macro uses the same parameters as the InsertItem macro, but it omits the position parameter. Menu items created using AppendItem are added to the end of the specified menu. The AppendItem macro can be easier to use in the Help project file because, rather than having to specify position values, you can just arrange the macros in the [CONFIG] section in the order you want the menu items to appear. To change the order of the menu items, you just change the order of the macros.

Removing Menu Items

To remove a menu item, use the following macro:

```
DeleteItem("item_id")
```

For example, to remove the Show Index Window menu item from the previous example, you use the following macro:

```
DeleteItem("item_showindex")
```

Disabling and Enabling Menu Items

When you disable a menu item, the menu text changes to gray, and the menu macro is made unavailable. To disable a menu item, use the following macro:

```
DisableItem("item_id")
```

For example, to disable the Options menu item, you use the following macro:

```
DisableItem("mnu_options")
```

The EnableItem macro activates a disabled macro. For example, to activate the Copy Bitmap menu item, you use the following macro:

```
EnableItem("mnu_copybmp")
```

Checking and Unchecking Menu Items

Many applications for Windows use check marks on menu items to indicate that an option is set. Help provides two macros, CheckItem and UncheckItem, that add and remove check marks from menu items. These macros have the following syntax:

```
CheckItem("item_id")
UncheckItem("item_id")
```

For example, to add a check mark next to an item with the ID item_showindex, you can use the following macro:

```
CheckItem("item_showindex")
```

In the example shown in Adding Menu Items, earlier in this chapter, a Help file uses a menu macro that sets or removes a mark. The menu item would be more effective if it displayed a check mark to indicate that the mark (and therefore the option) was set. By adding CheckItem and UncheckItem macros to the menu macro, the Help file will display a check mark at the appropriate times. The following example shows the resulting InsertItem macro:

```
InsertItem("mnu_options", "item_showindex", "Show &Index Window",
"IfThenElse(IsMark(`show_index'),
`DeleteMark(`show_index');UncheckItem(`item_showindex')',
`SaveMark(`show_index');CheckItem(`item_showindex')")",0)
```

When the user chooses the resulting menu item, the macro saves or deletes a marker, depending on whether the marker is already set in the Help file. It also displays a check mark to indicate the state of the option.

Changing the Menu Item's Function

Help lets you change the macro associated with a menu item. To change the macro for an item, use the following macro:

```
ChangeItemBinding("item_id", "new_macro")
```

For example, to change the macro assigned to a menu item with the ID item_time, you could use the following macro:

```
ChangeItemBinding("item_time", "ExecProgram(`clock', 0)")
```

In the example in the previous section, a Help file uses a check mark to indicate whether an option is set. In the following example, the InsertItem() macro is changed so the menu item changes its function when the user sets or clears the option:

```
InsertItem("mnu_options", "item_showindex", "Show &Index Window",
"IfThenElse(IsMark(`show_index'), `DeleteMark(`show_index');
ChangeItemBinding("item_showindex", "JumpContents(`index.hlp')")',
`SaveMark(`show_index');
ChangeItemBinding("item_showindex", "JumpID(`index.hlp', `sub_cont')")'), 0)
```

Defining Buttons

Help can display up to 22 buttons on the button bar. Using Help macros, you can customize the button bar in the following ways:

- Add buttons to the button bar
- Remove buttons
- Disable or enable a button
- Change the macro associated with a button

The following sections describe how to use Help macros to perform each of these tasks. For comprehensive information about Help macros, see Chapter 14, Help Macros, and Chapter 15, Help Macro Reference.

> Note: Some button macros dont work when run from a topic displayed in a secondary window. In those cases, you must run the macros from a topic displayed in the main window.

Using the Standard Buttons

Help provides six buttons for use with Help files. The standard buttons provide access to six commonly used Help macros. Four of the buttons are required, and two are optional buttons in your Help file. The standard button bar is shown in Figure 13.3.:

You can access the standard button functionality if you use the specified button IDs. The standard buttons have the following IDs, macros, and functions.

Button	ID	Macro	Description
Contents	btn_contents	Contents	Displays the Contents topic, which is the first topic in the Help file.
Index	btn_search	Search	Displays the Search dialog box.
Go Back	btn_back	Back	Jumps to the last topic the user displayed in the main window.
History	btn_history	History	Displays the History window.
<< (Browse Previous)	btn_previous	Prev	Jumps to the previous topic in the browse sequence.
>> (Browse Next)	btn_next	Next	Jumps to the next topic in the browse sequence.

How Help Disables Standard Buttons

In certain situations, Help automatically disables buttons with the following button IDs.

Button ID	Disabled when
btn_back	Using the Go Back button or the history list, the user returns to the first topic in the history list.
btn_previous	Help is displaying the first topic in the browse sequence.
btn_next	Help is displaying the last topic in the browse sequence.

This behavior provides useful feedback to the users. If you define custom Previous and Next buttons without using the button IDs listed in the preceding table, Help cannot disable the buttons in the situations described above. To define these buttons using custom paging devices, be sure to use the specified button IDs in your CreateButton or InsertButton macros.

Adding and Removing Buttons

Buttons are displayed on the button bar. When the user chooses a button, Help runs the macro associated with the button. You can add as many as 22 buttons to the button bar, although your users might find it difficult to use a button bar containing more than seven to nine buttons.

Help automatically sizes buttons to fit the text displayed on the button. You cant control the width of the buttons. You can have as many as 29 characters of text on a button.

Creating Buttons

Help provides the CreateButton macro for adding a new button to the button bar. The CreateButton macro has the following syntax:

```
CreateButton("button_id", "button_text", "macro")
```

For example, the following macro creates an Options button:

```
CreateButton("btn_options", "&Options", "JumpId(`current.hlp',
`option1_id')")
```

The button is added to the button bar after the standard buttons in the order that it is listed in the [CONFIG] section of the Help project file. For example, the Options button might be added before the Browse buttons and an Exit button:

```
[CONFIG]
CreateButton("btn_options", "&Options", "JumpId(`current.hlp',
`option1_id')")
BrowseButtons()
CreateButton("E&xit", "Exit()")
```

Figure 13.4 shows the resulting button bar:

Removing Buttons

To remove a button, use the following macro:

```
DestroyButton("button_id")
```

For example, to remove the Options button, you use the following macro:

```
DestroyButton("btn_options")
```

Disabling and Enabling Buttons

When you disable a button, the button text changes to gray and the button macro is made unavailable. To disable a button, use the following macro:

```
DisableButton("button_id")
```

For example, to disable the Options button, you use the following macro:

```
DisableButton("btn_options")
```

The EnableButton macro activates a disabled button. For example, to activate the Options button, you use the following macro:

```
EnableButton("btn_options")
```

Changing the Function of Buttons

Help lets you change the macro associated with a button. To change the macro assigned to a button, use the following macro:

```
ChangeButtonBinding("button_id", "macro")
```

For example, to change the Options button to jump to a topic with a context string of option2_id (instead of topic1_id), you can use the following macro:

```
CreateButton("btn_options", "&Options", "JumpId(`current.hlp',
`option2_id')")
```

Defining Accelerator Keys

You can assign keystrokes to macros used within your Help file. These accelerator keys provide keyboard equivalents to macros displayed on menus or the button bar. To define an accelerator key, use the following macro:

```
AddAccelerator(key, shift_state, "macro")
```

The key parameter specifies the numeric code for the key, and shift_state specifies the required state of the CTRL, ALT, and SHIFT keys. For information on these numeric codes, see Appendix A, Windows Virtual-Key Codes.

For example, you can define CTRL+C to be the Clock key. When the user presses CTRL+C, your Help file displays the Windows Clock. The following macro assigns CTRL+C as the accelerator key for the ExecProgram macro:

```
AddAccelerator(0x43, 2, "ExecProgram(`clock.exe', 0)")
```

To remove an accelerator key that you have assigned to a macro, use the RemoveAccelerator macro. To remove an accelerator key, use the following macro:

```
Remove Accelerator(key, shift_state)
```

The following macro removes CTRL+C as the accelerator key for the ExecProgram macro:

```
RemoveAccelerator(0x43, 2)
```

For more information about assigning accelerator keys, see Chapter 15, Help Macro reference..

Running Applications from Help

You can run a Windows-based application from a Help file using the ExecProgram macro. You might find this useful for the following reasons:

- If your Help file describes an application, you can run that application without forcing users to leave Help.
- You can use the Help file as a way to organize and execute various applications.
- You can incorporate applications to augment the information presented.

Once users quit the program, they return to Help and can continue reading information.

ExecProgram runs the specified program much the same as using the Run command from the File menu in Windows Program Manager. The .EXE and other necessary files for the application must either be on the PATH or be in the same directory as the .HPJ file for the Help file.

Usually, the ExecProgram macro is started when the user clicks a hot spot. But you can also include the macro as part of a hypergraphic or as the action taken when users choose a custom button or menu item.

The ExecProgram macro has the following syntax:

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

Command-line is the filename, enclosed in double quotation marks, of the application you want to run. Display-state is a number that specifies how the application window initially appears. Use 0 for normal size, 1 for minimized, and 2 for maximized.

In the following example, ExecProgram runs the Calendar application (in normal size) whenever a user clicks a CALENDAR.BMP bitmap:

```
!ExecProgram("calendar.exe", 0)
```

The application appears on top of the Help window, as shown in Figure 13.5.

> Note: For information about using the ExecProgram macro, see Chapter 14, Help Macros for more information about how to include macros in the Help file, see Chapter 15, Help Macro Reference. for information about using the ExecProgram macro.

Creating a Custom Window Title

By default, Help displays the words Windows Help as the window title. This title remains displayed as long as the Help file is open. To override this default and assign a custom title to the Help file, you use the TITLE option in the [OPTIONS] section of the Help project file.

The TITLE option can assign a new title with as many as 50 characters. The TITLE option uses the following syntax:

```
TITLE=title
```

Title is the title you want displayed in the Help title bar.

For example, the following entry in the Help project file tells Help to display Paintbrush as the title:

```
[OPTIONS]
TITLE=Paintbrush
```

This title appears in the Help window title bar, as shown in Figure 13.6.

For more information abuot project file options, see Chapter 16, The Help Project File.

Creating a Custom Icon for Your Help File

To display your own icon instead of the standard Help icon when users minimize the Help window with your Help file open, you can include the ICON option in the [OPTIONS] section of the Help project file:

```
ICON=icon-file
```

Icon-file is the name of an icon file you have created with the Microsoft Windows Icon Editor or a similar tool. It can be an absolute path or a path relative to the Help project directory.

For example, the following entry in the Help project file uses a custom icon:

```
[OPTIONS]
ICON=hyper.ico
```

This icon appears when the Help window is minimized, as shown in Figure 13.7.

For more information abuot project file options, see Chapter 16, The Help Project File.

Creating a Custom How To Use Help File

Windows Help provides a default Help file (called WINHELP.HLP) that explains to users how to use basic Help features. This default file, however, does not include any information about what is displayed in the Help window. In other words, it simply documents the Windows Help application. The default Help file is provided for your convenience so that you can include it with your product if you choose.

To provide users with more specific instructions about how to use your particular Help file, you can create a custom How To Use Help file. This is especially important if you customize Help in ways that make the default Help file inadequate. Or, if you are using the Windows Help application as a delivery medium for nonstandard Help files, you should certainly create a custom Help file that users can read while viewing your information.

Creating the File

Windows Help treats the How To Use Help file as just another Help file. The process for creating the instructional Help file is identical to the process for creating the main Help file: you create topic files, make links between topics, add graphics, create a project file, and build the Help file using the Help compiler.

You can use the same Help features in your instructional Help file that you can use in the main Help file. For example, you might:

- Display definitions of important terms in pop-up windows.
- Use hypergraphics to describe the information model youre using.
- Add your own custom buttons that let users access information in the file.

Users can take advantage of any features you add, as well as the standard Help features, when looking for information in the instructional Help file.

Source Files

As previously mentioned, Windows Help provides a basic Help file named WINHELP.HLP that you can customize for your own project. The following topic files and graphics, which are used to build the basic How To Use Help file are also included:

Filename	Description
WINHELP.BAS	Basic instructions to get users started using Help
WINHELP.BUT	Explanations of the Windows Help buttons
WINHELP.CMD	Explanation of the Windows Help menu commands
WINHELP.GLY	Definitions of terms used in the Help file
WINHELP.HOW	Step-by-step instructions for using Help features
WINHELP.IDX	Contents screen for the Help file
WINHELP.KBD	Keyboard equivalents used in Help
WINHELP.HPJ	Help project file used to build the Help file
BULLET.BMP	Bullet symbol used in bulleted lists
DOIT.BMP	Symbol used to indicate step-by-step instructions
HNDPOINT.BMP	Picture of the Help hot-spot cursor

You can customize these files in any way that you want, or you can create your own instructional file from scratch.

> Note: If you use these files, you should change the name of the Help project file used to build the Help file so that your custom Help file doesnt overwrite the default Help file when you install your product on a users machine. The standard Help file is used to display Help for all other applications for Windows. You should also install your custom Help file in the same directory as your main Help file.

Customizing the Help Menu

Users choose the How To Use Help command from the Help menu in the Windows Help application. To change the name of this command to something that more closely matches your instructional Help file, you can include the following Help macros in the [CONFIG] section of your Help project file:

```
[CONFIG]
   .
   .
   .
SetHelpOnFile("hlpbasic.hlp")
DeleteItem("mnu_HelpOn")
InsertItem("mnu_help", "mnu_hlpbas", "&Custom Help", "JC(`hlpbasic.hlp')",
0)
AddAccelerator(0x70, 0, "JC(`hlpbasic.hlp')")
```

These macros perform the following actions:

- The SetHelpOnFile macro sets the custom Help file as HLPBASIC.HLP for this Help file.
- The DeleteItem macro removes the default How To Use Help command from the Help menu.
- The InsertItem macro adds the custom Help file (identified by mnu_hlpbas) to the Help menu (mnu_help) as the first item, before the Always On Top command.
- The JumpContents (or JC) macro in the InsertItem macro opens the custom Help file, HLPBASIC.HLP, and jumps to the Contents topic for that file.
- The AddAccelerator macro sets up the F1 key (hexadecimal code 0x70) as the keyboard accelerator for the custom Help command on the Help menu. This accelerator key opens the custom Help file the same as if the user chooses the Help command.

For more information about any of the macros discussed in this example, see Chapter 15, Help Macro Reference.

Adding Additional Items to the Help Menu

A Help menu may also include additional items. For example, it might include items that describe the following Help topics:

- Tour of your Help file
- Quick Reference

You add additional menu items the same way as you added the main Help item: by using the InsertItem macro. Assuming the Help file is named HLPBASIC.HLP, the [CONFIG] section of HLPBASIC.HPJ might include the following macros:

```
[CONFIG]
   .
   .
   .
InsertItem("mnu_help", "mnu_tour", "Help &Tour", "JI(`hlpbasic.hlp>tour',
`tour_start')", 1)
AddAccelerator(27, 0, "CloseWindow(`tour')")
InsertItem("mnu_help", "mnu_qkref", "&Quick Reference", "JI(`hlpbasic.hlp',
`ref_idx')", 2)
AddAccelerator(0x52, 5, "JI(`hlpbasic.hlp', `ref_idx')")
```

These macros perform the following actions:

- Create menu items named Help Tour and Quick Reference that appear on the Help menu in the

second and third positions.

- Specify which topic to display when users choose the commands: a topic identified by the context string tour_start for the Help Tour and a topic identified by the context string ref_idx for the Quick Reference.
- Assign keyboard equivalents to each menu item: the ESC key to close the Help Tour window and ALT+SHIFT+R to access the Quick Reference.

Adding a Custom Help Icon

As previously mentioned, you can create a separate icon for your instructional Help file to distinguish it from the main Help file. To specify a custom icon, insert an ICON option in the [OPTIONS] section of the project file for your Help file, as in this example:

```
[OPTIONS]
    .
    .
    .
ICON=custom.ico
```

Chapter 14 Help Macros

Windows Help provides a complete set of functions necessary to display and navigate Help files or other hypertext documents. In addition to the standard functionality described in this guide, Windows Help also provides a set of custom commands, or macros, that let Help authors control and customize Help functionality.

This chapter describes the standard Help macros included with Windows Help and explains how to use them in your Help file. Each Help macro has a different purpose, and each can be used to improve the effectiveness and usability of your finished Help file. But the real power of Help macros emerges when you combine several macros to create unique (and sometimes amazing) features. The more creative the Help author, the more powerful the Help macros become. And if you find that the standard Help macro set falls short of your ultimate goal, you can also create your own Help macros using DLL functions to provide just the right effect for your Help file.

Before you read further, you might want to review the Macro Quick Reference section, later in this chapter, to see what macros are available to you.

What Are Help Macros?

A macro is a custom-made command that you can use in a Help file to change the way Windows Help works. A typical Help macro consists of a specific action that Help performs when the macro is executed. In general, you use Help macros to:

- Customize the Help menu bar by creating and modifying your own menus and menu items. You can also use macros to access the standard menu functions and dialog boxes.
- Customize the Help button bar by changing the function of the standard Help buttons or by creating and modifying your own custom Help buttons. You can also use macros to access any of the button-bar dialog boxes.
- Control the location and behavior of the main Help window or any secondary windows you create.
- Create jumps to specific topics in a Help file or display specific topics in pop-up windows.
- Save text markers at specific locations within the Help file, and then create conditional jumps to those locations.
- Assign a keyboard access (accelerator) key or key combination to a Help macro.
- Start other applications.
- Register a function within a dynamic-link library (DLL) and then use the function as a custom Help macro.

Executing Help Macros

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

- Placing macros in the Help project file so that Windows Help executes the macros whenever the user opens the Help file.
- Placing macros in a topic footnote so that the macros execute when the user displays the topic.
- Configuring the menu bar and button bar so that Help executes a macro when the user chooses the menu item or button.
- Adding hot spots to your topic that execute a macro when the user chooses the hot spot.
- Using an external application to send a function call to Windows Help that requests Help to execute a macro.

The following sections provide details about each of these methods.

> Note: Some Help macros dont work when run from a pop-up or secondary window. For example, macros that configure the menu bar or button bar are ignored when run from a topic displayed in a pop-up or secondary window. For each macro listed in Chapter 15, Help Macro Reference, the Comments section indicates the situation(s) in which the macro will not work. Before using any macro, you should review its description in Chapter 15.

Executing Macros when Opening a Help File

If a macro appears in the [CONFIG] section of the Help project file, Windows Help executes that macro when it first opens the Help file. If more than one macro is listed in the [CONFIG] section, Windows Help executes them in the order listed.

The following example shows two macros listed in the [CONFIG] section of a sample Help project file:

```
[CONFIG]
FocusWindow("index")
SetHelpOnFile("APPHELP.HLP")
```

These macros perform the following operations:

- FocusWindow makes a secondary window called index the active window.
- SetHelpOnFile replaces the standard How To Use Help file with a customized version.

Executing Macros from a Topic Footnote

If a Help macro is included in a topic footnote, Windows Help executes that macro when the user displays that topic. Users can execute a macro footnote in a topic by:

- Choosing a hot spot that jumps to that topic.
- Choosing the Back button or a browse button.
- Selecting a topic from the history list or keyword search list.
- Selecting a command or other feature in the application that is programmed to display the topic.

Macro footnotes are executed only when the user first displays the topic, not when the user completes any other action within the same topic.

You use an exclamation point (!) as the footnote character to execute topic-entry macros.

To insert a topic-entry macro

1. Position the insertion point at the beginning of the topic (to the right of the other footnotes).
2. From the Insert menu, choose Footnote.
   The Footnote dialog box appears.
3. Type an exclamation point as the custom footnote mark, and then choose OK.
   A superscript exclamation point (!) appears in the text window, and the insertion point moves to the footnote window.
4. Type the macro to the right of the exclamation point in the footnote window.
   Use only a single space between the exclamation point and the first word of the macro. For example, you might type the following:

   ```
   ! SaveMark("Creating Groups")
   ```

   Figure 14.1 shows this footnote window.

Note that you can include spaces in macros.

For more information about how to insert macro footnotes in your Help topics, see Chapter 6, Creating Topics.

Executing Macros from a Menu Item or Button

If a macro is defined for a menu item or button, Windows Help executes that macro when the user chooses the menu item or button. The macros can be defined in the [CONFIG] section of the Help project file or in a topic footnote. Macros that affect Help buttons, menus, or menu items remain in effect until the user displays a topic that changes the items function, opens a new Help file, or quits Windows Help.

The following macro executes when the user chooses a menu item:

```
AppendItem("mnu_util", "mnu_icon", "&Create Icon",
"ExecProgram(`imagedit.exe', 0)")
```

This macro performs the following operation:

- AppendItem adds an item to the Utilities menu that starts the ImageEdit application when the user chooses the menu item.

The following macro executes when the user chooses a button:

```
ChangeButtonBinding("btn_contents", "JumpID(`hgcd.hlp', `acc_idx_hg')")
```

This macro performs the following operation:

- ChangeButtonBinding changes the standard function of the Contents to jump to a specific topic within the Help file.

Executing Macros from a Hot Spot

Windows Help executes macro hot spots within a topic when the user chooses the hot spot containing the macro. The topic displays continuously while Windows Help executes the macro, unless the macro causes a jump to another topic.

Macro hot spots are formatted the same as jump hot spotsthe hot spot (text or bitmap reference) is formatted as double-underlined text and the macro string (preceded by an exclamation point) is formatted as hidden text.

To create a macro hot spot in a topic

1. Select the macro hot-spot text.
2. From the Format menu, choose Character.
3. Select the Double Underline check box, and then choose OK.
4. Position the insertion point immediately after the last letter in the double-underlined macro hot-spot text.
5. From the Format menu, choose Character again.
6. Clear the Double Underline check box, select the Hidden check box, and then choose OK.
7. Insert an exclamation point (!) as the first character of the macro string.
   Note The exclamation point must be formatted as hidden text.
8. Type the macro string that you want Help to execute when the user chooses this hot-spot text.
   Note You do not need to include a space between the exclamation point and macro string.
9. From the Format menu, choose Character again, clear the Hidden check box, and then choose OK.
   Figure 14.2 shows a correctly coded macro hot spot in a topic file.

For more information about how to include macro hot spots in your Help topics, see Chapter 8, Creating Links and Hot Spots.

Executing Macros in a WinHelp Function Call

An application can send a HELP_COMMAND parameter in the WinHelp function call that specifies a macro to execute. The WinHelp function uses the following C-language syntax:

```
BOOL WinHelp (hWnd, lpHelpFile[>WindowName], wCommand, dwData)
```

When sending a macro request, the dwData parameter should point to a null-terminated string that contains the macro. The macro string can have as many as 255 characters.

The following example uses a Help macro to specify the context string IDM_HELP_KEYBOARD for the Keyboard topic in MYHELP.HLP:

```
case IDM_HELP_KEYBOARD:
    WinHelp (hWnd, "myhelp.hlp", HELP_COMMAND,
           (LPSTR) "JumpID(\"myhelp.hlp\",\"IDM_HELP_KEYBOARD\")");
    return 0L;
```

For more information about the WinHelp function, see Chapter 19, The WinHelp API.

Constructing Help Macros

For those of you unfamiliar with programming languages or macro languages, the Windows Help macros may seem intimidating. The rules for constructing macros are technical and somewhat complex. The Help macros are designed to imitate standard C-language format. However, the standard Help macros do not support variables or expression evaluation.

Macro Guidelines

The following sections provide guidelines to follow when constructing Help macros. Read each section carefully and study the examples.

Macro Syntax

Help macro statements have two main components: the macro name and the macro parameters enclosed in parentheses. The most important rule to remember is that the macro name must be spelled exactly as it is given in the syntax and the parameters must be used in the order they are given in the syntax. Parameters provide information for the macro; for example, the JumpId macro, which executes a jump to a topic with a specific context string, has parameters for the name of the Help file and the topics context string.

All Help macros use the following format (or syntax):

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

The entire macro, including macro name, parentheses, and parameter list, can have a maximum of 512 characters. The opening parenthesis, closing parenthesis, quotation marks, and commas are required characters when included in the syntax statement for a macro.

Macro names are not case sensitive, so you can either use the capitalization shown in Chapter 15, Help Macro Reference, or you can adopt a different convention. For example, you can use any of the following forms:

IfThenElse

ifthenelse

IFTHENELSE

IFthenELSE

The parameter list consists of a series of parameters separated by commas. Parameters can be text strings or numbers. For example, the following macro creates a custom Help menu called Utilities:

```
InsertMenu("menu_util", "&Utilities", 3)
```

Some macros have no parameters, but the parentheses are still required. For example, the following macro displays the Search dialog box:

```
Search()
```

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

```
PlayAudio()
```

You can include more than one macro in a macro string by placing a semicolon (;) between each macro in the string. The Help compiler processes the macro strings as a unit and executes the macros sequentially. The following macro contains three different macro strings:

```
ChangeButtonBinding("btn_contents", "JumpID(`hgcd.hlp', `acc_idx_hg')");
EnableButton("btn_up"); ChangeButtonBinding("btn_up", "JumpID(`cdcd.hlp',
`hlpidx_idx_card')")
```

Using String Parameters

You must enclose all string parameters within quotation marks. Quotation marks can be either double quotation marks or matching single quotation marks, as follows:

"string parameter"

`string parameter'

Note: On US keyboards, the single opening quotation mark is different from the single

closing quotation mark. The single opening quotation mark (`) is paired with the tilde (~) above the TAB key on extended keyboards; the single closing quotation mark ('), or apostrophe, is paired with the double quotation mark.

For example, the JumpId macro takes two string parameters enclosed in double quotation marks:

```
JumpID("hgcd.hlp", "acc_idx_hg")
```

You can also enclose the string parameters in single quotation marks, as follows:

```
JumpID(`hgcd.hlp', `acc_idx_hg')
```

Using the single quotation marks eliminates ambiguities in situations where strings are nested within other strings (see the following section).

Nested Macros and Nested String Parameters

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

For example, the following macro creates a button called Time that uses the ExecProgram macro as a parameter. When the user chooses the button, Help starts the Microsoft Windows Clock application. Since the ExecProgram macro takes a string as its first parameter, the string is enclosed in single quotation marks:

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

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

You can also use single quotation marks for the outermost parameters. The following example produces the same results as the previous one:

```
CreateButton(`btn_time', `&Time', `ExecProgram(`clock', 0)')
```

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

Some Incorrect Examples

The following example is incorrect because the clock string is enclosed in double quotation marks, even though it is nested within another string delimited by double quotation marks:

```
CreateButton("btn_time", "&Time", "ExecProgram("clock", 0)")
```

The following example is also incorrect because the clock string is enclosed in single quotation marks that are not matched correctly (two closing quotation marks):

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

The ExecProgram(string will be interpreted as the third parameter, and the rest of the string will produce a syntax error.

Some Complex Examples

You can use single quotation marks at any level of nesting. For example, the following macro creates a menu item that, when chosen, creates a button that, when chosen, displays the Windows Clock:

```
AppendItem("mnu_fun", "mnu_fun_makebutton", "Display Clock &button",
"CreateButton(`btn_time', `&Time',`ExecProgram(`clock', 0)')")
```

Macro strings may not contain more than three other macro strings as parameters. The following macro shows the correct way to nest macros:

```
IfThen(1, `IfThen(1, `IfThen(IsMark(`Managing Memory'), `JumpId(`trb.hlp',
```

```
`man_mem')')')')
```

The following macro string is nested too deeply:

```
IfThen(1, `IfThen(1, `IfThen(1, `IfThen(1, `BrowseButtons()')')')')
```

> Note: The Help compiler displays an error message if macros are nested too deeply, but it passes the macro to the Help file. The Help application, however, does not display an error message if the macro string is nested too deeply. Therefore, both of the above macros would supposedly work, even though one generates an error message during compilation and one does not.

Using Special Characters in Strings

To use certain characters as values within a macro string, you must preface the character with a backslash. (Adding a backslash before a character is known as escaping the character.) Use the following guidelines when using the double quotation mark ("), single opening quotation mark (`), single closing quotation mark ('), and backslash (\) in macros:

- To use a backslash in a string parameter, you must type two backslashes.
- In the following example, the JumpContents macro string includes two backslashes for each backslash character to specify the \\ROOT\PROJECT\SUBDIR\MYHELP.HLP network path for a Help file:

  ```
  JumpContents("\\\\root\\project\\subdir\\myhelp.hlp")
  ```

- To use a double quotation mark within a string that is enclosed in double quotation marks, you must preface the double quotation mark with a backslash:

  ```
  "Chapter 8, \"Making Links Between Topics\""
  ```

- If you must use quotation marks as part of the parameter, you can enclose the entire parameter in single quotation marks and omit the backslash escape character required for the double quotation marks delimiting the string:

  ```
  ExecProgram(`command "string as parameter"', 0)
  ```

- To use a single quotation mark within a string that is enclosed in single quotation marks, you must preface the single quotation mark with a backslash:

  ```
  `This isn\'t easy'
  ```

- You dont have to use this form in a string delimited with double quotation marks. For example, you could use the following string in place of the previous example:

  ```
  "This isn't easy"
  ```

- You never have to escape commas (,) or parentheses () within a macro string.

Using Numeric Parameters

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

64

0x40

To specify a negative number, add a minus sign before the number. For example, the following numbers both specify decimal 18:

18

0x12

> Note: To accept a negative number, you must specify a numeric parameter as a signed number. If you use a negative number with an unsigned parameter, Help displays a Parameter type wrong error message.

Making Arbitrary DLL Calls in Help Macros

You can make arbitrary DLL calls only if you inform Help about the call by using the RegisterRoutine macro. For example, Microsoft Help Author defines a function called ClearRTFEditor with one valid parametera null string:

```
[CONFIG]
RegisterRoutine("hcparse.dll", "ClearRTFEditor", "")
```

For more information, see Using DLL Calls as Help Macros, later in this chapter.

Return Values in Help Macros

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

```
IfThenElse(IsMark("Managing Memory"),"JumpID(`trb.hlp',`man_mem')",
"JumpContents(`trb.hlp')")
```

The first parameter of the IfThenElse macro is a number. Since IsMark returns a number, it can be used as the first parameter. Help executes the IsMark and IfThenElse macros as follows:

1. Help executes the IsMark macro and obtains a numeric return value from it.
2. Help executes the IfThenElse macro. Help passes the number returned from IsMark to the first parameter and passes the JumpId and JumpContents macro strings to the second and third parameters.
3. If the number passed to the first parameter is not zero, the IfThenElse macro executes the JumpID macro; otherwise, it executes the JumpContents macro.

Note: In this example, the use of the IsMark macro differs from the use of the jump macros. Help does not use the return values of the JumpID and JumpContents macros in the IfThenElse macro. Since they are enclosed in quotation marks, Help treats them as simple strings, not as macros.

Note: You can also use DLL calls as conditions for the IfThen and IfThenElse macros.

Macro Error Checking

The Windows Help compiler checks the validity of each macro included in a build of the Help file during compilation. If the compiler finds an error, it gives an error message and continues the build. The compiler checks for the following:

- If the macro name is spelled correctly
- If the macro syntax is correctmatching parentheses, matching quotation marks, or missing commas
- If the macro has the correct number of parameters
- If the parameter type matches the specified typenumeric or string, for example
- If the prototype is valid for a RegisterRoutine macro

Although the Help compiler can check for these predictable errors, as the Help author you are responsible for verifying that the macros you use work correctly in the built Help file.

Using DLL Calls as Help Macros

If you want to add a certain feature to your Help file that Windows Help does not support, and you have programming experience for Windows (or access to someone who does), you can develop custom extensions to support the functions you need. Extending Help involves using the Windows software libraries, called dynamic-link libraries (DLLs). These libraries are commonly used in Windows applications, and most programmers for Windows are familiar with their construction and use. You can use calls to functions in the DLLs as Help macros by registering the functions in the Help project file.

This section provides a general discussion of these tools and describes the authoring tasks related to using DLLs in a Help file. As a Help author, you should understand the capabilities of these tools. To develop your Help file to its fullest potential, youll want to be fully informed about the opportunities they present.

Using DLLs in Help

Help provides the following mechanisms for calling DLL routines:

- You use the RegisterRoutine macro to identify a DLL routine to be called from the Help file as a Help macro.
- You can create an embedded window in a topic and use a DLL to control the objects displayed in the embedded window.

Many Windows-based programs use DLLs. A DLL is a library of programming routines that is automatically loaded when needed. DLLs are useful because they can use memory efficiently, and their routines can be called from multiple applications.

Figure 14.3 illustrates the two DLL interfaces.

In the example, EWDEMO.DLL contains routines to display a list of printers in the special embedded window. MMLIB.DLL contains a PlayAudio routine that is registered in the Help project file and is called using a macro hot spot.

The following sections describe the two DLL interfaces. For important information about writing DLLs to support external macros and embedded windows, see Chapter 20, Writing DLLs for Windows Help.

Registering DLL Routines

If you create a DLL for use with Windows Help, you can identify the functions within the DLL. The RegisterRoutine macro gives you the power to extend Windows Help using your own DLLs. After you register a DLL function with Windows Help, you can create menu items or macro buttons that execute the function. In this way, you can offer specialized capabilities that are not offered in the standard Windows Help application and make them available to users of your Help file.

When registering a DLL function, you provide Help the following information:

- DLL filename
- Function name
- Data type returned by the function
- Number and type of function parameters

To register the DLL function, you enter a RegisterRoutine macro in the [CONFIG] section of the Help project file. (If you dont register the DLL function in the [CONFIG] section, you must register it another way before using the function.) The RegisterRoutine macros are executed when the Help file is opened, so the registered functions are available during the entire Help session. You must register a DLL routine before using it, or the Help compiler will report an error when it encounters the unregistered macro in the RTF source files and the macro will not work when executed in the built Help file.

The RegisterRoutine macro has the following syntax:

```
RegisterRoutine("DLL-name", "function-name", "parameter-spec")
```

Parameter	Description
DLL-name	String specifying the name of the DLL in which the function resides. The filename must be enclosed in quotation marks. You can omit the .DLL filename extension.Specify the directory only if necessary. Generally, DLLs are installed in the directory where Windows Help resides. For more information, see the following section.
function-name	String specifying the name of the function to use as a Help macro. The function name must be enclosed in quotation marks.
parameter-spec	String specifying the formats of parameters passed to the function. Characters in the string represent C parameter types.

For complete information on the RegisterRoutine macro, see Chapter 15, Help Macro Reference.

How Help Locates .DLL and .EXE Files

When executing custom DLLs and applications using the RegisterRoutine and ExecProgram macros, Windows Help loads .DLL and .EXE files only when they are needed by the Help file. To load a DLL or application, Help must be able to find it on the users system. When preparing to use a .DLL or .EXE file, Help looks in the following locations:

1. Helps current directory
2. The MS-DOS current directory
3. The users Windows directory
4. The Windows SYSTEM directory
5. The directory containing WINHELP.EXE
6. The directories listed in the users PATH environment variable
7. The directories specified in WINHELP.INI

If Help cannot find the .DLL or .EXE file after searching in all these locations, it displays an error message.

To increase the likelihood that Help will locate the .DLL or .EXE file quickly, follow these guidelines:

- Use unique names for all .DLL and .EXE files accessed by the Help file.
- When installing your application on a users hard disk drive, your setup program should copy all custom DLLs and executable files to the directory where Help is located.
- If your product is distributed on CD-ROM, copy the WINHELP.EXE and custom .DLL and .EXE files to the users hard disk drive.
- Define a WINHELP.INI entry for each custom .DLL or .EXE file that your Help file is using so that Help knows where to locate them.

For an explanation of the WINHELP.INI file, see Creating Links Between Help Files in Chapter 8, Creating Links and Hot Spots.

Windows Help Internal Variables

Windows Help defines a series of internal variables that you can use with macros. After you register a DLL function as a Help macro, you can specify the Windows Help internal variables as parameters to that function when the function appears in hot spots or macro footnotes. These variables are always available, and their values change depending on the current state of the Help file. Many DLL routines, as well as some standard Help macros, require the information residing in these variables.

You can use any of the following Windows Help internal variables in DLL functions.

Variable	Format spec	Description
hwndApp	U	Number specifying the handle (a numeric identifier used by Windows) to the main Help window. This variable is guaranteed to be valid only while the DLL function is executing.
hwndContext	U	Number specifying the handle of the Help window (either the main Help window or a secondary window) that is active at the time the DLL is called.
qchPath	S	String specifying a fully qualified path of the currently open Help file.
qError	U	Long pointer to a structure containing information about the most recent Windows Help error.
lTopicNo	U	Number specifying the current topic number. This number is relative to the order of topics in the RTF files used to build the Help file. The current topic is the topic displayed in the Help window that is active when the DLL is called.
hfs	U	Number specifying the handle (a numeric identifier used by Windows) to the file system for the currently open Help file.
coForeground	U	Number specifying the RGB value of the foreground color of the window that is active when the DLL is called.
coBackground	U	Number specifying the RGB value of the background color of the window that is active when the DLL is called.

For more information about Windows Help internal variables in DLL calls, see Chapter 20, Writing DLLs for Windows Help.

Macro Quick Reference

The tables in this Quick Reference organize the Help macros according to function so that you can get a quick overview of the related macros you may want to use to achieve certain effects when customizing your Help file. Refer to Chapter 15, Help Macro Reference, for full details about each macro.

Button Macros

Use the following macros to access standard Help buttons, to create new buttons, or to modify button functionality.

Back	Displays the previous topic in the Back list.
BrowseButtons	Adds the Browse buttons to the Help button bar.
ChangeButtonBinding	Changes the assigned function of a Help button.
Contents	Displays the Contents topic of the current Help file.
CreateButton	Creates a new button and adds it to the button bar.
DestroyButton	Removes a button from the button bar.
DisableButton	Disables a button on the button bar.
EnableButton	Enables a disabled button.
History	Displays the history list.
Next	Displays the next topic in a browse sequence.
Prev	Displays the previous topic in a browse sequence.
Search	Displays the Search dialog box.
SetContents	Designates a specific topic as the Contents topic.

Menu Macros

Use the following macros to access standard Help menu items, to create new menus and menu items, or to modify menus and menu items.

About	Displays the About dialog box.
Annotate	Displays the Annotate dialog box.
AppendItem	Appends a menu item to the end of a custom menu.
BookmarkDefine	Displays the Bookmark Define dialog box.
BookmarkMore	Displays the Bookmark dialog box.
ChangeItemBinding	Changes the assigned function of a menu item.
CheckItem	Displays a check mark next to a menu item.
CopyDialog	Displays the Copy dialog box.
CopyTopic	Copies the current topic to the Clipboard.
DeleteItem	Removes a menu item from a menu.
DisableItem	Disables a menu item.
EnableItem	Enables a disabled menu item.
Exit	Exits the Windows Help application.
FileOpen	Displays the Open dialog box.
HelpOn	Displays the How To Use Help file.
InsertItem	Inserts a menu item at a given position on a menu.
InsertMenu	Adds a new menu to the Help menu bar.
Print	Sends the current topic to the printer.
PrinterSetup	Displays the Print Setup dialog box.
SetHelpOnFile	Specifies a custom How To Use Help file.
UncheckItem	Removes a check mark from a menu item.

Linking Macros

Use the following macros to create hypertext links to specific Help topics.

JumpContents	Jumps to the Contents topic of a specific Help file.
JumpContext	Jumps to the topic with a specific context number.
JumpHelpOn	Jumps to the Contents of the How To Use Help file.
JumpId	Jumps to the topic with a specific context string.
JumpKeyword	Jumps to the first topic containing a specified keyword.
PopupContext	Displays the topic with a specific context number in a pop-up window.
PopupId	Displays the topic with a specific context string in a pop-up window.

Window Macros

Use the following macros to control or modify the behavior of the main Help window or secondary Help windows.

CloseWindow	Closes the main or secondary Help window.
FocusWindow	Changes the focus to a specific Help window.
HelpOnTop	Places all Help windows on top of other windows.
PositionWindow	Sets the size and position of a Help window.

Keyboard Macros

Use the following macros to add keyboard access to a Help macro.

AddAccelerator Assigns an accelerator key to a Help macro.

RemoveAccelerator Removes an accelerator key from a Help macro.

Auxiliary Macros

Use the following macros to access applications and functionality not available in Windows Help.

ExecProgram	Starts an application.
RegisterRoutine	Registers a function within a DLL as a Help macro.

Text-Marker Macros

Use the following macros to create and manipulate text markers.

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 macros if a given marker exists.
IsMark	Tests whether a marker set by SaveMark exists.
Not	Reverses the result returned by IsMark.
SaveMark	Saves a marker for the current topic and Help file.

Chapter 15 Help Macro Reference

Windows Help provides a complete set of functions necessary to display and navigate Help files or other hypertext documents. In addition to the standard functionality described in this guide, Windows Help includes a set of 56 custom commands, or macros, that let Help authors control and customize Help functionality.

This chapter describes the standard Help macros that you can use to customize the way Help works with your Help file. For information on the rules for constructing and using macros, see Chapter 14, Help Macros.

Macro Reference

This section lists all macros in alphabetic order and contains complete information on each macro. Macro descriptions provide the following information.

Heading	Information
Syntax	Syntax for each macro. The table following the syntax describes the parameters that the macro requires. For information about the typographic conventions used in syntax descriptions, see the Document Conventions section in the Introduction to this guide.
Example	Example of the macro.
Comments	Notes about using the macro, including any restrictions.
See Also	Cross-references to other Help macros with similar functionality.

About

Displays the About dialog box. Executing this macro is the same as choosing the About command on the Help menu.

Syntax

About()

Parameter	Description
none	

AddAccelerator (or AA)

Assigns an accelerator (keyboard access) key or key combination to a Help macro so that the user can execute the macro simply by pressing the accelerator key(s).

Syntax

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

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

Parameter	Description
key	Windows virtual-key value of the accelerator key the user must press to execute the macro. For the list of virtual keys, see Appendix A, Windows Virtual-Key Codes.
macro	Help macro or macro string that executes when the user presses the accelerator key(s). The macro must be enclosed in quotation marks. Separate multiple macros in a string with semicolons (;).
shift-state	Number specifying the key or key combination to use with the accelerator key. Valid modifier keys are ALT, SHIFT, and CTRL.

Number	Modifier key(s)
0	(No modifier key)
1	SHIFT
2	CTRL
3	SHIFT+CTRL
4	ALT
5	ALT+SHIFT
6	ALT+CTRL
7	ALT+SHIFT+CTRL

Example

The following macro assigns a key combination to the JumpID macro so that the user can display an alphabetic index of Help topics by pressing ALT+CTRL+F10:

```
AddAccelerator(0x79, 6, "JumpID(`index.hlp', `cont_idx')")
```

Comments

The Help macro that AddAccelerator executes might not work in secondary windows, or its use may not be recommended if the macro it executes is prohibited or not recommended in secondary windows. Check the macros Comments section before using AddAccelerator to execute it in a secondary window.

See Also

RemoveAccelerator

Annotate

Displays the Annotate dialog box. Executing this macro is the same as choosing the Annotate command on the Edit menu.

Syntax

Annotate()

Parameter	Description
none	

Comments

If the Annotate macro executes from a pop-up window, the annotation is attached to the topic that contains the pop-up hot spot (parent topic) rather than to the topic displayed in the pop-up window.

AppendItem

Appends a menu item to the end of a menu created with the InsertMenu macro.

Syntax

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

Parameter	Description
menu-id	Name used in the InsertMenu macro to create the menu. This name must be enclosed in quotation marks. The new item is appended to this menu.
item-id	Name that Windows Help uses internally to identify the menu item. This name is case sensitive and must be enclosed in quotation marks.
item-name	Name that Windows Help displays on the menu for the item. This name is case sensitive and must be enclosed in quotation marks. Within the quotation marks, place an ampersand (&) before the character you want to use for the macros accelerator key.
macro	Help macro or macro string that executes when the user chooses the menu item. The macro must be enclosed in quotation marks. Separate multiple macros in a string with semicolons (;).

Example

The following macro appends a menu item labeled Example to a View menu identified by the mnu_view context string:

```
AppendItem("mnu_view", "mnu_example", "E&xample", "JI(`charts.hlp',
`eg_012_topic')")
```

Choosing the menu item causes a jump to a topic with the eg_012_topic context string in the CHARTS.HLP file. Note that the letter x serves as the accelerator key for this menu item.

Comments

Be sure that the accelerator key you assign to a menu item is unique. If you assign a key that conflicts with another menu access key, Windows Help displays an Unable to add item error message and ignores the macro.

Windows Help ignores this macro if it is executed in a secondary window.

See Also

ChangeItemBinding, CheckItem, DeleteItem, DisableItem, EnableItem, InsertItem, InsertMenu, UncheckItem

Back

Displays the previous topic in the Back list. (The Back list is an internal mechanism that tracks all the topics the user has displayed since starting Windows Help.)

Syntax

Back()

Parameter **Description**

none

Comments

If the Back macro is executed when the Back list is empty, Windows Help takes no action.

Windows Help ignores this macro if it is executed in a secondary window.

See Also

History

BookmarkDefine

Displays the Bookmark Define dialog box. Executing this macro is the same as choosing the Define command on the Bookmark menu.

Syntax

BookmarkDefine()

Parameter	Description
none	

Comments

If the BookmarkDefine macro executes from a pop-up window, the bookmark is attached to the topic that contains the pop-up hot spot (parent topic) rather than to the topic that is displayed in the pop-up window.

BookmarkMore

Displays the Bookmark dialog box. Executing this macro is the same as choosing the More command on the Bookmark menu.

> Note: The More command appears on the Bookmark menu if the user defines more than nine bookmarks.

Syntax

BookmarkMore()

Parameter **Description**

none

Comments

If this macro is executed in a secondary window, Help displays the bookmarked topic in the secondary window, regardless of where the topic appeared when the user set the bookmark. For that reason, using this macro in secondary windows is not recommended.

BrowseButtons

Adds browse buttons (<< and >>) to the button bar in Windows Help.

Syntax

BrowseButtons()

Parameter	Description
none	

Comments

If the BrowseButtons macro is used with one or more CreateButton macros in the [CONFIG] section of the Help project file, the left-to-right order of the browse buttons on the Windows Help button bar is determined by the top-down order of the BrowseButtons macro in relation to the other macros listed in the [CONFIG] section. For example, the following [CONFIG] section tells Help to add the Browse buttons between a Clock button and an Exit button:

```
[CONFIG]
CreateButton("btn_time", "&Clock", "ExecProgram(`clock', 0)")
BrowseButtons()
CreateButton("btn_close", "E&xit", "Exit()")
```

Depending on how its used, the BrowseButtons macro may interfere with the DisableButton macro. See DisableButton for details.

Windows Help ignores this macro if it is executed in a secondary window.

See Also

CreateButton, DisableButton

ChangeButtonBinding (or CBB)

Changes the assigned function of a button on the Windows Help button bar. You can change the function of the standard Help buttons or any button created with the CreateButton macro.

Syntax

ChangeButtonBinding("button-id", "button-macro")
CBB("button-id", "button-macro")

Parameter	Description
button-id	Identifier assigned to the button in the CreateButton macro, or one of the following standard Help button IDs:

Button ID	Button
btn_contents	Contents
btn_search	Search
btn_back	Back
btn_history	History
btn_previous	Browse previous (<<)
btn_next	Browse next (>>)

The button ID must be enclosed in quotation marks.

button-macro	Help macro that executes when the user chooses the button. The macro must be enclosed in quotation marks.

Example

The following macro changes the function of the Contents button so that choosing it causes a jump to the Table of Contents topic (identified by the dict_contents context string) in the DICT.HLP file:

```
ChangeButtonBinding("btn_contents", "JumpId(`dict.hlp', `dict_contents')")
```

Comments

Windows Help ignores this macro if it is executed in a secondary window.

See Also

CreateButton, DestroyButton, DisableButton, EnableButton

ChangeItemBinding (or CIB)

Changes the assigned function of a menu item added to a Windows Help menu with the AppendItem macro. This macro can also change the function of one (and only one) standard Help menu item: How To Use Help.

Syntax

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

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

Parameter	Description
item-id	Identifier assigned to the item in the AppendItem macro, or, for the standard How To Use Help menu item, use mnu_helpon as the identifier. The item ID must be enclosed in quotation marks.
item-macro	Help macro that executes when the user chooses the item. The macro must be enclosed in quotation marks.

Example

The following macro changes the menu item identified by time_item so that it starts the Clock application:

```
ChangeItemBinding("time_item", "ExecProgram(`clock', 0)")
```

The following macro changes the How To Use Help menu item so that it opens the Contents topic of a custom Help file:

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

Comments

Use the DeleteItem macro to remove the standard How To Use Help item from the Help menu. Use the SetHelpOnFile macro to specify the custom How To Use Help file you want to use. Then use the InsertItem macro to place the new menu item on the Help menu.

Windows Help ignores this macro if it is executed in a secondary window.

See Also

AppendItem, CheckItem, DeleteItem, DisableItem, EnableItem, InsertItem, InsertMenu, SetHelpOnFile, UncheckItem

CheckItem (or CI)

Displays a check mark next to a menu item added to a Windows Help menu with the AppendItem macro. The check mark indicates the state of a toggle-type command: on or off, show or hide, and so on.

Syntax

CheckItem("item-id")
CI("item-id")

Parameter	Description
item-id	Identifier assigned to the item in the AppendItem macro. The item ID must be enclosed in quotation marks.

Example

The following macro checks the menu item identified by time_item:

```
CheckItem("time_item")
```

Comments

To clear the check mark from the item, use the UncheckItem macro.

Windows Help ignores this macro if it is executed in a secondary window.

See Also

AppendItem, ChangeItemBinding, DeleteItem, DisableItem, EnableItem, InsertItem, InsertMenu, UncheckItem

CloseWindow

Closes either the main Help window or a secondary window.

Syntax

CloseWindow("window-name")

Parameter	Description
window-name	Name of the window to close. The name main is reserved for the primary Help window. For secondary windows, the window name is defined in the [WINDOWS] section of the Help project file. This name must be enclosed in quotation marks.

Example

The following macro closes the index secondary window:

```
CloseWindow("index")
```

Comments

If the window does not exist, Windows Help ignores the macro.

See Also

Exit

Contents

Displays the Contents topic of the Help file that executes the macro. The Contents topic is defined by the CONTENTS option in the [OPTIONS] section of the Help project file.

Syntax

Contents()

Parameter **Description**

none

Comments

If the Help project file does not have a CONTENTS option, Help displays the first topic in the first RTF file specified in the [FILES] section of the Help project file.

Windows Help ignores this macro if it is executed in a secondary window.

See Also

JumpContents, SetContents

CopyDialog

Displays the Copy dialog box and places the text from the current topic in the copy box where the user can select a portion to copy to the Clipboard. Executing this macro is the same as choosing the Copy command on the Edit menu.

Syntax

CopyDialog()

Parameter	Description
none	

Comments

If the CopyDialog macro executes from a pop-up window, the text from the topic that contains the macro hot spot (parent topic) is copied to the Copy dialog box rather than the text that is displayed in the pop-up window.

Using this macro in secondary windows is highly recommended because it is the only way that a user can copy the text displayed in a secondary window.

CopyTopic

Copies all the text in the currently displayed topic to the Clipboard. Executing this macro is the same as pressing CTRL+INS in the main Help window.

Syntax

CopyTopic()

Parameter	Description
none	

Comments

This macro does not copy bitmaps or any other images in the Help topic, only text.

If the CopyTopic macro executes from a pop-up window, the text from the topic that contains the macro hot spot (parent topic) is copied to the Clipboard rather than the text that is displayed in the pop-up window.

Using this macro in secondary windows is highly recommended because it is the only way that a user can copy the text displayed in a secondary window.

CreateButton (or CB)

Creates a new button and adds it to the Windows Help button bar.

Syntax

CreateButton("button-id", "name", "macro")

CB("button-id", "name", "macro")

Parameter	Description
button-id	Name that Windows Help uses internally to identify the button. This name must be enclosed in quotation marks.
name	Text that appears on the button. This name must be enclosed in quotation marks. Within the quotation marks, place an ampersand (&) before the character you want to use for the buttons accelerator key. The button name is case sensitive and can have as many as 29 charactersafter which Help ignores any additional characters.
macro	Help macro or macro string that executes when the user chooses the button. The macro must be enclosed in quotation marks. Separate multiple macros in a string with semicolons (;).

Example

The following macro creates a new button labeled Ideas that, when chosen, jumps to a topic with the directory context string in the IDEAS.HLP file:

```
CreateButton("btn_ideas", "&Ideas", "JumpId(`ideas.hlp', `directory')")
```

Notice that the letter I serves as the buttons accelerator key.

Comments

Windows Help allows a maximum of 16 author-defined buttons on the button bar, making a total of 22 buttons, including the Browse buttons. However, designing a button bar with more than seven to nine buttons may cause usability problems.

If more than one button is created with the CreateButton and BrowseButtons macros in the [CONFIG] section of the Help project file, the left-to-right order of the added buttons on the Windows Help button bar is determined by the top-down order of the macros listed in the [CONFIG] section. For example, the following [CONFIG] section adds a Clock button, then the Browse buttons, and then an Exit button to the right of the standard Help buttons:

```
[CONFIG]
CreateButton("btn_time", "&Clock", "ExecProgram(`clock', 0)")
BrowseButtons()
CreateButton("btn_close", "E&xit", "Exit()")
```

Windows Help ignores this macro if it is executed in a secondary window.

See Also

BrowseButtons, ChangeButtonBinding, DestroyButton, DisableButton, EnableButton

DeleteItem

Removes a menu item added with the AppendItem macro.

Syntax

DeleteItem("item-id")

Parameter	Description
item-id	Item identifier string used in the AppendItem macro. The item ID must be enclosed in quotation marks.

Example

The following macro removes the Example menu item that was created in the example for the AppendItem macro:

```
DeleteItem("mnu_example")
```

Comments

Windows Help ignores this macro if it is executed in a secondary window.

See Also

AppendItem, ChangeItemBinding, CheckItem, DisableItem, EnableItem, InsertItem, InsertMenu, UncheckItem

DeleteMark

Removes a text marker added with the SaveMark macro.

Syntax

DeleteMark("marker-text")

Parameter	Description
marker-text	Text marker previously added by the SaveMark macro. The marker text must be enclosed in quotation marks.

Example

The following macro removes the Managing Memory marker from a Help file:

```
DeleteMark("Managing Memory")
```

Comments

If the marker does not exist when the DeleteMark macro is executed, Windows Help displays a Topic not found error message.

See Also

GotoMark, IfThen, IfThenElse, IsMark, Not, SaveMark

DestroyButton

Removes a button added with the CreateButton macro.

Syntax

DestroyButton("button-id")

Parameter	Description
button-id	Identifier assigned to the button in the CreateButton macro. The button ID must be enclosed in quotation marks.

Example

The following macro removes the Ideas button that was created in the example for the CreateButton macro:

```
DestroyButton("btn_ideas")
```

Comments

You cannot use this macro to remove a standard Help button: Contents, Search, Back, or History. Therefore, the button identifier cannot be the same as an identifier used for the standard Help buttons. (The standard Help button identifiers are listed in the ChangeButtonBinding macro.)

Windows Help ignores this macro if it is executed in a secondary window.

See Also

ChangeButtonBinding, CreateButton, DisableButton, EnableButton

DisableButton (or DB)

Disables and greys out a button added with the CreateButton macro.

Syntax

DisableButton("button-id")
DB("button-id")

Parameter	Description
button-id	Identifier assigned to the button in the CreateButton macro. The button ID must be enclosed in quotation marks.

Example

The following macro disables the Ideas button that was created in the example for the CreateButton macro:

```
DisableButton("btn_ideas")
```

Comments

You cannot use this macro to disable a button in a topic until the button has been enabled using the EnableButton macro.

If you use this macro to disable a standard Help button (Contents, Search, Back, or History), the users next action may reactivate the button. For example, when the user displays a new topic, the History and Back buttons will become enabled. Also, each time the history list is updated, Help refreshes the History button. The Contents and Search buttons remain disabled until the user chooses an interfile jump or executes an EnableButton macro.

When the BrowseButtons macro is used with one or more DisableButton macros, it may interfere with the results of the DisableButton macro. When it follows the DisableButton macros, the BrowseButtons macro forces the standard buttons to refresh, creating the same effect as if the DisableButton macro had failed. The order in the following example causes the standard buttons to be enabled rather than disabled:

```
[CONFIG]
DisableButton("btn_contents")
DisableButton("btn_search")
DisableButton("btn_back")
DisableButton("btn_history")
BrowseButtons()
```

To ensure that the DisableButton macro works as you intend it to, place the BrowseButtons macro first in the order:

```
[CONFIG]
BrowseButtons()
DisableButton("btn_contents")
DisableButton("btn_search")
DisableButton("btn_back")
DisableButton("btn_history")
```

You can also disable the Search button in a Help file by not assigning any keywords to the topics.

Windows Help ignores this macro if it is executed in a secondary window.

See Also

BrowseButtons, ChangeButtonBinding, CreateButton, DestroyButton, EnableButton

DisableItem (or DI)

Disables and greys out a menu item added with the AppendItem macro.

Syntax

DisableItem("item-id")
DI("item-id")

Parameter	Description
item-id	Identifier assigned to the menu item in the AppendItem macro. The item ID must be enclosed in quotation marks.

Example

The following macro disables the Example marker that was created in the AppendItem macro example:

```
DisableItem("mnu_example")
```

Comments

You cannot use this macro to disable a menu item in a topic until the item has been enabled using the EnableItem macro.

Windows Help ignores this macro if it is executed in a secondary window.

See Also

AppendItem, ChangeItemBinding, CheckItem, DeleteItem, EnableItem, InsertItem, InsertMenu, UncheckItem

EnableButton (or EB)

Re-enables a button disabled with the DisableButton macro.

Syntax

EnableButton("button-id")
EB("button-id")

Parameter	Description
button-id	Identifier assigned to the button in the CreateButton macro. The button ID must be enclosed in quotation marks.

Example

The following macro re-enables the Ideas button that was disabled in the DisableButton macro example:

```
EnableButton("btn_ideas")
```

Comments

If you use this macro to enable a standard Windows Help button (Contents, Search, Back, or History), the users next action may disable the button. For example, if you enable the Contents button in one topic, it may be disabled again when the user jumps to a different topic.

Windows Help ignores this macro if it is executed in a secondary window.

See Also

ChangeButtonBinding, CreateButton, DestroyButton, DisableButton

EnableItem (or EI)

Re-enables a menu item disabled with the DisableItem macro.

Syntax

EnableItem("item-id")
EI("item-id")

Parameter	Description
item-id	Identifier assigned to the menu item in the AppendItem macro. The item ID must be enclosed in quotation marks.

Example

The following macro enables the Example menu item that was disabled in the DisableItem macro example:

```
EnableItem("mnu_example")
```

Comments

Windows Help ignores this macro if it is executed in a secondary window.

See Also

AppendItem, ChangeItemBinding, CheckItem, DeleteItem, DisableItem, InsertItem, InsertMenu, UncheckItem

ExecProgram (or EP)

Starts an application.

Syntax

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

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

Parameter	Description
command-line	Command line for the application to be started. The command line must be enclosed in quotation marks.
display-state	Specifies a value indicating how the application is shown when started.

Value	Display
0	Normal
1	Minimized
2	Maximized

Example

The following macro runs the Clock program in its normal window size:

```
ExecProgram("clock.exe", 0)
```

Comments

When using this macro to start an application, Windows Help searches for the application in this order: the current directory, the Windows directory, the Windows SYSTEM directory, the users path, and then the directory of the currently displayed Help file.

The ExecProgram macro does not change the directory before starting an application, so if you need to set the working directory for the application so that Help can find the correct files, you must create your own custom DLL.

If your application includes an online tutorial, you can use the ExecProgram macro to create hot spots within the Help file that link to tutorial lessons. For example, the following macro starts the TUTOR.EXE tutorial application (located in the LESSONS subdirectory) and displays the Toolbox lesson in a maximized window:

```
ExecProgram("c:\\lessons\\tutor.exe toolbox.cbt", 2)
```

The ExecProgram macro converts display-state values into the appropriate ShowWindow values. However, some applications may ignore the display-state you specify (because they ignore the nCmdShow parameter). For example, some tutorial applications may operate only in a maximized window; therefore, the value you specify in the display-state parameter may or may not work. To test whether an application ignores these values, hold down the SHIFT key while double-clicking the applications icon in Program Manager. If the application starts minimized, it will probably accept the value you specify. If the application starts in any other state, it will ignore the display-state value.

If you must use quotation marks as part of the command-line parameter, you can enclose the entire parameter in single quotation marks and omit the backslash escape character required for the double quotation marks delimiting the string, as shown in this example:

```
ExecProgram(`command "string as parameter"', 0)
```

Exit

Exits the Windows Help application. Executing this macro is the same as choosing the Exit command on the File menu.

Syntax

Exit()

Parameter **Description**

none

Comments

Executing this macro will close any secondary windows associated with the open Help file.

See Also

CloseWindow

FileOpen

Displays the Open dialog box. Executing this macro is the same as choosing the Open command on the File menu.

Syntax

FileOpen()

Parameter **Description**

none

Comments

Using this macro in secondary windows is not recommended because the Help file may be displayed in the secondary window, leaving the user without Help menus and navigation buttons. (Help authors can ensure that their Help file is opened in the main window, but there is no guarantee that all Help files have done this.)

FocusWindow

Changes the focus to the specified window, either the main Help window or a secondary window.

Syntax

FocusWindow("window-name")

Parameter	Description
window-name	Name of the window to receive the focus. The name main is reserved for the primary Help window. Secondary window names are defined in the [WINDOWS] section of the Help project file. This name must be enclosed in quotation marks.

Example

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

```
FocusWindow("keys")
```

Comments

If the window does not exist, Windows Help ignores the macro.

See Also

CloseWindow, PositionWindow

GotoMark

Jumps to a marker set with the SaveMark macro.

Syntax

GotoMark("marker-text")

Parameter	Description
marker-text	Text marker previously defined by the SaveMark macro. The marker text must be enclosed in quotation marks.

Example

The following macro jumps to the Managing Memory marker:

```
GotoMark("Managing Memory")
```

Comments

If the GotoMark macro specifies a marker that has not been previously defined by the SaveMark macro, Windows Help displays a Topic not found error message.

See Also

DeleteMark, IfThen, IfThenElse, IsMark, Not, SaveMark

HelpOn

Displays the How To Use Help file for the Windows Help application. Executing this macro is the same as choosing the How To Use Help command on the Help menu.

Syntax

HelpOn()

Parameter	Description
none	

Comments

If you replace the default How To Use Help file (WINHELP.HLP) with a custom version using the SetHelpOnFile macro, executing this macro will display the custom version of How To Use Help.

See Also

SetHelpOnFile

HelpOnTop

Changes the state of all Help windows to on-top. An on-top window appears visually on top of other application windows, except certain windows that may also use the topmost window attribute, such as the Windows Task Manager. Executing this macro is the same as choosing the Always On Top command on the Help menu.

Syntax

HelpOnTop()

Parameter	Description
none	

Comments

Microsoft does not recommend executing this macro in the main Help window. Instead use the on-top attribute when defining secondary windows. For complete information about creating secondary windows with the on-top attribute, see Chapter 9, Defining Topic Windows.

Windows Help does not provide a macro to check the current state of the Always On Top command and place a check mark next to the command when the state has changed to on-top. Therefore, it is up to Help authors to decide whether they want to use a macro to change the state of the command on the menu.

History

Displays the history list, which shows the last 40 topics the user has viewed since opening a Help file. Executing this macro is the same as choosing the History button.

Syntax

History()

Parameter **Description**

none

Comments

Windows Help ignores this macro if it is executed in a secondary window.

See Also

Back

IfThen

Executes a Help macro if a given marker exists. It uses the IsMark macro to make the test. You can also use a DLL function as a condition for this macro.

Syntax

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

Parameter	Description
marker-text	Text marker previously created by the SaveMark macro. The IsMark macro tests the marker you specify. If the marker value that the test returns is zero, the macro does not execute. If the value is something other than zero, the macro executes. The marker text must be enclosed in quotation marks.
macro	Help macro or macro string that executes if the marker exists. The macro must be enclosed in quotation marks. Separate multiple macros in a string with semicolons (;).

Example

The following macro jumps to the topic with the man_mem context string if the SaveMark macro has set a marker named Managing Memory:

```
IfThen(IsMark("Managing Memory"), "JI(`trb.hlp', `man_mem')")
```

Comments

You can use the IfThen macro to create many custom effects in your Help file. For example, you can use it to add a button to some topics and not have it appear in other topics. In the topic(s) where you want the button to appear, you create a macro footnote with the following sample macro string:

```
IfThen(Not(IsMark(`B')), "SaveMark(`B') : CreateButton(`misc_btn', `&Test',
`JumpContents(`pmcd.hlp')')")
```

In the topics where you dont want the button to appear, you create a macro footnote with this macro string:

```
IfThen(IsMark(`B'), "DeleteMark(`B') : DestroyButton(`misc_btn')")
```

If a topic does not have this footnote, it will have the same button characteristics as the previously viewed topic.

See Also

DeleteMark, GotoMark, IfThenElse, IsMark, Not, SaveMark

IfThenElse

Executes one of two Help macros depending on whether a marker exists. It uses the IsMark macro to make the test. You can also use a DLL function as a condition for this macro.

Syntax

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

Parameter	Description
marker-text	Text marker previously created by the SaveMark macro. The IsMark macro tests the marker you specify. The marker text must be enclosed in quotation marks.
macro1	Windows Help executes macro1 if the test returns a nonzero marker value. This macro must be enclosed in quotation marks. Separate multiple macros in the string with semicolons (;).
macro2	Windows Help executes macro2 if the test returns a marker value of zero. This macro must be enclosed in quotation marks. Separate multiple macros in the string with semicolons (;).

Example

The following macro jumps to the topic with the man_mem context string if the SaveMark macro has set a marker named Managing Memory. If the marker does not exist, the macro jumps to the Contents topic in the TRB.HLP file:

```
IfThenElse(IsMark("Managing Memory"), "JumpID(`trb.hlp', `man_mem')",
"JumpContents(`trb.hlp')")
```

See Also

DeleteMark, GotoMark, IfThen, IsMark, Not, SaveMark

InsertItem

Inserts a menu item at a given position on an existing menu. The menu can be one you create with the InsertMenu macro or a standard Windows Help menu.

Syntax

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

Parameter	Description
menu-id	Name used in the InsertMenu macro to create the menu or the name of a standard Windows Help menu. Standard menu names and identifiers are: **Name** IdentifierFile mnu_fileEdit mnu_editBookmark mnu_bookmarkHelp mnu_helpon The menu ID must be enclosed in quotation marks.
item-id	Name that Windows Help uses internally to identify the menu item. The item ID must be enclosed in quotation marks.
item-name	Name that Windows Help displays on the menu for the item. This name is case sensitive and must be enclosed in quotation marks. Within the quotation marks, place an ampersand (&) before the character you want to use for the items accelerator key.
macro	Help macro or macro string that executes when the user chooses the menu item. The macro must be enclosed in quotation marks. Separate multiple macros in a string with semicolons (;).
position	Number specifying the position in the menu where the new item will appear. The number must be an integer. Position 0 is the first or topmost position in the menu.

Example

The following macro inserts a menu item labeled Example as the fourth item on a View menu that has a mnu_view identifier:

```
InsertItem("mnu_view", "mnu_example", "E&xample", "JI(`charts.hlp',
`eg_012_topic')", 3)
```

Choosing the menu item causes a jump to a topic with the eg_012_topic context string in the CHARTS.HLP file. Note that the letter x serves as the items accelerator key.

Comments

Be sure that the access key you assign to a menu item is unique. If you assign a key that conflicts with another menu accelerator key, Windows Help displays the Unable to add item error message and ignores the macro.

The item-id parameter defined in this macro can be used in other menu item macros to modify the menu item, such as disable it or change its macro function.

Windows Help ignores this macro if it is executed in a secondary window.

See Also

AppendItem, ChangeItemBinding, CheckItem, DeleteItem, DisableItem, EnableItem, InsertMenu, UncheckItem

InsertMenu

Adds a new menu to the Windows Help menu bar.

Syntax

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

Parameter	Description
menu-id	Name that Windows Help uses internally to identify the menu. The menu ID must be enclosed in quotation marks. Use this identifier in the AppendItem macro to add menu items (commands) to the menu.
menu-name	Name for the menu that Windows Help displays on the menu bar. This name is case sensitive and must be enclosed in quotation marks. Within the quotation marks, place an ampersand (&) before the character you want to use for the menus accelerator key.
menu-position	Number specifying the position on the menu bar that the new menu name will have. This number must be an integer. Positions are numbered from left to right, with position 0 being the leftmost menu.

Example

The following macro adds a menu named Utilities to Windows Help:

```
InsertMenu("menu_util", "&Utilities", 3)
```

Utilities appears as the fourth menu on the Windows Help menu bar, between the Bookmark and Help menus. The user presses ALT+U to open the menu.

Comments

When adding menus, remember that The Windows Interface: An Application Design Guide requires that the File and Edit menus be the first two menus and that the Help menu be the last menu on the menu bar. Therefore, add new menus between the Edit menu and the Help menu.

Be sure that the accelerator key you assign to a menu is unique. If you assign a key that conflicts with another menu accelerator key, Windows Help displays an Unable to add menu error message and ignores the macro.

Windows Help ignores this macro if it is executed in a secondary window.

See Also

AppendItem, ChangeItemBinding, CheckItem, DeleteItem, DisableItem, EnableItem, InsertItem, UncheckItem

IsMark

Tests whether a marker set by the SaveMark macro exists. Use this macro as a parameter to the conditional macros IfThen and IfThenElse. The IsMark macro returns nonzero if the marker exists or zero if it does not.

Syntax

IsMark("marker-text")

Parameter	Description
marker-text	Marker text tested by the IsMark macro. The marker text must be enclosed in quotation marks.

Example

The following macro jumps to the topic with the man_mem context string if the SaveMark macro has set a marker named Managing Memory. The IsMark macro tests for the Managing Memory marker:

```
IfThen(IsMark("Managing Memory"), "JI(`trb.hlp', `man_mem')")
```

Comments

Use the Not macro to reverse the results of the IsMark macro.

See Also

DeleteMark, GotoMark, IfThen, IfThenElse, Not, SaveMark

JumpContents

Executes a jump to the Contents topic of a specified Help file. The Contents topic is defined by the CONTENTS option in the [OPTIONS] section of the Help project file.

Syntax

JumpContents("filename")

Parameter	Description
filename	Name of the destination Help file for the jump. The filename must be enclosed in quotation marks.

Example

The following macro jumps to the Contents topic of the PROGMAN.HLP file:

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

Comments

If the Help project file does not have a CONTENTS option, Help displays the first topic in the first RTF file specified in the [FILES] section of the Help project file.

If Windows Help cannot find the specified Help file, it displays an error message and does not perform the jump.

Windows Help ignores this macro if it is executed in a secondary window.

See Also

Contents, SetContents

JumpContext (or JC)

Executes a jump to a specific context within a Help file. The context is identified by an entry in the [MAP] section of the Help project file.

Syntax

JumpContext("filename", context-number)

JC("filename", context-number)

Parameter	Description
filename	Name of the destination Help file for the jump. The filename must be enclosed in quotation marks.
context-number	Context number of the topic in the destination Help file. The context number must be defined in the [MAP] section of the destination Help files project file.

Example

The following macro jumps to the topic mapped to the 801 context ID number in the PIFEDIT.HLP file:

```
JumpContext("pifedit.hlp", 801)
```

Comments

Use regular context jumps (double underlining, context string in hidden text) for jumps within the Help file rather than the JumpContext macro.

If Windows Help cannot find the specified Help file, it displays an error message and does not perform the jump.

If the context number does not exist or cannot be found in the [MAP] section, Windows Help jumps to the Contents topic or the first topic in the Help file and displays an error message. (For more information about context numbers, see [MAP] Section in Chapter 16, The Help Project File.)

See Also

JumpId, PopUpContext

JumpHelpOn

Executes a jump to the Contents topic of the How To Use Help file, which is either WINHELP.HLP (provided with Windows Help version 3.1) or the Help file designated by the SetHelpOnFile macro in the [CONFIG] section of the Help project file.

Syntax

JumpHelpOn()

Parameter	Description
none	

Comments

If Windows Help cannot find the specified Help file, it displays an error message and does not perform the jump.

See Also

HelpOn, SetHelpOnFile

JumpId (or JI)

Executes a jump to the topic with the specified context string in the specified Help file. Unlike the JumpContext macro, this macro does not require the topic to be defined in the [MAP] section because it takes the topic context string (as defined in the # footnote) as the second parameter.

Syntax

JumpId("filename", "context-string")
JI("filename", "context-string")

Parameter	Description
filename	Name of the Help file containing the context string. The filename must be enclosed in quotation marks.
context-string	Context string of the topic in the destination Help file. The context string must be enclosed in quotation marks.

Example

The following macro jumps to a topic with the groups_how_pm context string in the PROGMAN.HLP file:

```
JumpId("progman.hlp", "groups_how_pm")
```

Comments

If Windows Help cannot find the specified Help file, it displays an error message and does not perform the jump.

If the context string does not exist, Windows Help jumps to the Contents topic for that Help file. (For more information about context strings, see Inserting Context Strings in Chapter 6, Creating Topics.)

You can use the JumpId macro to display topics in secondary windows by adding the window name to the filename parameter, as in this example:

```
JumpId("progman.hlp>proc", "groups_how_pm")
```

The topic identified by the groups_how_pm context string would appear in the proc secondary window.

If you use the JumpId macro without specifying a filename, Help performs the jump in the current Help file, as in this example:

```
JumpId("", "groups_how_pm")
```

This method is not recommended, but it may be helpful under certain circumstances (for example, you repeat the macro many times in the same Help file and you want to save disk space).

See Also

JumpContext, PopupId

JumpKeyword (or JK)

Opens the indicated Help file, searches through the K keyword table, and displays the first topic containing the keyword specified in the macro.

Syntax

JumpKeyword("filename", "keyword")
JK("filename", "keyword")

Parameter	Description
filename	Name of the Help file containing the desired keyword table. The filename must be enclosed in quotation marks.
keyword	Keyword that the macro passes to Help to search for. The keyword must be enclosed in quotation marks.

Example

The following macro displays the first topic that has hands as a keyword in the CLOCK.HLP file:

```
JumpKeyword("clock.hlp", "hands")
```

Comments

If Windows Help cannot find the specified Help file, it displays an error message and does not perform the jump.

If Windows Help finds more than one keyword match, it displays the first matched topic. If it does not find any matches, it displays a Not a keyword message and displays the Contents topic of the destination Help file.

See Also

Search

Next

Displays the next topic in the browse sequence for the Help file. Executing this macro is the same as choosing the Browse next button (>>).

Syntax

Next()

Parameter	Description
none	

Comments

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

Windows Help ignores this macro if it is executed in a secondary window.

See Also

BrowseButtons, Previous

Not

Reverses the result (nonzero or zero) returned by the IsMark macro. Use it with the IsMark macro as a parameter to the conditional macros IfThen and IfThenElse.

Syntax

Not(IsMark("marker-text"))

Parameter	Description
marker-text	Text marker previously created by the SaveMark macro. The IsMark macro tests the marker you specify. The Not macro returns zero if the mark exists (IsMark returns nonzero) or nonzero if the mark does not exist (IsMark returns zero). The marker text must be enclosed in quotation marks.

Example

The following macro executes a jump to the topic with the exp_mem context string if the SaveMark macro has not set a marker named Managing Memory:

```
IfThen(Not(IsMark("Managing Memory")), "JI(`trb.hlp', `exp_mem')")
```

See Also

DeleteMark, GotoMark, IfThen, IfThenElse, IsMark, SaveMark

PopupContext (or PC)

Displays in a pop-up window a topic identified by a specific context number. An entry in the [MAP] section of the Help project file identifies the context.

Syntax

PopupContext("filename", context-number)

PC("filename", context-number)

Parameter	Description
filename	Name of the Help file that contains the topic to be displayed in the pop-up window. The filename must be enclosed in quotation marks.
context-number	Context number of the topic to be displayed in the pop-up window. The context number must be defined in the [MAP] section of the specified Help files project file.

Example

The following macro displays in a pop-up window the topic mapped to the 801 context ID number in the PIFEDIT.HLP file:

```
PopupContext("pifedit.hlp", 801)
```

Comments

If Windows Help cannot find the specified Help file, it displays an error message.

If the context number does not exist or cannot be found in the [MAP] section, Windows Help displays the Contents topic or the first topic in the Help file. (For more information about context numbers, see [MAP] Section in Chapter 16, The Help Project File.)

See Also

JumpContext

PopupId (or PI)

Displays in a pop-up window the topic with a specified context string in a specified Help file. Unlike the PopupContext macro, this macro does not require the topic to be defined in the [MAP] section because it takes the topic context string (as defined in the # footnote) as the second parameter.

Syntax

PopupId("filename", "context-string")
PI("filename", "context-string")

Parameter	Description
filename	Name of the Help file containing the pop-up window topic. The filename must be enclosed in quotation marks.
context-string	Context string of the topic in the destination Help file. The context string must be enclosed in quotation marks.

Example

The following macro displays in a pop-up window a topic identified by the 019_eg_pm context string in the PROGMAN.HLP file:

```
PopupId("progman.hlp", "019_eg_pm")
```

Comments

If Windows Help cannot find the specified Help file, it displays an error message.

If the context string does not exist or cannot be found, Windows Help displays the Contents topic or the first topic in the Help file. (For more information about context strings, see Inserting Context Strings in Chapter 6, Creating Topics.)

See Also

JumpId

PositionWindow (or PW)

Sets the size and position of either the main Help window or a secondary window.

Syntax

PositionWindow(x-coord, y-coord, width, height, window-state, "window-name")

PW(x-coord, y-coord, width, height, window-state, "window-name")

Parameter	Description
x-coord	X-coordinate, in Help units, of the upper-left window corner. Positions are defined in terms of Windows Helps 1024-by-1024 coordinate system, regardless of screen resolution. For example, if the x-coordinate is 512, the left edge of the Help window is in the middle of the screen. (For more information about determining actual coordinates for different video resolutions, see Secondary Windows in Chapter 9, Defining Topic Windows.)
y-coord	Y-coordinate, in Help units, of the upper-left window corner.
width	Default width, in Help units, of the window.
height	Default height, in Help units, of the window.
window-state	Specifies the windows state when it is displayed. This parameter is passed to the Windows ShowWindow function, which determines how the window is to be shown. The values for the ShowWindow function are explained in the following table. If the window is maximized (value=3), Windows Help ignores the x-coord, y-coord, width, and height parameters.

Value	Constant	Action
0	SW_HIDE	Hides the window and passes activation to another window.
1	SW_SHOWNORMAL	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_RESTORE).
2	SW_SHOWMINIMIZED	Activates a window and displays it as an icon.
3	SW_SHOWMAXIMIZED	Activates a window and displays it as a maximized window.
4	SW_SHOWNOACTIVATE	Displays a window in its most recent size and position. The window that is currently active remains active.
5	SW_SHOW	Activates a window and displays it in its current size and position.
6	SW_MINIMIZE	Minimizes the specified window and activates the top-level window in the systems list.
7	SW_SHOWMINNOACTIVE	Displays a window as an icon. The window that is currently active remains active.
8	SW_SHOWNA	Displays a window in its current state. The window that is currently active remains active.
9	SW_RESTORE	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_SHOWNORMAL).

window-name	Name of the window to position. The name main is reserved for the primary Help window. Secondary window names are defined in the [WINDOWS] section of the Help project file. This name must be enclosed in quotation marks.

Example

The following macro displays and positions the Samples secondary window in the upper-left corner (100,

100) with a width and height of 500 in Help units:

```
PositionWindow(100, 100, 500, 500, 5, "Samples")
```

Comments

If the window to be positioned does not exist, Windows Help ignores the macro.

Microsoft does not guarantee platform independence for the ShowWindow function because the values are Windows constants. Therefore, they may change on other platforms.

See Also

CloseWindow, FocusWindow

Prev

Displays the previous topic in the browse sequence for the Help file. Executing this macro is the same as choosing the Browse previous button (<<).

Syntax

Prev()

Parameter	Description
none	

Comments

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

Windows Help ignores this macro if it is executed in a secondary window.

See Also

BrowseButtons, Next

Print

Sends the currently displayed topic to the printer.

Syntax
Print()

Parameter	Description
none	

Comments

Use this macro only to print topics in windows other than the main Help window. For example, use it to print topics displayed in secondary windows provided no dialog boxes are open at the time of printing.

Using this macro in secondary windows is highly recommended because it is the only way that a user can print the text in a secondary window.

If the Print macro executes from a pop-up window, the topic that contains the pop-up hot spot is printed rather than the topic that is displayed in the pop-up window.

PrinterSetup

Displays the Print Setup dialog box. Executing this macro is the same as choosing the Print Setup command on the File menu.

Syntax

PrinterSetup()

Parameter	Description
none	

RegisterRoutine (or RR)

Registers a function within a dynamic-link library (DLL) as a Help macro. Registered functions can be used in macro hot spots or footnotes within topic files or in the [CONFIG] section of the Help project file, the same as standard Help macros.

> Note: The RegisterRoutine macro ignores all return values.

Syntax

RegisterRoutine("DLL-name", "function-name", "parameter-spec")
RR("DLL-name", "function-name", "parameter-spec")

Parameter	Description
DLL-name	String specifying the filename of the DLL being called. The filename must be enclosed in quotation marks. You can omit the .DLL filename extension.Specify the directory only if necessary. Generally, DLLs are installed in the directory where Windows Help resides. For more information, see How Help Locates .DLL and .EXE Files in Chapter 14, Help Macros.
function-name	String specifying the name of the function you want to use as a Help macro. The function name must be enclosed in quotation marks.
parameter-spec	String specifying the formats of parameters passed to the function. Characters in the string represent C parameter types. Valid parameter types include the following:

Character	Data type	Equivalent Windows data type
u	Unsigned short integer	UINT, WORD, WPARAM
U	Unsigned long integer	DWORD
i	Signed short integer	BOOL (also C int or short)
I	Signed long integer	LONG, LPARAM, LRESULT
s	Near pointer to a null-terminated text string	PSTR, NPSTR
S	Far pointer to a null-terminated text string	LPSTR, LPCSTR
v	Void (means no type; used only with return values)	None. Equivalent to C void data type.

The parameter-spec must be enclosed in quotation marks. Windows Help checks the format string to ensure that it matches the function prototype defined in the DLL.

To determine the data type of the functions parameters, consult the application programming interface (API) documentation for the DLL, or ask the person who developed the DLL. When using Windows functions with Windows Help, be sure that you fully understand how the function will affect Help. For information on Windows functions, their parameters, and parameter types, see the Microsoft Windows version 3.1 Software Development Kit.

Example

The following DLL call registers a routine named PlayAudio in the DLL named HELPLIB.DLL:

```
RegisterRoutine("helplib", "PlayAudio", "SIU")
```

Comments

If Windows Help cannot find the DLL, it displays an error message and does not perform the call. When loading DLLs, Help looks for a routine first in the directory from which Help was started. Therefore, WINHELP.EXE can be located anywhereeven on a drive not on the users machineand Help will look for DLLs in that directory first. Generally, it is not a good idea to place application-specific DLLs in the

Windows directory or in the Windows SYSTEM directory.

RemoveAccelerator (or RA)

Removes an accelerator keyboard (access) key or key combination assigned to a Help macro.

Syntax

RemoveAccelerator(key, shift-state)

RA(key, shift-state)

Parameter	Description
key	Windows virtual-key value assigned to the macro using the AddAccelerator macro. For the list of virtual keys, see Appendix A, Windows Virtual-Key Codes.
shift-state	Number specifying the key or key combination to use with the accelerator key. Valid modifier keys are ALT, SHIFT, and CTRL.Number

Modifier key(s)

0	(No modifier key)
1	SHIFT
2	CTRL
3	SHIFT+CTRL
4	ALT
5	ALT+SHIFT
6	ALT+CTRL
7	ALT+SHIFT+CTRL

Example

The following macro removes the ALT+CTRL+F10 key combination that was assigned in the AddAccelerator macro example:

```
RemoveAccelerator(0x79, 6)
```

Comments

Windows Help does not display an error message if the author attempts to remove an unassigned accelerator key.

See Also

AddAccelerator

SaveMark

Saves the location of the currently displayed topic and Help file and associates a text marker with that location. The GotoMark macro can then be used to jump to this location.

Syntax

SaveMark("marker-text")

Parameter	Description
marker-text	Text marker used to identify the topic location. The marker text must be enclosed in quotation marks, and it must be unique.

Example

The following macro saves the Managing Memory marker in the current topic in the Troubleshooting Help file:

```
SaveMark("Managing Memory")
```

Comments

Text markers are not saved if the user exits and then restarts Windows Help.

If you use the same text for more than one marker, Windows Help uses the most recently entered marker.

See Also

DeleteMark, GotoMark, IfThen, IfThenElse, IsMark, Not

Search

Displays the Search dialog box, which allows users to search for topics using keywords defined in K footnotes. Executing this macro is the same as choosing the Search button.

Syntax

Search()

Parameter **Description**

none

Comments

Windows Help ignores this macro if it is executed in a secondary window.

See Also

JumpKeyword

SetContents

Designates a specific topic as the Contents topic in the specified Help file.

Syntax

SetContents("filename", context-number)

Parameter	Description
filename	Name of the Help file that contains the desired Contents topic. The filename must be enclosed in quotation marks.
context-number	Context number of the topic in the specified Help file. The context number must be defined in the [MAP] section of the destination Help files project file.

Example

The following macro sets the topic mapped to the 101 context ID number in the PROGMAN.HLP file as the Contents topic:

```
SetContents("progman.hlp", 101)
```

After this macro executes, choosing the Contents button causes a jump to the topic mapped to 101.

Comments

If Windows Help cannot find the Help file, it displays an error message and does not perform the jump.

If the context number does not exist or cannot be found in the [MAP] section, Windows Help displays an error message. (For more information about context numbers, see [MAP] Section in Chapter 16, The Help Project File.)

See Also

Contents, JumpContents

SetHelpOnFile

Designates the Help file that is to replace WINHELP.HLP, the How To Use Help file provided with Windows Help version 3.1. The replacement file opens when the user chooses the How To Use Help command or presses F1 in Windows Help.

Syntax

SetHelpOnFile("filename")

Parameter	Description
filename	Name of the replacement How To Use Help file. The filename must be enclosed in quotation marks.

Example

The following macro sets the How To Use Help file as QUIKHELP.HLP:

```
SetHelpOnFile("quikhelp.hlp")
```

To ensure that the How To Use Help file is always displayed in the main Help window, add the window name main to the macro and place the macro in the [CONFIG] section of the Help project file, as in this example:

```
[CONFIG]
SetHelpOnFile("quikhelp.hlp>main")
```

Comments

If Windows Help cannot find the Help file, it displays an error message.

If this macro appears within a topic in the Help file, the replacement Help file is set after execution of the macro. If this macro appears in the [CONFIG] section of the Help project file, the replacement Help file is set when the Help file is opened.

If this macro is executed from a secondary window, the replacement file will appear in the secondary window.

If you use this macro to replace the default How To Use Help file, executing the HelpOn macro will display the custom version of How To Use Help.

See Also

HelpOn, JumpHelpOn

UncheckItem (or UI)

Removes the check mark from a menu item added to a Windows Help menu with the CheckItem macro. The check mark indicates the state of a toggle-type command: on or off, show or hide, and so on.

Syntax

UncheckItem("item-id")
UI("item-id")

Parameter	Description
item-id	Identifies the menu item to uncheck. Use the identifier assigned to the item in the AppendItem macro. The item ID must be enclosed in quotation marks.

Example

The following macro removes the check mark from the menu item identified by time_item:

```
UncheckItem("time_item")
```

Comments

To check a menu item, use the CheckItem macro.

Windows Help ignores this macro if it is executed in a secondary window.

See Also

AppendItem, ChangeItemBinding, CheckItem, DeleteItem, DisableItem, EnableItem, InsertItem, InsertMenu

Chapter 16 The Help Project File

This chapter describes the format and contents of the Help project file (.HPJ), which is used to build the Help file. The Help project file contains all the information the Help compiler needs to combine RTF files and other elements into a Help file. Among other information, the Help project file tells the compiler:

- Where to find the files used to build the Help file.
- Which topic contains the table of contents for the Help file.
- Which custom elements (including menus, buttons, and windows) are added to the Help file.
- Which custom DLLs (if any) are used with the Help file.
- Which options to include during the build process.

Help Project File Sections

A Help project file contsists of several sections, each of which specifies information about the Help file. Section names appear within square brackets using the following syntax:

[sectionname]

The following table describes the nine sections that can be used in a Help project file.

Section	Function
[OPTIONS]	Specifies options that control the build process. This section is optional. If this section is used, it should appear before any other section in the Help project file.
[FILES]	Specifies topic files to be included in the build. This section is required.
[BUILDTAGS]	Specifies valid build tags. This section is optional.
[CONFIG]	Specifies author-defined menus and buttons used in the Help file and registers DLLs and DLL functions used as macros within the Help file. This section is required if the Help file uses any of these features.
[BITMAPS]	Specifies bitmap files to be included in the build. This section is not required if the Help project file lists a path for bitmap files using the BMROOT or the ROOT option.
[MAP]	Associates context strings with context numbers for context-sensitive Help within the application. This section is optional.
[ALIAS]	Assigns one or more context strings to the same topic. This section is optional.
[WINDOWS]	Defines the characteristics of the primary Help window and secondary window types used in the Help file. This section is required if the Help file uses secondary windows.
[BAGGAGE]	Lists files that are to be placed within the Help files .HLP file (which contains its own file system). This section is optional.

Semicolons (;) can be used to indicate a comment in the Help project file. Comments can be a single line or multiple lines. The compiler ignores all text from the semicolon to the end of the line on which it occurs.

Project-File Features

To create many features in the Help file, you modify topic files in the word processor. After you create the topic files, you build the file with the Help project file and compiler. But thats only the simple case. In fact, you use the Help project file to do more than just pass technical information to the compiler. You also use the Help project file to create unique features within the Help file. Some of these features can be added only by using project-file sections and options. Each section and option in the Help project file has a different purpose, and each can be used to improve the effectiveness and usability of your finished Help file.

Before you go on to the Help Project File Reference section, you might want to review the following table to find out what kinds of features you can create with project-file sections and options.

Project-file feature	[Section] or option
Customize the look and feel of the Help application, such as add menus and buttons and use DLLs to create new features.	[CONFIG]
Control the size and position of Help windows and add secondary windows that display information independently of the main window.	[WINDOWS]
Display a custom icon when the user minimizes the Help window.	ICON
Display the name of your Help file in the title bar of the Help window.	TITLE
Designate a certain topic as the Contents screen for the entire Help file.	CONTENTS
Create a custom copyright message for your Help file that users will see when viewing the Help file or copying information from it.	CITATION, COPYRIGHT
Customize the keyword search so that it uses more than one keyword table or a non-English language sorting order.	MULTIKEY, LANGUAGE
Tell the compiler which source files to include in the build and where to find them.	[FILES], [BITMAPS], BMROOT, ROOT
Compress the Help file so it uses less disk space.	COMPRESS, OLDKEYPHRASE
Control the amount of information the compiler displays during a build and where the information is displayed.	ERRORLOG, REPORT, WARNING
Create the topic files using one set of fonts and font sizes and have Help display the Help file using a different font and font size.	FORCEFONT, MAPFONTSIZE
Improve the performance of the Help file when it is delivered to users on a CD-ROM drive.	OPTCDROM
Perform a partial build of the Help file that excludes certain topics.	[BUILDTAGS], BUILD
Define a set of identifiers so that the application can display context-sensitive Help topics when the user requests Help.	[MAP], [ALIAS]
Include a group of non-standard files, such as multimedia files, in the Help file.	[BAGGAGE]

Sample Help Project File

The following shows a sample Help project file for the Cardfile application. Comments in the file text indicate the purpose of each section in the file

```
; Options used to define the
; Help project root, bitmap directory,
; contents topic, title, minimized icon,
; compression, warning level,
; report, and error file
;
[OPTIONS]
ERRORLOG=CARD.BUG
ROOT=C:\HELP
BMROOT=C:\HELP\ART
CONTENTS=cont_idx_card
TITLE=Cardfile Help
ICON=CARDHLP.ICO
COMPRESS=OFF
WARNING=3
REPORT=ON
;files used to build Cardfile Help
[FILES]
\.\RTFTXT\COMMANDS.RTF
\.\RTFTXT\HOWTO.RTF
\.\RTFTXT\KEYS.RTF
\.\RTFTXT\GLOSSARY.RTF
;button macros and How to Use Help file
[CONFIG]
CreateButton("btn_up", "&Up", "JumpContents(`HOME.HLP')")
BrowseButtons()
SetHelpOnFile("APPHELP.HLP")
;secondary-window characteristics
[WINDOWS]
picture="Samples", (123, 123, 256, 256), 0, (0, 255, 255), (255, 0, 0)
```

Help Project File Reference

The Help Project File Reference describes the different sections and options in a Help project file in alphabetic order and gives examples of their use. Reference descriptions provide the following information.

Heading	Information
Syntax	Syntax for the section or option. For information about the typographic conventions used in syntax descriptions, see the Document Conventions section in the Introduction to this guide.
Parameters	Describes the parameters that the section or option requires.
Comments	Notes about using the section or option, including any restrictions.
Example	Example of the section or option.
See Also	Cross-references to similar sections and options.

[ALIAS] Section

Syntax

[ALIAS]

context_string=alias context string . . .

The [ALIAS] section associates one set of context strings with an alternate set of context strings. The alias strings correspond to context strings assigned to topics in the # footnotes of the Help file. This section is optional; however, if it is included, it must precede the [MAP] section in the Help project file.

Parameters

context_string Specifies the application ID or other context ID that you want to reassign.

alias context string

Specifies the context string that appears in the # footnote of the topic you want Help to recognize. An alias context string has the same form and follows the same conventions as standard context string. That is, it is not case-sensitive and may contain the alphabetic characters A through Z, the numeric characters 0 through 9, and the period (.) and underscore (_) characters.

Comments

Because context strings must be unique for each topic and cannot be used for any other topic in the Help project, the [ALIAS] section provides a way to remap context strings that are no longer used or invalid. For example, suppose the application defines a context ID for each field in a dialog box, but your Help file only provides one topic for all the fields. You can use the [ALIAS] section to map all the application context IDs to your one Help topic. In this way, no matter which field the user has selected in the dialog box, Help will display your Help topic when the user requests context-sensitive Help.

You can also use the [ALIAS] section to combine Help topics without recoding your files. For example, if you create a topic that replaces the information in three other topics, you could manually search through your files for invalid cross-references to the deleted topics. The easier approach, however, would be to use the [ALIAS] section to assign the name of the new topic to the deleted topics.

You can use alias names in the [MAP] section of the Help project file. If you do, however, the [ALIAS] section must precede the [MAP] section.

Example

The following example creates several aliases within an [ALIAS] section:

```
[ALIAS]
sm_key=key_shrtcuts
cc_key=key_shrtcuts
st_key=key_shrtcuts        ;combined into keyboard shortcuts topic
clskey=us_dlog_bxs
maakey=us_dlog_bxs         ;covered in using dialog boxes topic
chk_key=dlogprts
drp_key=dlogprts
lst_key=dlogprts
opt_key=dlogprts
tbx_key=dlogprts                ;combined into parts of dialog box topic
frmtxt=edittxt
wrptxt=edittxt
seltxt=edittxt                  ;covered in editing text topic
```

See Also

[MAP] Section

[BAGGAGE] Section

Syntax

[BAGGAGE]

filename . . .

The [BAGGAGE] section lists files (typically multimedia elements) that the Windows Help compiler stores within the .HLP files internal file system. Windows Help can access data files stored in the Help file system more efficiently than it can access files stored in the normal MS-DOS file system because it doesnt have to read the file allocation table from CD-ROM when it accesses the files.

In some circumstances, a Help file may use many DLLs or execute several programs that use external data elements, such as graphics, animations, or audio files. In those situations, the Help file will benefit if it can include all the data elements needed by those programs in one MS-DOS file. To store Help-related data files within the Help file, Help authors can include a [BAGGAGE] section in the Help project file.

Parameter

filename	Specifies the full path of a baggage file. If a file cannot be found, the compiler reports an error.

Comments

The Help compiler stores all files listed in the [BAGGAGE] section exactly as they are typed. In other words, baggage filenames are case sensitive. To retrieve a baggage file, Help uses the MS-DOS filename without the path. This means that the Help author must specify the filename exactly as it appears in the [BAGGAGE] section, as in the following example.

Help does not limit the number of baggage files that you can define; however, adding too many baggage files will waste disk space and decrease performance when building and displaying the Help file. For that reason, you should define no more than 1000 baggage files. If you have more than 1000 files to include, you should store the data outside the .HLP file or concatenate the files into a few baggage files that you list in the [BAGGAGE] section.

To access the data from the Help files internal file system, Windows Help provides specialized source code. This source code can be built into an application or custom DLL so that it can retrieve the appropriate data file from the Help files [BAGGAGE] section. For more information and the baggage include file, see Appendix C, Baggage Access Functions.

To avoid having to specify a full path, use the ROOT option in the [OPTIONS] section to specify the path. Then all filenames that you give in the topic file are relative to the ROOT path.

Example

The following [BAGGAGE] section stores multimedia files in the Help file:

```
[BAGGAGE]
\.\ANIM\ASTER.AMF
\.\AUDIO\APATO.WAV
\.\AUDIO\PACHY.WAV
\.\AUDIO\STAR.WAV
\.\AUDIO\TREX.WAV
\.\AVI\APATO.AVI
\.\AVI\PACHY.AVI
\.\AVI\TREX.AVI
```

See Also

ROOT Option

[BITMAPS] Section

Syntax

[BITMAPS]

filename . . .

The [BITMAPS] section tells the Help compiler where to find bitmaps placed using the bmc, bml, or bmr reference.

Parameter

filename	Specifies the full path of a bitmap file. If a file cannot be found, the compiler reports an error.

Comments

The [BITMAPS] section is not required if the bitmaps are located in the Help project directory or if the path containing the bitmaps is listed in the BMROOT or ROOT option. If the Help project file does not include either of these options, each bitmap filename must be listed in the [BITMAPS] section of the Help project file.

Example

The following example specifies three bitmaps:

```
[BITMAPS]
BMP01.BMP
BMP02.BMP
BMP03.BMP
```

See Also

BMROOT Option, ROOT Option

BMROOT Option

Syntax

BMROOT=pathname[, pathname]...

The BMROOT option designates the bitmap root directory, which tells the Help compiler where to find the bitmap files to include in the build.

If the bitmap files reside in the root directory of the project, you dont need to include a BMROOT directory; use the ROOT option instead. However, if you set up your Help project so that bitmap files are not stored in the project root directory, you may want to designate one or more bitmap root directories in the BMROOT option.

Parameter

pathname	Specifies a drive and full path to each bitmap directory or to a path relative to the project directory. If you give more than one path for this option, use a comma (,) to separate the paths. If the path is invalid, the compiler reports an error.

Comments

If the Help project file has a BMROOT option, you dont need to list bitmap files in the [BITMAPS] section.

If the Help project file does not have a BMROOT option, the Help compiler looks for bitmaps in the directories specified by the ROOT option. If the Help project file doesnt have a ROOT option, or if none of the ROOT directories contain the bitmap files, you can:

- List the bitmap filenames in the [BITMAPS] section of the Help project file.
  or
- Place all bitmap files in the same directory as the Help project file.

Example

The following entry specifies that bitmap files reside in the \HELP\BMP1 and \HELP\BMP2 directories on drive C and in the \GRAPHICS\ART directory on drive D:

```
[OPTIONS]
BMROOT=C:\HELP\BMP1, C:\HELP\BMP2, D:\GRAPHICS\ART
```

See Also

[BITMAPS] Section, [OPTIONS] Section, ROOT Option

BUILD Option

Syntax

BUILD=expression

The BUILD option specifies which topics containing build tags are included in or excluded from a build. Use this option only if the RTF topic files have build tags.

A topic contains a build tag if it includes a build tag footnote (*). Topics without build tags are always compiled, regardless of the current build expression.

Parameter

expression — A logical statement that specifies which topics to include in or exclude from the build. This parameter consists of a combination of build tags (specified in the [BUILDTAGS] section) and the following logical operators.

Operator	Description
~	Applies the NOT operator to a single build tag. The Help compiler compiles a topic only if the tag is not present. This operator has the highest precedence; the compiler applies it before any other operator.
&	Combines two build tags by using the AND operator. The Help compiler compiles a topic only if it contains both build tags used in the expression. The compiler applies this operator only after the ~ operator has been applied.
|	Combines two build tags by using the OR operator. The Help compiler compiles a topic if it has at least one of the build tags used in the expression. This operator has the lowest precedence; the compiler applies it only after all other operators have been applied.

Parentheses may be used to override operator precedence. Expressions enclosed within parentheses are always evaluated first.

Comments

Only one BUILD option can be given per Help project file.

The Help compiler evaluates all build expressions from left to right, using the specified precedence rules.

Example

The following examples assume that the [BUILDTAGS] section in the Help project defines the build tags DEMO, MASTER, and TEST_BUILD. Although the following examples show several BUILD options on consecutive lines using these build tags, only one BUILD option would be allowed in the Help project file.

Example	Compile all topics...
`BUILD=DEMO`	That have the DEMO tag
`BUILD=DEMO & MASTER`	With both the DEMO and MASTER tag
`BUILD=DEMO | MASTER`	With either the DEMO or MASTER tag
`BUILD=(DEMO | MASTER) & TEST_BUILD`	That have either the DEMO or MASTER tag and also the TEST_BUILD tag
`BUILD=~ MASTER`	That do not have the MASTER tag

See Also

[BUILDTAGS] Section, [OPTIONS] Section

[BUILDTAGS] Section

Syntax

[BUILDTAGS]

tag . . .

The [BUILDTAGS] section defines the valid build tags for a Help file. The Help compiler uses the build tags to determine which topics to include when building the Help file.

This section is used in conjunction with the build tag footnote (*) and the BUILD option. The build tag footnote associates a particular build tag with a given topic. If the build tag is included in the [BUILDTAGS] section and defined in the BUILD expression, the Help compiler compiles the topic; otherwise, it excludes the topic from the build.

Parameter

tag	Specifies a build tag consisting of any combination of characters except spaces. The Help compiler strips any spaces it finds between the start of the build tag and the end of the tag. Build tags are case insensitive, so the compiler treats uppercase and lowercase characters as the same. Each build tag can have as many as 32 characters.

Comments

The [BUILDTAGS] section is optional. If used, it can include as many as 30 build tags.

Example

The following example defines four build tags in a sample Help project file:

```
[BUILDTAGS]
DEMO            ;topics to include in demo build
MASTER          ;topics to include in master Help file
DEBUGBUILD      ;topics to include in debugging build
TESTBUILD       ;topics to include in a mini-build for testing
```

See Also

BUILD Option

CITATION Option

Syntax

CITATION=copyright-notice

The CITATION option appends a custom copyright notice to the end of any information that is copied from the Help file using the Copy command. You can use this option if a publisher owns some material in your Help file and requires that you attach a copyright notice to that material.

Parameter

copyright-notice Specifies the copyright information you want to append to copied material. The notice appears at the end of text displayed in the Copy dialog box (the notice is appended whenever the user copies text from the original Help file).

Comments

The CITATION option differs from the COPYRIGHT option in two ways: a citation does not appear in the About dialog box, and a citation can be much larger than the standard copyright notice. The maximum size of the citation copyright notice is about 2K (compared to 50 characters for COPYRIGHT).

Example

The following shows a sample citation:

```
CITATION=Information in this document is subject to change without notice
and does not represent a commitment on the part of Microsoft Corporation.
The software, which includes information contained in any databases,
described in this document is furnished under a license agreement or
nondisclosure agreement and may be used or copied only in accordance with
the terms of that agreement. It is against the law to copy the software
except as specifically allowed in the license or nondisclosure agreement.
No part of this document may be reproduced in any form or by any means,
electronic or mechanical, for any purpose without the express written
permission of Microsoft Corporation.
```

See Also

COPYRIGHT Option, [OPTIONS] Section

COMPRESS Option

Syntax

COMPRESS=compression-level

The COMPRESS option specifies the level of compression to be used when building the Help file. Compression levels indicate either no compression, medium compression (approximately 40 percent), or high compression (approximately 50 percent). The higher the compression level, the smaller the Help file. However, the higher the compression, the longer the file takes to build.

Parameter

compression-level Specifies the level of compression. This parameter can be one of the following values.

Value	Meaning
OFF	The Help compiler does not compress the Help file.
MEDIUM	The compiler uses a medium level of compression (block compression).
HIGH	The compiler uses high compression (block and key-phrase compression).

Comments

Depending on the level of compression requested, the Help compiler uses either block compression or a combination of block and key-phrase compression:

- Block compression compresses the topic data into predefined units known as blocks.
- Key-phrase compression combines repeated phrases found within the source file(s).

The Help compiler creates a phrase-table file with a .PH extension if it doesnt find one in the project root directory. If the Help compiler finds a file with a .PH extension, it uses the file for the current build. Because the .PH file speeds up the compression process when little text has changed since the last build, you might want to keep the phrase file around if you compile the same Help file several times with compression. However, you get maximum compression if you delete the .PH file before starting each build.

Because of these improvements to Help compression, the resulting Help files you build will take up less space on users disks, but your setup program may not be able to compress the smaller Help files as much as version 3.0 files. Therefore, when estimating disk-space requirements, you should use sizes relatively close to those achieved with high compression.

Key-phrase compression is the only compression method supported by the version 3.0 Help compiler.

Example

The following shows a typical compression entry in the Help project file:

```
COMPRESS=off
```

See Also

[OPTIONS] Section

[CONFIG] Section

Syntax

[CONFIG]

macro . . .

The [CONFIG] section contains one or more Windows Help macros that carry out actions, such as creating buttons or menus. This section can also contain macros that register routines in external DLLs as Windows Help macros. These routines can then be used the same as Windows Help macros. Windows Help executes the macros when it opens the Help file.

Parameter

macro	Specifies a Windows Help macro or a DLL function registered as a Help macro. For descriptions of the standard Help macros and the RegisterRoutine macro, see Chapter 15, Help Macro Reference.

Comments

The [CONFIG] section can include any number of lines, and each line can have as many as 512 characters. When listing macros in the [CONFIG] section, include only one macro per line, instead of stringing them together and separating them with semicolons as you do in macro hot spots and macro footnotes.

When opening a Help file, Help does not necessarily execute the macros listed in the [CONFIG] section in the same order.

Example

The following example registers two DLLs, creates a button, enables the browse buttons, and sets the name of the How To Use Help file:

```
[CONFIG]
RegisterRoutine("bmp","HDisplayBmp","USSS")
RegisterRoutine("bmp","CopyBmp", "v=USS")
CreateButton("btn_up", "&Up", "JumpContents(`HOME.HLP')")
BrowseButtons()
SetHelpOnFile("APPHELP.HLP")
```

CONTENTS Option

Syntax

CONTENTS=context-string

The CONTENTS option identifies the context string of the highest-level or home topic (usually a Table of Contents or index within the Help file). Windows Help displays the Contents whenever a user chooses Contents from the Help menu, clicks the Contents button, or presses F1 in the application without a specific context.

Parameter

context-string	Specifies the context string of a topic in the Help file. The string can be any combination of characters, except spaces, and must also be specified in a context string footnote (#) in some topic in the Help file.

Comments

If the [OPTIONS] section does not include a CONTENTS option, the compiler assumes that the first topic it encounters in the first listed topic file in the [FILES] section of the Help project file is the contents topic.

Example

The following example sets the topic containing the main_contents context string as the Contents topic for this Help file:

```
CONTENTS=main_contents
```

See Also

[FILES] Section, [OPTIONS] Section

COPYRIGHT Option

Syntax

COPYRIGHT=copyright-notice

The COPYRIGHT option places a custom copyright notice in the About dialog box of Windows Help. Help displays the notice immediately below the Microsoft copyright notice. You can use this option to copyright the material found in your Help file.

Parameter

copyright-notice Specifies the copyright notice you want to display for users of your Help file. The notice can be any combination of characters, and can have from 35 to 75 characters depending on the characters you use. A notice of 50 characters generally fits in the dialog box.

Comments

The copyright notice also appears at the end of text displayed in the Copy dialog box. The notice is appended whenever the user copies text from the original Help file using the Copy command.

If you need to display a longer copyright notice, use the CITATION option.

Example

The following example adds a short copyright notice to the Help file:

```
COPYRIGHT="Copyright (C) 1992,  Microsoft Corporation."
```

See Also

CITATION Option, [OPTIONS] Section

ERRORLOG Option

Syntax

ERRORLOG=error-filename

The ERRORLOG option directs the compiler to write all error messages generated during the build to an error file. The compiler also displays the error messages on the screen.

Parameter

error-filename Specifies the name of the file to which the compiler will write the error messages. The filename can be any valid MS-DOS filename. This parameter can be a full or partial path if you want the file to be written to a directory other than the Help project root directory.

Comments

If you use the ERRORLOG option, it should be the first line in the [OPTIONS] section.

The error file contains the Windows Help copyright notice and the name of the Help project file at the top of the file, followed by any build errors that occurred. Error messages are listed on separate lines. The periods representing compiler progress are included only in the screen display, not in the written file.

If the Help compiler cannot create or open the error file, it displays an error message on the screen and continues the build.

Example

The following example writes all errors during the build to the HLPBUGS.TXT file in the Help project root directory:

```
ERRORLOG=HLPBUGS.TXT
```

See Also

[OPTIONS] Section, REPORT Option, WARNING Option

[FILES] Section

Syntax

[FILES]

filename . . .

The [FILES] section lists all RTF topic files used to build the Help file. A Help project file must have a [FILES] section.

Parameter

filename — Specifies the full or partial path of a topic file. If a partial path is given, the help compiler uses the directories specified by the ROOT option to construct a full path. If a file is not on the defined path and cannot be found, the compiler reports an error.

Comments

You can use the #include directive in the [FILES] section to specify the topic files indirectly by designating a file that contains a list of the topic files that are to be included in the build.

The #include directive has the following syntax:

```
#include <filename>
```

The filename must reside in the Help project directory, or it must include a complete path specification. The Help compiler does not use the INCLUDE environment variable to search for files.

Example

The following example specifies four topic files:

```
[FILES]
\.\rtftxt\COMMANDS.RTF ;comment 1
\.\rtftxt\HOWTO.RTF    ;comment 2
\.\rtftxt\KEYS.RTF     ;comment 3
\.\rtftxt\GLOSSARY.RTF ;comment 4
```

The following example uses the #include directive to specify the topic files indirectly:

```
[FILES]
#include <rtffiles.h>
```

See Also

ROOT Option

FORCEFONT Option

Syntax

FORCEFONT=fontname

The FORCEFONT option forces the Help file to substitute the specified font for all requested fonts. Use this option to create Help files that can be viewed on systems that do not have all fonts available.

Parameter

fontname	Specifies the name of an available font. Font names must be spelled the same as they are in the Fonts dialog box in Control Panel. Font names cannot exceed 20 characters. If an invalid font name is given, the Help compiler uses the MS Sans Serif font as the default.

Comments

The fontname can be any of the following standard fonts installed in Windows version 3.1:

- Courier 10,12,15
- Modern
- MS Sans Serif 8,10,12,14,18,24
- MS Serif 8,10,12,14,18,24
- Roman
- Script
- Small
- Symbol 8,10,12,14,18,24

Windows version 3.1 also includes the following scalable TrueType™ fonts; however, you cannot use this option to specify a TrueType font:

- Arial®
- Arial Bold
- Arial Bold Italic
- Arial Italic
- Courier
- Courier Bold
- Courier Bold Italic
- Courier Italic
- Times New Roman®
- Times New Roman Bold
- Times New Roman Bold Italic
- Times New Roman Italic
- Symbol

Example

The following example forces all fonts to be displayed in the MS Serif font:

```
FORCEFONT=MS SERIF
```

See Also

MAPFONTSIZE Option, [OPTIONS] Section

ICON Option

Syntax

ICON=icon-file

The ICON option identifies the icon file to display when the user minimizes the Windows Help application.

Parameter

icon-file	Specifies the name of the icon file. This file must have the standard Windows icon-file format (.ICO file). You must create this file in an application such as Microsoft Windows Image Editor (IMAGEDIT.EXE) or other application that generates the .ICO file format. You can specify either an absolute or relative path if the file resides in a directory other than the Help project root directory.

Comments

If you do not include the ICON option in your Help project file, Help will use the standard question-mark icon when the user minimizes the Help file.

If the icon file is in an invalid format, or if Help cannot find the icon file, the Help compiler displays an error message on the screen during the build and ignores this option. In that case, Help will display the standard question-mark icon.

The ICON option only appears when the user minimizes Help. To have your custom icon appear in a Program Manager group, you must provide users with the icon file and then instruct them to use the Properties command in Program Manager to change the standard Help icon to your custom icon.

Example

The following example creates a custom icon for the Help file:

```
ICON=HYPER.ICO
```

See Also

[OPTIONS] Section

LANGUAGE Option

Syntax

LANGUAGE=language-name

The LANGUAGE option sets the sort order for keywords in the Search dialog box.

Parameter

language-name Specifies the language on which to base sorting. This parameter can have only one value.

Value	Meaning
scandinavian	Sets the sorting order to the Scandinavian-language order.

Comments

The default sorting order is the English-language order.

Microsoft Windows Help version 3.1 supports only English and Scandinavian sorting.

Example

The following example changes the sorting order to Scandinavian:

```
LANGUAGE=SCANDINAVIAN
```

See Also

[OPTIONS] Section

[MAP] Section

Syntax

[MAP]

context_string context number . . .

The [MAP] section associates context strings (or aliases) to context numbers for context-sensitive Help. The context number corresponds to a value the parent application passes to Windows Help to display a particular topic. This section is optional.

Parameters

context-string Specifies the context string of a topic in the Help file. The string can be any combination of characters, except spaces, and must also be specified in a context string footnote (#) in some topic in the Help file.

context-number Specifes the context number to associate with the context string. The number can be in either decimal or standard C hexadecimal format.

Comments

Only one context number may be assigned to a context string or alias. Assigning the same number to more than one context string generates a compiler error.

You can separate context numbers and context strings by an arbitrary amount of white space using space characters or tabs, but there must be at least one space between the context number and the context string.

If you do not explicitly assign context numbers to topics, the Help compiler generates default values by converting topic context strings into context numbers.

You can define the context strings listed in the [MAP] section either in a Help topic or in the [ALIAS] section. The compiler generates a warning message if a context string appearing in the [MAP] section is not defined in any of the topic files or in the [ALIAS] section.

If you use an alias, the [ALIAS] section must precede the [MAP] section in the Help project file.

If you remove a Help topic that the application defines as context sensitive, users will get a Topic does not exist error message when they request Help on the item. To prevent the error from occurring, you must either change the application so that it no longer sends the context number to Help or map that context number to an existing topic.

The [MAP] section supports two additional statements for specifying context strings and their associated context numbers: #include and #define.

The #include statement has the following form:

```
#include <filename>
```

The filename parameter, which can be enclosed in either angle brackets (<>) or double quotation marks, specifies the name of a file containing one or more #define statements. The file may contain additional #include statements as well, but files may not be nested in this way more than five deep.

The #define statement has the following form:

```
#define  context-string  context-number
```

The context-string and context-number parameters are the same as those described in the Parameters section above.

If context numbers use the #define directive and the file containing the #define statements is included in both the application code and the Help file, updates made to the context numbers by the application programmers are reflected in the next Help build.

When using the #define statement in the [MAP] section, observe these rules:

- You can use C-style comments (/* open comment and */ close comment) with the #define directive. The comments can occur anywhere in the line.

```
#define context_string    context_number     /* comment */
```

- The Help compiler supports 32-bit constants in #define statements. It also accepts (as 32-bit constants) numbers that end with L and are accepted by the C compiler for a long constant:

```
#define vscroll      1234000L
```

- The Help compiler does not perform arithmetic on the object of the #define statement. It does not support the following forms of #define:

```
#define A  1
#define B  (A+1)
#define C  (A+2)
```

- The Help compiler only accepts #define statements; it does not support other forms such as #ifdef and #endif.

Examples

The following example uses a decimal number to specify the context number:

```
dtb_scr  34    ;document title bar
```

The following example uses a hexadecimal number to specify the context number:

```
Minimize_Icon  0x0004
```

The following example uses an #include directive to point to another file containing the context strings and context numbers:

```
#include <sample.h>
```

The following example uses a #define statement to specify the mapping:

```
#define up_scroll    0x0112      /* up scroll arrow */
```

See Also

[ALIAS] Section

MAPFONTSIZE

Syntax

MAPFONTSIZE=m[-n]:p

The MAPFONTSIZE option maps font sizes specified in topic files to different sizes when displayed in the Help window. You can use one font size in your topic files and have the compiler change them to an appropriate size for the actual Help file display.

This option is especially useful if there is a significant size difference between the authoring display and the intended user display, as there is if the RTF file is created using Word for the Macintosh.

Parameters

m[-n]	Specifies the size of the source font. This parameter is either a single point size or a range of point sizes, as indicated by the optional parameter n, which specifies a font range to be mapped. A range of point sizes consists of the low and high point sizes separated by a hyphen (-). If a range is specified, all fonts in the range are changed to the size specified by the p parameter.
p	Specifies the size of the desired font for the Help file.

Comments

Although you can specify as many as five font ranges in the [OPTIONS] section of the Help project file, you can map only one font size or range with each MAPFONTSIZE statement. If you include more than one MAPFONTSIZE statement, the source font size or range specified in subsequent statements cannot overlap previous mappings.

Examples

The following examples illustrate the use of the MAPFONTSIZE option:

```
MAPFONTSIZE=8:12            ; display all 8 pt. fonts as 12 pt.
MAPFONTSIZE=12-24:16     ; display fonts from 12 to 24 pts. as 16 pt.
```

The following two statements show an incorrect use of the MAPFONTSIZE option because the second statement contains a point size already mapped in the preceding statement (14 falls in the 1224 range):

```
MAPFONTSIZE=12-24:16
MAPFONTSIZE=14:20
```

See Also

FORCEFONT Option, [OPTIONS] Section

MULTIKEY Option

Syntax

MULTIKEY=footnote-character

The MULTIKEY option specifies the footnote character to use for an alternate keyword table. This option is intended to be used in conjunction with topic files that contain keyword footnotes for alternative keyword tables.

Parameter

footnote-character Specifies the case-sensitive letter to be used for the keyword footnote.

Comments

Multiple keyword tables enable Help authors to differentiate terminology from different sources. For example, the standard keyword table can be used to define search queries for the parent application, and a second keyword table can map commands in an auxiliary application. Users can then look up topics using both sets of keywords.

Because keyword footnotes are case sensitive, you should limit your keyword-table footnotes to one case, usually uppercase. If an uppercase letter is specified, the compiler will not include footnotes with the lowercase form of the same letter in the keyword table.

You may use any alphanumeric character for a keyword table except K and k, which are reserved for Helps standard keyword table. There is an absolute limit of five keyword tables, including the standard table. However, depending upon system configuration and the structure of your Help system, a practical limit of only two or three tables may be more realistic. If the compiler cannot create an additional keyword table, the additional table is ignored in the build.

Example

The following example illustrates how to enable the letter L for a keyword-table footnote:

```
MULTIKEY=L
```

See Also

[OPTIONS] Section

OLDKEYPHRASE Option

Syntax

OLDKEYPHRASE=yes/no

The OLDKEYPHRASE option specifies whether to use an existing key-phrase file for the current build.

Parameters

yes/no	Specifies whether the existing file should be used.

Comments

The Help compiler creates a phrase-table file with a .PH extension if it doesnt find one in the project root directory. If the Help compiler finds a file with a .PH extension, it uses the file for the current build. Because the .PH file speeds up the compression process when little text has changed since the last build, you might want to keep the phrase file around if you compile the same Help file several times with compression. However, you get maximum compression if you delete the .PH file before starting each build.

If you do not include this option in the Help project file, the Help compiler will use the old keyphrase file by default.

Example

The following example illustrates this option:

```
OLDKEYPHRASE=NO
```

See Also

[OPTIONS] Section

OPTCDROM Option

Syntax

OPTCDROM=yes/no

The OPTCDROM option optimizes a Help file for CD-ROM display by aligning topic files on predefined 2K block boundaries.

Parameters

yes/no	Specifies whether the file should be optimized for CD-ROM.

Comments

The CD-ROM optimization allows Windows Help to read data from the CD-ROM drive faster and more efficiently. On average, sequential reads from the CD are twice as fast when the topics are aligned using the OPTCDROM option.

The disadvantage to using the OPTCDROM option is that it slightly increases the size of the built Help file (approximately 10K). If you are delivering your Help file on CD-ROM, the extra file size should not be significant, given the large storage capacity of a CD-ROM disc.

Using the OPTCDROM option will also improve performance when Help files are read from a standard hard disk; however, because of the size increase you may not want to use this option if you are shipping your Help file on floppy disks. Help authors must evaluate the size difference and performance improvement of their individual files to decide whether to use this option.

Example

The following example illustrates this option:

```
OPTCDROM=YES
```

See Also

[OPTIONS] Section

[OPTIONS] Section

Syntax

[OPTIONS]

option . . .

The [OPTIONS] section includes options that control how a Help file is built and what feedback the build process displays. If you include this section in the Help project file, list it first so that the options apply during the entire build process.

Parameters

option Specifies one of the following project-file options.

Option	Description
BMROOT	Specifies the directory containing the bitmap files named in bmc, bml, and bmr references in the Help topic files.
BUILD	Determines which topics to include in the build.
CITATION	Adds a unique copyright message whenever users copy information in the Help file using the Copy command.
COMPRESS	Specifies the type of compression to use during the build.
CONTENTS	Specifies the context string of the Help files Contents topic.
COPYRIGHT	Adds a unique copyright message for the Help file to the About dialog box.
ERRORLOG	Puts compilation errors in a file during the build.
FORCEFONT	Forces all authored fonts in the topic files to appear in a different font when displayed in the Help file.
ICON	Specifies the icon file to be displayed when the Help file is minimized.
LANGUAGE	Specifies a different sorting order for Help files authored in a Scandinavian language.
MAPFONTSIZE	Maps a font size in the topic file to a different font size in the compiled Help file.
MULTIKEY	Specifies an alternate keyword table to use for mapping topics.
OLDKEYPHRASE	Specifies whether the compiler should use the existing key phrase table or create a new one during the build.
OPTCDROM	Optimizes the Help file for CD-ROM use.
REPORT	Controls the display of messages during the build process.
ROOT	Specifies the directories containing the topic and data files listed in the Help project file.
TITLE	Specifies the text that is displayed in the title bar of the Help window when the file is open.
WARNING	Indicates the level of error-message reporting the compiler is to display during the build.

Comments

These options can appear in any order within the [OPTIONS] section. The [OPTIONS] section is not required.

Example

The following shows the [OPTIONS] section in a sample Help project file:

```
[OPTIONS]
ERRORLOG=CARD.LOG
ROOT=C:\HELP\PROJECT
BMROOT=C:\HELP\PROJECT\BMP1;C:\HELP\PROJECT\BMP2;C:\HELP\PROJECT\BMP3;
TITLE=My Help File
```

```
CONTENTS=IDX_CONTENTS
COMPRESS=HIGH
OLDKEYPHRASE=NO
ICON=C:\HELP\PROJECT\BMP2\HLPFILE.ICO
WARNING=3
REPORT=ON
```

This sample [OPTIONS] section indicates the following to the Help compiler:

- Write messages displayed during the build to the CARD.LOG file.
- The project directory is C:\HELP\PROJECT.
- Bitmap files reside in three subdirectories off of the project root directory: \BMP1, \BMP2, and \BMP3.
- The title bar caption for the Help file is My Help File.
- The context string of the Help file's Contents topic is IDX_CONTENTS.
- Use high compression during the build.
- Do not use the old key-phrase table during compression.
- Use HLPFILE.ICO as the minimized Help icon.
- Report all errors and warnings.
- Display messages throughout the build to indicate when processes are occurring.

See Also

BMROOT Option, BUILD Option, CITATION Option, COMPRESS Option, CONTENTS Option, COPYRIGHT Option, ERRORLOG Option, FORCEFONT Option, ICON Option, LANGUAGE Option, MAPFONTSIZE Option, MULTIKEY Option, OLDKEYPHRASE Option, OPTCDROM Option, REPORT Option, ROOT Option, TITLE Option, WARNING Option

REPORT Option

Syntax

REPORT=on/off

The REPORT option displays messages on the screen during the build. These messages indicate when the compiler is performing the different phases of the build, including scanning the file for compression, compiling the file, verifying context strings, and resolving jumps, keywords, and browse sequences.

Parameters

on/off	Specifies whether the compiler should display messages on the screen during the build.

Comments

Unlike the ERRORLOG option, messages displayed by using the REPORT option are not written to a file.

Example

The following example illustrates this option:

```
REPORT=ON
```

See Also

[OPTIONS] Section, WARNING Option

ROOT Option

Syntax

ROOT=pathname[, pathname]...

The ROOT option specifies the project root directory where the Help compiler looks for the source files to include in the build.

Parameter

pathname	Specifies either a drive and full path or a relative path from the project directory. If you specify more than one project directory, a comma (,) separates each path. If the Help project file has a ROOT option, all relative paths in the Help project file refer to one of these paths. If the Help project file does not have a ROOT option, all paths are relative to the directory containing the Help project file.

Comments

If the Help project file does not have a BMROOT option, the compiler looks in the directories specified in the ROOT option to find bitmaps positioned by using the bmc, bml, and bmr references. If the Help project file doesnt have a ROOT option or if none of the ROOT directories contain the bitmaps, the bitmap filenames must be listed in the [BITMAPS] section of the Help project file.

Example

The following entry specifies that the project root directory is C:\HELP\PROJECT and is found on drive C:

```
[OPTIONS]
ROOT=C:\HELP\PROJECT
```

Given this root directory, topic files can be listed relative to the project root directory in the [FILES] section:

```
[FILES]
.\RTF\TOPICS1.RTF
.\RTF\TOPICS2.RTF
.\RTF\TOPICS3.RTF
.\RTF\TOPICS4.RTF
.\RTF\TOPICS5.RTF
```

The full path for a topic file is C:\HELP\PROJECT\RTF\TOPICS1.RTF.

See Also

[BITMAPS] Section, BMROOT Option, [OPTIONS] Section

TITLE Option

Syntax

TITLE=titlename

The TITLE option assigns a title to the Help file. Windows Help displays this title in the title bar of the Help window whenever it displays the Help file.

Parameter

titlename	Specifies the title displayed in the title bar of the Help window. The title can have as many as 50 characters.

Comments

If you do not specify a title using the TITLE option, the title bar displays Microsoft Help.

Example

The following example sets the Help file title to Cardfile Help:

```
[OPTIONS]
TITLE=Cardfile Help
```

See Also

[OPTIONS] Section

WARNING Option

Syntax

WARNING=level

The WARNING option specifies the amount of debugging information that the compiler is to report during the build.

Parameter

level	Specifies the warning level. This parameter may be any one of the following values.

Value	Meaning
1	Report only the most severe errors.
2	Report an intermediate number of errors.
3	Report all fatal errors and warnings.

Comments

This option only specifies the amount of information to report. It should be used in conjunction with the REPORT option and the ERRORLOG option if you want to see the error messages on the screen and save them in a file.

Example

The following example specifies an intermediate level of error reporting:

```
[OPTIONS]
WARNING=2
```

See Also

ERRORLOG Option, [OPTIONS] Section, REPORT Option

[WINDOWS] Section

Syntax

[WINDOWS]

window-name="caption", (x-coord, y-coord, width, height), window-state,
(scrolling-RGB), (nonscrolling-RGB), ontop-state . . .

The [WINDOWS] section defines the size, location, and colors for the primary Help window and any secondary window types used in a Help file. You can define as many as five secondary window types.

Parameters

window-name — Specifies the name of the window that uses the defined attributes. For the primary Help window, this parameter is main. For a secondary window, this parameter may be any unique name (other than main) with as many as eight characters. Any jumps that display a topic in a secondary window give this window name as part of the jump.

caption — Specifies the text that appears in the title bar of the window. The caption can have as many as 50 characters. If no caption is specified, Windows Help is displayed. Generally, to set the caption of the main window, you use the TITLE option in the [OPTIONS] section of the Help project file. If the caption specified here differs from the TITLE option, the [WINDOWS] caption determines the title displayed in the title bar.

x-coord — Specifies the x-coordinate, in Help units, of the windows upper-left corner. The horizontal position is defined in terms of Windows Helps 01023 coordinate system. (Help always assumes the screen is 1024 units wide, regardless of resolution. For information about how to determine actual coordinates for different video resolutions, see the Comments section.) The x-coordinate is relative to the upper-left corner of the screen, which is 0,0.

y-coord — Specifies the y-coordinate, in Help units, of the windows upper-left corner. The vertical position is defined in terms of Helps 01023 coordinate system. (Help always assumes the screen is 1024 units wide, regardless of resolution. For information about how to determine actual coordinates for different video resolutions, see the Comments section.) The y-coordinate is relative to the upper-left corner of the screen, which is 0,0.

width — Specifies the windows default width in Helps 01023 coordinate system.

height — Specifies the windows default height in Helps 01023 coordinate system.

window-state — Specifies how the window is sized when Windows Help first opens it. This parameter can be one of the following values.

Value	Meaning
0	Size the window according to the values specified in the x-coord, y-coord, width, and height parameters. The default is 0 or normal size.
1	Maximize the window, and ignore the x-coord, y-coord, width, height, and ontop-state parameters given in the type definition.

scrolling-RGB — Specifies the background color for the windows scrolling region. Colors are given as standard RGB valueswhere RRR, GGG, and BBB are three-digit numbers in the range 0 to 255 representing the red, green, and blue components of the color. If this parameter is not given, Help uses the default Windows system color specified by the end-user in Control Panel.

nonscrolling-RGB — Specifies the background color for the windows nonscrolling region (if any). Colors are given as standard RGB valueswhere RRR, GGG, and BBB are three-digit numbers in the range 0 to 255 representing the red, green, and blue components of the color. If this parameter is not given, Help uses the default Windows system color specified by the end-user in Control Panel.

ontop-state — Specifies whether a secondary window type stays on top of other windows. The main Help window cannot be authored as a topmost window. This parameter can be one of the

following values.

Value	Meaning
0	Display the window normally, not on top of other windows. The default is 0 or normal behavior.
1	Display the window on top of other windows. If this value is given, the user cannot change the window behavior using the Always On Top command in Help.

Comments

Window attributes defined in the [WINDOWS] section follow these rules:

- A single comma may be substituted for an entry or a group of entries enclosed by parentheses. Preceding commas are required if you want to use the default settings. Trailing commas are optional. (See the main example below.)
- Windows attributes are set for the Help windows whenever a file containing predefined attributes is opened.
- If a user performs an operation, such as an interfile jump, that opens a file with predefined window attributes, the Help windows will adjust to the new settings, change size or location, for example.
- When an application requests Help using the WinHelp function, Help uses the values defined in the [WINDOWS] section to determine window attributes. If the Help file does not include settings in the Help project file, Help uses the default settings.
- When Help is closed, the window settings are updated in the [Windows Help] section of the WIN.INI file.

The Help coordinate system ranges from 0 through 1023 in both directions, so the vertical position plus the height must be less than or equal to 1023. Similarly, the horizontal position plus the width must be less than or equal to 1023. This 1024-by-1024 coordinate system is mapped to the horizontal and vertical resolutions of the video card. To convert from pixels to Windows Help coordinates, you invert the ratio between Helps resolution and the video resolution. Assuming the video card resolution is horiz by vert pixels, and the horizontal and vertical locations (or dimensions) you want are in pixels, the x-coordinate (or width), in Help coordinates, is as follows:

x-coord = pixel location * (1024/horiz)width = number of pixels * (1024/horiz)

The y-coordinate (or height), in Help coordinates, is:

y-coord = pixel location* (1024/vert)height = number of pixels * (1024/horiz)

For example, if you want the windows upper-left corner to appear at horizontal pixel 320 and at vertical pixel 120, and the Help file is being displayed on a standard VGA monitor with 640 by 480 resolution, the x-coord is (320 * (1024/640)) = 512, and y-coord is (120 * (1024/480)) = 256.

Example

The following example defines two windows, the main Help window and a secondary window named picture. The main window definition sets the background color to magenta (128, 0, 128) and leaves the other values empty (for which Help will supply its default values). The secondary window definition sets the caption to Samples, positions it in the upper-left part of the screen (123, 123), and sets the windows width and height to about one-quarter of the screens width and height (256). The window will not be maximized (0). The windows background colors are set to cyan (0, 255, 255) for the scrolling region and light gray (192, 192, 192) for the nonscrolling region. The secondary window will appear on top of other windows when it is open (1).

```
[WINDOWS]
main=, , , (128, 0, 128)
picture="Samples", (123, 123, 256, 256), 0, (0, 255, 255), (192, 192, 192),
1
```

See Also

[OPTIONS] Section, TITLE Option

Chapter 17 Building the Help File

After you create the topic file(s) and Help project file, you can build the Help file. This chapter describes how to build, debug, and test a Help file.

The Windows Help Compiler

To create a Help file from your topic files, you use the Windows Help compiler. The Help compiler is an application that converts RTF source files into a binary Help file that can be displayed by the Windows Help application. Windows Help version 3.1 offers several versions of the Help compiler. The following table lists each version and describes its function.

Compiler	Description
HC30.EXE	The Help compiler that ships with the Microsoft Windows Version 3.0 Software Development Kit. Use this compiler to create Help files compatible with Windows Help 3.0 or 3.1.
HC31.EXE	The Help compiler that ships with the Microsoft Windows Version 3.0 Software Development Kit. Use this compiler to create Help files compatible with Windows Help 3.1.
HCP.EXE	The MS-DOS extended version of the Help compiler. Use this compiler to create large Help files or to make better use of memory. This Help compiler should be able to compile in MS-DOS any file that can be compiled in OS/2.

Choosing the Correct Version for your Help File

Windows version 3.1 includes Windows Help version 3.1 as the system Help application. The Microsoft Windows Version 3.1 Software Development Kit and the Microsoft Developer Network CD-ROM include a batch file (HC.BAT) that gives applications the choice of creating Help files either in version 3.0 or 3.1 format (3.0 being the default). Applications can also choose to ship Windows Help version 3.1 and replace a users system-wide Help application when installing their product on the users computer. All Help files created using the 3.0 Help compiler will display properly in Windows Help version 3.1. However, Help files created using the 3.1 Help compiler cannot be displayed with Windows Help version 3.0. Therefore, when deciding which version of the Help compiler to use to build your Help files, follow these guidelines:

- Your application requires Microsoft Windows version 3.1.

  If your application requires Windows version 3.1 and will not run in Windows version 3.0, you should use the 3.1 Help compiler to build your Help files. Your application does not need to include the version 3.1 Windows Help application on the product disk(s) nor should it install the Help application on the users computer. In this scenario, you can use any 3.1 feature in your Help file.
- Your application is designed to work with Microsoft Windows version 3.0 and 3.1 and does not ship the version 3.1 Windows Help application on the product disk(s).

  In this case, you must use the 3.0 Help compiler to build your Help files to ensure that they will display properly on both versions of Windows Help. This also means that you cannot use any feature specific to Windows Help version 3.1 in your Help file.
- Your application is designed to work with Microsoft Windows versions 3.0 and 3.1, and does ship the version 3.1 Windows Help application on the product disk (s).

  In this case, you should use the 3.1 Help compiler to build your Help files. When installing your product, the setup program should detect the version number of the users Help application and replace it if it is version 3.0. In this scenario, you can use any 3.1 feature in your Help file.

Building a Help File

Building the Help file is quite simple.

To build a Help file

1. Edit the Help project file and enter the following lines in the [OPTIONS] section so you can monitor the build process on your screen:

   ```
   WARNING=3
   REPORT=ON
   ```

   When the WARNING option is set as above, the compiler reports all warnings and errors it encounters during the build. When the REPORT option is set, the compiler displays its messages as it performs the different phases of the build, including compiling the file, resolving jumps, and verifying browse sequences.
2. Change to the directory that holds the Help project file.
3. Type the HC command using the following syntax:

   ```
   HC projectfile
   ```

   The projectfile is the name of your Help project file. The compiler assumes that your Help project file has an .HPJ file extension. If it does, or if it has no extension, you can type just the filename, as in the following example:

   ```
   hc myhelp
   ```

   The compiler displays periods on the screen to show its progress and displays warnings and error messages when it encounters problems. For more information, see Chapter 18, Help Error Messages.

If everything goes smoothly, the Help compiler creates a Help file in the project directory and then returns to the MS-DOS prompt. The Help file has the same name, but with an .HLP extension, as the Help project file. For example, the command in the previous example would produce MYHELP.HLP.

Building Large Help Files

The number of topic files affects the amount of time required to complete the build. Help files consisting of a small number of files compile quickly; Help files with large numbers of files can require several hours to compile. For large Help files, you might want to start the build process at the end of the day and let it compile at night. (You can also start a build at the MS-DOS prompt within Windows, and then change its settings to run in the background while you use other applications. For more information, see the Microsoft Windows Users Guide.)

If your Help file is too large to build under MS-DOS, you can compile under OS/2 or use the DOS Protected Mode Interface (DPMI), or extended, version of the Help compiler (HCP.EXE). The extended version of the compiler uses the same syntax as HC.EXE, but it makes better use of memory. All instructions for building under MS-DOS apply to these other versions unless otherwise indicated.

Getting Around Memory Problems

Depending on the size of your Help file and the computer you are using to build on, you may encounter out-of-memory problems during compilation. Although there is no single solution that addresses all out-of-memory conditions, the following list offers some suggestions you might try to ensure that your Help file will build successfully:

- If possible, compile without compression.

  The Help file will build fastest when no compression is used. Compressing the Help file can take up to ten times as long to build as an uncompressed file, depending on the content. In most cases, you can build with no compression while youre developing the Help file, and use compression for only the final builds of the product.
- Compile in a Windows MS-DOS box rather than from the MS-DOS prompt.

  In most instances, you are better off running the Help compiler in a Windows MS-DOS box because HC31.EXE uses extended memory, which is not always available in regular MS-DOS but is available in an MS-DOS box under enhanced mode Windows. To verify that you are using the enhanced version of the compiler, you can run the compiler and look at the copyright message. It should include the word extended after the version number.
- Close other applications that are also running.

  Frequently you want to have Word for Windows running so that you can edit your RTF source files and then do a quick compile. However, if you have a limited amount of memory in your computer, you may have to close Word for Windows or all other running applications to free enough memory to complete the build.
- Compile the Help file under OS/2 or use HCP.EXE under MS-DOS.

  OS/2 does not have the MS-DOS 640K memory limitation and manages available memory much more efficiently than does MS-DOS. If you do not have an OS/2 machine, you can use the DPMI version of the Help compiler (HCP.EXE), which should produce similar results under MS-DOS.
- Be sure that you have sufficient free disk space.

  Sometimes the Help compiler can run out of memory if the disk runs out of space. Compare the amount of free disk space you have with the size of your topic files. If you are compiling with compression, you may need as much free disk space as five times the size of your source files. This requirement can be more or less (usually less), depending on the Help file. Compiling without compression also reduces the amount of disk space the compiler requires.
- Replace any inline bitmaps in the topic files with bitmap references.

  The 3.1 Help compiler fragments memory when compiling bitmaps under MS-DOS, which often causes out-of-memory problems. Therefore, if you have pasted any bitmaps directly into the topic files, you should replace them with bitmap references. If the pasted-in bitmaps are larger than 64K, the compiler will always run out of memory during the build. If you have severe memory problems, you may have to limit the number of bitmaps you include in the Help file.
- If you are building with compression, perform partial builds until you achieve a successful build of the Help file.

  To perform a partial build, you edit the Help project file and comment out specific lines (topic files) in the [FILES] section. Commenting out topic files will produce an incomplete Help file with unresolved jumps and other problems; however, the compiler may be able to create a .PH file with compression information. After you have a phrase file, you can build the Help file again and include all the topic files. (Remember to set OLDKEYPHRASE=yes in the Help project file before building the Help file.) The compiler will create a working Help file with some level of compression. If necessary, you can use the phrase file with several topic file subsets to compress the complete Help file.
- If you are building a version 3.0 Help file with compression, use the 3.1 Help compiler to create the phrase file.

  The 3.0 Help compiler often runs out of memory when creating a phrase file (.PH) for large Help files, even when compiling under OS/2. That is because creating the phrase file is a very memory intensive process, and phrase files over 32K cause the build to fail. However, this is not usually a problem when using the 3.1 Help compiler. So, if you are having trouble compiling a file with the 3.0 Help

compiler, you can delete the 3.0-generated phrase file and use the 3.1 compiler to generate a phrase file. After you have a phrase file, you can build the Help file again with the 3.0 compiler. (Remember to set OLDKEYPHRASE=yes in the Help project file before building the Help file so the 3.0 compiler will not try to create a phrase file.)

Speeding Up the Build

When compiling large Help files, or when compiling many Help files using a batch file or make file, the build can take several hours. This is often true even if you have a fast machine, large hard disk, and plenty of memory. Although you cannot completely avoid slow compile times with large Help projects, you can do a few things to achieve faster builds.

- If possible, compile without compression or use medium compression.

  The Help file will build fastest when no compression is used. Using the Compress=high option can require 4 to 10 times as long to build as uncompressed files, depending on the content. Another option is to use medium compression. The difference in size between high and medium compression is small, but the build is a lot faster with medium compression.
- Compile the Help file on an OS/2 machine with HPFS.

  OS/2s high-performance file system (HPFS) is much faster than the MS-DOS file allocation table (FAT). If you have a machine that you can configure with OS/2, you might try doing some performance tests to see if your Help file builds faster under OS/2.
- Create a RAM drive and use it to build the Help file.

  If your Help file is not too large and you have sufficient memory in your build machine, you can use a RAM drive to build your Help file. For this method to work, the RAM drive should be roughly two and one-half times larger than the Help file you are building. For example, if the Help file is about 2 MB, the RAM drive should be at least 5 MB.
- Divide the Help file into separate topic files, and then build and test these files independently.

  The easiest way to compile individual files separately is by editing the Help project file and commenting out lines in the [FILES] section for topic files you dont want included. To comment out a line, insert a semicolon (;) at the beginning of the line. Remember, however, that if you comment out particular topic files that contain hot spots referencing excluded topics, the compiler will display warnings as the Help file builds. You can also use the View Topic, View Selection, and View File commands in the Help Authoring Templates (WHAT30.DOT or WHAT31.DOT) to perform partial builds.

Compiling a 3.0 Help File Under 3.1 Help

You need to make some changes to get your 3.0 Help file to compile under Windows Help version 3.1.

To prepare a 3.0 source file for compiling under version 3.1

1. Change INDEX= to CONTENTS= in the Help project file.
2. Change side-by-side paragraphs (if any) to tables.
3. Check to make sure there are no hidden paragraph marks or hidden page breaks.

For more information about the CONTENTS option, see Chapter 16, The Help Project File. For more information about creating tables, see Chapter 7, Formatting Topics.

Performing Simultaneous Builds

Generally, it is not a good idea to run simultaneous builds on the same machine, and we do not recommend it. However, if you want to attempt a simultaneous build, you can do so at your own risk. As a minimum, take the following precaution: set the TMP environment variable to a different directory for each MS-DOS or OS/2 screen running the Help compiler. Otherwise, temporary filenames that the Help compiler creates during the build may collide, producing very bad results.

Building Identical Help Files

Sometimes (when performing quality assurance tests, for example) it may be important to compare the binary versions of a Help file compiled from the same source files. However, if the Help compiler displays any error messages on the screen (other than progress periods), subsequent builds of the Help file will not compare to previous ones, even though they are using exactly the same source files. (The Help files work the same; they just arent identical.)

This happens because Help files include a time stamp indicating when they were compiled. In addition, the Help compiler writes uninitialized parts of memory buffers to disk every time a screen message is printed. Therefore, Help files will not be identical when compiled from the same source.

If binary compares are important to you, you can try this workaround (which is not guaranteed to work).

To compile a Help file that may be recreated exactly

1. Create a batch file that:
   - Zeros the machines memory.
   - Resets the clock.
   - Compiles the Help file.
2. Run the batch file to create the identical version.

Building Files Saved in Word for the Macintosh version 4.00

When the Help Compiler processes RTF files created by Microsoft Word for the Macintosh, colon (:) characters may disappear from the built Help file. That is because version 4.00 of Word for the Macintosh prefixes each colon with a backslash character (\), which makes each colon a special RTF control word. Because the Help Compiler ignores these RTF statements, the colons drop out of the compiled Help file.

There are two methods to correct this problem:

- Contact Microsoft Customer Service to update your copy of Word for the Macintosh to version 4.00a or later.
- Perform the following procedure to remove the backslash characters from the RTF file.

To remove backslash characters from the RTF file

1. Open the file in Word for the Macintosh version 4.00.
2. When asked if Word should interpret RTF text, choose No.
3. Use the Change feature to find each occurrence of \: (backslash, colon) and replace it with : (a plain colon).
4. Save the file as Text Only and close the document.
5. Recompile the Help file.

Building Help Projects with Multiple Files

In some situations, you may want to create a Help project that has multiple Help files. A multi-file Help project is one in which several .HLP files work together to provide users complete information about a product. Even though each .HLP file is separate, users can jump from one Help file to another just as they can within a single Help file.

The process for building a multi-file Help project is slightly different from the process for building a single Help file.

To build a multi-file Help project

1. Set up a directory structure appropriate for your Help project.
2. Create a Help project file (.HPJ) for each Help file in the project.
3. Create a batch file to build all the files in the project.

The following sections examine each of these steps in more detail.

Setting Up the Project Directories

First, create a directory structure to fit the organization of your Help project. Figure 17.1 shows one possible structure.

This sample Help project consists of three Help files: CPCD.HLP, FMCD.HLP, and PMCD.HLP. As shown in the illustration, the HYPER project directory is a subdirectory of the directory where you installed Windows Help. The HYPER subdirectory contains the three .HPJ files for the three Help files in the project and a batch file to build the entire project.

HYPER also has a subdirectory for each Help file in the project, a subdirectory for error files, and a subdirectory for archiving built files. In turn, each Help file subdirectory has two subdirectories for the different file types used in the Help file: RTF for topic files and ART for bitmap files.

Creating Help Project Files for Each Help file

After you have set up the directory structure for a multi-file Help project, you create a Help project file for each Help file in the project. The exact contents of the Help project file will vary depending on the type of Help system you are creating, but at a minimum it should include an [OPTIONS] section and a [FILES] section. The Help project file for one of the Help files in this sample project might look like this:

```
[OPTIONS]
ERRORLOG=PROGMAN.BUG
ROOT=C:\HYPER\PROGMAN
BMROOT=.\PROGMAN\ART
CONTENTS=cont_progman
TITLE=Program Manager
COMPRESS=OFF
REPORT=ON
WARNING=3
[FILES]
.\RTF\PROGMAN.IDX
.\RTF\PROGMAN.CON
.\RTF\PROGMAN.CMD
.\RTF\PROGMAN.HOW
.\RTF\PROGMAN.KBD
.\RTF\PROGMAN.QCK
```

Notice the following points:

- The ERRORLOG option appears first so that all error messages will be recorded in the error file.
- The ROOT option gives a full path to specify the project root directory. In this example, the root directory is the PROGMAN subdirectory of the \HYPER project directory.
- The BMROOT option gives a path relative to the project directory. The full path for the bitmap root would be \HYPER\PROGMAN\ART.
- Similarly, the relative paths in the [FILES] section refer to the project root directory.
- For the first few builds of a new Help project, it is usually more important to build quickly than to get good compression in the built file, so compression is turned off.
- To make debugging the Help files easier, the [OPTIONS] section includes the REPORT and WARNING options.

Building the Individual Help files

If you set up your Help project files as described in the previous section, you should be able to build each Help file just as you would for a single-file Help project. Simply change to the project directory and enter the HC command for the Help file you want to build.

For example, to build PMCD.HLP, you would do the following:

1. Change to the \HYPER project directory using the following command:

   ```
   cd \HYPER
   ```

2. Enter an HC command to build the Help file:

   ```
   hc pmcd.hpj
   ```

To build the other Help files, just repeat the same process for each of the three .HPJ files in the project. All the .HLP files you build will be created in the project directory (in this example, in the \HYPER directory).

Creating a Batch File

You can also create a batch file to build all the Help files in the project. Using a batch file is optional, but it simplifies the build process. The following example shows what a simple batch file for this Help project might look like:

```
@echo off
echo delete files from last build
del *.hlp
del *.ph
del *.bug
del hlperr.all
echo building CONTROL PANEL
hc cpcd.hpj
copy control.bug \hyper\error
copy cpcd.hlp \hyper\archive
echo building FILE MANAGER
hc fmcd.hpj
copy winfile.bug \hyper\error
copy fmcd.hlp \hyper\archive
echo building PROGRAM MANAGER
hc pmcd.hpj
copy progman.bug \hyper\error
copy pmcd.hlp \hyper\archive
echo Building composite error file
copy *.bug hlperr.all
```

The batch file begins by deleting all the files produced during the previous build. This step is optional, but it ensures that the project directory is cleaned up before each build. The next three sections build each individual Help file and place a copy of the resulting error file in the \ERROR subdirectory and a copy of the built Help file in the \ARCHIVE subdirectory. The final section of the batch file creates a single error file from the three individual error files so that all the error messages from the build can be viewed or printed more easily.

Depending on the Help system, you may want to create a master Contents topic that accesses topics in each of the separate Help files. Or you may simply want to create interfile jumps that allow users to move through the information in each of the files. How you interconnect the Help files in a multi-file Help system is up to you.

Debugging the Help File

This section examines ways you can investigate problems that might arise during the build process.

The Help compiler displays messages when it encounters errors in building the Help file. (Chapter 18, Help Error Messages, contains a list of messages that might appear during the build process.) Whenever possible, the compiler displays the name of the file that contains the error and the number that identifies the specific line of the Help project file or the topic that produced the error.

Because topics are not actually numbered, the topic number given with an error message refers to that topics sequential position in the RTF file (first topic, second topic, and so on). These numbers may be identical to the page numbers shown by your word processor, depending on the number of lines you have assigned to the hypothetical printed page. Remember that topics are separated by hard page breaks, even though there is no such thing as a page in the Help system.

A single error condition in the topic file can cause many error messages. For example, an incorrectly identified topic causes an error every time a jump term refers to the correct topic identifier. Such a mistake is easily fixed by correcting the footnote that contains the wrong context string. In general, if you get an error (for example, an unresolved jump), correct the error(s) and recompile.

> Note: An easy way to avoid build errors is to make sure that the compiler can access all the RTF files and all the graphic files needed to build the Help file.

Displaying Error Messages on the Screen

The REPORT option tells the compiler to display messages on the screen while it is building. These messages indicate when the compiler is performing the different phases of the build, including compiling the file, resolving jumps, and verifying browse sequences.

The following option turns on the message display:

```
REPORT=ON
```

Warning Message Reporting

You can specify the level of warnings reported by the compiler during the build process. Warning messages alert you to conditions that are not serious enough to stop the build but that might cause problems in your Help file.

The WARNING option in your Help project file controls the amount of warning information that the compiler displays. You set this in the [OPTIONS] section, using the following command:

WARNING=level

The following table describes the three reporting levels for the build process.

Level	Information reported
1	Only the most severe warnings
2	Severe and less serious warnings
3	All warnings

The following example specifies the intermediate level of error reporting:

```
[OPTIONS]
WARNING=2
```

Redirecting Errors to a File

By default, the Help compiler displays all errors on your screen. However, you can redirect errors to a file so that you can examine the errors more carefully and fix problems in your Help file. You can examine the redirected file using any ASCII text editor. Use the standard MS-DOS redirection syntax to redirect error and status messages to a file; for example:

```
hc myhelp.hpj > errors.txt
```

This command places all the messages sent during the build process into the ERRORS.TXT file. You can also enter the following in the [OPTIONS] section of your Help project file:

```
ERRORLOG=error-file
```

The Help compiler places compilation errors in the file given by error-file. The error-file can be an absolute or relative path if the file is in a directory other than the Help project root directory. If you use the ERRORLOG option, it should be the first line in the [OPTIONS] section.

Using ERRORLOG has two advantages over redirecting error messages:

- Using redirection, the Help compiler puts unnecessary characters (including the periods it displays to indicate the progress of the build) and status messages along with error messages in the output file. Using ERRORLOG, you get only the error messages.
- Using redirection, the Help compiler doesnt display error messages on the screen, so you cant monitor the messages during the build. Using ERRORLOG, the Help compiler displays the error messages on your screen and writes them to the file.

Testing the Built Help File

After youve successfully built the Help file, youll want to look at the finished product to see if there are problems in displaying the topics.

To test a built Help file

1. From within Windows, start Windows Help by clicking the Windows Help icon or using the Run command from the Program Manager.
2. From the File menu, choose Open and open the .HLP file youve built.
3. Test all components to ensure that text and graphics have been formatted correctly, no typographical errors appear, and that elements such as jumps, keywords, browse sequences, pop-up windows, and macros work correctly.

Using the Keyboard Debug Option

If you want to use a keyboard option to debug your Help file, you can add a switch to the WIN.INI file that lets you display every topic in a Help file in sequential order, whether or not you include browse sequences. The order is determined by the physical location of each topic in the compiled Help file.

To implement the keyboard debug option

1. Add the following line to the [Windows Help] section of your WIN.INI file:

   ```
   SeqTopicKeys=1
   ```

2. Restart Windows and open the Help file you want to test.
3. Press SHIFT+CTRL+HOME to display the first topic in the Help file.
4. Press SHIFT+CTRL+RIGHT ARROW to display each successive topic in the Help file.

> Note: If the Help file contains a blank topic, it means that one of your RTF topic files ends with a hard page break.

Chapter 18 Help Error Messages

When you build a Help file, the compiler displays messages on the screen, indicating the progress of the build and any errors or problems that may occur. This chapter describes the build and error messages for the version 3.1 and 3.0 Help compilers.

Help Compiler 3.1 Error Messages

This section lists the error messages that the version 3.1 Help compiler version 3.1 displays when it encounters problems building a Help file. Whenever possible, the Help compiler displays the name of the file that contains the error, as well as the number used to identify the specific line of the Help project file or the topic that produced the error. Since topics do not contain numbers, the topic number specified in the error message refers to the topics sequential order in the topic file. In most cases, the topic number given in the error message will match the topics page number in the Word for Windows document.

The Error Message File

In Windows Help version 3.1, error messages are stored in a text-only ASCII file, rather than in the compilers resource file (as in version 3.0). Because Help error messages are stored independently of the compiler, Help authors can customize the error messages in two ways. They can:

- Change the text that appears in the error message.
- Insert tabs into the error messages.

When customizing error messages, Help authors cannot:

- Change the number assigned to the error message.
- Change the order of the error message in HC.ERR.
- Add, delete, or change the order of parameter (%) tags in the error messages.

The default name of the error message file is HC.ERR; however, this name may vary according to the specific version of the compiler that you are using to build the Help file: HC.ERR, HC31.ERR, or HCP.ERR, for example. To work correctly, the error message file must be located in the same directory as the compiler and its name must match the name of the compiler you are using. For example, if you rename HC31.EXE to HC.EXE, you must rename the HC31.ERR error file to HC.ERR. If the error message file is missing, the build will fail, and the compiler will display this message:

Warning 0101: Cannot open filename.ERR in filename.EXE directory.

If you see this error message, check to be sure you are running matched versions of the compiler and error message file and that the files are in the same directory. Also, be sure you are using the version of the compiler you think you are using. If the problem persists, reinstall the error message file from the original source disk.

Interpreting Error Messages

The Help compiler displays either warning or fatal error messages. A warning message begins with the word Warning. A warning error indicates a problem encountered during the build that is not severe enough to prevent the Help compiler from completing the build. Therefore, a build with warnings produces a Help file that Windows Help should be able to open, but the file may contain problems in certain topics, such as missing graphics or hot spots that arent hot. Fatal error messages begin with the word Error. A fatal error indicates a problem that prevents the Help compiler from creating a Help file.

As stated previously, you use the REPORT option to display error messages on the screen and the WARNING option to specify the amount of warning information you want the Help compiler to provide. The Help compiler always reports fatal errors, regardless of the current warning level or report option specified in the Help project file, since no usable Help file results from the build.

While the Help compiler processes the Help project file, it ignores lines that contain errors and attempts to continue with the build. This means that errors encountered early in a project file may result in many more errors being reported as the build continues.

Similarly, when the Help compiler processes the RTF topic files, it reports any errors it encounters and, if the errors are not fatal, the compiler continues with the build. A single error in a topic file may result in more than one error message being displayed by the compiler. For instance, a typographic mistake in a topics context string will cause an error to be reported every time the compiler encounters a reference to the correct topic identifier. On the other hand, some warning errorssuch as hot spots with the same hidden text codes (broken jumps) or any bitmap file errors, including File not found are reported once per file, regardless of the number of times the problem occurs in the source file.

Error Message Categories

In Windows Help version 3.0, error messages included the prefixes P and R to indicate whether the error occurred in the topic file or in the Help project file. In Help version 3.1, theses prefixes have been replaced by a numbering scheme that differentiates build errors into eight categories.

Error message numbers have four digits. The first one or two digits identify the message category. The third and fourth digits establish the message order within the category. The message number prefixes and the categories they identify are listed in the following table.

Prefix	Error category
1	Problems with files used to build the Help file
2	Problems with the Help project file
3031	Problems with build tags or build tag expressions
3536	Problems with Help macros
4041	Problems with context strings
4245	Problems with topic footnotes
4647	Problems with RTF topic files
5	Other problems

You might encounter errors other than the build errors described here. For more information, see the README.TXT file.

Error Message Reference

The Error Message Reference lists all error messages by category and in the order that they appear in the HC.ERR file. The wording of the error messages is also the same as in HC.ERR, except that variables (% tags) are replaced with descriptive words. Error message descriptions contain one or more of the following types of information.

Heading	Information
Problem	Explains what the error message means.
Result	Describes how the Help compiler handles this particular error.
Solution	Offers suggestions about how to correct the problem.

File Errors

The following messages result from problems with the files used to build the Help file.

1019 Project file extension cannot be .HLP or .PH.

PROBLEM: You cannot specify a Help project file with an.HLP or a .PH extension. Project files must use the .HPJ extension.

RESULT: The compiler aborts the build.

SOLUTION: Rename the Help project file, and then recompile.

1030 File name exceeds limit of 259 characters.

PROBLEM: The combined length of the path and filename must not be more than the MS-DOS limit of 259 characters.

RESULT: The compiler ignores the file.

SOLUTION: Shorten the path, and then recompile.

1079 Out of file handles.

PROBLEM: The compiler does not have enough available file handles to continue the build. This error is likely to occur if you are compiling in an MS-DOS box or if you are using Help Author and have Word for Windows running at the same time.

RESULT: The compiler aborts the build.

SOLUTION: If possible, increase the FILES setting in the CONFIG.SYS file to Files=50. Then reboot your computer and recompile.

1100 Cannot open file filename: permission denied.

PROBLEM: You do not have the required file privileges to open the requested file. Requested files must have at least read privilege to be opened.

RESULT: The compiler ignores the file.

SOLUTION: Change your access privileges if you are working on a secured network, or change the file attributes to read only or read/write.

1150 Cannot overwrite file filename.

PROBLEM: The Help compiler cannot overwrite files with a read-only attribute. For example, this error occurs if you rebuild an existing .HLP file and the .HLP file is read-only.

RESULT: The compiler aborts the build.

SOLUTION: Change the files read-only attribute if you want the Help compiler to overwrite the file; otherwise, rename the file.

1170 File filename is a directory.

PROBLEM: A directory in the Help project directory has the same name as the requested Help file. This is an MS-DOS file error.

RESULT: The compiler ignores the file.

SOLUTION: Move or rename the directory or the Help project file, and then recompile.

1190 Cannot use reserved MS-DOS file name filename.

PROBLEM: The file has a reserved MS-DOS filename, such as COM1, LPT2, or PRN. This is an MS-DOS file error.

RESULT: The compiler ignores the file.

SOLUTION: Rename the file, and then recompile.

1230 File filename not found.

PROBLEM: The specified file could not be found or is unreadable. This is an MS-DOS file error or an

out-of-memory condition.

RESULT: If the file not found is an RTF topic file, the compiler ignores the file. If the file is a bitmap file, the built Help file displays the Unable to display picture message in the topic instead of the bitmap.

SOLUTION: Check to see if the file exists, and also check the amount of available memory.

1292 File filename is not a valid bitmap.

PROBLEM: The specified bitmap file could not be found or is not in a recognizable bitmap format. This is an MS-DOS file error or an out-of-memory condition.

RESULT: The Help file displays the Unable to display picture message in the compiled topic instead of the bitmap.

SOLUTION: Check to see if the file exists; if it does, check its format. If necessary save the file again in your paint or draw program, and then recompile.

1319 Disk full.

PROBLEM: There is not enough disk space to write the Help file to disk.

RESULT: The compiler aborts the build.

SOLUTION: Create more space on the destination disk, and then recompile.

1513 Bitmap name filename duplicated.

PROBLEM: The [BITMAPS] section contains duplicate bitmap names.

RESULT: The compiler uses the first occurrence of the name.

SOLUTION: Rename one of the duplicate bitmap files, and then recompile.

1536 Not enough memory to compress bitmap filename.

PROBLEM: The specified bitmaps cannot be compressed due to insufficient memory, or the compiler does not have enough memory to check bitmap files. The compiler also displays this error if any of the specified bitmaps are segmented hypergraphics (.SHG files), since the .SHG files may contain searchable hot spots.

RESULT: The Help compiler does not check or compress the bitmap files.

SOLUTION: Free more memory and then recompile. Refer to the Getting Around Memory Problems section in Chapter 17 for suggestions on how to free more memory for the build. If you cant overcome your memory constraints, you may have to compile without compression or remove some of the graphics in the topic files.

1000 UNKNOWN ERROR, Contact Microsoft Product Support Services

PROBLEM: This error message is the equivalent of an assertion failure.

RESULT: The compiler aborts the build.

SOLUTION: Contact Microsoft Product Support Services.

Project-File Errors

The following messages result from errors in the Help project file used to build a Help file.

2010 Include statements nested more than 5 deep.

PROBLEM: The #include statement on the specified line exceeds the maximum of five include levels.

RESULT: The compiler aborts the build.

SOLUTION: Simplify the #include statement so that it nests fewer than include levels, and then recompile.

Syntax Errors

The following messages result from syntax errors in the Help project file.

2030 Comment starting at line linenumber of file filename unclosed at end of file.

PROBLEM: The compiler has unexpectedly come to the end of the Help project file. There may be an open comment in the Help project file or in an include file.

RESULT: The compiler aborts the build.

SOLUTION: Close any open comments within the Help project file, and then recompile.

2050 Invalid #include syntax.

PROBLEM: The #include statement is missing the required filename.

RESULT: The compiler ignores the #include statement.

SOLUTION: Correct the syntax, and then recompile. The correct #include syntax is:

#include <filename>

2091 Bracket missing from section heading [sectionname].

PROBLEM: The right bracket (]) is missing from the specified section heading.

RESULT: The compiler ignores the section heading on this line.

SOLUTION: Insert the right section heading bracket, and then recompile.

2111 Section heading missing.

PROBLEM: The section heading on the specified line is not complete, or the first entry in the Help project file is not a section heading.

RESULT: The compiler ignores the section heading on this line and all succeeding lines until it encounters a valid section heading.

SOLUTION: Insert a valid section heading at the appropriate place in the Help project file, and then recompile.

2131 Invalid OPTIONS syntax: option=value expected.

PROBLEM: An entry in the [OPTIONS] section is missing the equal sign, or it has some other syntax error.

RESULT: The compiler ignores the invalid option on this line.

SOLUTION: Check the syntax of each option used in the [OPTIONS] section. Fix the incorrect option, and then recompile. For the correct syntax, refer to Chapter 16, The Help Project File.

2141 Invalid ALIAS syntax: context=context expected.

PROBLEM: An entry in the [ALIAS] section is missing the equal sign, or it has some other syntax error.

RESULT: The compiler ignores the invalid alias string on this line.

SOLUTION: Check the syntax of the entries in the [ALIAS] section. Fix the incorrect entry, and then

recompile. For the correct syntax, refer to Chapter 16, The Help Project File.

2151 Incomplete line in [sectionname] section.

PROBLEM: An entry in the specified section is incomplete.

RESULT: The compiler ignores the incomplete line.

SOLUTION: Check the section for any incomplete lines and correct them. Then recompile.

2171 Unrecognized text.

PROBLEM: The compiler found some text in the Help project file that it cant recognize.

RESULT: The compiler ignores the line with the unrecognized text.

SOLUTION: Remove the unsupported text from the Help project file, and then recompile.

2191 Section heading [sectionname] unrecognized.

PROBLEM: The Help project file includes a section heading that the compiler does not recognize or does not support.

RESULT: The compiler ignores the section heading.

SOLUTION: Check all bracketed headings in the Help project file for typographical or other errors. Fix the section heading, and then recompile.

2214 Line in .HPJ file exceeds length limit of 2047 characters.

PROBLEM: There is a line in the Help project file that exceeds the maximum length of 2047 characters.

RESULT: The compiler ignores the line.

SOLUTION: Shorten the line, and then recompile.

General Section Errors

The following messages result from general errors in the different sections of the Help project file.

2273 [OPTIONS] should precede [FILES] and [BITMAPS] for all options to take effect.

PROBLEM: The [OPTIONS] section should be the first section in the Help project file so that all the options will be used during compilation. Also, if the ERRORLOG option is used, it should be the first line in the [OPTIONS] section.

RESULT: The Help compiler does not use the ROOT or BMROOT option to locate topic or bitmap files that are listed before the [OPTIONS] section. Also, if the ERRORLOG option is used, the compiler does not write in the log file any errors that were generated before it processed the ERRORLOG option.

SOLUTION: Move the [OPTIONS] section to the beginning of the Help project file, and then recompile.

2291 Section sectionname previously defined.

PROBLEM: The compiler found more than one section with the specified heading in the Help project file.

RESULT: The compiler ignores the duplicate section and continues from the next valid section heading.

SOLUTION: Remove or rename one of the duplicate section headings, and then recompile.

2305 No valid files in [FILES] section.

PROBLEM: The [FILES] section is empty or contains only invalid files.

RESULT: The compiler aborts the build.

SOLUTION: Check the [FILES] section to see if it lists any files. If it does, make sure that the files listed are saved as RTF. After you have a list of valid topic files to include in the build, recompile.

[ALIAS] or [MAP] Section Errors

The following messages result from errors in the [ALIAS] or [MAP] section of the Help project file.

2322 Context string context_name cannot be used as alias string.

PROBLEM: A context string that has been assigned an alias cannot be used later as an alias for another context string. That is, you cannot map a=b and then c=a in the [ALIAS] section.

RESULT: The compiler ignores the attempted reassignment on this line.

SOLUTION: Correct the alias string mapping, and then recompile.

2331 Context number already used in [MAP] section.

PROBLEM: A context number in the [MAP] section of the Help project file was previously mapped to a different context string.

RESULT: The compiler ignores the line with the duplicate context number.

SOLUTION: Check all context numbers to make sure that they are unique. Remove or reassign any duplicate context numbers, and then recompile.

2341 Invalid or missing context string.

PROBLEM: A line in the [MAP] or [ALIAS] section is missing a context string before the equal sign.

RESULT: The compiler ignores the line with the invalid context string.

SOLUTION: Check all context strings to the left of the equal sign. Correct any missing or invalid entries, and then recompile.

2351 Invalid context identification number.

PROBLEM: A context number in the [MAP] or [ALIAS] section is empty or contains invalid characters.

RESULT: The compiler ignores the line with the invalid context number.

SOLUTION: Check the format of all context numbers. Context numbers can be specified as decimal or hexadecimal. Correct any invalid numbers, and then recompile.

2362 Context string context_name already assigned an alias.

PROBLEM: The specified context string has already been assigned an alias in the [ALIAS] section. A context string can have only one alias. That is, you cannot map a=b and then a=c in the [ALIAS] section.

RESULT: The compiler ignores the attempted reassignment on this line.

SOLUTION: Correct the alias string mapping, and then recompile.

2372 Alias string aliasname already assigned.

PROBLEM: You cannot alias an alias. That is, an alias string cannot, in turn, be assigned another alias. You cannot map a=b and then b=c in the [ALIAS] section.

RESULT: The compiler ignores the attempted reassignment on this line.

SOLUTION: Correct the alias string mapping, and then recompile.

[WINDOWS] Section Errors

The following messages result from errors in the definitions of secondary window types given in the [WINDOWS] section of the Help project file.

2391 Limit of 6 window definitions exceeded.

PROBLEM: The maximum number of window definitions is one main-window definition and five secondary-window definitions.

RESULT: The compiler ignores the additional window definitions.

SOLUTION: Reduce the number of window definitions in the Help project file to five or fewer, and then recompile.

2401 Window maximization state must be 0 or 1.

PROBLEM: The value of the window-state parameter is not zero or 1.

RESULT: The compiler ignores the window definition.

SOLUTION: Correct the entry in the Help project file, and then recompile. The window-state value must be either 0 (normal) or 1 (maximized).

2411 Invalid syntax in window color.

PROBLEM: A window definition in the Help project file has incorrectly defined a window color. A window color definition consists of three decimal numbers, representing the red, green, and blue (RGB) components of the color. The three numbers must be enclosed in parentheses and separated by commas.

RESULT: The compiler ignores the window definition.

SOLUTION: Correct the syntax, and then recompile. The correct syntax for a window color is:

(RRR, GGG, BBB)

2421 Invalid window position.

PROBLEM: The window position in a window definition consists of four decimal numbers that define the windows location on the screen and its width and height. The four numbers must be given in Helps 1024-by-1024 coordinate system, and they must be enclosed in parentheses and separated by commas.

RESULT: The compiler ignores the window definition.

SOLUTION: Correct the syntax, and then recompile. The correct syntax to indicate the predefined window position is:

(x-coord, y-coord, width, height)

2431 Missing quote in window caption.

PROBLEM: The window caption in a window definition must be enclosed in quotation marks.

RESULT: The compiler ignores the window definition.

SOLUTION: Add the missing quotation marks, and then recompile.

2441 Window name windowname is too long.

PROBLEM: The window name exceeds the maximum length of eight characters.

RESULT: The compiler ignores the window definition.

SOLUTION: Shorten the window name to eight or fewer characters, and then recompile.

2451 Window position value out of range 01023.

PROBLEM: One or more of the window position coordinates (x-coord, y-coord, width, height) exceed the limit of 1023.

RESULT: The compiler ignores the window definition.

SOLUTION: Correct the numeric values of the window position coordinates so that they fall within the valid range (01023), and then recompile.

2461 Window name missing.

PROBLEM: A window definition in the Help project file is missing the window name.

RESULT: The compiler ignores the window definition.

SOLUTION: Specify a name for the window, and then recompile.

2471 Invalid syntax in [WINDOWS] section.

PROBLEM: The entry for a main or a secondary window is incorrect.

RESULT: The compiler ignores the invalid window definition.

SOLUTION: Check the syntax of each window definition, and then recompile. The correct syntax for a

window definition is:

window-name="caption", (x-coord, y-coord, width, height), window-state,

(scrolling-RGB), (nonscrolling-RGB), ontop-state

2481 Secondary window position required.

PROBLEM: A window definition for a secondary window must specify all four window position parametersx-coord, y-coord, width, and heightin the Help project file.

RESULT: The compiler ignores the window definition.

SOLUTION: Supply the missing window position parameters, and then recompile.

2491 Duplicate window name windowname.

PROBLEM: The [WINDOWS] section contains window definitions with duplicate window names.

RESULT: The compiler ignores the duplicate window definition.

SOLUTION: Check the uniqueness of each window name, and then recompile.

2501 Window caption windowcaption exceeds limit of 50 characters.

PROBLEM: The caption for the window exceeds the limit of 50 characters.

RESULT: The compiler ignores the window definition.

SOLUTION: Shorten the window caption, and then recompile.

[OPTIONS] Section Errors

The following error messages are caused by problems in the [OPTIONS] section of the Help project file.

2511 Unrecognized option optionname in [OPTIONS] section.

PROBLEM: The compiler does not recognize or support the specified option.

RESULT: The compiler ignores the unrecognized option.

SOLUTION: Check the [OPTIONS] section for typing errors or invalid options, and then recompile.

2532 Option optionname previously defined.

PROBLEM: The specified option has been defined on a previous line.

RESULT: The compiler uses the first definition and ignores the attempted redefinition.

SOLUTION: Remove one of the duplicate options, and then recompile.

ROOT Option Errors

The following error messages are caused by problems with the ROOT or the BMROOT option in the [OPTIONS] section of the Help project file.

2550 Invalid path pathname in optionname option.

PROBLEM: The compiler cannot find the path specified by the ROOT or BMROOT option.

RESULT: The compiler uses the current working directory.

SOLUTION: Correct the path in the ROOT or BMROOT option, and then recompile.

2570 Path in optionname option exceeds number of characters.

PROBLEM: The specified root path exceeds the MS-DOS limit.

RESULT: The compiler ignores the path and uses the current working directory.

SOLUTION: Shorten the path, and then recompile.

MAPFONTSIZE Option Errors

The following error messages are caused by problems with the MAPFONTSIZE option in the [OPTIONS] section of the Help project file.

2591 Invalid MAPFONTSIZE option.

PROBLEM: The font range syntax used is invalid. A font range consists of a low and high point size, separated by a hyphen (-).

RESULT: The compiler ignores the option.

SOLUTION: Correct the syntax, and then recompile. The correct syntax is:

MAPFONTSIZE=m[-n]:p

2612 Maximum of 5 font ranges exceeded.

PROBLEM: The maximum number of font ranges that can be specified is five.

RESULT: The compiler ignores additional ranges.

SOLUTION: Remove the additional font ranges so that there are five or fewer, and then recompile.

2632 Current font range overlaps previously defined range.

PROBLEM: A font size range overlaps a previously defined mapping.

RESULT: The compiler uses the first range and ignores the second mapping.

SOLUTION: Adjust one or both of the font ranges to remove any overlapping.

FORCEFONT Option Errors

The following error messages are caused by problems with the FORCEFONT option in the [OPTIONS] section of the Help project file.

2651 Font name exceeds limit of 20 characters.

PROBLEM: Font names cannot exceed 20 characters.

RESULT: The compiler ignores the option on this line.

SOLUTION: Shorten the font name to 20 or fewer characters, and then recompile.

2672 Unrecognized font name fontname in FORCEFONT option.

PROBLEM: The compiler has encountered a font name that it does not recognize or support.

RESULT: The compiler ignores the font name and uses the default MS Sans Serif font.

SOLUTION: Change the font name to a supported font, and then recompile. For the list of supported fonts, refer to Chapter 16, The Help Project File.

MULTIKEY Option Errors

The following error messages are caused by problems with the MULTIKEY option in the [OPTIONS] section of the Help project file.

2691 Invalid MULTIKEY option.

PROBLEM: The MULTIKEY option must specify a single capital letter other than the letter K.

RESULT: The compiler ignores the attempted keyword table assignement on this line.

SOLUTION: Correct the syntax, and then recompile. The correct syntax is:

MULTIKEY=char

2711 Maximum of 5 keyword tables exceeded.

PROBLEM: The [OPTIONS] section contains more than five MULTIKEY entries, which exceeds the limit of five keyword tables.

RESULT: The compiler ignores the additional keyword tables.

SOLUTION: Reduce the number of keyword tables to five or fewer, and then recompile.

2732 Character already used.

PROBLEM: A character used for indicating the alternate keyword table (MULTIKEY=char) was previously used.

RESULT: The compiler ignores the duplicate keyword table assignment on this line.

SOLUTION: Change the duplicate character to one that has not been used, and then recompile.

2752 Characters K and k cannot be used.

PROBLEM: These characters are reserved for Helps standard keyword table.

RESULT: The compiler ignores the attempted keyword table assignment on this line.

SOLUTION: Choose another character for the alternate keyword table, and then recompile.

Other [OPTIONS] Errors

The following error messages are caused by problems with other options in the [OPTIONS] section of the Help project file.

2771 REPORT option must be ON or OFF.

PROBLEM: The REPORT option specifies an incorrect value.

RESULT: The compiler uses the default value off.

SOLUTION: Correct the entry, and then recompile. The correct syntax is:

REPORT=on/off

2811 OLDKEYPHRASE option must be ON or OFF.

PROBLEM: The OLDKEYPHRASE option specifies an incorrect value.

RESULT: The compiler uses the default value on.

SOLUTION: Correct the entry, and then recompile. The correct syntax is:

OLDKEYPHRASE=[off, 0, no, false] or [on, 1, yes, true]

2832 COMPRESS option must be OFF, MEDIUM, or HIGH.

PROBLEM: The COMPRESS option specifies an incorrect value.

RESULT: The compiler uses the default value off.

SOLUTION: Correct the entry, and then recompile. The correct syntax is:

COMPRESS=[off, 0, no, false] or [medium] or [on, 1, yes, true, high]

2842 OPTCDROM option must be TRUE or FALSE.

PROBLEM: The OPTCDROM option specifies an incorrect value.

RESULT: The compiler uses the default value false.

SOLUTION: Correct the entry, and then recompile. The correct syntax is:

OPTCDROM=[off, 0, no, false] or [on, 1, yes, true]

2852 Invalid TITLE option.

PROBLEM: The TITLE option defines a string that is empty or contains more than 32 characters.

RESULT: The compiler truncates the title at 32 characters.

SOLUTION: Add a title if the option is empty or shorten the title if it is too long, and then recompile.

2872 Invalid LANGUAGE option.

PROBLEM: You have specified an ordering that is not supported by the compiler.

RESULT: The compiler uses the default English (U.S.) sorting order.

SOLUTION: Correct the entry, and then recompile. The correct syntax is:

LANGUAGE=scandinavian

2893 Warning option must be 1, 2, or 3.

PROBLEM: The warning reporting level can be set only to 1, 2, or 3.

RESULT: The compiler uses full reporting (level 3).

SOLUTION: Correct the entry, and then recompile.

2911 Invalid icon file filename.

PROBLEM: The compiler cannot find the icon file specified in the ICON option, or the file is not a valid icon file.

RESULT: The compiler ignores the file.

SOLUTION: Check to see if the icon file exists and if it is a valid .ICO file. Or if the file is in not in a directory defined by the ROOT or BMROOT option, move the file so that the Help compiler can find it, and then recompile.

2932 Copyright string exceeds limit of 50 characters.

PROBLEM: The maximum length of the copyright string in the About box is 50 characters.

RESULTS: The compiler truncates the copyright string at 50 characters.

SOLUTION: Shorten the copyright notice, and then recompile.

2000 UNKNOWN ERROR, Contact Microsoft Product Support Services

PROBLEM: This error message is the equivalent of an assertion failure.

RESULT: The compiler aborts the build.

SOLUTION: Contact Microsoft Product Support Services.

Build Tag or Build Expression Errors

The following messages are caused by errors in build tag footnotes or build expressions in the [BUILDTAGS] section of the Help project file.

3011 Maximum of 32 build tags exceeded.

PROBLEM: The maximum number of build tags that can be defined is 32.

RESULT: The compiler ignores the additional tags.

SOLUTION: Reduce the number of build tags to 32 or fewer, and then recompile.

3031 Build tag length exceeds 32 characters.

PROBLEM: The build tag on the specified line exceeds the limit of 32 characters.

RESULT: The compiler ignores the build tag.

SOLUTION: Shorten the build tag to 32 or fewer characters, and then recompile.

3051 Build tag tagname contains invalid characters.

PROBLEM: Build tags can contain only alphanumeric characters or the underscore (_) character.

RESULT: The compiler ignores the line with invalid characters.

SOLUTION: Correct the build tag so that it contains only valid characters, and then recompile.

3076 [BUILDTAGS] section missing.

PROBLEM: The BUILD option declares a conditional build, but there is no [BUILDTAGS] section in the Help project file.

RESULT: The compiler includes all topics in the build.

SOLUTION: Add a [BUILDTAGS] section to the Help project file, if you want to perform a conditional build, and then recompile.

3096 Build expression too complex.

PROBLEM: The build expression has too many expressions (~, |, or &) or is nested too deeply.

RESULT: The compiler includes all topics in the build.

SOLUTION: Reduce the number of operators or nested levels, and then recompile. Refer to Chapter 16, The Help Project File, for details.

3116 Invalid build expression.

PROBLEM: The syntax used in the build expression on the specified line contains one or more logical or syntax errors.

RESULT: The compiler ignores the line with the invalid expression.

SOLUTION: Correct the entry, and then recompile. For the correct syntax and information about build expressions, refer to Chapter 16, The Help Project File.

3133 Duplicate build tag in [BUILDTAGS] section.

PROBLEM: A build tag appears more than once in the [BUILDTAGS] section.

RESULT: The compiler uses the first build tag and ignores the duplicate build tag.

SOLUTION: Remove or rename one of the duplicate build tags, and then recompile.

3152 Build tag tagname not defined in [BUILDTAGS] section.

PROBLEM: The specified build tag has been assigned to a topic but not declared in the Help project file.

RESULT: The compiler ignores the build tag defined in the topic.

SOLUTION: Include the build tag in the [BUILDTAGS] section, and then recompile.

3178 Build expression missing from project file.

PROBLEM: The topics have build tag footnotes, but there is no BUILD= expression in the Help project file.

RESULT: The compiler includes all topics in the build.

SOLUTION: Add a build expression to the Help project file and recompile if you want your build tags to work. Otherwise, you can safely ignore this message.

Macro Errors

The following messages result from errors in the use of Help macros in footnotes, hot spots, and the [CONFIG] section of the Help project file.

3511 Macro macrostring exceeds limit of 254 characters.

PROBLEM: The macro string exceeds the limit of 254 characters.

RESULT: The compiler passes the macro to the Help file, where, in most cases, it will cause the Help application to display an error when it is executed.

SOLUTION: Shorten the length of the macro string, and then recompile. In some cases, you may be able to substitute an abbreviation for the macro name to shorten the macro string. For the supported macro abbreviations, check Chapter 15, Help Macros Reference.

3532 Undefined function in macro macroname.

PROBLEM: The specified macro is not on the list of macros supported by the compiler, nor is it specified in the RegisterRoutine macro.

RESULT: The compiler passes the macro to the Help file, where, in most cases, it will cause the Help application to display an error when it is executed.

SOLUTION: If the function resides in a custom DLL, check the Help project file to make sure that you have registered the function using the RegisterRoutine macro. Otherwise, change the macro function to one that is supported or remove it, and then recompile.

3552 Undefined variable in macro macroname.

PROBLEM: The specified macro contains a variable that is not recognized by the compiler.

RESULT: The compiler passes the macro to the Help file, where, in most cases, it will cause the Help application to display an error when it is executed.

SOLUTION: Change the macro so that it uses only defined variables, and then recompile. If the variable is defined in a custom DLL, check the RegisterRoutine macro in the Help project file to be sure that the variable is defined and that you are using the function correctly. Otherwise, check Chapter 15, Help Macro Reference.

3571 Wrong number of parameters to function in macro macroname.

PROBLEM: There are too many or too few parameters in the macro.

RESULT: The compiler passes the macro to the Help file, where, in most cases, it will cause the Help application to display an error when it is executed.

SOLUTION: Change the macro so that it uses the correct number of parameters, and then recompile. If the macro is defined in a custom DLL, check the RegisterRoutine macro in the Help project file for the correct number of parameters. Otherwise, check Chapter 15, Help Macro Reference.

3591 Syntax error in macro macroname.

PROBLEM: The syntax of the macro is invalid.

RESULT: The compiler passes the macro to the Help file, where, in most cases, it will cause the Help application to display an error when it is executed.

SOLUTION: Correct the syntax, and then recompile. For the correct syntax of the macro you are using, refer to Chapter 15, Help Macro Reference.

3611 Function parameter type mismatch in macro macroname.

PROBLEM: There is a type mismatch in the function call. In other words, a string is given for a numeric parameter or a number is given for a string parameter.

RESULT: The compiler passes the macro to the Help file, where, in most cases, it will cause the Help application to display an error when it is executed.

SOLUTION: Change the macro so that it uses the correct parameter types, and then recompile. If the macro is defined in a custom DLL, check the RegisterRoutine macro in the Help project file for the

correct parameter types. Otherwise, check Chapter 15, Help Macro Reference.

3631 Bad macro prototype.

PROBLEM: The prototype string passed to RegisterRoutine is invalid.

RESULT: The compiler passes the macro to the Help file, where, in most cases, it will cause the Help application to display an error when it is executed.

SOLUTION: Change the macro so that it uses a valid prototype string, and then recompile.

3652 Empty macro string.

PROBLEM: The ! footnote or a hidden text code starting with ! does not contain a valid macro.

RESULT: The compiler passes the macro to the Help file, where, in most cases, it will cause the Help application to display an error when it is executed.

SOLUTION: Correct the macro string in the topic footnote, and then recompile.

3672 Macro macroname nested too deeply.

PROBLEM: Macro strings may not contain more than three other macro strings as parameters.

RESULT: The compiler passes the macro to the Help file.

SOLUTION: Remove one or more of the nested macro strings, and then recompile. Or ignore the error if you want to include the macro in the Help file. The following macro correctly nests three macros:

```
IfThen(1, `IfThen(1, `IfThen(1, `BrowseButtons()')')')
```

The following macro string is nested too deeply:

```
IfThen(1, `IfThen(1, `IfThen(1, `IfThen(1, `BrowseButtons()')')')')
```

The Help application will not display an error message if a macro string is nested too deeply. Therefore, both of the above macros will work, even though one generates an error during compilation and one does not.

3000 UNKNOWN ERROR, Contact Microsoft Product Support Services

PROBLEM: This error message is the equivalent of an assertion failure.

RESULT: The compiler aborts the build.

SOLUTION: Contact Microsoft Product Support Services.

Context String Errors

The following messages are caused by problems with context string footnotes or with context strings specified in jumps or in Help project file options.

4011 Context string contextname already used.

PROBLEM: The specified context string was previously assigned to another topic.

RESULT: The compiler ignores the duplicate context string, and the topic has no identifier.

SOLUTION: Change the second context string so that it is unique, and then recompile.

4031 Invalid context string contextname.

PROBLEM: The context string footnote contains non-alphanumeric characters or is empty.

RESULT: The compiler does not assign the topic an identifier.

SOLUTION: Change the characters in the context string so that it is valid, and then recompile.

4056 Unresolved context string specified in CONTENTS option.

PROBLEM: The Contents topic defined in the Help project file could not be found.

RESULT: The compiler uses the first topic in the build as the Contents.

SOLUTION: Compare the context string in the CONTENTS option to the context string used in the Contents topic to be sure they are the same. Change one of the strings to match the other, and then recompile.

4072 Context string exceeds limit of 255 characters.

PROBLEM: The context string hidden text cannot exceed 255 characters.

RESULT: The compiler ignores the context string.

SOLUTION: Shorten the context string, and then recompile.

4098 Context string(s) in [MAP] section not defined in any topic.

PROBLEM: The compiler cannot find a context string listed in the [MAP] section in any of the topics in the build.

RESULT: The compiler ignores the line in the [MAP] section. If the application uses that number in the WinHelp function, Help will display the Help topic not found error message.

SOLUTION: Check to be sure that all the context strings listed in the [MAP] section are assigned to topics included in the build and that they are all spelled correctly. Then recompile.

4113 Unresolved jump or pop-up contextname.

PROBLEM: The specified topic contains a context string that identifies a nonexistent topic.

RESULT: The Help application displays the Help topic not found error message when the user chooses a hot spot with the unresolved context string.

SOLUTION: Check the topic for spelling errors in the context string, and also check to see if the requested topic is included in the build.

4131 Hash conflict between contextname and contextname.

PROBLEM: The hash algorithm has generated the same hash value for both of the listed context strings.

RESULT: The Help application displays the topic with the first context string when the user chooses a hot spot containing either context string.

SOLUTION: Change one of the context strings, and then recompile.

4151 Invalid secondary window name windowname.

PROBLEM: The hot spot contains a window name that is not defined in the Help project file.

RESULT: The hot spot is not active (not hot) in the built Help file.

SOLUTION: Change the window name to a unique name that is eight or fewer characters, and then recompile.

4171 Cannot use secondary window with pop-up.

PROBLEM: The hidden text defining the pop-up identifier contains a secondary window name.

RESULT: The hot spot is not active (not hot) in the built Help file.

SOLUTION: Remove the right angle bracket and secondary window name from the pop-up hot spot, and then recompile.

4196 Jumps and lookups not verified.

PROBLEM: Due to low memory conditions, the Help compiler cannot verify all jump and pop-up hot spots. (The reference to lookups (keywords) in the error message is incorrect.)

RESULT: The compiler continues the build but does not verify the validity of jump and pop-up hot spots. The resulting Help file may still be valid.

SOLUTION: Free more memory and then recompile. For suggestions on how to free more memory for the build, refer to the Getting Around Memory Problems section in Chapter 17. If you cant overcome your memory constraints, you may have to compile and test the hot spots manually.

Footnote Errors

The following messages are caused by problems with footnotes in topic files.

4211 Footnote text exceeds limit of 1023 characters.

PROBLEM: The footnote text cannot exceed the limit of 1023 characters.

RESULT: The compiler ignores the footnote.

SOLUTION: Reduce the length of the footnote text, and then recompile.

Browse Sequence Errors

4251 Browse sequence not in first paragraph.

PROBLEM: The browse sequence footnote is not in the first paragraph of the topic.

RESULT: The compiler ignores the browse sequence footnote.

SOLUTION: Move the browse sequence footnote so that it is first, and then recompile.

4272 Empty browse sequence string.

PROBLEM: The browse sequence footnote for the specified topic contains no sequence characters.

RESULT: The compiler ignores the browse sequence footnote.

SOLUTION: Add sequence characters to the browse sequence footnote, and then recompile.

4292 Missing sequence number.

PROBLEM: A browse sequence number ends with a colon (:) for the specified topic.

RESULT: The compiler ignores the browse sequence footnote.

SOLUTION: Remove the colon or enter a minor sequence number, and then recompile.

4312 Browse sequence already defined.

PROBLEM: A browse sequence footnote already exists for the specified topic.

RESULT: The compiler uses the first browse sequence footnote and ignores the duplicate footnote.

SOLUTION: Remove the duplicate browse sequence footnote, and then recompile.

Topic Title Errors

4331 Title not in first paragraph.

PROBLEM: The title footnote ($) is not in the first paragraph of the topic.

RESULT: The compiler ignores the title footnote, and the topic does not have a title.

SOLUTION: Move the title footnote to the beginning of the topic, and then recompile.

4352 Empty title string.

PROBLEM: The title footnote for the specified topic contains no characters.

RESULT: The compiler ignores the title footnote, and the topic does not have a title.

SOLUTION: Open the footnote window and type the title string next to the topic title footnote character for the topic, and then recompile.

4372 Title defined more than once.

PROBLEM: There is more than one title footnote in the specified topic.

RESULT: The compiler uses the first title footnote and ignores the duplicate title footnote.

SOLUTION: Remove the duplicate title footnote, and then recompile.

4393 Title exceeds limit of 128 characters.

PROBLEM: The title for the specified topic exceeds 128 characters.

RESULT: The compiler truncates the title string at 128 characters.

SOLUTION: Shorten the length of the title string, and then recompile.

Keyword Errors

4412 Keyword string exceeds limit of 255 characters.

PROBLEM: The keyword string exceeds the limit of 255 characters.

RESULT: The compiler ignores the keyword footnote.

SOLUTION: Shorten the length of the keyword string, or add another keyword footnote for the extra keywords, and then recompile.

4433 Empty keyword string.

PROBLEM: There are no characters in the keyword footnote.

RESULT: The compiler ignores the keyword footnote, and the topic does not have any keywords.

SOLUTION: Open the footnote window and type some keywords next to the keyword footnote character for the topic, and then recompile.

4452 Keyword(s) defined without title.

PROBLEM: The topic has a keyword assigned to it, but no title.

RESULT: The topic will appear as >>Untitled Topic<< in the Search dialog box and in the history list.

SOLUTION: Add a topic title footnote to the topic if you want a title to appear in the Search dialog box when the user selects this keyword.

Build Tag Errors

4471 Build tag footnote not at beginning of topic.

PROBLEM: The build tag footnote marker, if used, has to be the first character in the topic.

RESULT: The compiler ignores the build tag footnote, and the topic is not assigned a build tag.

SOLUTION: Move the build tag footnote to the beginning of the topic, before other footnotes, and then recompile.

4492 Build tag exceeds limit of 32 characters.

PROBLEM: A build tag for the specified topic exceeds the limit of 32 characters.

RESULT: The compiler ignores the build tag assigned to the topic.

SOLUTION: Shorten the length of the build tag to 32 or fewer characters, and then recompile.

Entry Macro Errors

4551 Entry macro not in first paragraph.

PROBLEM: The ! footnote (for executing a macro) is not in the first paragraph of the topic.

RESULT: The compiler ignores the macro footnote.

SOLUTION: Move the macro footnote to the beginning of the topic, next to the other footnotes, and then recompile.

Topic File Errors

The following messages result from problems in the use of Word for Windows formatting in one or more topic files.

4616 File filename is not a valid RTF topic file.

PROBLEM: The specified file is not an RTF file.

RESULT: The compiler ignores the file.

SOLUTION: Make sure that you save the topic as RTF from your word processor.

4639 Error in file filename at byte offset 0xoffset.

PROBLEM: The specified file contains unrecognized RTF at that byte offset. Unrecognized RTF includes any formatting the Help compiler does not recognize, including mismatched braces or nonstandard RTF produced by applications other than Word for Windows. This message should not appear if Word for Windows, Word for the Macintosh, or PCWord is used.

RESULT: The compiler aborts the build.

SOLUTION: Open the file in Word for Windows and save the file again as RTF, and then recompile. If you are using Word for the Macintosh, transfer the file to the PC again, and then recompile. If the problem persists, open the file without converting it to .DOC format, and look for errors in the source RTF.

4649 File filename contains more than 32767 topics.

PROBLEM: The maximum number of topics allowed in one RTF file is 32,767.

RESULT: The compiler aborts the build.

SOLUTION: Reduce the number of topics in the file by dividing it into two or more topic files, and then recompile.

4652 Table formatting too complex.

PROBLEM: The compiler encountered a table with borders, shading, or right justification.

RESULT: The compiler ignores the formatting.

SOLUTION: Remove the unsupported formatting, and then recompile.

4662 Side by side paragraph formatting not supported.

PROBLEM: The side-by-side paragraph formatting is not supported in Windows Help version 3.1.

RESULT: The compiler ignores the side-by-side paragraphs.

SOLUTION: Convert the side-by-side paragraphs to a table by opening the file in Word for Windows and saving it.

4671 Table contains more than 32 columns.

PROBLEM: The maximum number of columns in one table is 32. Some word processors may have different limits for the number of columns supported.

RESULT: The compiler treats the additional columns as one table cell.

SOLUTION: Reduce the number of columns to 32 or fewer, and then recompile.

4680 Font fontname in file filename not in RTF font table.

PROBLEM: A font not defined in the RTF header has been entered into the topic.

RESULT: The compiler uses the default system font.

SOLUTION: Check to be sure the font you are using is supported by the word processor. Then save the file again as RTF and recompile.

4692 Unrecognized graphic format.

PROBLEM: The compiler supports only Windows bitmaps (.BMP), Windows device-independent

bitmaps (.DIB), Windows metafiles (.WMF), segmented hypergraphics (.SHG), and multiresolution bitmaps (.MRB).

RESULT: The compiler ignores the graphic.

SOLUTION: Make sure that you have not used Macintosh picture format or some other unsupported graphic format.

4733 Hidden page break.

PROBLEM: The page break is part of the hidden text.

RESULT: The compiler ignores the page break formatted as hidden text, so the two topics will not be separated.

SOLUTION: Reformat the page break as plain text and recompile.

4753 Hidden paragraph.

PROBLEM: A paragraph marker is part of the hidden text.

RESULT: The compiler ignores the paragraph marker formatted as hidden text, so the two paragraphs will run together.

SOLUTION: Reformat the paragraph marker as plain text and recompile.

4763 Hidden carriage return.

PROBLEM: A carriage return is part of the hidden text.

RESULT: The compiler ignores the carriage return formatted as hidden text and does not create a new line.

SOLUTION: Reformat the carriage return as plain text and recompile.

4774 Paragraph exceeds limit of 64K.

PROBLEM: A single paragraph has more than 64K of text or 64K of graphics. If the paragraph contains an inline graphic (pasted directly into the topic in Word for Windows) or a bitmap reference stored with data (bmcwd, for example), the paragraph size cannot exceed 64K. This limit does not apply to graphics stored separately from the topic data using the standard bitmap references bmc, bml, or bmr.

RESULT: The compiler aborts the build.

SOLUTION: Replace any inline graphics with bitmap references, or reduce the amount of text in the paragraph, or reduce the size of the graphic, and then recompile.

4792 Non-scrolling region defined after scrolling region.

PROBLEM: A paragraph that was authored as Keep With Next is not the first paragraph in the topic.

RESULT: The compiler ignores the Keep With Next attribute; the paragraph is treated as regular text and will be part of the topic text displayed in the scrolling region.

SOLUTION: Format the first paragraph(s) as Keep With Next and remove the Keep With Next formatting from the subsequent paragraphs. Then recompile.

4813 Non-scrolling region crosses page boundary.

PROBLEM: The Keep With Next paragraph formatting crosses a page break boundary.

RESULT: The topic includes a nonscrolling region, whether or not one is intended.

SOLUTION: Check the file to be sure that all topics end with a hard carriage return immediately before the page break. Remove the Keep With Next formatting attribute from any unwanted paragraphs, and then recompile.

4000 UNKNOWN ERROR, Contact Microsoft Product Support Services

PROBLEM: This error message is the equivalent of an assertion failure.

RESULT: The compiler aborts the build.

SOLUTION: Contact Microsoft Product Support Services.

Miscellaneous Errors

The following messages are caused by conditions such as MS-DOS file errors or out-of-memory conditions.

5035 File filename not created.

PROBLEM: There are no topics to compile, or the build expression is false for all topics.

RESULT: The compiler does not create a Help file.

SOLUTION: Add a topic filename to the [FILES] section in the Help project file, or change the build expression so that all topics are not excluded from the build, and then recompile.

5059 Not enough memory to build help file.

PROBLEM: The Help compiler does not have enough memory to complete the build process.

RESULT: The compiler does not create a Help file.

SOLUTION: To free memory, unload any unneeded applications, device drivers, and memory-resident programs. You can also turn off compression in the Help project file, replace any inline bitmaps pasted directly in the topic files with bitmap references, or compile under OS/2 rather than MS-DOS.

5075 Help Compiler corrupted. Please reinstall HC.EXE.

PROBLEM: Virus-checking code has detected a corruption in the compiler.

RESULT: The compiler does not run.

SOLUTION: Reinstall the compiler from the original source disk.

5098 Using old key-phrase table.

PROBLEM: The compiler found an existing key-phrase table for the Help file you are building, and the Help project file does not specify not to use the table.

RESULT: The compiler uses the key-phrase table that was generated from an earlier build, instead of creating a new table.

SOLUTION: To achieve maximum compression during a build, you must delete the old key-phrase table (.PH file) before each recompilation, or set the OLDKEYPHRASE option to off.

5115 Write failed.

PROBLEM: Write to disk failed.

RESULT: The compiler aborts the build.

SOLUTION: Contact Microsoft Product Support Services.

5139 Aborted by user.

PROBLEM: The user ended the build prematurely by pressing CTRL+C or CTRL+BREAK. Note: The extended version of the compiler (HCP.EXE) does not display this error message.

RESULT: The Help compiler aborts the build.

SOLUTION: Restart the build when ready.

5000 UNKNOWN ERROR, Contact Microsoft Product Support Services

PROBLEM: This error message is the equivalent of an assertion failure.

RESULT: The compiler aborts the build.

SOLUTION: Contact Microsoft Product Support Services.

Help Compiler 3.0 Error Messages

This section lists the error messages that the Help compiler version 3.0 displays when it encounters problems building a Help file. Although many of the messages are similar to those displayed by the version 3.1 Help compiler, the 3.0 message scheme and numbering system are different. This section describes the version 3.0 error messages.

Whenever possible, the Help compiler displays the name of the file that contains the error, as well as the number used to identify the specific line of the Help project file or the topic that produced the error. Since topics do not contain numbers, the topic number specified in the error message refers to the topics sequential order in the topic file. In most cases, the topic number given in the error message will match the topics page number in the Word for Windows document.

Interpreting Help 3.0 Error Messages

The Help compiler version 3.0 displays either warning or fatal error messages. A warning message begins with the word Warning. A warning error indicates a problem encountered during the build that is not severe enough to prevent the Help compiler from completing the build. Therefore, a build with warnings produces a Help file that Windows Help should be able to open, but the file may contain problems in certain topics, such as missing graphics or hot spots that arent hot. Fatal error messages begin with the word Error. A fatal error indicates a problem that prevents the Help compiler from creating a Help file.

As stated previously, you use the REPORT option to display error messages on the screen and the WARNING option to specify the amount of warning information you want the Help compiler to provide. The 3.0 compiler always reports fatal errors, regardless of the current warning level or report option specified in the Help project file, since no usable Help file results from the build.

While the Help compiler processes the Help project file, it ignores lines that contain errors and attempts to continue with the build. This means that errors encountered early in a project file may result in many more errors being reported as the build continues.

Similarly, when the Help compiler processes the RTF topic files, it reports any errors it encounters and, if the errors are not fatal, the compiler continues with the build. A single error in a topic file may result in more than one error message being displayed by the compiler. For instance, a typographic mistake in a topics context string will cause an error to be reported every time the compiler encounters a reference to the correct topic identifier. On the other hand, some warning errorssuch as hot spots with the same hidden text codes (broken jumps) or any bitmap file errors, including File not found are reported once per file, regardless of the number of times the problem occurs in the source file.

Help 3.0 Error Message Categories

In Windows Help version 3.0, errors that occur during processing of the Help project file are prefixed with the letter P and appear as in the following examples:

```
Error P1025: line...7 of filename.HPJ : Section heading sectionname
unrecognized.

Warning P1039: line...38 of filename.HPJ : [BUILDTAGS] section missing.
```

Errors that occur during processing of the RTF topic file(s) are prefixed with the letter R and appear as in the following examples:

```
Error R2025: File environment error.

Warning R2501: Using old key-phrase table.
```

Help 3.0 Error Message Reference

This reference lists all Help 3.0 error messages by category. The wording of the error messages is the same as that displayed by the 3.0 Help compiler, except that variables (% tags) are replaced with descriptive words. Error message descriptions contain one or more of the following types of information.

Heading	Information
Problem	Explains what the error message means.
Result	Describes how the Help compiler handles this particular error.
Solution	Offers suggestions about how to correct the problem.

Project File Error Messages

This section lists all the possible errors that might occur when the 3.0 Help compiler is processing the Help project file.

P1001 Unable to read file filename.

PROBLEM: The specified file could not be found or is unreadable. This is an MS-DOS file error or an out-of-memory condition.

RESULT: If the file not found is an RTF topic file, the compiler ignores the file. If the file is a bitmap file, the built Help file displays the Unable to display picture message in the topic instead of the bitmap.

SOLUTION: Check to see if the file exists, and also check the amount of available memory.

P1003 Invalid path specified in Root option.

PROBLEM: The compiler cannot find the path specified by the ROOT option.

RESULT: The compiler uses the current working directory.

SOLUTION: Correct the path in the ROOT option, and then recompile.

P1005 Path and filename exceed limit of 79 characters.

PROBLEM: The combined length of the path and filename must not be more than the MS-DOS limit.

RESULT: The compiler ignores the file.

SOLUTION: Shorten the path, and then recompile.

P1007 Root path exceeds maximum limit of 66 characters.

PROBLEM: The specified root path exceeds the MS-DOS limit.

RESULT: The compiler ignores the path and uses the current working directory.

SOLUTION: Shorten the path, and then recompile.

P1009 [FILES] section missing.

PROBLEM: The [FILES] section is required.

RESULT: The compiler aborts the build.

SOLUTION: Create a [FILES] section that lists the topic files to include in the build, and then recompile.

P1011 Option optionname previously defined.

The specified option was defined previously. The compiler ignores the attempted redefinition.

P1013 Project file extension cannot be .HLP.

PROBLEM: You cannot specify a Help project file with a .HLP extension. Project files must use the .HPJ extension.

RESULT: The compiler aborts the build.

SOLUTION: Rename the Help project file, and then recompile.

P1015 Unexpected end-of-file.

PROBLEM: The compiler has unexpectedly come to the end of the Help project file. There may be an open comment in the Help project file or in an include file.

RESULT: The compiler aborts the build.

SOLUTION: Close any open comments within the Help project file, and then recompile.

P1017 Parameter exceeds maximum length of 128 characters.

PROBLEM: An option, context name or number, build tag, or other parameter on the specified line exceeds the limit of 128 characters.

RESULT: The compiler ignores the line.

SOLUTION: Shorten the line, and then recompile.

P1021 Context number already used in [MAP] section.

PROBLEM: A context number in the [MAP] section of the Help project file was previously mapped to a different context string.

RESULT: The compiler ignores the line with the duplicate context number.

SOLUTION: Check all context numbers to make sure that they are unique. Remove or reassign any duplicate context numbers, and then recompile.

P1023 Include statements nested too deeply.

PROBLEM: The #include statement on the specified line exceeds the maximum of five include levels.

RESULT: The compiler aborts the build.

SOLUTION: Simplify the #include statement so that it nests fewer than five include levels, then recompile.

P1025 Section heading sectionname unrecognized.

PROBLEM: The Help project file includes a section heading that the compiler does not recognize or does not support.

RESULT: The compiler ignores the section heading.

SOLUTION: Check all bracketed headings in the Help project file for typographical or other errors. Fix the section heading, and then recompile.

P1027 Bracket missing from section heading sectionname.

PROBLEM: The right bracket (]) is missing from the specified section heading.

RESULT: The compiler ignores the section heading on this line.

SOLUTION: Insert the right section heading bracket, and then recompile.

P1029 Section heading missing.

PROBLEM: The section heading on the specified line is not complete, or the first entry in the Help project file is not a section heading.

RESULT: The compiler ignores the section heading on this line and all succeeding lines until it encounters a valid section heading.

SOLUTION: Insert a valid section heading at the appropriate place in the Help project file, and then recompile.

P1030 Section sectionname previously defined.

PROBLEM: The compiler found more than one section with the specified heading in the Help project file.

RESULT: The compiler ignores the duplicate section and continues from the next valid section heading.

SOLUTION: Remove or rename one of the duplicate section headings, and then recompile.

P1031 Maximum number of build tags exceeded.

PROBLEM: The maximum number of build tags that can be defined is 32.

RESULT: The compiler ignores the additional tags.

SOLUTION: Reduce the number of build tags to 32 or fewer, and then recompile.

P1033 Duplicate build tag in [BUILDTAGS] section.

PROBLEM: A build tag appears more than once in the [BUILDTAGS] section.

RESULT: The compiler uses the first build tag and ignores the duplicate build tag.

SOLUTION: Remove or rename one of the duplicate build tags, and then recompile.

P1035 Build tag length exceeds maximum.

PROBLEM: The build tag on the specified line exceeds the limit of 32 characters.

RESULT: The compiler ignores the build tag.

SOLUTION: Shorten the build tag to 32 or fewer characters, and then recompile.

P1037 Build tag tagname contains invalid characters.

PROBLEM: Build tags can contain only alphanumeric characters or the underscore (_) character.

RESULT: The compiler ignores the line with invalid characters.

SOLUTION: Correct the build tag so that it contains only valid characters, and then recompile.

P1039 [BUILDTAGS] section missing.

PROBLEM: The BUILD option declares a conditional build, but there is no [BUILDTAGS] section in the Help project file.

RESULT: The compiler includes all topics in the build.

SOLUTION: Add a [BUILDTAGS] section to the Help project file, if you want to perform a conditional build, and then recompile.

P1043 Too many tags in Build expression.

PROBLEM: The Build expression on the specified line has used more than the maximum of 20 build tags.

RESULT: The compiler ignores the line.

SOLUTION: Reduce the number of build tags to 20 or fewer, and then recompile.

P1045 [ALIAS] section found after [MAP] section.

PROBLEM: When used, the [ALIAS] section must precede the [MAP] section in the Help project file.

RESULT: The compiler ignores the [ALIAS] section.

SOLUTION: Move the [ALIAS] section so that it precedes the [MAP] section, and then recompile.

P1047 Context string contextname already assigned an alias.

PROBLEM: The specified context string has already been assigned an alias in the [ALIAS] section. A context string can have only one alias. That is, you cannot map a=b and then a=c in the [ALIAS] section.

RESULT: The compiler ignores the attempted reassignment on this line.

SOLUTION: Correct the alias string mapping, and then recompile.

P1049 Alias string aliasname already assigned.

PROBLEM: You cannot alias an alias. That is, an alias string cannot, in turn, be assigned another alias. You cannot map a=b and then b=c in the [ALIAS] section.

RESULT: The compiler ignores the attempted reassignment on this line.

SOLUTION: Correct the alias string mapping, and then recompile.

P1051 Context string contextname cannot be used as alias string.

PROBLEM: A context string that has been assigned an alias cannot be used later as an alias for another context string. That is, you cannot map a=b and then c=a in the [ALIAS] section.

RESULT: The compiler ignores the attempted reassignment on this line.

SOLUTION: Correct the alias string mapping, and then recompile.

P1053 Maximum number of font ranges exceeded.

PROBLEM: The maximum number of font ranges that can be specified is five.

RESULT: The compiler ignores additional ranges.

SOLUTION: Remove the additional font ranges so that there are five or fewer, and then recompile.

P1055 Current font range overlaps previously defined range.

PROBLEM: A font size range overlaps a previously defined mapping.

RESULT: The compiler uses the first range and ignores the second mapping.

SOLUTION: Adjust one or both of the font ranges to remove any overlapping.

P1056 Unrecognized font name in Forcefont option.

PROBLEM: The compiler has encountered a font name that it does not recognize or support.

RESULT: The compiler ignores the font name and uses the default MS Sans Serif font.

SOLUTION: Change the font name to a supported font, and then recompile. For the list of supported fonts, refer to Chapter 16, The Help Project File.

P1057 Font name too long.

PROBLEM: Font names cannot exceed 20 characters.

RESULT: The compiler ignores the option on this line.

SOLUTION: Shorten the font name to 20 or fewer characters, and then recompile.

P1059 Invalid multiple-key syntax.

PROBLEM: The MULTIKEY option must specify a single capital letter other than the letter K.

RESULT: The compiler ignores the attempted keyword-table assignement on this line.

SOLUTION: Correct the syntax, and then recompile. The correct syntax is:

MULTIKEY=char

P1061 Character already used.

PROBLEM: A character used for indicating the alternate keyword table (MULTIKEY=char) was previously used.

RESULT: The compiler ignores the duplicate keyword-table assignment on this line.

SOLUTION: Change the duplicate character to one that has not been used, and then recompile.

P1063 Characters K and k cannot be used.

PROBLEM: These characters are reserved for Helps standard keyword table.

RESULT: The compiler ignores the attempted keyword-table assignment on this line.

SOLUTION: Choose another character for the alternate keyword table, and then recompile.

P1065 Maximum number of keyword tables exceeded.

PROBLEM: The [OPTIONS] section contains more than five MULTIKEY entries, which exceeds the limit of five keyword tables.

RESULT: The compiler ignores the additional keyword tables.

SOLUTION: Reduce the number of keyword tables to five or fewer, and then recompile.

P1067 Equal sign missing.

PROBLEM: An entry in the [OPTIONS] section is missing the equal sign, or it has some other syntax error.

RESULT: The compiler ignores the invalid option on this line.

SOLUTION: Check the syntax of each option used in the [OPTIONS] section. Fix the incorrect option, and then recompile. For the correct syntax, refer to Chapter 16, The Help Project File.

P1069 Context string missing.

PROBLEM: A line in the [MAP] or [ALIAS] section is missing a context string before the equal sign.

RESULT: The compiler ignores the line with the invalid context string.

SOLUTION: Check all context strings to the left of the equal sign. Correct any missing or invalid entries, and then recompile.

P1071 Incomplete line in sectionname section.

PROBLEM: An entry in the specified section is incomplete.

RESULT: The compiler ignores the incomplete line.

SOLUTION: Check the section for any incomplete lines and correct them. Then recompile.

P1073 Unrecognized option in [OPTIONS] section.

PROBLEM: The compiler does not recognize or support the specified option.

RESULT: The compiler ignores the unrecognized option.

SOLUTION: Check the [OPTIONS] section for typing errors or invalid options, and then recompile.

P1075 Invalid build expression.

PROBLEM: The syntax used in the build expression on the specified line contains one or more logical or syntax errors.

RESULT: The compiler ignores the line with the invalid expression.

SOLUTION: Correct the entry, and then recompile. For the correct syntax and information about build expressions, refer to Chapter 16, The Help Project File.

P1077 Warning level must be 1, 2, or 3.

PROBLEM: The warning reporting level can be set only to 1, 2, or 3.

RESULT: The compiler uses full reporting (level 3).

SOLUTION: Correct the entry, and then recompile.

P1079 Invalid compression option.

PROBLEM: The COMPRESS option specifies an incorrect value.

RESULT: The compiler uses the default value off.

SOLUTION: Correct the entry, and then recompile. The correct syntax is:

COMPRESS=[off, 0, no, false] or [on, 1, yes, true]

P1081 Invalid title string.

PROBLEM: The TITLE option defines a string that is empty or contains more than 32 characters.

RESULT: The compiler truncates the title at 32 characters.

SOLUTION: Add a title if the option is empty or shorten the title if it is too long, and then recompile.

P1083 Invalid context identification number.

PROBLEM: A context number in the [MAP] or [ALIAS] section is empty or contains invalid characters.

RESULT: The compiler ignores the line with the invalid context number.

SOLUTION: Check the format of all context numbers. Context numbers can be specified as decimal or hexadecimal. Correct any invalid numbers, and then recompile.

P1085 Unrecognized text.

PROBLEM: The compiler found some text in the Help project file that it cant recognize.

RESULT: The compiler ignores the line with the unrecognized text.

SOLUTION: Remove the unsupported text from the Help project file, and then recompile.

P1086 Invalid font-range syntax.

PROBLEM: The font range syntax used is invalid. A font range consists of a low and high point size, separated by a hyphen (-).

RESULT: The compiler ignores the option.

SOLUTION: Correct the syntax, and then recompile. The correct syntax is:

MAPFONTSIZE=m[-n]:p

P1089 Unrecognized sort ordering.

PROBLEM: You have specified an ordering that is not supported by the compiler.

RESULT: The compiler uses the default English (U.S.) sorting order.

SOLUTION: Correct the entry, and then recompile. The correct syntax is:

LANGUAGE=scandinavian

RTF Topic File Error Messages

This section lists all the possible errors that might occur when the 3.0 Help compiler is processing the RTF topic file(s).

R2001 Unable to open bitmap file filename.

PROBLEM: The specified bitmap file could not be found or is not in a recognizable bitmap format. This is an MS-DOS file error or an out-of-memory condition.

RESULT: The Help file displays the Unable to display picture message in the compiled topic instead of the bitmap.

SOLUTION: Check to see if the file exists; if it does, check its format. If necessary save the file again in your paint or draw program, and then recompile.

R2003 Unable to include bitmap file filename.

PROBLEM: The specified file could not be found or is unreadable. This is an MS-DOS file error or an out-of-memory condition.

RESULT: The built Help file displays the Unable to display picture message in the topic instead of the bitmap.

SOLUTION: Check to see if the file exists, and also check the amount of available memory.

R2005 Disk full.

PROBLEM: There is not enough disk space to write the Help file to disk.

RESULT: The compiler aborts the build.

SOLUTION: Create more space on the destination disk, and then recompile.

R2009 Cannot use reserved DOS device name for file filename.

PROBLEM: The file has a reserved MS-DOS filename, such as COM1, LPT2, or PRN. This is an MS-DOS file error.

RESULT: The compiler ignores the file.

SOLUTION: Rename the file, and then recompile.

R2013 Output file filename already exists as a directory.

PROBLEM: A directory in the Help project directory has the same name as the requested Help file. This is an MS-DOS file error.

RESULT: The compiler ignores the file.

SOLUTION: Move or rename the directory or the Help project file, and then recompile.

R2015 Output file filename already exists as read-only.

PROBLEM: The Help compiler cannot overwrite files with a read-only attribute. For example, this error occurs if you rebuild an existing .HLP file and the .HLP file is read-only.

RESULT: The compiler aborts the build.

SOLUTION: Change the files read-only attribute if you want the Help compiler to overwrite the file; otherwise, rename the file.

R2017 Path for file filename exceeds limit of 79 characters.

PROBLEM: The combined length of the path and filename must not be more than the MS-DOS limit of 79 characters.

RESULT: The compiler ignores the file.

SOLUTION: Shorten the path, and then recompile.

R2019 Cannot open file filename.

PROBLEM: The specified file could not be found or is unreadable. This is an MS-DOS file error or an

out-of-memory condition.

RESULT: The compiler ignores the file.

SOLUTION: Check to see if the file exists, and also check the amount of available memory.

R2021 Cannot find file filename.

PROBLEM: The specified file could not be found or is unreadable. This is an MS-DOS file error or an out-of-memory condition.

RESULT: The compiler ignores the file.

SOLUTION: Check to see if the file exists, and also check the amount of available memory.

R2023 Not enough memory to build Help file.

PROBLEM: The Help compiler does not have enough memory to complete the build process.

RESULT: The compiler does not create a Help file.

SOLUTION: To free memory, unload any unneeded applications, device drivers, and memory-resident programs. You can also turn off compression in the Help project file, replace any inline bitmaps pasted directly in the topic files with bitmap references, or compile under OS/2 rather than MS-DOS.

R2025 File environment error.

PROBLEM: The compiler does not have enough available file handles to continue the build. This error is likely to occur if you are compiling in an MS-DOS box or if you are using Help Author and have Word for Windows running at the same time.

RESULT: The compiler aborts the build.

SOLUTION: If possible, increase the FILES setting in the CONFIG.SYS file to FILES=50. Then reboot your computer and recompile.

R2027 Build tag tagname not defined in [BUILDTAGS] section of project file.

PROBLEM: The specified build tag has been assigned to a topic but not declared in the Help project file.

RESULT: The compiler ignores the build tag defined in the topic.

SOLUTION: Include the build tag in the [BUILDTAGS] section, and then recompile.

R2033 Context string in Map section not defined in any topic.

PROBLEM: The compiler cannot find a context string listed in the [MAP] section in any of the topics in the build.

RESULT: The compiler ignores the line in the [MAP] section. If the application uses that number in the WinHelp function, Help will display the Help topic not found error message,

SOLUTION: Check to be sure that all the context strings listed in the [MAP] section are assigned to topics included in the build and that they are all spelled correctly. Then recompile.

R2035 Build expression missing from project file.

PROBLEM: The topics have build tag footnotes, but there is no BUILD= expression in the Help project file.

RESULT: The compiler includes all topics in the build.

SOLUTION: Add a build expression to the Help project file and recompile if you want your build tags to work. Otherwise, you can safely ignore this message.

R2037 File filename cannot be created, due to previous error(s).

PROBLEM: The compiler cannot continue because there are topics remaining to be processed.

RESULT: The compiler does not create a Help file.

SOLUTION: Correct the errors that preceded this error, and then recompile.

R2039 Unrecognized table formatting in topic topicnumber of file filename.

PROBLEM: The compiler ignores table formatting that is unsupported in Help.

RESULT: The compiler aborts the build.

SOLUTION: Reformat the entries as text paragraphs with hanging indents or as side-by-side paragraphs, and then recompile.

R2041 Jump context_string unresolved in topic topicnumber of file filename.

PROBLEM: The specified topic contains a context string that identifies a nonexistent topic.

RESULT: The Help application displays the Help topic not found error message when the user chooses a hot spot with the unresolved context string.

SOLUTION: Check the topic for spelling errors in the context string, and also check to see if the requested topic is included in the build.

R2043 Hotspot text cannot spread over paragraphs.

PROBLEM: A paragraph marker is part of the hidden text.

RESULT: The compiler ignores the paragraph marker formatted as hidden text, so the two paragraphs will run together.

SOLUTION: Reformat the paragraph marker as plain text, and then recompile.

R2045 Maximum number of tab stops reached in topic topicnumber of file filename.

PROBLEM: The maximum number of custom tab stops that can be defined for a paragraph is 32.

RESULT: The compiler uses the default tab stops after the 32nd tab.

SOLUTION: Reduce the number of tab stops to 32 or fewer, and then recompile.

R2047 File filename not created.

PROBLEM: There are no topics to compile, or the build expression is false for all topics.

RESULT: The compiler does not create a Help file.

SOLUTION: Add a topic filename to the [FILES] section in the Help project file, or change the build expression so that all topics are not excluded from the build, and then recompile.

R2049 Context string text too long in topic topicnumber of file filename.

PROBLEM: The context string hidden text cannot exceed 64 characters.

RESULT: The compiler ignores the context string.

SOLUTION: Shorten the context string, and then recompile.

R2051 File filename is not a valid RTF topic file.

PROBLEM: The specified file is not an RTF file.

RESULT: The compiler ignores the file.

SOLUTION: Make sure that you save the topic as RTF from your word processor, and then recompile.

R2053 Font fontname in file filename not in RTF font table.

A font not defined in the RTF header has been entered into the topic. The compiler uses the default system font.

R2055 File filename is not a usable RTF topic file.

PROBLEM: The specified file contains a valid RTF header, but the content is not RTF or is corrupted.

RESULT: The compiler ignores the file.

SOLUTION: Resave the topic as RTF from your word processor, and then recompile.

R2057 Unrecognized graphic format in topic topicnumber of file filename.

PROBLEM: The compiler supports only Windows bitmaps (.BMP) and Windows device-independent bitmaps (.DIB).

RESULT: The compiler ignores the graphic.

SOLUTION: Make sure that you have not used a metafile, a Macintosh picture format, or some other unsupported graphic format.

R2059 Context string identifier already defined in topic topicnumber of file filename.

PROBLEM: There is more than one context-string identifier footnote for the specified topic.

RESULT: The compiler uses the identifier defined in the first # footnote.

SOLUTION: Remove one of the context string identifiers, and then recompile.

R2061 Context string contextname already used in file filename.

PROBLEM: The specified context string was previously assigned to another topic.

RESULT: The compiler ignores the duplicate context string, and the topic has no identifier.

SOLUTION: Change the second context string so that it is unique, and then recompile.

R2063 Invalid context-string identifier for topic topicnumber of file filename.

PROBLEM: The context string footnote contains non-alphanumeric characters or is empty.

RESULT: The compiler does not assign the topic an identifier.

SOLUTION: Change the characters in the context string so that it is valid, and then recompile.

R2065 Context string defined for index topic is unresolved.

PROBLEM: The Index topic defined in the Help project file could not be found.

RESULT: The compiler uses the first topic in the build as the Index.

SOLUTION: Compare the context string in the INDEX option to the context string used in the Index topic to be sure they are the same. Change one of the strings to match the other, and then recompile.

R2067 Footnote text too long in topic topicnumber of file filename.

PROBLEM: The footnote text cannot exceed the limit of 1000 characters.

RESULT: The compiler ignores the footnote.

SOLUTION: Reduce the length of the footnote text, and then recompile.

R2069 Build tag footnote not at beginning of topic topicnumber of file filename.

PROBLEM: The build tag footnote marker, if used, has to be the first character in the topic.

RESULT: The compiler ignores the build tag footnote, and the topic is not assigned a build tag.

SOLUTION: Move the build tag footnote to the beginning of the topic, before other footnotes, and then recompile.

R2071 Footnote text missing in topic topicnumber of file filename.

PROBLEM: The specified topic contains a footnote that has no characters.

RESULT: The compiler ignores the footnote.

SOLUTION: Add characters to the footnote, and then recompile.

R2073 Keyword string is null in topic topicnumber of file filename.

PROBLEM: There are no characters in the keyword footnote.

RESULT: The compiler ignores the keyword footnote, and the topic does not have any keywords.

SOLUTION: Open the footnote window and type some keywords next to the keyword footnote character for the topic, and then recompile.

R2075 Keyword string too long in topic topicnumber of file filename.

PROBLEM: The keyword string exceeds the limit of 255 characters.

RESULT: The compiler ignores the keyword footnote.

SOLUTION: Shorten the length of the keyword string, or add another keyword footnote for the extra

keywords, and then recompile.

R2077 Keyword(s) defined without title in topic topicnumber of file filename.

PROBLEM: The topic has a keyword assigned to it, but no title.

RESULT: The topic will appear as >>Untitled Topic<< in the Search dialog box and in the history list.

SOLUTION: Add a topic title footnote to the topic if you want a title to appear in the Search dialog box when the user selects this keyword.

R2079 Browse sequence string is null in topic topicnumber of file filename.

PROBLEM: The browsesequence footnote for the specified topic contains no sequence characters.

RESULT: The compiler ignores the browse sequence footnote.

SOLUTION: Add sequence characters to the browse sequence footnote, and then recompile.

R2081 Browse sequence string too long in topic topicnumber of file filename.

PROBLEM: The browse sequence string exceeds the limit of 128 characters.

RESULT: The compiler ignores the browse sequence footnote.

SOLUTION: Shorten the length of the sequence string, and then recompile.

R2083 Missing sequence number in topic topicnumber of file filename.

PROBLEM: A browsesequence number ends with a colon (:) for the specified topic.

RESULT: The compiler ignores the browse sequence footnote.

SOLUTION: Remove the colon or enter a minor sequence number, and then recompile.

R2085 Sequence number already defined in topic topicnumber of file filename.

PROBLEM: A browse-sequence footnote already exists for the specified topic.

RESULT: The compiler uses the first browse sequence footnote and ignores the duplicate footnote.

SOLUTION: Remove the duplicate browse sequence footnote, and then recompile.

R2087 Build tag too long.

PROBLEM: A build tag for the specified topic exceeds the limit of 32 characters.

RESULT: The compiler ignores the build tag assigned to the topic.

SOLUTION: Shorten the length of the build tag to 32 or fewer characters, and then recompile.

R2089 Title string null in topic topicnumber of file filename.

PROBLEM: The title footnote for the specified topic contains no characters.

RESULT: The compiler ignores the title footnote, and the topic does not have a title.

SOLUTION: Open the footnote window and type the title string next to the topic title footnote character for the topic, and then recompile.

R2091 Title too long in topic topicnumber of file filename.

PROBLEM: The title for the specified topic exceeds 128 characters.

RESULT: The compiler truncates the title string at 128 characters.

SOLUTION: Shorten the length of the title string, and then recompile.

R2093 Title titlename in topic topicnumber of file filename used previously.

PROBLEM: The specified title has previously been assigned to another topic.

RESULT: The compiler uses the first title and ignores the duplicate title.

SOLUTION: Remove the duplicate title, and then recompile.

R2095 Title defined more than once in topic topicnumber of file filename.

PROBLEM: There is more than one title footnote in the specified topic.

RESULT: The compiler uses the first title footnote and ignores the duplicate title footnote.

SOLUTION: Remove the duplicate title footnote, and then recompile.

R2501 Using old key-phrase table.

PROBLEM: The compiler found an existing key-phrase table for the Help file you are building, and the Help project file does not specify not to use the table.

RESULT: The compiler uses the key-phrase table that was generated from an earlier build, instead of creating a new table.

SOLUTION: To achieve maximum compression during a build, you must delete the old key-phrase table (.PH file) before each recompilation.

R2503 Out of memory during text compression.

PROBLEM: The compiler encountered a memory limitation during compression.

RESULT: The Help compiler continues the build without compressing the Help file.

SOLUTION: Free more memory and then recompile. For suggestions on how to free more memory fo the build, refer to the Getting Around Memory Problems in Chapter 17. If you cant overcome your memory constraints, you may have to compile without compression or remove some of the graphics in the topic files.

R2505 File environment error during text compression.

PROBLEM: The compiler does not have enough available file handles to compress the Help file. This error is likely to occur if you are compiling in an MS-DOS box or if you are using Help Author and have Word for Windows running at the same time.

RESULT: The Help compiler continues the build without compressing the Help file.

SOLUTION: If possible, increase the FILES setting in the CONFIG.SYS file to FILES=50. Then reboot your computer and recompile.

R2507 DOS file error during text compression.

PROBLEM: The compiler encountered a problem accessing a disk file while it was compressing the Help file. This is an MS-DOS file error.

RESULT: The Help compiler continues the build without compressing the Help file.

SOLUTION: Check to make sure you have enough free disk space or look for other MS-DOS related problems, and then recompile.

R2509 Error during text compression.

PROBLEM: One of the three compression errorsR2503, R2505, or R2507has occurred.

RESULT: The Help compiler continues the build without compressing the Help file.

SOLUTION: Check the solutions section for each of the other errors to see what the problem might be. When you have corrected the problem, recompile.

R2701 Internal error.

PROBLEM: The compiler is unable to create the data structure (b-tree) containing all the titles in the Help file.

RESULT: The compiler aborts the build.

SOLUTION: Contact Microsoft Product Support Services.

R2703 Internal error.

PROBLEM: The compiler cannot create the bitmap file because of a low-memory condition, lack of sufficient free disk space, or insufficient file handles.

RESULT: The compiler aborts the build.

SOLUTION: Contact Microsoft Product Support Services.

R2705 Internal error.

PROBLEM: The phrase file used for compression has disappeared between the time it is created (or last checked) and when the compiler needs to use it. This will likely occur if the hard disk drive is low on disk space or if you are building the Help file over a network.

RESULT: The compiler aborts the build.

SOLUTION: Contact Microsoft Product Support Services.

R2707 Internal error.

PROBLEM: The compiler has encountered a disk error other than out of memory, out of disk space, or out of file handles. This error may indicate a problem with your hard disk drive, such as bad sectors or too many files in a directory.

RESULT: The compiler aborts the build.

SOLUTION: Contact Microsoft Product Support Services.

R2709 Internal error.

PROBLEM: The compiler cant open the output system file. This problem should only occur if you are using a nonstandard release or beta release of Help version 3.0.

RESULT: The compiler aborts the build.

SOLUTION: Contact Microsoft Product Support Services.

Chapter 19 The WinHelp API

If Windows Help is used as an online Help system with context sensitivity, the application must be programmed so that the user can access the Help application and the appropriate Help file. The WinHelp API supports both context-sensitive and topical searches of the Help file.

This chapter explains the WinHelp function and describes different ways to call Help from a Windows application.

The WinHelp Function

An application makes a Help system available to the user by calling the WinHelp function. The WinHelp function uses the following C-language syntax.

Syntax

```
BOOL WinHelp (hWnd, lpszHelpFile, wCommand, dwData)

HWND hwnd;          /* handle of window requesting help    */
LPCSTR lpszHelpFile; /* address of directory-path string   */
UINT wCommand;      /* type of help                        */
DWORD dwData;       /* additional data                     */
```

The WinHelp function starts Windows Help (WINHELP.EXE) and passes optional data indicating the nature of the help requested by the application. The application specifies the name and, where required, the path of the Help file that the Help application is to display.

Parameters

hWnd — Identifies the window requesting Help. The WinHelp function uses this handle to keep track of which applications have requested Help.

lpszHelpFile — Points to a null-terminated string containing the path, if necessary, and the name and extension of the Help file that the Help application is to display. The Help file extension (usually .HLP) is required, and a directory path to the Help file is recommended.

The filename may be followed by an angle bracket (>) and the name of a secondary window if the topic is to be displayed in a secondary window rather than the main Help window. The name of the secondary window must have been defined in the [WINDOWS] section of the Help project file for the Help file being called. For a description of the possible effects on the main and secondary Help windows, see the following Comments section.

wCommand — Specifies the type of help requested. For a list of possible values and how they affect the value to place in the dwData parameter, see the following Comments section.

dwData — Specifies additional data. The value used depends on the value of the wCommand parameter. For a list of possible values, see the following Comments section.

Return Value — WinHelp returns a nonzero value if the function is successful. Otherwise, the return value is zero.

Comments

Before closing the window that requested Help, the application must call WinHelp with wCommand set to HELP_QUIT. Until all applications have done this, Windows Help does not terminate.

The following table describes the possible effects on the main and secondary Help windows when using the >WindowName parameter:

State at time of call	Effect on window after API call
Main	Secondary
closed	closed
open	open

The following table shows the possible values for the wCommand parameter and the corresponding formats of the dwData parameter.

wCommand valuedwData format Action

HELP_CONTEXT
An unsigned long integer containing the context number for the topic. Displays the topic identified by a context number that has been defined in the [MAP] section of the Help project file.

HELP_CONTEXTNOFOCUS
An unsigned long integer containing the context number for the topic. Displays the topic

identified by a context number that has been defined in the [MAP] section of the Help project file. Help does not change the focus to the window displaying the topic.

HELP_CONTEXTPOPUP

An unsigned long integer containing the context number for the topic. Displays in a pop-up window the topic identified by a context number that has been defined in the [MAP] section of the Help project file. The main Help window is not displayed.

HELP_CONTENTS

Ignored; applications should set to 0L Displays the topic defined by the CONTENTS option in [OPTIONS] section of the Help project file.

HELP_SETCONTENTS

An unsigned long integer containing the context number for the topic the application wants to designate as the Contents topic. Determines which Contents topic Help should display when a user presses F1 or chooses the Contents button in Help. This call should never be used with HELP_CONTENTS.If a Help file has two or more Contents topics, the application must assign one as the default. To ensure that the correct Contents topic remains set, the application should call WinHelp with wCommand set to HELP_SETCONTENTS and dwData specifying the corresponding context identifier. Each call to WinHelp should be followed with a command set to HELP_CONTEXT.

HELP_POPUPID

A long pointer to a string that contains the context string of the topic to be displayed. Displays in a pop-up window the topic identified by a specific context string. The main Help window is not displayed.

HELP_KEY

A long pointer to a string that contains a keyword for the requested topic. Displays the topic in the keyword list that matches the keyword passed in the dwData parameter if there is one exact match. If there is more than one match, it displays the first topic found. If there is no match, it displays an error message.

HELP_PARTIALKEY

A long pointer to a string that contains a keyword for the requested topic. Displays the topic in the keyword list that matches the keyword passed in the dwData parameter if there is one exact match. If there is more than one match, it displays the Search dialog box with the topics found listed in the Go To box. If there is no match, it displays the Search dialog box.If you just want to bring up the Search dialog box without passing a keyword (the third result), you should use a long pointer to an empty string.

HELP_MULTIKEY

A long pointer to the MULTIKEYHELP structure, as defined in WINDOWS.H. This structure specifies the table footnote character and the keyword. Displays the topic identified by a keyword in an alternate keyword table.

HELP_COMMAND

A long pointer to a string that contains a Help macro to be executed. Executes the Help macro string specified in the dwData parameter. Help must be running and the Help file must be open when Help receives this API message; otherwise, Help may ignore this message.

HELP_SETWINPOS

A far pointer to the HELPWININFO structure, as defined in WINDOWS.H. This structure specifies the size and position of the main Help window or a secondary window. Positions the Help window according to the data passed. If the Help window is minimized, it is opened first and then positioned.

HELP_CLOSEWINDOW

Ignored; applications should set to 0L. Closes the main Help window, or a secondary window if specified in the lpszHelpFile argument.

HELP_FORCEFILE

Ignored; applications should set to 0L. Ensures that the correct Help file is displayed. If the correct Help file is currently displayed, there is no action. If the correct Help file is not displayed, WinHelp opens the correct file and displays the topic defined by the CONTENTS option in the [OPTIONS] section of the Help project file.

HELP_HELPONHELP

Ignored; applications should set to 0L. Displays the Contents topic of the designated How To Use Help file.

HELP_QUIT

Ignored; applications should set to 0L. Informs the Help application that Help is no longer needed. If no other applications have requested Help, Windows closes the Help application.

The following table shows the complete list of #defines for WinHelp commands.

wCommand	**Hexadecimal value**
#define HELP_CONTEXT	0x0001
#define HELP_QUIT	0x0002
#define HELP_INDEX	0x0003 (Windows Help version 3.0)
#define HELP_CONTENTS	0x0003
#define HELP_HELPONHELP	0x0004
#define HELP_SETINDEX	0x0005 (Windows Help version 3.0)
#define HELP_SETCONTENTS	0x0005
#define HELP_CONTEXTPOPUP	0x0008
#define HELP_FORCEFILE	0x0009
#define HELP_KEY	0x0101
#define HELP_COMMAND	0x0102
#define HELP_POPUPID	0x0104
#define HELP_PARTIALKEY	0x0105
#define HELP_CLOSEWINDOW	0x0107
#define HELP_CONTEXTNOFOCUS	0x0108
#define HELP_MULTIKEY	0x0201
#define HELP_SETWINPOS	0x0203

Using Help in a Windows Application

Windows applications can offer help to their users by using the WinHelp function to start Windows Help and display topics in the applications Help file. The WinHelp function gives a Windows application complete access to the Help file, as well as to the menus and commands of Windows Help. Many applications use WinHelp to implement context-sensitive Help. Context-sensitive Help enables users to view topics about specific windows, menus, menu items, and control windows by selecting the item with the keyboard or the mouse. For example, a user can learn about the Open command on the File menu by selecting the command (using the direction keys) and pressing the F1 key.

Choosing Help from the Help Menu

Every application should provide a Help menu to allow the user to open the Help file with either the mouse or the keyboard. The Help menu should contain at least one Contents menu item that, when chosen, displays the Contents or the main topic in the Help file. To support the Help menu, the applications main window procedure should check for the Contents menu item and call the WinHelp function, as in the following example:

```
case WM_COMMAND:
   switch (wParam) {
   case IDM_HELP_CONTENTS:
   WinHelp(hwnd, "myhelp.hlp", HELP_CONTENTS, 0L);
   return 0L;
   .
   .
   .
   }
   break;
```

You can add other menu items to the Help menu for topics containing general information about the application. For example, if your Help file contains a topic that describes how to use the keyboard, you can place a Keyboard menu item on the Help menu. To support additional menu items, your application must specify either the context string or the context identifier for the corresponding topic when it calls the WinHelp function. The following example uses a Help macro to specify the context string IDM_HELP_KEYBOARD for the Keyboard topic:

```
case IDM_HELP_KEYBOARD:
   WinHelp(hwnd, "myhelp.hlp", HELP_COMMAND,
   (DWORD)(LPSTR)"JumpID(\"myhelp.hlp\",\"IDM_HELP_KEYBOARD\")");
   return 0L;
```

A better way to display a topic is to use a context identifier. To do this, the Help file must assign a unique number to the corresponding context string, in the [MAP] section of the Help project file. For example, the following section assigns the number 101 to the context string IDM_HELP_KEYBOARD:

```
[MAP]
IDM_HELP_KEYBOARD      101
```

An application can display the Keyboard topic by specifying the context identifier in the call to the WinHelp function, as in the following example:

```
#define IDM_HELP_KEYBOARD   101
WinHelp(hwnd, "myhelp.hlp", HELP_CONTEXT, (DWORD)IDM_HELP_KEYBOARD);
```

To make maintenance of an application easier, most programmers place their defined constants (such as IDM_HELP_KEYBOARD in the previous example) in a single header file. As long as the names of the defined constants in the header file are identical to the context strings in the Help file, you can include the header file in the [MAP] section to assign context identifiers, as shown in the following example:

```
[MAP]
#include <myhelp.h>
```

If the defined constants in the header file are different from the context strings in the Help file, you can use Help Author to perform the context mapping.

Defining More Than One Help Contents

Some applications may require more than one Help Contents topic, depending on the state of the application. For example, the interface and options in PIF Editor are different for Standard mode and Enhanced mode. Therefore, the Help offered by PIF Editor depends on which mode the user is running. When running in Standard mode, the user sees the Help Contents tailored to that mode, and the user see a different Help Contents when running in Enhanced mode.

If a Help file contains two or more Contents topics, the application can assign one as the default by using the context identifier and the HELP_SETCONTENTS value in a call to the WinHelp function.

The sample application Helpex applies a somewhat different model by defining a function that the application can use instead of WinHelp. This function sends the HELP_SETCONTENTS value and sets the Contents topic without opening Windows Help.

```
BOOL MyWinHelp(hwnd, lpHelpfile, wCommand, dwData)
HWND     hwnd;
LPSTR    lpHelpfile;
WORD     wCommand;
DWORD    dwData;
{
   static DWORD      ctxContents = (DWORD)-1L;
   if (wCommand == HELP_SETCONTENTS) {
   ctxContents = dwData;
   return (TRUE);
   }
   if (wCommand == HELP_CONTENTS && ctxContents != (DWORD)-1L) {
   WinHelp(hwnd, lpHelpfile, HELP_CONTEXT, ctxContents);
   }
   else {
   WinHelp(hwnd, lpHelpfile, wCommand, dwData);
   }
   if (wCommand != HELP_QUIT && ctxContents != (DWORD)-1L) {
   WinHelp (hwnd, lpHelpfile, HELP_SETCONTENTS, dwData);
   }
}
```

After the Contents topic is set, the application can use any WinHelp API, including HELP_CONTENTS.

Choosing Help with the Keyboard

An application can enable the user to choose a help topic with the keyboard by intercepting the F1 key. Intercepting this key lets the user select a menu, menu item, dialog box, message box, or control window and view Help for it with a single keystroke.

> Note: The sample code in this section assumes that resource ID numbers are all unique and correspond directly to context ID numbers used for Help. Although this assumption makes things more restrictive than they have to be, it is a worthwhile option to consider using because it simplifies the code.

To intercept the F1 key, the application must install a message filter procedure by using the SetWindowsHook function. This allows the application to examine all keystrokes for the application, regardless of which window has the input focus. If the filter procedure detects the F1 key, it posts a WM_F1DOWN message (application-defined) to the applications main window procedure. The procedure then determines which Help topic to display.

The filter procedure should have the following form:

```
int FAR PASCAL FilterFunc(nCode, wParam, lParam)
int nCode;
WORD wParam;
DWORD lParam;
{
   LPMSG lpmsg = (LPMSG)lParam;
   if ((nCode == MSGF_DIALOGBOX || nCode == MSGF_MENU) &&
         lpmsg->message == WM_KEYDOWN && lpmsg->wParam == VK_F1) {
      PostMessage(hWnd, WM_F1DOWN, nCode, 0L);
   }
   DefHookProc(nCode, wParam, lParam, &lpFilterFunc);
   return 0;
}
```

The application should install the filter procedure after creating the main window, as shown in the following example:

```
lpProcInstance = MakeProcInstance(FilterFunc, hInstance);
if (lpProcInstance == NULL)
   return FALSE;
lpFilterFunc = SetWindowsHook(WH_MSGFILTER, lpProcInstance);
```

> Note: Be sure that FilterFunc is exported in the .DEF file.

Like all callback functions, the filter procedure must be exported by the application.

The filter procedure sends a WM_F1DOWN message only when the F1 key is pressed in a dialog box, message box, or menu. Many applications also display the Contents topic if no menu, dialog box, or message box is selected when the user presses the F1 key. In this case, the application should define the F1 key as an accelerator key that starts Help.

To create an accelerator key, the applications resource definition file must define an accelerator table, as follows:

```
1 ACCELERATORS
BEGIN
```

```
    VK_F1, IDM_HELP_CONTENTS, VIRTKEY
END
```

To support the accelerator key, the application must load the accelerator table by using the LoadAccelerators function and translate the accelerator keys in the main message loop by using the TranslateAccelerator function.

In addition to installing the filter procedure, the application must keep track of which menu, menu item, dialog box, or message box is currently selected. In other words, when the user selects an item, the application must set a global variable indicating the current context. For dialog and message boxes, the application should set the dwCurrentHelpId global variable immediately before calling the DialogBox or MessageBox function. For menus and menu items, the application should set the variable whenever it receives a WM_MENUSELECT message. As long as identifiers for all menu items and controls in an application are unique, an application can use code similar to the following example to monitor menu selections:

```
case WM_MENUSELECT:
    /*
     * Set dwCurrentHelpId to the Help ID of the menu item that is
     * currently selected.
     */
    if (HIWORD(lParam) == 0)            /* no menu selected */
        dwCurrentHelpId = ID_NONE;
    else if (lParam & MF_POPUP) {             /* pop-up selected */
        if ((HMENU)wParam == hMenuFile)
            dwCurrentHelpId = ID_FILE;
        else if ((HMENU)wParam == hMenuEdit)
            dwCurrentHelpId = ID_EDIT;
        else if ((HMENU)wParam == hMenuHelp)
            dwCurrentHelpId = ID_HELP;
        else
            dwCurrentHelpId = ID_SYSTEM;
    }
    else        /* menu item selected */
        dwCurrentHelpId = wParam;
    break;
```

In this example, the hMenuFile, hMenuEdit, and hMenuHelp parameters must previously have been set to specify the corresponding menu handles. An application can use the GetMenu and GetSubMenu functions to retrieve these handles.

When the main window procedure finally receives a WM_F1DOWN message, it should use the current value of the global variable to display a Help topic. The application can also provide Help for individual controls in a dialog box by determining which control has the focus at this point, as shown in the following example:

```
case WM_F1DOWN:
    /*
     * If there is a current Help context, display it.
     */
    if (dwCurrentHelpId != ID_NONE) {
        DWORD dwHelp = dwCurrentHelpId;
```

```
      /*
       * Check for context-sensitive Help for individual dialog box
       * controls.
       */
      if (wParam == MSGF_DIALOGBOX) {
         WORD wID = GetWindowWord(GetFocus(), GWW_ID);
         if (wID != IDOK && wID != IDCANCEL)
         dwHelp = (DWORD) wID;
      }
      WinHelp(hWnd, szHelpFileName, HELP_CONTEXT, dwHelp);
      /*
       * This call is used to remove the highlighting from the system
       * menu, if necessary.
       */
      DrawMenuBar(hWnd);
   }
   break;
```

When the application ends, it must remove the filter procedure by using the UnhookWindowsHook function and free the procedure instance for the function by using the FreeProcInstance function.

Choosing Help with the Mouse

An application can let the user choose a Help topic with the mouse by intercepting mouse input messages and calling the WinHelp function. To distinguish requests to view Help from regular mouse input, the user must press the SHIFT+F1 key combination. In such cases, the application sets a global variable when the user presses the key combination and changes the cursor shape to a question-mark pointer to indicate that the mouse can be used to choose a Help topic.

To detect the SHIFT+F1 key combination, an application checks for the VK_F1 virtual-key value in each WM_KEYDOWN message sent to its main window procedure. It also checks for the VK_ESCAPE virtual-key code. The user presses the ESC key to quit Help and restore the mouse to its regular function. The following example checks for these keys:

```
case WM_KEYDOWN:
   if (wParam == VK_F1) {
      /* If Shift+F1, turn on Help mode and set Help cursor. */
      if (GetKeyState(VK_SHIFT)) {
         bHelp = TRUE;
         SetCursor(hHelpCursor);
         return (DefWindowProc(hWnd, message, wParam, lParam));
      }
      /* If F1 without shift, call Help main index topic. */
      else {
         WinHelp(hWnd,szHelpFileName,HELP_INDEX,0L);
      }
   }
   else if (wParam == VK_ESCAPE && bHelp) {
      /* To escape during Help mode, turn off Help mode. */
      bHelp = FALSE;
      SetCursor((HCURSOR) GetClassWord(hWnd, GCW_HCURSOR));
   }
   break;
```

Until the user clicks the mouse or presses the ESC key, the application responds to WM_SETCURSOR messages by resetting the cursor to the arrow and question-mark combination.

```
case WM_SETCURSOR:
   /*
    * In Help mode it is necessary to reset the cursor in response to
    * every WM_SETCURSOR message. Otherwise, by default, Windows resets
    * the cursor to that of the window class.
    */
   if (bHelp) {
         SetCursor(hHelpCursor);
         break;
   }
   return (DefWindowProc(hWnd, message, wParam, lParam));
case WM_INITMENU:
   if (bHelp) {
```

```
            SetCursor(hHelpCursor);
        }
        return (TRUE);
```

If the user clicks the mouse button in a nonclient area of the application window while in Help mode, the application receives a WM_NCLBUTTONDOWN message. By examining the wParam value of this message, the application can determine which context identifier to pass to WinHelp.

```
    case WM_NCLBUTTONDOWN:
        /*
         * If we are in Help mode (Shift+F1), display context-sensitive
         * Help for nonclient area.
         */
        if (bHelp) {
            dwHelpContextId =
                (wParam == HTCAPTION) ?(DWORD) HELPID_TITLE_BAR:
                (wParam == HTSIZE) ? (DWORD) HELPID_SIZE_BOX:
                (wParam == HTREDUCE) ? (DWORD) HELPID_MINIMIZE_ICON:
                (wParam == HTZOOM) ? (DWORD) HELPID_MAXIMIZE_ICON:
                (wParam == HTSYSMENU) ?(DWORD) HELPID_SYSTEM_MENU:
                (wParam == HTBOTTOM) ? (DWORD) HELPID_SIZING_BORDER:
                (wParam == HTBOTTOMLEFT) ? (DWORD) HELPID_SIZING_BORDER:
                (wParam == HTBOTTOMRIGHT) ?(DWORD) HELPID_SIZING_BORDER:
                (wParam == HTTOP) ?(DWORD) HELPID_SIZING_BORDER:
                (wParam == HTLEFT) ?(DWORD) HELPID_SIZING_BORDER:
                (wParam == HTRIGHT) ?(DWORD) HELPID_SIZING_BORDER:
                (wParam == HTTOPLEFT) ?(DWORD) HELPID_SIZING_BORDER:
                (wParam == HTTOPRIGHT) ? (DWORD) HELPID_SIZING_BORDER:
                (DWORD) 0L;
            if (!((BOOL) dwHelpContextId))
                return (DefWindowProc(hWnd, message, wParam, lParam));
            bHelp = FALSE;
            WinHelp(hWnd, szHelpFileName, HELP_CONTEXT, dwHelpContextId);
            break;
        }
        return (DefWindowProc(hWnd, message, wParam, lParam))
```

If the user clicks a menu item while in Help mode, the application intercepts the WM_COMMAND message and sends the Help request:

```
    case WM_COMMAND:
        /* Are we in Help mode (Shift+F1)? */
        if (bHelp) {
            bHelp = FALSE;
            WinHelp(hWnd,szHelpFileName,HELP_CONTEXT, (DWORD)wParam);
            return NULL;
        }
```

Searching for Help with Keywords

An application can enable the user to search for Help topics based on full or partial keywords. This method is similar to employing the Search dialog box in Windows Help to find useful topics. The following example searches for the keyword Keyboard and displays the corresponding topic, if found:

```
WinHelp (hWnd, "myhelp.hlp", HELP_KEY, "Keyboard");
```

If the topic is not found, Windows Help displays an error message. If more than one topic has the same keyword, Windows Help displays only the first topic.

An application can give the user more options in a search by specifying partial keywords. When a partial keyword is given, Windows Help usually displays the Search dialog box to allow the user to continue the search or return to the application. However, if there is an exact match and no other topic exists with the given keyword, Windows Help displays the topic. The following example opens the Search dialog box and selects the first keyword in the list starting with the letters Ke:

```
WinHelp(hwnd, "myhelp.hlp", HELP_PARTIALKEY, "Ke");
```

When the HELP_KEY and HELP_PARTIALKEY values are specified in the WinHelp function, Windows Help searches the K keyword table. This table contains keywords generated by using the letter K with \ footnote statements in the topic file. However, your application may have commands or terms that correspond to terms in a similar, but different, application.

An application can search alternative keyword tables by specifying the HELP_MULTIKEY value in the WinHelp function. In this case, the application must specify the footnote character for the alternate keyword table and the full keyword in a MULTIKEYHELP structure. The MULTIKEYHELP structure specifies a keyword table and an associated keyword to be used by the Windows Help application.

The MULTIKEYHELP structure has the following form:

```
typedef struct tagMULTIKEYHELP { /* mkh */
   WORD  mkSize;
   BYTE  mkKeyList;
   BYTE  szKeyPhrase[1];
} MULTIKEYHELP;
```

where

mkSize	Specifies the length, in bytes, of the MULTIKEYHELP structure, including the keyword (or phrase) and the associated keyword-table letter.
mkKeyList	Contains a single character that identifies the keyword table to be searched.
szKeyPhrase	Contains a null-terminated text string that specifies the keyword to be located in the alternate keyword table.

The following example illustrates a keyword search for the word frame in the alternate keyword table designated with the footnote character L:

```
HANDLE hqmk;
MULTIKEYHELP far *qmk;
char szKeyword[] = "frame";
case MULTIKEY:
   hqmk = GlobalAlloc(GHND, (sizeof(MULTIKEYHELP) + lstrlen(szKeyword)));
   if (hqmk == NULL)
      break;
   qmk = (MULTIKEYHELP far *) GlobalLock(hqmk);
   qmk -> mkSize = sizeof(MULTIKEYHELP) + strlen(szKeyword);
   qmk -> mkKeylist = 'L';
   lstrcpy(qmk->szKeyphrase, szKeyword);
```

```
WinHelp(hWnd, szHelpFileName, HELP_MULTIKEY, (DWORD) (LPSTR) qmk);
GlobalUnlock(hqmk);
GlobalFree(hqmk);
break;
```

Displaying Help in a Secondary Window

An application can display Help topics in secondary windows instead of in Windows Helps main window. Secondary windows are useful whenever the user does not need the full capabilities of Windows Help. The Windows Help menus and buttons are not available in secondary windows.

To display Help in a secondary window, the application specifies the name of the secondary window along with the name of the Help file. The following example displays the Help topic having the context identifier IDM_FILE_SAVE in the secondary window named wnd_menu:

```
WinHelp(hwnd, "myhelp.hlp>wnd_menu", HELP_CONTEXT, IDM_FILE_SAVE);
```

The name and characteristics of the secondary window must be defined in the [WINDOWS] section of the Help project file, as in the following example:

```
[WINDOWS]
wnd_menu = "Menus", (128, 128, 256, 256), 0
```

Windows Help displays the secondary window with the initial size and position specified in the [WINDOWS] section. However, an application can set a new size and position by specifying the HELP_SETWINPOS value in the WinHelp function. In this case, the application sets the members in a HELPWININFO structure to specify the window size and position. The following example sets the wnd_menu secondary window to a new size and position:

```
HANDLE hhwi;
LPHELPWININFO lphwi;
WORD wSize;
char *szWndName = "wnd_menu";
wSize = sizeof(HELPWININFO) + lstrlen(szWndName);
hhwi = GlobalAlloc(GHND, wSize);
lphwi = (LPHELPWININFO)GlobalLock(hhwi);
lphwi->wStructSize = wSize;
lphwi->x       = 256;
lphwi->y       = 256;
lphwi->dx= 767;
lphwi->dy= 512;
lphwi->wMax    = 0;
lstrcpy(lphwi->rgchMember, szWndName);
WinHelp(hwnd, "myhelp.hlp", HELP_SETWINPOS, lphwi);
GlobalUnlock(hhwi);
GlobalFree(hhwi);
```

Canceling Help

Windows Help requires an application to explicitly cancel Help so that Windows Help can free any resources it used to keep track of the application and its Help files. The application can do this at any time.

An application cancels Windows Help by calling the WinHelp function and specifying the HELP_QUIT value, as shown in the following example:

```
case WM_DESTROY:
   WinHelp (hwnd, "myhelp.hlp", HELP_QUIT, NULL);
```

If the application has made any calls to the WinHelp function, it must cancel Help before it closes its main window (for example, in response to the WM_DESTROY message in the main window procedure). If the application has opened more than one Help file, it must call WinHelp to cancel Help for each file. Windows Help remains running until all applications or dynamic-link libraries that have called WinHelp function have canceled Help.

We strongly recommend that an application use the HELP_QUIT parameter in its exit routine even if it has not actually sent any other WinHelp calls.

Chapter 20 Writing DLLs for Windows Help

A dynamic-link library (DLL) is an executable module containing functions that Windows-based applications (like Windows Help) can call to perform useful tasks. Windows Help accesses DLLs in two ways:

- Through DLL functions registered as Help macros in a Help Project file. These functions can then be used in hot spots and macro footnotes in topic files.
- Through embedded window (ew) references in topic files.

If Windows Helps internal macro set (described in Chapter 15, Help Macro Reference) doesnt provide all the functionality you need for a Help file, you can write your own DLLs to add extensions to Windows Help. In this chapter, youll learn two ways to extend Windows Help: by providing custom DLLs containing author-defined Windows Help macros and by providing DLL access through embedded-window references.

When creating DLLs for embedded windows, you must follow all the standard design requirements for Windows DLLs. This section assumes that youre familiar with these requirements, which are described in Chapter 20, Dynamic-Link Libraries, of the Microsoft Windows version 3.1 Software Development Kit (SDK), Guide to Programming manual..

To create DLLs for Windows Help, you need the Microsoft Windows Version 3.1 SDK and the Microsoft C Optimizing Compiler, version 6.0 or greater, or Microsoft Quick C for Windows version 1.0.

Included with this Authoring Guide is the source code for a sample DLL and a MAKEFILE for building the sample DLL. The source code is contained in the files DLLDEMO.C (for the Help macro DLL) and EWDEMO.C (for the embedded window DLL). You can use this sample code as a template for creating your own DLL. Be sure you study these source files and understand how they work before you continue in this chapter, since the sample DLL illustrates the concepts presented in this chapter and the remainder of this chapter assumes you are familiar with these files.

Creating Custom Help Macros

As explained in Chapter 14, Help Macros, you can use the RegisterRoutine macro to register any DLL function as a Windows Help macro. You might want to create a DLL whose sole purpose is to provide new Windows Help macros for your Help authors. This section will show you how to do that.

Examples in this section use the sample source code in the DLLDEMO.C file. DLLDEMO.C contains a HelloWorld function, which plays a beep on the speaker.

The examples also refer to a small Windows Help file called DLLDEMO.HLP, along with the DLLDEMO.RTF source file and DLLDEMO.HPJ Help project File used to build it. HelloWorld is registered as a Help macro in DLLDEMO.HPJ, and the following hot spot in the DLLDEMO.HLP Help file executes HelloWorld to play a beep.

```
Call HelloWorld.!HelloWorld(hwndapp,qchPath)
```

Note: To debug your DLL or observe any of the operations described in this section, you can compile DLLDEMO for the CodeView for Windows symbolic debugger and load the DLL into CodeView while you view the sample title. See the DLLDEMO.C file for specific instructions.

Registering DLL Functions as Help Macros

When registering a DLL function, you provide Help the following information:

- DLL filename
- Function name
- Data type returned by the function
- Number and type of function parameters

To register the DLL function, you enter a RegisterRoutine macro in the [CONFIG] section of the Help project file. (If you dont register the DLL function in the [CONFIG] section, you must register it another way before using the function.) The RegisterRoutine macros are executed when the Help file is opened, so the registered functions are available during the entire Help session. You must register a DLL routine before using it, or the Help compiler will report an error when it encounters the unregistered macro in the RTF source files, and the macro will not work when executed in the built Help file.

The RegisterRoutine macro has the following syntax:

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

Parameter	Description
DLL-name	String specifying the name of the DLL in which the function resides. The filename must be enclosed in quotation marks. You can omit the .DLL filename extension.Specify the directory only if necessary. Generally, DLLs are installed in the directory where Windows Help resides. See the next section, How Help Locates .DLLs, for more information.
function-name	String specifying the name of the function to use as a Help macro. The function name must be enclosed in quotation marks.
parameter-spec	String specifying the formats of parameters passed to the function. Characters in the string represent C parameter types. Valid parameter types include the following:

Character	Data Type	Equivalent Windows Data Type
u	Unsigned short integer	UINT, WORD, WPARAM
U	Unsigned long integer	DWORD, RGBQUAD
i	Signed short integer	BOOL (also C int or short)
I	Signed long integer	LONG, LPARAM, LRESULT
S	Far pointer to a null-terminated text string	LPSTR, LPCSTR
v	Void (means no type; used only with return values)	None. Equivalent to C void data type.

The parameter-spec must be enclosed in quotation marks. Windows Help checks the format string to ensure that it matches the function prototype defined in the DLL. (For more information about the RegisterRoutine macro, see Chapter 15, Help Macro Reference.)

To determine the data type of the functions parameters, consult the application programming interface (API) documenation for the DLL, or ask the person who developed the DLL. For information on parameter types of Windows functions, see the Windows version 3.1 SDK.

The HelloWorld function is defined as follows in the DLLDEMO.C file:

```
PUBLIC BOOL PASCAL EXPORT HelloWorld( LONG hwndContext, LPSTR lszHlpFile)
{
   MessageBeep(0);
}
```

HelloWorld takes, as parameters, two internal Windows Help variables: a handle to the Windows Help context window and a pointer to the name of the .HLP file that Windows Help has opened. Although

HelloWorld doesnt do anything with these parameters, they illustrate the types of internal Windows Help variables you might pass to a DLL. For a complete list of these variables, see the Windows Help Internal Variables section, later in this chapter.

HelloWorld is registered in the DLLDEMO.HPJ Help project file as follows:

```
[CONFIG]
RegisterRoutine("dlldemo", "HelloWorld", "US")
```

As described above, the first parameter to RegisterRoutine is the DLL name (DLLDEMO); the second is the name of the function being registered; and the third is a format string representing the types of parameters in the function. Windows Help compares the parameter-spec with the functions parameter types and issues an error if they dont match.

The format string for the parameters to HelloWorld are U (unsigned long) for the LONG parameter and S for the LPSTR parameter. All internal variables that are handles should be declared in RegisterRoutine with a U specifier, even though they are normally WORDs, for upward compatibility with future versions of the Windows graphical environment. All internal string variables should be declared with an S specifier.

How Help Locates DLLs

When executing custom DLLs using the RegisterRoutine macro, Windows Help loads the DLL only when it is needed by the Help file. To load a DLL , Help must be able to find it on the users system. When preparing to use a .DLL, Help looks in the following locations, in the following order:

1. Helps current directory
2. The MS-DOS current directory
3. The users Windows directory
4. The Windows SYSTEM directory
5. The directory containing WINHELP.EXE
6. The directories listed in the users PATH environment variable
7. The directories specified in WINHELP.INI

If Help cannot find the DLL after searching in all these locations, it displays an error message.

To increase the likelihood that Help will locate the DLL quickly, you should also observe the following guidelines:

- Use unique names for all DLLs accessed by the Help file.
- When installing your application on a users hard disk drive, your setup program should copy all custom DLLs to the directory where Help is located.
- If your product is distributed on CD-ROM, copy WINHELP.EXE and any custom DLLs to the users hard disk drive.
- Define a WINHELP.INI entry for each custom DLL that your Help file is using so that Help knows where to locate them.

For an explanation of the WINHELP.INI file, see the Creating Links Between Help Files section in Chapter 8, Creating Links and Hot Spots.

Windows Help Internal Variables

In the hot-spot example earlier in this section, the HelloWorld function used the internal Windows Help variables hwndContext (a handle to the Windows Help context window) and qchPath (a pointer to the Help filename). In general, after you register a DLL function as a Help macro, you can specify Windows Help internal variables as parameters to that function when the function appears in hot spots or macro footnotes.

In this section, youll learn about other Windows Help internal variables that you can pass to DLL functions registered as Windows Help macros. This section also examines in detail one variable that controls how the DLL handles Windows Help errors.

List of Variables

You can use any of the following Windows Help internal variables in DLL functions.

Variable	Format Spec	Description
hwndApp	U	32-bit handle to the main Help window. This variable is guaranteed to be valid only while the function is executing.
hwndContext	U	Handle to the current active window (either the main Help window or a secondary window).
qchPath	S	Fully qualified path of the currently open Help (.HLP) file.
qError	S	Long pointer to a structure containing information about the most recent Windows Help error (described later in this section).
lTopicNo	U	Topic number. This number is relative to the order of topics in the RTF files used to build the Help file.
hfs	U	Handle to the file system for the currently open Help (.HLP) file.
coForeground	U	Current foreground color.
coBackground	U	Current background color.

Error Handling

The qError internal variable points to a structure containing information about the most recent Windows Help error. The error structure is defined as follows:

```
struct
 { WORD  fwFlags;
     WORD  wError;
     char  rgchError[wMACRO_ERROR];
 } QME;
```

fwFlags

The fwFlags field contains flags indicating how Windows Help responds to errors. The following are possible error flags and their values.

Flag	Value	Description
fwMERR_ABORT	0x0001	Allows the Abort option. This flag is set by default.
fwMERR_CONTINUE	0x0002	Allows the Continue option.
fwMERR_RETRY	0x0004	Allows the Retry option.

wError

The wError field is a number indicating the type of error that occurred. The following are possible error numbers and their values.

Error	Number	Description
wMERR_NONE	0	No error (initial value)
wMERR_MEMORY	1	Out of memory (local)
wMERR_PARAM	2	Invalid parameter passed
wMERR_FILE	3	Invalid file parameter
wMERR_ERROR	4	General Help macro error
wMERR_MESSAGE	5	Help macro error with message

rgchError

If the wError field is wMERR_MESSAGE, the rgchError field contains the error message that Windows Help displays.

Notifying DLLs of Windows Help Events

You might want your DLL to receive notification of Windows Help events (for example, when the user selects a jump or changes input focus to an application other than Windows Help). To do this, you add a LDLLHandler function to the DLL. The LDLLHandler function has the job of processing messages sent from Windows Help to the DLL.

When the user clicks the Call HelloWorld hot spot in DLLDEMO.HLP, Windows Help loads DLLDEMO.DLL and looks for a function named LDLLHandler. If Windows Help finds LDLLHandler, it calls LDLLHandler. A sample LDLLHandler function is defined in the DLLDEMO.C file.

> Note: To debug your DLL or observe any of the operations described in this section, you can compile DLLDEMO for the CodeView for Windows symbolic debugger and open the DLL in CodeView while you view the sample title. See the DLLDEMO.C file for specific instructions.

The LDLLHandler function is defined in DLLDEMO.C as illustrated in the following example:

```
PUBLIC   LONG PASCAL EXPORT LDLLHandler(
   WORD  wMsg,
   LONG  lParam1,
   LONG  lParam2)
   {
         switch(wMsg) {
               case DW_WHATMSG:
                     return DC_INITTERM | DC_JUMP;
               case DW_INIT:
                     return TRUE;
               case DW_TERM:
                     return TRUE;
               case DW_ACTIVATE:
                     return TRUE;
               case DW_CHGFILE:
                     return TRUE;
   }
   return FALSE;
}
```

In this definition, wMsg identifies the message Windows Help has sent to the DLL. The two LONG parameters are passed with the message and depend on the message type.

When the user first clicks the Call HelloWorld hot spot, Windows Help calls LDLLHandler with the wMsg parameter set to DW_WHATMSG. This message asks the DLL what types of Windows Help messages it wants to receive. LDLLHandler should return one or more of the flags in the following table with this information. Windows Help uses the flags returned by LDLLHandler to determine which messages the DLL wants to receive in the LDLLHandler function.

LDLLHandler returns	Windows Help sends	Description
DC_MINMAX	DW_MINMAX, DW_SIZE	Minimize, maximize, or resize Windows Help
DC_INITTERM	DW_INIT, DW_TERM	Initialize or terminate DLL
DC_JUMP	DW_STARTJUMP,	Jump or change .HLP file

	DW_ENDJUMP, DW_CHGFILE	
DC_ACTIVATE	DW_ACTIVATE	Give Windows Help or another application input focus
DC_CALLBACKS	DW_CALLBACKS	Give DLL access to Windows Help entry points (see Calling Windows Help Internal Functions, later in this chapter, for more information)

You combine these flags using the standard C bitwise-OR (|) operator. For example, many DLLs request notification when Windows Help is about to initialize or terminate them. The sample LDLLHandler function requests this notification and asks to be notified when the user has moved the input focus to or from Windows Help. It processes the DW_WHATMSG message as follows:

```
case DW_WHATMSG:
   return DC_INITTERM | DC_ACTIVATE;
```

The remainder of this section looks at the different Windows Help messages that LDLLHandler processes.

DC_INITTERM Flag

The DC_INITTERM flag tells Windows Help that the DLL should receive initialization and termination messages.

DW_INIT

After LDLLHandler returns a DC_INITTERM flag, Windows Help sends a DW_INIT message. This message tells LDLLHandler to perform any required initialization operations (such as initializing variables or loading strings). The lParam1 and lParam2 parameters are not used.

If LDLLHandler returns FALSE, Windows Help unloads the DLL and stops sending messages. If LDLLHandler returns TRUE, the DLL remains loaded.

In the DLLDEMO sample, LDLLHandler returns TRUE for this message.

DW_TERM

When the user ends a Windows Help session, Windows Help sends a DW_TERM message. This message tells LDLLHandler to perform any required cleanup operations before the DLL is unloaded from memory. The lParam1 and lParam2 parameters are not used, and the return value has no meaning.

DC_MINMAX Flag

The DC_MINMAX flag tells Windows Help that the DLL should receive messages when the user minimizes, maximizes, or resizes the Windows Help window.

DW_MINMAX

Windows Help sends the DLL a DW_MINMAX message if the user minimizes or maximizes the context window.

The lParam1 parameter to DW_MINMAX indicates which operation was performed: 1L for minimized or 2L for maximized.

The lParam2 parameter is not used.

If LDLLHandler returns TRUE for this message, Windows Help continues sending this message throughout the remainder of the session; if LDLLHandler returns FALSE, Windows Help stops sending it.

DW_SIZE

Windows Help sends the DLL a DW_SIZE message if the user resizes the context window.

The lParam1 message specifies the horizontal size of the window in the low-order word and the vertical size in the high-order word.

The lParam2 parameter is not used.

If LDLLHandler returns TRUE for this message, Windows Help continues sending this message throughout the remainder of the session; if LDLLHandler returns FALSE, Windows Help stops sending it.

DC_JUMP Flag

The DC_JUMP flag tells Windows Help that the DLL should receive messages when the user executes a jump in the Help file.

DW_STARTJUMP

Windows Help sends the DLL a DW_STARTJUMP message after the user executes a jump. The parameters to DW_STARTJUMP are not used.

If LDLLHandler returns TRUE for this message, Windows Help continues sending this message throughout the remainder of the session; if LDLLHandler returns FALSE, Windows Help stops sending it.

DW_ENDJUMP

Windows Help sends the DLL a DW_ENDJUMP message after it displays the jump-destination topic.

The lParam1 parameter specifies the byte offset of that topic within the Help files file system. This value can be passed to the LSeekHf function (defined in the DLL.H file) to perform a seek to that topic within the file.

The lParam2 parameter specifies the position of the scroll box in the topic. This value can be used in any of the standard Windows scrolling functions that accept scroll-position parameters.

If LDLLHandler returns TRUE for this message, Windows Help continues sending this message throughout the remainder of the session; if LDLLHandler returns FALSE, Windows Help stops sending it.

DW_CHGFILE

Windows Help sends the DLL a DW_CHGFILE message if the user selects Open from the File menu to change .HLP files or jumps to a topic in a different .HLP file. If the message is the result of a jump to a new file, Windows Help sends the DW_CHGFILE message between the DW_STARTJUMP and DW_ENDJUMP messages.

The lParam1 parameter specifies a long pointer to the .HLP filename. The lParam2 parameter is not used.

If LDLLHandler returns TRUE for this message, Windows Help continues sending this message throughout the remainder of the session; if LDLLHandler returns FALSE, Windows Help stops sending it.

DC_ACTIVATE Flag

The DC_ACTIVATE flag tells Windows Help that the DLL should receive a DW_ACTIVATE message when the user gives either Windows Help or another application the input focus.

DW_ACTIVATE

The lParam1 parameter to the DW_ACTIVATE message specifies whether Windows Help received (nonzero LONG value) or lost (0L) the input focus. The lParam2 parameter is not used.

If LDLLHandler returns TRUE for this message, Windows Help continues sending this message throughout the remainder of the session; if LDLLHandler returns FALSE, Windows Help stops sending it.

Calling Windows Help Internal Functions

Windows Help provides DLL authors with access to 16 of its internal functions. These functions perform operations in the internal .HLP file system, get global information about the currently open Help file, or display information in a standard Windows Help dialog box. They are documented in the DLL.H header file.

In this section, youll learn how to access six of these internal functions. An overview of the mechanism is as follows:

1. Windows Help sends a DW_WHATMSG message to the DLL.
2. The LDLLHandler function in the DLL processes the DW_WHATMSG message by returning a DC_CALLBACKS flag.
3. Windows Help sends a DW_CALLBACKS message specifying a pointer to an array of Windows Help internal functions.
4. The LDLLHandler function uses the pointer passed in the DW_CALLBACKS message to obtain pointers to the Windows Help functions it will use.

The ExportBag function in the DLLDEMO.C sample file is used for the examples in this section. ExportBag copies a file from the .HPJ internal file system to an MS-DOS file. (The file from the .HPJ file system is initially stored using an entry in the [BAGGAGE] section of the DLLDEMO.HPJ project file.) You can use the mechanism illustrated in this example to access files stored in any .HLP file system.

Sample Code

ExportBag is executed from a two topic entry macros in the sample DLLDEMO.HLP Help file. Like the HelloWorld function earlier in this chapter, ExportBag is registered as a Help macro in the DLLDEMO.HPJ Help project file as follows:

```
RegisterRoutine("dlldemo","ExportBag","SSSS")
```

The two entry macros in DLLDEMO.RTF that execute ExportBag are coded as follows:

```
! ExportBag(qchPath, "dlldemo.hpj", "decomp.hpj", qError )
! ExportBag(qchPath, "dlldemo.rtf", "decomp.rtf", qError )
```

Note: To debug your DLL or observe any of the operations described in this section, you can compile DLLDEMO for the CodeView for Windows symbolic debugger and open the DLL in CodeView while you view the sample title. See the DLLDEMO.C file for specific instructions.

Accessing Windows Help Functions (DW_CALLBACKS)

If a DLL wants access to Windows Help internal functions, its LDLLHandler function handles the DW_WHATMSG message by returning a DC_CALLBACKS flag. In DLLDEMO.C, the LDLLHandler function requests access as follows:

```
case DW_WHATMSG:
         return DC_INITTERM | DC_CALLBACKS;
```

When it gets this flag, Windows Help sends a DW_CALLBACKS message to the DLL. The lParam1 parameter is a long pointer to an array containing pointers to each of the 16 internal Windows Help functions. The DLL.H file defines symbolic names for indexing each function in the array.

In processing the DW_CALLBACKS message, the LDLLHandler function calls a function named GetCallBacks to specify which functions it wants to access. GetCallBacks is defined as follows in DLLDEMO:

```
PUBLIC   BOOL PASCAL EXPORT GetCallBacks(
VPTR     VPtr,
LONG     lVersion)
{
   // hfs level:
   lpfn_HfsOpenSz      = VPtr[HE_HfsOpenSz];
   lpfn_RcCloseHfs    = VPtr[HE_RcCloseHfs];
   lpfn_RcLLInfoFromHfs = VPtr[HE_RcLLInfoFromHfs];
   // bag level routines
   lpfn_FAccessHfs =  VPtr[HE_FAccessHfs];
   lpfn_HfOpenHfs  =  VPtr[HE_HfOpenHfs];
   lpfn_LcbReadHf  =  (LPFN_LCBREADHF) VPtr[HE_LcbReadHf];
   lpfn_RcCloseHf  =  VPtr[HE_RcCloseHf];
   return TRUE;
}
```

The VPtr parameter in GetCallBacks is the lParam1 pointer passed by the DW_CALLBACKS message from Windows Help. The GetCallBacks function gets pointers to the following Windows Help internal functions, which it uses later in the example.

Function	What it does
HfsOpenSz	Opens the .HLP file system (the compound file containing the files internal to the Help file)
RcCloseHfs	Closes the open .HLP file system
RcLLInfoFromHfs	Maps a handle to the open .HLP file system (returned by HfsOpenSz) to low-level file information
FAccessHfs	Determines whether a file within the .HLP file system is accessible
HfOpenHfs	Opens a file within the .HLP file system
LcbReadHf	Reads bytes from a file within the .HLP file system

Note: This sample code does not take multiple instances of Windows Help into account. If more than one instance is started, the DLL receives different pointers to these callback functionsone pointer for each instance. You must keep track of which pointer is associated with each Windows Help instance. One way to do this is to create an array associating the task with the callback pointer.

Copying Files from the .HLP File System

Now that DLLDEMO has access to the Windows Help functions it needs, it is ready to copy a file from the .HLP file system. The ExportBag function performs this operation. ExportBag uses the following variables:

```
PUBLIC               BOOL PASCAL
EXPORT ExportBag(LPSTR lszHLPname,
               LPSTRlszBagFName,
               LPSTRlszExportName,
               QME         qError)
{
   HANDLE      hfsHlp;           // handle to .HLP file
   HANDLE      hfBag;            // file handle to bag file
   HANDLE      hFile;            // handle to output file
   BOOL        fClean = TRUE;
                     //if set, Del file in Error state Machine.
   DWORD  dwBytesRead;           // input bytes read
   HANDLE             hMem;      // handle to copy buffer
   LPBYTE             lpMem;     // ptr to copy buffer
   OFSTRUCT ofs;
```

Getting a Handle to the .HLP File System

First, ExportBag uses the HfsOpenSz function to get a handle to the .HLP file system, as follows:

```
if ((hfsHlp = (*lpfn_HfsOpenSz)(lszHLPname,
   fFSOpenReadOnly)) == NULL)
   {
   qError->fwFlags = fwMERR_ABORT;
   qError->wError = wMERR_PARAM; goto ExitBag;
   }
```

The lszHLPname parameter identifies the .HLP file for which it wants the handle. This parameter takes its value from Windows Helps qchPath internal variable.

As youll recall, ExportBag is executed from two entry macros in the DLLDEMO.HLP Help file. The DLLDEMO.HPJ project file for that Help file includes a RegisterRoutine macro that registers ExportBag as a Help authoring macro. When ExportBag is run from the macros, qchPath is given as the first parameter to ExportBag.

Verifying the .HLP File

Next, ExportBag makes sure the baggage file within the .HPJ file system is valid, as follows:

```
if (((*lpfn_FAccessHfs)(hfsHlp, lszBagFName, NULL)) == FALSE)
   {
   qError->fwFlags = fwMERR_RETRY;
   qError->wError = wMERR_MESSAGE;
   lstrcpy(qError->rgchError, "Could not open baggage file `");
   lstrcat(qError->rgchError, lszBagFName);
   lstrcat(qError->rgchError, "'.");
   goto ExitBag;
   }
```

The lszBagFName parameter to ExportBag gives this filename. Like the lszHLPname parameter, the baggage filename in lszBagFName is passed from the entry macros encoded in the DLLDEMO.HLP file. However, the baggage filename is hard-coded in the macro rather than obtained from a Windows Help internal variable.

In this and other examples, the return code rc would normally be serviced in a common error routine at label UNDOstateEXIT. This error routine would inform the user of the problem. For simplicity, this error routine has been omitted from DLLDEMO.C.

Copying the Baggage File

ExportBag now opens the baggage file for reading, as folows:

```
if ((hfBag = (*lpfn_HfOpenHfs)(hfsHlp, lszBagFName, fFSOpenReadOnly))
   == NULL)
   {
   qError->fwFlags = fwMERR_ABORT;
   qError->wError = wMERR_ERROR; goto ExitBag;
   }
```

It creates a buffer to improve efficiency in the copy:

```
if ((hMem = GlobalAlloc(GMEM_MOVEABLE,
   (DWORD)wCOPY_SIZE)) == NULL)
   {
   qError->fwFlags = fwMERR_ABORT;
   qError->wError = wMERR_MEMORY;
   goto ExitBag;
   }
lpMem = GlobalLock(hMem);
```

To make sure the MS-DOS file its copying to is not already present, ExportBag calls the Windows API OpenFile. This function executes an MS-DOS function to purge the filename ExportBag is using:

```
OpenFile(lszExportName, &ofs, OF_DELETE);
```

ExportBag creates the MS-DOS file its copying to and obtains a handle to the new file, as follows:

```
if ((hFile = _lcreat(lszExportName,0)) == -1)
   {
   qError->fwFlags = fwMERR_ABORT;
   qError->wError = wMERR_ERROR; goto ExitBag;
   }
```

Now ExportBag starts the write loop that copies the baggage file to the new MS-DOS file, as follows:

```
do  {    if ((dwBytesRead =
         (*lpfn_LcbReadHf)(hfBag, lpMem, wCOPY_SIZE)) == -1L)
         {
         qError->fwFlags = fwMERR_ABORT;
         qError->wError = wMERR_ERROR;
         goto ExitBag;
         }
if (_lwrite(hFile, lpMem,(WORD) dwBytesRead) != (WORD) dwBytesRead)
         {
         qError->fwFlags = fwMERR_ABORT | fwMERR_RETRY;
         qError->wError = wMERR_MESSAGE;
         lstrcpy(qError->rgchError, "Out of disk space.");
         goto ExitBag;
         }
   } while (dwBytesRead == wCOPY_SIZE);
```

ExportBag reads the size of the buffer it created until it reaches the end of the baggage file. The end-of-file condition is indicated when the LcbReadhf function returns 1L. ExportBag writes to the new MS-DOS file using the standard Windows _lwrite function.

When ExportBag has finished writing the MS-DOS file, it closes the baggage and MS-DOS files and cleans up its memory, as follows:

```
ExitBag:
if (hfBag != NULL)    (*lpfn_RcCloseHf)(hfBag);
if (hfsHlp != NULL)   (*lpfn_RcCloseHfs)(hfsHlp);
if (hMem != NULL)
   {
   GlobalUnlock(hMem);
   GlobalFree(hMem);
   }
if (hFile != -1)
   _lclose(hFile);
if (qError->wError != wMERR_NONE && fClean)
   OpenFile(lszExportName, &ofs, OF_DELETE);
```

Writing DLLs for Embedded Windows

In Windows Help version 3.1 you can create embedded windows to extend the functionality of Windows Help by placing objects in a fixed-size window under the control of a DLL. For example, you can create a DLL to display animation sequences or to play audio segments in an embedded window. However, in version 3.1 you cannot create an embedded window that functions as a hot spot. This section explains how to write custom DLLs for Windows Help.

Creating Embedded Windows

To create an embedded window, authors insert an embedded window reference in the RTF source file. This reference has the following general form:

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

Parameter	Description
x	A character specifying the alignment of the window: l for a left-aligned window, r for a right-aligned window, or c for a character-aligned window.
DLL-name	The name of the DLL that controls the embedded window. The file name should not include an extension or be fully qualified, but it can include a relative path. Windows Help assumes .DLL or .EXE to be the default extension.
window-class	The name of the embedded window class as defined in the source code for the DLL.
author-data	An arbitrary string, which Windows Help passes to the embedded window when it creates the window. This string can be one or more substrings separated by any punctuation mark except a comma. The DLL is responsible for parsing this string. The string is terminated by the closing brace (})

The following example shows a valid embedded window reference:

How Embedded Windows Work in Help

Windows Help displays embedded windows within topic windows. To Windows Help, an embedded window is simply a child window of that topic window. Users cannot minimize, maximize, or resize an embedded window. Embedded windows cannot be used as hot spots. However, you can include hot spots in an embedded window if they are controlled by the DLL, but users will not be able to use the keyboard equivalents to access the hot spots. That is because embedded windows cannot receive the input focus, which means they cannot process keystrokes. So, you should not place anything in an embedded window that requires keyboard input from users.

Windows Help positions the embedded window using the justification character (left, right, or character) specified by the author in the embedded window reference. The embedded window DLL is expected to display the information in the window and resize the window appropriately. Help expects the window size to remain fixed as long as the topic is displayed. The window element and DLL determine the size and content of the embedded window, and Help arranges the other elements of the topic around the embedded window.

Windows Help displays embedded windows only when necessarywhile the topic containing the embedded window is being displayed. However, an embedded window may exist while it is not being displayed (if the user scrolls the topic past the embedded window, for example). Because it is part of a specific topic, an embedded window goes away when the user displays a different topic.

Initialization

Windows Help creates embedded windows with window style WS_CHILD and a default size. It initializes the DLL for an embedded window before it displays the window.

When Windows Help creates the embedded window, it passes two strings in the WM_CREATE message. The lparam parameter points to a CREATESTRUCT structure, and the lpCreateParams field of the CREATESTRUCT structure points to a second structure defined as follows in the DLL.H header file:

```
typedef struct tagCreateInfo
      {
      short       idMajVersion;
      short       idMinVersion;
      LPSTR       lpstrFileName;
      LPSTR       lpstrAuthorData;
      HANDLE         hfs;
      DWORD       coForeground;
      DWORD       coBackground;
      }
   CREATEINFO;
```

The fields in the structure are defined in the following table.

Field	Use
idMajVersion,idMinVersion	Identifies the version of Windows Help (and the version of the CREATESTRUCT structure used for embedded windows in that version of Windows Help). Currently, these fields are 0. If the idMinVersion field is a value other than 0, additional interfaces may be available to DLLs, but all the interfaces described in this chapter are still supported.The DLL should always verify that the value in the idMajVersion and idMinVersion fields of the CREATESTRUCT structure are 0. If they are not, the DLL might not work with that version of Windows Help.
lpstrFileName	Points to the fully qualified name of the .HLP file containing the embedded window. This field will be NULL if the data is not available. If the DLL uses the string contained in this

field, it should make a copy of the string.

lpstrAuthorData Points to the author-data parameter in the ewl, ewc, or ewr reference from the source RTF file. This field will be NULL if the data is not available. If the DLL uses the string contained in this field, it should make a copy of the string.

hfs Specifies a handle to the file system for the .HLP file.

CoForeground,CoBackground
Specify the foreground and background colors of the main Windows Help window. If the embedded window uses the same colors as the main Windows Help window, the DLL can use the color values in these fields to set those colors in the embedded window.

Embedded Window Behavior

While a topic is being displayed, Windows Help creates embedded windows when it needs to lay them out and destroys them when they are no longer needed. The window can be created and destroyed, then created again if necessary, while Windows Help tries to lay out the embedded window object. If you load information, you might want to do so during WM_SHOWWINDOW processing so that you dont have to load this information twice.

An embedded window may exist while it is not being displayed (for example, if the topic requires scrolling and the embedded window resides in the portion not currently displayed in the Help window). Do not set the window style to WS_VISIBLE or call the ShowWindow function during WM_CREATE processing. Windows Help displays the window when necessary.

As an embedded window DLL processes a WM_CREATE message, it is expected to set up any window styles the window uses and resize the window appropriately.

Windows Help positions the window using the justification character (left, right, or character) authored into the RTF file. Windows Help expects the window size to remain fixed during the windows lifetime.

Because it is a child window of the main topic window, an embedded window does not need to destroy itself; it will be destoyed when the parent window is destroyed.

Message Processing for Embedded Windows

Embedded windows receive most standard Windows messages, including mouse and WM_PAINT messages. Embedded windows should not take the input focus; therefore, they do not need to process keystrokes.

Embedded window DLLs must also process the following messages, which are specifically defined for use with embedded windows:

- EWM_RENDER (0x706A)
- EWM_QUERYSIZE (0x706B)
- EWM_ASKPALETTE (0x706C)

EWM_RENDER

Windows Help sends the EWM_RENDER message to an embedded window to get information about the image in the window. It uses this information when printing the image or placing it in the Clipboard. The wParam parameter to EWM_RENDER indicates the type of information returned by the message, and the LOWORD of the return value contains a handle to this information. Possible wParam values and return values are shown in the following table.

wParam	Return Value
CF_TEXT	Sharable global handle to a null-terminated ASCII string. Windows Help frees this handle.
CF_BITMAP	Handle to a bitmap. Windows Help removes this handle.

EWM_QUERYSIZE

Windows Help sends the EWM_QUERYSIZE message to an embedded window to obtain the size of the window. The wParam parameter is a handle to the device context for the window. The lParam parameter points to a POINT structure. The embedded window DLL should fill in the x field of the structure with the windows height and the y field of the structure with the windows width. Windows Help uses this information when laying out the topic in the topic window, when printing the topic, or when placing it in the Clipboard.

EWM_ASKPALETTE

Windows Help sends the EWM_ASKPALETTE message to all embedded windows displayed within a topic to get information about the palettes used in each window. The wParam and lParam parameters are not used.

When an embedded window receives an EWM_ASKPALETTE message, it should return a handle to the palette that it uses. Windows Help then selects the most appropriate palette for use in that topic. Usually, this palette is the one used for the first embedded window in the currently displayed part of the topic.

Before the DLL repaints an embedded window, it should send an EWM_ASKPALETTE message to the parent window to determine which palette to use. The following code fragment illustrates how to do this:

```
hpal = (HPALETTE)SendMessage(GetParent(hwnd), EWM_ASKPALETTE, 0, 0L)
```

Sample Embedded Window DLL

In this section, well analyze more source code in the sample DLL, and point out the functions and definitions required for embedded windows. The EWDEMO.C file contains the source code for a sample embedded window. The sample DLL displays a dynamic list of network printers in an embedded window. You can use this sample code as a template for creating your own embedded window DLLs.

Functions

The DLL consists of four functions: LibMain, WEP, PrinterListProc, and InitPrinterList. LibMain is the standard DLL function that registers the window class, and WEP is the standard function that terminates the DLL. PrinterListProc performs the message processing, including processing for the embedded window messages EWM_RENDER and EWM_QUERYSIZE. It also calls InitPrinterList to initialize the printer list after it receives a WM_CREATE message from Windows Help.

Initialization (WM_CREATE)

The PrinterListProc function is responsible for creating an embedded window when the window first receives a WM_CREATE message. The type for the structure passed in the WM_CREATE message is defined in DLL.H, as shown below:

```
typedef struct
   {
    short      idMajVersion;
    short      idMinVersion;
    LPSTR      szFileName;
    LPSTR      szAuthorData;
    HANDLE     hfs;
    DWORD      coFore;
    DWORD      coBack;  }
 EWDATA, FAR * QEWDATA;
```

As noted earlier, the szFileName field points to the relative path of the .HLP file, and the szAuthorData field points to the author-data parameter in the ewl, ewc, or ewr command from the source RTF file. The local variable qci is defined using this type.

PrinterListProc first takes the information passed in the lParam parameter to WM_CREATE and copies it into the fields of the qci structure, as follows:

```
case WM_CREATE:
   qci = (QCI)((CREATESTRUCT FAR *)lParam)->lpCreateParams;
   /*-------------------------------------------------------*\
   | Save the WM_CREATE information.
   \*-------------------------------------------------------*/
   lstrcpy( rgchFileName, qci->szFileName );
   lstrcpy( rgchAuthorText, qci->szAuthorData );
```

Next, PrinterListProc adds a border to the embedded window and initializes the printer list, as follows:

```
/*-------------------------------------------------------*\
| Add a border to this window.
\*-------------------------------------------------------*/
   SetWindowLong( hwnd, GWL_STYLE,
         GetWindowLong( hwnd, GWL_STYLE ) | WS_BORDER );
```

```
/*--------------------------------------------------------*\
| Initialize the printer list.  This could be updated more
| dynamically, say on each WM_PAINT message.
\*--------------------------------------------------------*/
   InitPrinterList();
   return 0L;
```

Painting (WM_PAINT)

PrinterListProc is also responsible for painting the embedded window. When the window receives a WM_PAINT message, PrinterListProc lists the names of the printers in the list initialized by InitPrinterList, as follows:

```
case WM_PAINT:
   BeginPaint( hwnd, &ps );
   GetTextMetrics( ps.hdc, &tm );
   SetTextColor( ps.hdc, GetSysColor( COLOR_WINDOWTEXT ) );
   SetBkColor( ps.hdc, GetSysColor( COLOR_WINDOW ) );
   for (irgch = 0; irgch  MAX_PRINTERS && rgrgchPrinters[irgch][0];
         irgch++)
         TextOut( ps.hdc, tm.tmMaxCharWidth/2,
                 (tm.tmHeight + tm.tmExternalLeading)/2 +
                 irgch*(tm.tmHeight + tm.tmExternalLeading),
                 rgrgchPrinters[irgch], lstrlen(rgrgchPrinters[irgch]) );
   EndPaint( hwnd, &ps );
   return 0L;
```

Obtaining Rendering Information (EWM_RENDER)

PrinterListProc also processes the EWM_RENDER message sent by Windows Help. The wParam parameter determines the type of rendering information that PrinterListProc returns. This parameter has the value CF_BITMAP or CF_TEXT.

CF_BITMAP

If wParam is CF_BITMAP, PrinterListProc creates a bitmap listing the system printers and returns a handle to this bitmap (hbm). PrinterListProc gets the information it needs to create the bitmap in two ways:

- Copying the values passed in the lParam parameter to the EWM_RENDER message.
- Sending an EWM_QUERYSIZE message to the embedded window and obtaining its size.

The lParam parameter points to a RENDERINFO structure. Type RENDERINFO is defined (in DLL.H), as follows:

```
typedef struct
  {
  RECT  rc;
  HDC   hdc;
  } RENDERINFO, FAR *QRI;
```

To create the bitmap, PrinterListProc copies the device context from lParam. It then obtains the size required for the bitmap by sending an EWM_QUERYSIZE message, as follows:

```
switch( wParam )
   {
   case CF_BITMAP:
   /*-------------------------------------------------------*\
   | Prepare a bitmap image of the printer list.  This will
   | appear in a similar manner as the layout, but we will need
   | to draw the border explicitly here, as well as making sure
   | the background is filled in correctly.  We must use the
   | default colors for this hdc.
   \*-------------------------------------------------------*/
qri = (QRI)lParam;
   hdc = CreateCompatibleDC( NULL );
   if (hdc)
   {
      hfont = 0;
      /*-------------------------------------------------------*\
      | Create a monochrome bitmap for the DC, sized for the screen.
      \*-------------------------------------------------------*/
         SendMessage( hwnd, EWM_QUERYSIZE, hdc, (long)(LPPOINT)&pt );
         rc.left = 0;
         rc.top = 0;
         rc.right = pt.x;
         rc.bottom = pt.y;
```

PrinterListProc creates the bitmap using the default foreground and background colors, but it draws the window border explicitly, as follows:

```
hbm = CreateCompatibleBitmap( qri->hdc, pt.x, pt.y );
      if (hbm)
      {
         hbmDefault = SelectObject( hdc, hbm );
         /*-------------------------------------------------------*\
         | Clear out the bitmap.
         \*-------------------------------------------------------*/
         hbrush = CreateSolidBrush( GetBkColor( hdc ) );
         if (hbrush)
         {
               FillRect( hdc, &rc, hbrush );
               DeleteObject( hbrush );
         }
         Rectangle( hdc, rc.left, rc.top, rc.right, rc.bottom );
         GetTextMetrics( hdc, &tm );
         for (irgch = 0; irgch  MAX_PRINTERS &&
            rgrgchPrinters[irgch][0]; irgch++)
         TextOut( hdc, tm.tmMaxCharWidth/2,
```

```
                (tm.tmHeight + tm.tmExternalLeading)/2 +
                irgch*(tm.tmHeight + tm.tmExternalLeading),
                rgrgchPrinters[irgch],
                lstrlen( rgrgchPrinters[irgch] ) );
        /*----------------------------------------------------------*\
        | At this point, hbm is the desired bitmap, sized for a
        | screen display.  This will be stretched as needed for
        | the target display.
        \*----------------------------------------------------------*/
        hbm = SelectObject( hdc, hbmDefault );
    }
    else
    {
        /*----------------------------------------------------------*\
        | Not enough memory.  Clean up and return NULL.
        \*----------------------------------------------------------*/
        hbm = NULL;
    }
    DeleteDC( hdc );
    }
    lReturn = (long)hbm;
    break;
```

PrinterListProc creates the bitmap using the default foreground and background colors for the device context, but it draws the window border explicitly, as follows:

```
  hdc = CreateCompatibleDC( qri->hdc );
        /*----------------------------------------------------------*\
        | Make hdc really compatible with the source hdc.
        \*----------------------------------------------------------*/
        if (hdc)
        {
            SetTextColor( hdc, GetTextColor( qri->hdc ) );
            SetBkColor( hdc, GetBkColor( qri->hdc ) );
            hbrushSource = SelectObject( qri->hdc,

  GetStockObject( NULL_BRUSH ) );
            if (hbrushSource)
                SelectObject( hdc, hbrushSource );
            hpenSource = SelectObject(qri->hdc, GetStockObject(NULL_PEN) );
            if (hpenSource)
                SelectObject( hdc, hpenSource );
        }
        hbm = CreateCompatibleBitmap( qri->hdc, rc.right, rc.bottom );
        if (hdc && hbm && (hbmDefault = SelectObject( hdc, hbm )))
```

```
        {
            GetTextMetrics( hdc, &tm );
            for (irgch = 0; irgch  MAX_PRINTERS && rgrgchPrinters[irgch][0];
                irgch++)
                TextOut( hdc, tm.tmMaxCharWidth/2,
                       (tm.tmHeight + tm.tmExternalLeading)/2 +
                       irgch*(tm.tmHeight + tm.tmExternalLeading),
                       rgrgchPrinters[irgch],
                       lstrlen( rgrgchPrinters[irgch] ) );
            hbm = SelectObject( hdc, hbmDefault );
        }
        .
        .
        .
        lReturn = (long)hbm;
        break;
```

CF_TEXT

If wParam is CF_TEXT, PrinterListProc simply creates an ASCII list of the system printers in a Clipboard compatible format, as follows:

```
   case CF_TEXT:
      /*-------------------------------------------------------*\
      | List out the printers in a format suitable for the Clipboard.
      | Since this list will be embedded in the text of the topic,
      | use blank lines for separators.
      \*-------------------------------------------------------*/
      gh = GlobalAlloc( GMEM_MOVEABLE | GMEM_NOT_BANKED,
                        sizeof(rgrgchPrinters) );
      lReturn = (long)gh;
      if (gh)
      {
            sz = GlobalLock( gh );
            lstrcpy( sz, "\r\n" );
            for (irgch = 0; irgch  MAX_PRINTERS && rgrgchPrinters[irgch][0];
                irgch++)
            {
                lstrcat( sz, rgrgchPrinters[irgch] );
                lstrcat( sz, "\r\n" );
            }
            GlobalUnlock( gh );
      }
      break;
```

Obtaining the Window Size (EWM_QUERYSIZE)

The PrinterListProc function is also responsible for processing EWM_QUERYSIZE messages. These messages can come from Windows Help, or they can be sent by the code in PrinterListProc that processes the EWM_RENDER message.

As described earlier, the EWM_QUERYSIZE message returns the dimensions of the embedded window. The wParam parameter to EWM_QUERYSIZE is the device context for the window display, and the lParam parameter points to a POINT structure that returns the length and height of the window display, as

shown below:

```
case EWM_QUERYSIZE:
   /*-------------------------------------------------------------*\
   *  Size query message from Windows Help
   *  wParam is the target hdc.
   *  wLong is the address of the point to return size in.
   *  Return non-zero to indicate that we did something.
   \*-------------------------------------------------------------*/
   hfont = SelectObject( (HDC)wParam, GetStockObject( SYSTEM_FONT ) );
   GetTextMetrics( (HDC)wParam, &tm );
   ((LPPOINT)lParam)->x = ((LPPOINT)lParam)->y = 0;
   for (irgch = 0; irgch < MAX_PRINTERS && rgrgchPrinters[irgch][0];
      irgch++)
      {
   DWORD dwExt = GetTextExtent( (HDC)wParam, rgrgchPrinters[irgch],
      lstrlen( rgrgchPrinters[irgch] ) );
   ((LPPOINT)lParam)->x = max( (WORD)((LPPOINT)lParam)->x,
      LOWORD(dwExt) );
   ((LPPOINT)lParam)->y += HIWORD(dwExt);
   }
   ((LPPOINT)lParam)->x += tm.tmMaxCharWidth;
   ((LPPOINT)lParam)->y += tm.tmHeight + tm.tmExternalLeading;
   if (hfont)
   SelectObject( (HDC)wParam, hfont );
   return 1;
   }
```

Appendix A Windows Virtual-Key Codes

The following table shows the symbolic constant names, hexadecimal values, and keyboard (or mouse) equivalents for the virtual-key codes used by the Microsoft Windows operating system version 3.1. The codes are listed in numeric order. Use these codes with the AddAccelerator and RemoveAccelerator macros.

Symbolic constant name	Hexadecimal value	Keyboard (or mouse) equivalent
VK_LBUTTON	01	Left mouse button
VK_RBUTTON	02	Right mouse button
VK_CANCEL	03	Used for control-break processing
VK_MBUTTON	04	Middle mouse button (three-button mouse)
--	05 07	Undefined
VK_BACK	08	BACKSPACE key
VK_TAB	09	TAB key
--	0A 0B	Undefined
VK_CLEAR	0C	CLEAR key
VK_RETURN	0D	RETURN key
--	0E 0F	Undefined
VK_SHIFT	10	SHIFT key
VK_CONTROL	11	CTRL key
VK_MENU	12	ALT key
VK_PAUSE	13	PAUSE key
VK_CAPITAL	14	CAPITAL key
--	15 19	Reserved for Kanji systems
--	1A	Undefined
VK_ESCAPE	1B	ESC key
--	1C 1F	Reserved for Kanji systems
VK_SPACE	20	SPACEBAR
VK_PRIOR	21	PAGE UP key
VK_NEXT	22	PAGE DOWN key
VK_END	23	END key
VK_HOME	24	HOME key
VK_LEFT	25	LEFT ARROW key
VK_UP	26	UP ARROW key
VK_RIGHT	27	RIGHT ARROW key
VK_DOWN	28	DOWN ARROW key
VK_SELECT	29	SELECT key
--	2A	OEM specific
VK_EXECUTE	2B	EXECUTE key
VK_SNAPSHOT	2C	PRINT SCREEN key for Windows version 3.0 or later
VK_INSERT	2D	INS key
VK_DELETE	2E	DEL key
VK_HELP	2F	HELP key
VK_0	30	0 key
VK_1	31	1 key
VK_2	32	2 key
VK_3	33	3 key
VK_4	34	4 key

VK_5	35	5 key
VK_6	36	6 key
VK_7	37	7 key
VK_8	38	8 key
VK_9	39	9 key
--	3A 40	Undefined
VK_A	41	A key
VK_B	42	B key
VK_C	43	C key
VK_D	44	D key
VK_E	45	E key
VK_F	46	F key
VK_G	47	G key
VK_H	48	H key
VK_I	49	I key
VK_J	4A	J key
VK_K	4B	K key
VK_L	4C	L key
VK_M	4D	M key
VK_N	4E	N key
VK_O	4F	O key
VK_P	50	P key
VK_Q	51	Q key
VK_R	52	R key
VK_S	53	S key
VK_T	54	T key
VK_U	55	U key
VK_V	56	V key
VK_W	57	W key
VK_X	58	X key
VK_Y	59	Y key
VK_Z	5A	Z key
--	5B 5F	Undefined
VK_NUMPAD0	60	Numeric keypad 0 key
VK_NUMPAD1	61	Numeric keypad 1 key
VK_NUMPAD2	62	Numeric keypad 2 key
VK_NUMPAD3	63	Numeric keypad 3 key
VK_NUMPAD4	64	Numeric keypad 4 key
VK_NUMPAD5	65	Numeric keypad 5 key
VK_NUMPAD6	66	Numeric keypad 6 key
VK_NUMPAD7	67	Numeric keypad 7 key
VK_NUMPAD8	68	Numeric keypad 8 key
VK_NUMPAD9	69	Numeric keypad 9 key
VK_MULTIPLY	6A	Multiply key
VK_ADD	6B	Add key
VK_SEPARATOR	6C	Separator key
VK_SUBTRACT	6D	Subtract key
VK_DECIMAL	6E	Decimal key
VK_DIVIDE	6F	Divide key

VK_F1	70	F1 key
VK_F2	71	F2 key
VK_F3	72	F3 key
VK_F4	73	F4 key
VK_F5	74	F5 key
VK_F6	75	F6 key
VK_F7	76	F7 key
VK_F8	77	F8 key
VK_F9	78	F9 key
VK_F10	79	F10 key
VK_F11	7A	F11 key
VK_F12	7B	F12 key
VK_F13	7C	F13 key
VK_F14	7D	F14 key
VK_F15	7E	F15 key
VK_F16	7F	F16 key
VK_F17	80	F17 key
VK_F18	81	F18 key
VK_F19	82	F19 key
VK_F20	83	F20 key
VK_F21	84	F21 key
VK_F22	85	F22 key
VK_F23	86	F23 key
VK_F24	87	F24 key
--	88 8F	Unassigned
VK_NUMLOCK	90	NUM LOCK key
VK_SCROLL	91	SCROLL LOCK key
--	92 B9	Unassigned
--	BA C0	OEM specific
--	C1 DA	Unassigned
--	DB E4	OEM specific
--	E5	Unassigned
--	E6	OEM specific
--	E7 E8	Unassigned
--	E9 F5	OEM specific
--	F6 FE	Unassigned

Appendix B Help RTF Statements

Windows Help RTF statements are an extended subset of tokens defined by the rich-text-format (RTF) standard. The RTF statements specify character and paragraph properties, such as font, color, spacing, and alignment, for text and graphics in the Help file.

This appendix describes the purpose and syntax of RTF statements used in topic files for the Microsoft Windows Help application version 3.1.

Rich-Text Format

The rich-text format (RTF) standard is a method of encoding formatted text and graphics for easy transfer between different applications and different operations. Generally, it is used by all Microsoft Word applicationsWord for Windows, Word for the Macintosh, and Word for MS-DOSin order to move word-processing documents between different platforms without having to rely on special translation software or conversion utilities. Because the RTF standard provides a format for text and graphics interchange that can be used with different output devices and operating systems, Windows Help also supports this standard. That means you can use any text editor that generates RTF output, including your own custom RTF editor, to create the source files that are built into Help files.

Software that takes a formatted file and turns it into an RTF file is referred to in this appendix as an RTF writer. Software that translates an RTF file into a formatted file is referred to as an RTF reader. An RTF writer separates the applications control information from the actual text and writes a new file containing the text and RTF groups associated with that text. An RTF reader does the converse of this operation.

RTF Syntax

Help RTF statements are presented to the Microsoft Help compiler in topic files, which are specified in the [FILES] section of a Help project file. To the Help compiler a topic file consists of RTF statements, control symbols, groups, and unformatted text. Each RTF statement consists of a backslash (\) followed by an RTF statement name and delimiter:

\statement-name<delimiter>

RTF statements must be separated from subsequent text or statement parameters by a delimiter. A delimiter can be one of the following:

- A space.
  In this case, the space is considered part of the statement.
- A digit or minus sign (-), which indicates that a numeric parameter follows.
  The subsequent digit sequence is then delimited by a space or any character other than a letter or digit. In other words, the parameter can be a positive or negative number. If a numeric parameter immediately follows the statement name, this parameter becomes part of the statement. The statement is then delimited by a space or a non-alphabetic or non-numeric character in the standard manner.
- Any character other than a letter or digit.
  In this case, the delimiting character terminates the statement but is not considered part of the statement. A letter is an upper- or lowercase ASCII letter.

For example, the following line demonstrates usage of the \tab statement and a space delimiter:

left column\tab right column

When a space is used as a delimiter, the Help compiler discards it. If any other character is used, the compiler processes it as text or the start of another RTF statement. For example, if a backslash is used as a delimiter, the compiler interprets it as the beginning of the next RTF statement.

Certain statements control properties (such as bold and italic) that have only two states. When the RTF statement has no parameter or has a non-zero parameter, the statement is used to turn the property on. When the RTF statement has a 0 (zero) parameter, the statement is used to turn the property off. For example, \b turns on bold, whereas \b0 turns off bold.

Some statements, referred to as destinations, mark the beginning of a collection of related text. An example of a desination is the \footnote group, where the Help-specific text follows the statement. Destination statements and their following text must be enclosed in braces, as in this example:

```
{\footnote main_contents}
```

A control symbol consists of a backslash (\) followed by a single non-letter. They require no further delimiting.

> Note: Although control symbols are compact, there arent too many of them. But the number of possible statements is not limited. The parameter is partially incorporated in control symbols, so that if an application does not understand a control symbol, it can ignore the corresponding parameter as well.

A group consists of Help RTF statements and text enclosed in braces ({ }). The opening brace { indicates the start of the group, and the closing brace } indicates the end of the group, as shown in this example:

```
{  group start, and
}  group end.
```

Formatting specified within a group affects only the text within that group. Text within a group inherits any formatting of the text preceding the group.

Unformatted text consists of any combination of 7-bit ASCII characters. Although characters whose values are greater than 127 are not permitted in topic files, the \ statement can be used to insert them in the final Help file. The Help compiler treats spaces as part of the text, but it discards carriage return and linefeed characters.

RTF statements, control symbols, and braces constitute control information. Text grouping is used to define the format and placement of text and graphics in the Help file. All other characters in RTF text constitute plain text.

Because the backslash character (\) and braces ({ }) have specific meanings in RTF, they must be preceded with a backslash if you want to use them as plain text, as shown by the control symbols \\, \{, and \}.

RTF Semantics

When reading a stream of RTF text, an RTF reader must accomplish the following tasks:

1. Separate RTF control information from plain text.
2. Act on control information.

   This is designed to be a relatively simple process, as described in the next section.

   Some control information just contributes special characters to the plain text stream. Other information changes the program state, which includes document properties as a whole and a stack of group states that apply to parts of the document. The group state is saved by the opening { brace and is restored by the closing } brace.

   The current group state specifies:

   - The destination or part of the document that the plain text is building up.
   - The section formatting properties.
   - The paragraph formatting properties.
   - The character formatting properties.
3. Collect and properly dispose of any remaining plain text as directed by the current group state.

Acting on Control Information

When parsing the RTF stream, the RTF reader should follow this process:

1. Read the next character.
2. If the character is an opening brace (char == '{'), store the current state on the stack.
   Or if the current state does not change, then continue.
3. If the character is a closing brace (char == '}'), retrieve the current state from the stack.
   Generally, this will change the state.
4. If the character is a backslash (char == '\'), collect the RTF statement or control symbol and its parameter.
5. If there is one, look up the statement or symbol in the symbol table (a constant table) and act according to the description found there.\sgmlli2p The parameter is left available for use by the action.
6. Leave a read pointer before or after the delimiter, as appropriate.
7. After completing the action, continue.\sgmlli2p The different actions are listed in the next section.
8. Otherwise, if the character is anything other than {, }, or \, write the plain text character to the current destination using the current formatting properties.
9. Read the next character.

Acting on Symbol Table Entries

When looking up RTF statements and symbols in the symbol table, the RTF reader can take any of the following actions:

- Change the destination to the destination described in the table entry.
  Most destination changes are legal only immediately after an opening brace. Other restrictions may also apply (for example, footnotes may not be nested).
- Change formatting property.
  The symbol table entry will describe the property and whether the parameter is required.
- Insert a special character.
  The symbol table entry will describe the character code.
- End of paragraph.
  This can also be viewed as just a special character.
- End of section.
  This can also be viewed as just a special character.
- Ignore the character.

The RTF File

An RTF file consists of unformatted text, control words (RTF statements), and control symbols. An entire RTF file is considered a group and must be enclosed in braces. The \rtfn statement must follow the first open brace in the Help file. The numeric parameter identifies the version of the RTF standard used. For Microsoft Help version 3.1, this parameter must be 1. The \rtf statement is followed by a character set statement. The RTF character set described in this appendix corresponds to the \ansi character set statement. System default values, such as font number (\deffn statement), follows the character set statement. The following example shows these opening statements in a typical RTF file:

```
{\rtf1\ansi\deff0
```

Font Table

The first group listed after the opening statements is the font table group. All of the fonts that the RTF writer will use must be declared in the font table, which begins with the \fonttbl statement. Each font in the table begins with a font number (used to reference the font within the file) and then lists the family name and the font name. Each font entry ends with a semicolon. For example, the following group defines nine fonts:

```
{\fonttbl
{\f0\froman Times New Roman;}
{\f1\froman Symbol;}
{\f2\fswiss Arial;}
{\f3\froman MS Serif;}
{\f4\fswiss MS Sans Serif;}
{\f5\fmodern Courier;}
{\f6\fdecor ZapfDingbats;}
{\f7\fswiss HelveticaCondensed;}
{\f8\fswiss Helvetica;}
}
```

Color Table

If the Help file uses color, the RTF file must also define a color table group to specify the individual colors. Unlike font entries in the font table, color entries in the color table do not specify a color number. Instead, the RTF reader assumes that the first color defined is color zero, the second color is color one, and so on. The following example defines the standard 16 colors in the Windows palette:

```
{\colortbl;
\red0\green0\blue0;
\red0\green0\blue255;
\red0\green255\blue255;
\red0\green255\blue0;
\red255\green0\blue255;
\red255\green0\blue0;
\red255\green255\blue0;
\red255\green255\blue255;
\red0\green0\blue127;
\red0\green127\blue127;
\red0\green127\blue0;
\red127\green0\blue127;
\red127\green0\blue0;
\red127\green127\blue0;
\red127\green127\blue127;
\red192\green192\blue192;
}
```

Topic Information

After setting up the RTF standards for the Help file, you define the topic information. Topics consist of Help-specific information, which is defined in topic footnotes, character and paragraph formatting information, plain text and graphics that the user sees, and a page break delimiter.

Context String

A topic is specified with the #{\footnote context string} group statement, which identifies each topic in a Help file. A context string is any string of up to 255 characters. Valid characters are the alphabetic characters AZ, the numeric characters 09, and the period (.) or underscore (_) character.

For example, this RTF entry defines the context string main_contents:

```
#{\footnote main_contents}
```

Topic Title

Typically, a topic will also contain a title specified with the ${\footnote title-text} group statement, which identifies the topic in Help dialog boxes. A title string is any string of up to 50 characters. Any printable ASCII character may appear in a topic title. However, ASCII characters such as braces ({ }), brackets ([ ]), and backslashes (\) that are used as special characters in RTF must be prefixed by backslashes.

This example defines the title Saving a Document:

```
${\footnote Saving a Document}
```

Font and Formatting Information

Before any text is placed in the RTF file, the font name and font size must be specified. The \fn statement specifes the font name (n matches the font number defined in the font table). The \fsn statement specifies the font size (n is given in half-points). For example, using the sample font table just described, the following example defines the text as 10-point MS Sans Serif:

```
\f4\fs20
```

If you want the text to have any special formatting characteristics, you must also define those before you write out the plain text. The Help compiler supports a number of character and paragraph formatting attributes that you can use to change the appearance and placement of text and graphics.

The following example defines the topic title text as 14-point MS Serif with 17 points of space before, 6 points space after, and 16 points of leading. The topic title paragraph is also indented 6 points from the left margin:

```
\pard\plain \li120\sb340\sa120\sl-320 \f3\fs28
```

Topic End

When there is more than one topic in an RTF file, each topic except the last one should endewith a \page statement:

```
\page
```

Sample RTF File

The following is an example of a complete (one topic) RTF file that can be compiled by the Help compiler:

```
{\rtf1\ansi \deff0\deflang1024
{\fonttbl
{\f0\froman Times New Roman;}
{\f1\froman Symbol;}
{\f2\fswiss Arial;}
{\f3\froman MS Serif;}
{\f4\fswiss MS Sans Serif;}
}
{\colortbl;
\red0\green0\blue0;
\red0\green0\blue255;
\red0\green255\blue0;
\red255\green0\blue0;
\red255\green255\blue255;
#{\footnote graphics_cont}
${\footnote Graphics}
\pard\plain \li120\sb340\sa120\sl-320 \f3\fs28 Graphics
\par \pard\plain \li120 \f4\fs20 Use graphic images to illustrate concepts
and present
information visually. Graphics include line art, icons, screen shots of the
interface,
and graphics with hot spots ("hypergraphics").
\par
}
```

Overview of Help RTF Statement Support

Although the Help compiler supports many RTF statements, it does not support them all. The following tables present the RTF statements by group, and briefly describe the statements that Help does and does not support in this release of Windows Help. The following tables are not intended to provide detailed information about the RTF statements. For complete information, see the Help RTF Statement Reference, later in this Appendix.

Statement descriptions follow these conventions.

Convention	**Meaning**
Bold Statement	Statement IS supported by the Help compiler.
Plain Statement	Statement is NOT supported by the Help compiler.
Parentheses	Indicates the default value for the statement.
OVERLOADED	Indicates that the RTF statement has a specific meaning in Help that is different from its print-based usage.

Overloaded Statements

The compiler interprets some RTF statements differently from their normal print-based usage. For example, standard RTF specifies that the \uldb statement indicates a double underline, but the Help compiler uses this statement to indicate a hot spot. The following table lists the RTF statements that are overloaded for the Help compiler.

Statement	Print-based usage	Help compiler usage
\footnote	Footnote	Special topic commands
\keep	Keeps paragraph intact	Makes text nonwrapping
\keepn	Keeps paragraph with next	Creates a nonscrolling region at the top of the topic
\page	Creates page break	Ends the current topic
\strike	Creates strikethrough	Indicates a hot spot
\trqc	Centers table row with respect to its containing column	Uses relative column widths
\ul	Creates continuous underline	Indicates a link to a pop-up topic
\uldb	Creates double underline	Indicates a hot spot
\v	Creates hidden text	Indicates the context string to jump to

Character Set

Microsoft Help supports all RTF character sets.

Statement	Character set
\ansi	ANSI character set
\windows	Windows (default)
\mac	Apple Macintosh
\pc	OEM code page 437
\pca	Internation English code page 850

Special Characters

Special characters are defined as they exist in Microsoft Word for the Macintosh. Other characters may be added for interchanging files with other applications. If the Help compiler does not recognize a character, it is ignored.

For simplicity, ASCII 9 is treated the same as \tab and ASCII 10 is treated the same as \par. ASCII 13 is ignored. The control code \<10> is also ignored, even though it may be used to indicate a soft carriage return.

Statement	Meaning
*	Custom RTF destination. Not supported.
\-	Optional hyphen. Not supported.
\:	Subentry in index entry. Not supported.
_	Nonbreaking hyphen. Not supported.
|	Formula character. Not supported.
\~	Nonbreaking space. Not supported.
\hh	Hexadecimal value of specified character.
\bullet	Bullet. Not supported.
\cell	End of table cell.
\chatn	Annotation reference. Not supported.
\chdate	Current date. Not supported.
\chftn	Auto-numbered footnote reference. Not supported.
\chftnsep	Anchoring character for footnote separator. Not supported.
\chftnsepc	Anchoring character for footnote continuation. Not supported.
\chpgn	Current page number. Not supported.
\chpict	Placeholder character for picture. Not supported.
\chtime	Current time. Not supported.
\column	Required column break. Not supported.
\emdash	Em-dash. Not supported.
\endash	En-dash. Not supported.
\ldbquote	Left double quotation mark. Not supported.
\line	Required line break; same as SHIFT+ENTER.
\lquote	Left single quotation mark. Not supported.
\page	End of current topic. OVERLOADED
\par	End of paragraph.
\rdbquote	Right double quotation mark. Not supported.
\row	End of table row.
\rquote	Right single quotation mark. Not supported.
\sect	End of section and paragraph.
\tab	Tab character.

Destinations

A destination change resets all properties to their default values. Changes are legal only at the beginning of a group (statement and text enclosed in braces).

Statement	Meaning
\colortbl	Color table. See RTF Reference for details.
\comment	Comment text. Not supported; ignored.
\fonttbl	Font table. See RTF Reference for details.
\footer	Footer for current section. Not supported.
\footnote	Topic information. OVERLOADED. See RTF Reference.
\header	Header for the current section. Not supported.
\info	Document information block. Not supported.
\nextfile	Next file to print or index. Not supported.
\pict	Picture.
\rtf	Version of RTF standard used.
\stylesheet	Document stylesheet. Not supported.
\template	Document template. Not supported.

Document Formatting

Windows Help version 3.1 does not support any document formatting statements.

Statement	Meaning
\defformat	Saves document in RTF format. Not supported.
\deftabn	Default tab width. Not supported.
\enddoc	Footnotes at end of document. Not supported.
\endnotes	Footnotes at end of section. Not supported.
\facingp	Facing pages. Not supported.
\fracwidth	Fractional character widths. Not supported.
\ftnbj	Footnotes at bottom of page. Not supported.
\ftncn	Footnote separator for continued footnote notice. Not supported.
\ftnrestart	Restart footnote numbers each page. Not supported.
\ftnsep	Footnote separator. Not supported.
\ftnsepc	Footnote separator for continued footnotes. Not supported.
\ftnstartn	Starting footnote number. Not supported.
\ftntj	Footnotes beneath text. Not supported.
\guttern	Gutter width. Not supported.
\hyphhotz	Hyphenation hot zone. Not supported.
\landscape	Print in landscape format. Not supported.
\linestartn	Starting line number. Not supported.
\makebackup	Backup document. Not supported.
\margbn	Bottom margin. Not supported.
\margln	Left margin. Not supported.
\margmirror	Switches margin definitions. Not supported.
\margrn	Right margin. Not supported.
\margtn	Top margin. Not supported.
\paperhn	Paper height. Not supported.
\paperwn	Paper width. Not supported.
\pgnstartn	Starting page number. Not supported.
\psover	Print PostScript over text. Not supported.
\revbarn	Vertical revision marks. Not supported.
\revisions	Turns on revision marking. Not supported.
\revpropn	Revision properties. Not supported.
\widowctrl	Enable widow control. Not supported.

Section Formatting

Windows Help version 3.1 does not support any section formatting statements.

Statement	Meaning
\colbreak	Break code. Not supported.
\colsn	Number of columns. Not supported.
\colsxn	Space between columns. Not supported.
\endnhere	Include endnotes in this section. Not supported.
\evenbreak	Break code. Not supported.
\footeryn	Footer y position from bottom of page. Not supported.
\headeryn	Header y position from top of page. Not supported.
\linebetcol	Line between columns. Not supported.
\linecont	Line number continued from previous section. Not supported.
\linemodn	Line number modulus. Not supported.
\lineppage	Line number restart on each page. Not supported.
\linerestart	Line number restart at 1. Not supported.
\linestartsn	Beginning line number. (1) Not supported.
\linexn	Line number, text distance. Not supported.
\nobreak	Break code. Not supported.
\oddbreak	Break code. Not supported.
\pagebreak	Break code. Not supported.
\pgncont	Continuous page numbering. (default) Not supported.
\pgndec	Page number format, decimal. Not supported.
\pgnlcltr	Page number format, lowercase letter. Not supported.
\pgnlcrm	Page number format, lower case roman. Not supported.
\pgnrestart	Restart page numbers at 1. Not supported.
\pgnstartsn	Beginning page number. (1) Not supported.
\pgnucltr	Page number format, uppercase letter. Not supported.
\pgnucrm	Page number format, uppercase roman. Not supported.
\pgnxn	Auto page number, x pos. Not supported.
\pgnyn	Auto page number, y pos. Not supported.
\sbkcol	Section break starts new column. Not supported.
\sbkeven	Section break starts at even page. Not supported.
\sbknone	No section break. Not supported.
\sbkodd	Section break starts at odd page. Not supported.
\sbkpage	Section break starts new page. Not supported.
\sectd	Reset to default section properties. Not supported.
\titlepg	Title page is special. Not supported.
\vertal	Bottom-aligned text. Not supported.
\vertalc	Vertically centered text. Not supported.
\vertalj	Vertically justified text. Not supported.
\vertalt	Top-aligned text. (default) Not supported.

Paragraph Formatting

The following statements specify paragraph formatting properties.

Statement	Meaning
\box	Boxed paragraph.
\brdrb	Bottom border.
\brdrbar	Ouside border.
\brdrbtw	Border between paragraphs. Not supported.
\brdrdb	Double border.
\brdrdot	Dotted border.
\brdrhair	Hairline border. Not supported.
\brdrl	Left border.
\brdrr	Right border.
\brdrs	Single-thickness border.
\brdrsh	Shadow border.
\brdrt	Top border.
\brdrth	Thick border.
\brspn	Space between border and object. Not supported.
\fin	First-line indent. (0)
\intbl	Table paragraph.
\keep	Nonwrapping text. OVERLOADED
\keepn	Nonscrolling region. OVERLOADED
\lin	Left indent. (0)
\noline	No line numbering. Not supported.
\pagebb	Page break before. Not supported.
\pard	Default paragraph properties.
\qc	Centered.
\qj	Justified.
\ql	Left-aligned (default).
\qr	Right-aligned.
\rin	Right indent. (0)
\san	Space after. (0)
\sbn	Space before. (0)
\sbys	Side-by-side paragraphs. Not supported in Help 3.1. Supported in Help 3.0.
\sln	Line spacing or leading.
\sn	Style. Not supported.
\tbn	Bar tab.
\tldot	Leader dots. Not supported.
\tlhyph	Leader hyphens. Not supported.
\tlth	Leader thick line. Not supported.
\tlul	Leader underscore. Not supported.
\tqc	Centered tab.
\tqdec	Decimal-aligned tab. Not supported.
\tqr	Flush-right tab.
\txn	Custom tab position.

Character Formatting

The following statements specify character formatting properties.

Statement	Meaning
\b	Bold
\caps	All capitals. Not supported.
\dnn	Subscript position. (6 pt.) Not supported.
\expandn	Expansion. (0) Not supported.
\fn	Font number
\fsn	Font size (24 pt.)
\i	Italic
\outl	Outline. Not supported.
\plain	Resets applications default character formatting properties.
\revised	Text added after revision marking turned on. Not supported.
\scaps	Small capitals
\shad	Shadow. Not supported.
\strike	Jump or macro hot spot; identical to \uldb OVERLOADED
\ul	Pop-up hot spot OVERLOADED
\uld	Dotted underline. Not supported.
\uldb	Jump or macro hot spot; identical to \strike OVERLOADED
\ulnone	Stop all underlining. Not supported.
\ulw	Word underline. Not supported.
\upn	Superscript position. (6 pt.) Not supported.
\v	Context string or macro OVERLOADED

Tables

The following statements specify table formatting properties.

Statement	Meaning
\cellxn	Set absolute position of a table cells right edge.
\clbrdrb	Bottom table cell border. Not supported.
\clbrdrl	Left table cell border. Not supported.
\clbrdrr	Right table cell border. Not supported.
\clbrdrt	Top table cell border. Not supported.
\clmgf	Mark first cell in a range of cells to be merged.
\clmrg	Merge current cell with preceding cell.
\trgaphn	Space between text in adjacent cells.
\trleftn	Set the position of left margin for the first cell in table row.
\trowd	Sets table row defaults.
\trqr	Right aligns text in each cell of table row. Not suppoted.
\trqc	Relative column widths. OVERLOADED
\trql	Left aligns text in each cell of table row. (default)
\trrhn	Height of table row. Not supported.

Pictures

The following statements specify picture formatting.

Statement	Meaning
\binn	Binary picture data.
\box	Boxed picture.
\brdrb	Bottom picture border.
\brdrbar	Outside picture border.
\brdrdb	Double picture border.
\brdrdot	Dotted picture border.
\brdrhair	Hairline picture border. Not supported.
\brdrl	Left picture border.
\brdrr	Right picture border.
\brdrs	Single-thickness picture border.
\brdrsh	Shadow picture border.
\brdrt	Top picture border.
\brdrth	Thick picture border.
\dibitmapn	Device independent bitmap.
\macpict	QuickDraw picture. Not supported.
\piccropbn	Bottom cropping value. Not supported.
\piccropln	Left cropping value. Not supported.
\piccroprn	Right cropping value. Not supported.
\piccroptn	Top cropping value. Not supported.
\pichgoaln	Desired picture height.
\pichn	Picture height.
\picscaled	Scale picture to fit in frame. Not supported.
\picscalexn	Horizontal scaling value. (100)
\picscaleyn	Vertical scaling value. (100)
\picwgoaln	Desired picture width.
\picwn	Picture width.
\wbitmapn	Picture type (Default is Windows bitmap).
\wbmbitspixeln	Bits per pixel. (1)
\wbmplanesn	Number of bitmap color planes. (1)
\wbmwidthbytesn	Bitmap width, in bytes.
\wmetafilen	Windows metafile.

Absolute-Positioned Objects

Windows Help version 3.1 does not support any absolute-positioned-object statements.

Statement	Meaning
\abswn	Absolute width of paragraph text. Not supported.
\dxfrtextn	Horizontal distance from text in main text flow. Not supported.
\phcol	Positions horizontally relative to column. Not supported.
\phmrg	Positions horizontally relative to margin. Not supported.
\phpg	Positions horizontally relative to page. Not supported.
\posxc	Centers horizontally within reference frame. Not supported.
\posxi	Positions horizontally inside reference frame. Not supported.
\posxl	Positions to left within reference frame. Not supported.
\posxn	Positions from left edge of reference frame. Not supported.
\posxo	Positions horizontally outside reference frame. Not supported.
\posxr	Positions to right within reference frame. Not supported.
\posyb	Positions at bottom of reference frame. Not supported.
\posyc	Centers vertically within reference frame. Not supported.
\posyil	Positions vertically to be in-line. Not supported.
\posyn	Positions from top edge of reference frame. Not supported.
\posyt	Positions at top of reference frame. Not supported.
\pvmrg	Positions vertically relative to margin. Not supported.
\pvpg	Positions vertically relative to page. Not supported.

Annotations

Windows Help version 3.1 does not support any annotation statements.

Statement	Meaning
\annotation	Annotation group reference. Not supported.
\antid	Annotation authors identification text. Not supported.
\chatn	Annotation reference character. Not supported.

Headers and Footers

Windows Help version 3.1 does not support any header and footer statements.

Statement	Meaning
\footerf	Footer for first page. Not supported.
\footerl	Footer for left-hand pages. Not supported.
\footerr	Footer for right-hand pages. Not supported.
\headerf	Header for first page. Not supported.
\headerl	Header for left-hand pages. Not supported.
\headerr	Header for right-hand pages. Not supported.

Document Information

Windows Help version 3.1 does not support any information statements.

Statement	Meaning
\author	Documents author. Not supported.
\buptim	Backup time. Not supported.
\comment	Comments; text is ignored. Not supported.
\creatim	Creation time. Not supported.
\doccomm	Comments in Edit Summary Info dialog box. Not supported.
\dyn	Day. Not supported.
\edminsn	Total editing time. Not supported.
\hrn	Hour. Not supported.
\idn	Internal identification number. Not supported.
\keywords	Selected document keywords. Not supported.
\minn	Minutes. Not supported.
\mon	Month. Not supported.
\nextfile	Next file. Not supported.
\nofcharsn	Number of characters. Not supported.
\nofpagesn	Number of pages. Not supported.
\nofwordsn	Number of words. Not supported.
\operator	Last person to make changes. Not supported.
\printtim	Last print time. Not supported.
\revtim	Revision time. Not supported.
\secn	Seconds. Not supported.
\subject	Subject matter. Not supported.
\title	Document title. Not supported.
\vernn	Internal version number. Not supported.
\versionn	Document version number. Not supported.
\yrn	Year. Not supported.

Fields

Windows Help version 3.1 supports only one field statement.

Statement	Meaning
\flddirty	Change made since last update. Not supported.
\fldedit	Text edited since last update. Not supported.
\fldinst	Field instructions. Not supported.
\fldlock	Filed is locked. Not supported.
\fldpriv	Result is in unknown format. Not supported.
\fldrslt	Most recent calculated result of the field.

Index Entries

Windows Help version 3.1 does not support any index statements.

Statement	Meaning
\bxe	Bold cross-reference or page number. Not supported.
\ixe	Italic cross-reference or page number. Not supported.
\rxe	Generate page numbers for range of text specified by bookmark name. Not supported.
\txe	Use text instead of page number. Not supported.

Table of Contents Entries

Windows Help version 3.1 does not support any table of contents statements.

Statement	Meaning
\tcfn	Type of table of contents. Not supported.
\tcln	Level number. (1) Not supported.

Bookmarks

Windows Help version 3.1 does not support any bookmark statements.

Statement	Meaning
\bkmkstartn	Start of specified bookmark. Not supported.
\bkmkend	End of specified bookmark. Not supported.

Help RTF Statement Reference

This section lists the Windows Help RTF statements in alphabetic order.

The Help RTF Statement Reference lists in alphabetic order all RTF statements supported by the Microsoft Help compiler version 3.1. Statement descriptions provide the following information.

Heading	Information
Statement	RTF statement supported by the Help compiler.
Comments	Notes about using the RTF statement, including any restrictions.
Example	Example of the RTF statement.
See Also	Cross-references to similar Help RTF statements.

\ansi

\ansi

The \ansi statement sets the American National Standards Institute (ANSI) character set. The Windows character set is essentially equivalent to the ANSI character set.

See Also

\windows

\b

\b

The \b statement starts bold text. The statement applies to all subsequent text up to the next \plain or \b0 statement.

Comments

No \plain or \b0 statement is required if the \b statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to just the enclosed text.

The \b0 statement was first supported in the Microsoft Help compiler version 3.1.

Example

The following example sets Note to bold:

```
{\b Note}   Setting the Auto option frees novice users from determining
their system configurations.
```

See Also

\i, \plain, \scaps

\bin

\bin n

The \bin statement indicates the start of binary picture data. The Help compiler interprets subsequent bytes in the file as binary data. This statement is used in conjunction with the \pict statement.

Parameter

n Specifies the number of bytes of binary data following the statement.

Comments

A single space character must separate the \bin statement from subsequent bytes. The Microsoft Help compiler assumes that all subsequent bytes, including linefeed and carriage return characters, are binary data. These bytes can have any value in the range 0 through 255. For this reason, the \bin statement is typically used in program-generated files only.

If the \bin statement is not given with a \pict statement, the default picture data format is hexadecimal.

See Also

\pict

\bmc

\{bmc filename\}

The bmc statement displays a specified bitmap or metafile in the current line of text. The statement positions the bitmap or metafile as if it were the next character in the line, aligning it on the base line and applying the current paragraph properties.

Parameter

filename — Specifies the name of a file containing either a Windows bitmap, a placeable Windows metafile, a multiresolution bitmap, or a segmented graphics bitmap.

Comments

Since the bmc statement is not a standard RTF statement, the Microsoft Help Compiler relies on the opening and closing braces, including the backslashes (\), to distinguish the statement from regular text.

If a file containing a metafile is specified, the file must contain a placeable Windows metafile; the Help compiler will not accept standard Windows metafiles. Furthermore, Windows Help sets the MM_ANISOTROPIC mode prior to displaying the metafile, so the placeable Windows metafile must either set the window origin and extents or set some other mapping mode.

Example

The following example inserts a bitmap representing a keyboard key in a paragraph:

```
\par
Press the \ key.bmp to return to the main window.
\par
```

See Also

bml, bmr, \wbitmap

\bml

\{bml filename\}

The bml statement displays a specified bitmap or metafile at the left margin of the Help window. The first line of subsequent text aligns with the upper-right corner of the image and subsequent lines wrap along the right edge of the image.

Parameters

filename — Specifies the name of a file containing either a Windows bitmap, a placeable Windows metafile, a multiresolution bitmap, or a segmented graphics bitmap.

Comments

Since the bml statement is not a standard RTF statement, the Help compiler relies on the opening and closing braces, including the backslashes (\), to distinguish the statement from regular text.

If a file containing a metafile is specified, the file must contain a placeable Windows metafile; the Help compiler will not accept standard Windows metafiles. Furthermore, Windows Help sets the MM_ANISOTROPIC mode prior to displaying the metafile, so the placeable Windows metafile must either set the window origin and extents or set some other mapping mode.

Example

The following example places a bitmap at the left margin. The subsequent paragraph wraps around the bitmap:

```
\par
\bml map.bmp\
The map at the left shows the easiest route to the school.
Although many people use Highway 125, there are fewer stops
and less traffic if you use Ames Road.
```

See Also

bmc, bmr, \wbitmap

\bmr

\{bmr filename\}

The bmr statement displays a specified bitmap or metafile at the right margin of the Help window. The first line of subsequent text aligns with the upper-left corner of the image and subsequent lines wrap along the left edge of the image.

Parameter

filename — Specifies the name of a file containing either a Windows bitmap, a placeable Windows metafile, a multiresolution bitmap, or a segmented graphics bitmap.

Comments

Since the bmr statement is not a standard RTF statement, the Help compiler relies on the opening and closing braces, including the backslashes (\), to distinguish the statement from regular text.

If a file containing a metafile is specified, the file must contain a placeable Windows metafile; the Help compiler will not accept standard Windows metafiles. Furthermore, Windows Help sets the MM_ANISOTROPIC mode prior to displaying the metafile, so the placeable Windows metafile must either set the window origin and extents or set some other mapping mode.

Example

The following example places a bitmap at the right margin. The subsequent paragraph wraps around the bitmap:

```
\par
\bml map.bmp\
The map at the right shows the easiest route to the school.
Although many people use Highway 125, there are fewer stops
and less traffic if you use Ames Road.
```

See Also

bmc, bml, \wbitmap

\box

\box

The \box statement draws a box around the current paragraph or picture. The statement applies to all subsequent paragraphs or pictures up to the next \pard statement.

Comments

For paragraphs, Windows Help uses the height of the paragraph, excluding space before or after the paragraph, as the height of the box. For pictures (as defined by \pict statements), Windows Help uses the specified height of the picture as the height of the box. For both paragraphs and pictures, the width of the box is equal to the space between the left and right indents.

Windows Help draws the box using the current border style.

Example

The following example draws a box around the paragraph:

```
\par \box
{\b Note}  Setting the Auto option frees novice users from
determining their system configurations.
\par \pard
```

See Also

\brdrb, \brdrl, \brdrr, \brdrt, \pard

\brdrb

\brdrb

The \brdrb statement draws a border below the current paragraph or picture. The statement applies to all subsequent paragraphs or pictures up to the next \pard statement.

Comments

Windows Help draws the border using the current border style.

See Also

\box, \brdrbar, \brdrl, \brdrr, \brdrt, \pard

\brdrbar

\brdrbar

The \brdrbar statement draws a vertical bar to the left of the current paragraph or picture. The statement applies to all subsequent paragraphs or pictures up to the next \pard statement.

Comments

Windows Help draws the border using the current border style.

In a print-based document, the \brdrbar statement draws the bar on the right side of paragraphs on odd-numbered pages, but on the left side of paragraphs on even-numbered pages.

See Also

\box, \brdrb, \brdrl, \brdrr, \brdrt, \pard

\brdrdb

\brdrdb

The \brdrdb statement selects a double line for drawing borders. The selection applies to all subsequent paragraphs or pictures up to the next \pard statement.

See Also

\brdrdot, \brdrs, \brdrsh, \brdrth, \pard

\brdrdot

\brdrdot

The \brdrdot statement selects a dotted line for drawing borders. The selection applies to all subsequent paragraphs or pictures up to the next \pard statement.

See Also

\brdrdb, \brdrs, \brdrsh, \brdrth, \pard

\brdrl

\brdrl

The \brdrl statement draws a border to the left of the current paragraph or picture. The statement applies to all subsequent paragraphs or pictures up to the next \pard statement.

Comments

Windows Help draws the border using the current border style.

See Also

\box, \brdrb, \brdrbar, \brdrr, \brdrt, \pard

\brdrr

\brdrr

The \brdrr statement draws a border to the right of the current paragraph or picture. The statement applies to all subsequent paragraphs or pictures up to the next \pard statement.

Comments

Windows Help draws the border using the current border style.

See Also

\box, \brdrb, \brdrbar, \brdrl, \brdrt, \pard

\brdrs

\brdrs

The \brdrs statement selects a standard-width line for drawing borders. The selection applies to all subsequent paragraphs or pictures up to the next \pard statement.

See Also

\brdrdb, \brdrdot, \brdrsh, \brdrth, \pard

\brdrsh

\brdrsh

The \brdrsh statement selects a shadow outline for drawing borders. The selection applies to all subsequent paragraphs or pictures up to the next \pard statement.

See Also

\bdrddb, \brdrdot, \brdrs, \brdrth, \pard

\brdrt

\brdrt

The \brdrt statement draws a border above the current paragraph or picture. The statement applies to all subsequent paragraphs or pictures up to the next \pard statement.

Comments

Windows Help draws the border using the current border style.

See Also

\box, \brdrb, \brdrbar, \brdrl, \brdrr, \pard

\brdrth

\brdrth

The \brdrth statement selects a thick line for drawing borders. The selection applies to all subsequent paragraphs or pictures up to the next \pard statement.

See Also

\brdrdb, \brdrdot, \brdrs, \brdrsh, \pard

\cell

\cell

The \cell statement marks the end of a cell in a table. A cell consists of all paragraphs from a preceding \ intbl or \cell statement to the ending \cell statement. Windows Help formats and displays these paragraphs using the left and right margins of the cell and any current paragraph properties.

Comments

This statement was first supported in the Microsoft Help compiler version 3.1.

Example

The following example creates a two-column table. The second column contains three separate paragraphs, each having different paragraph properties:

```
\cellx2880\cellx5760
\intbl
Alignment\cell
\ql
Left-aligned
\par
\qc
Centered
\par
\qr
Right-aligned\cell
\row \pard
```

See Also

\cellx, \intbl, \row, \trgaph, \trleft, \trowd

\cellx

\cellxn

The \cellx statement sets the absolute position of the right edge of a table cell. One \cellx statement must be given for each cell in the table. The first \cellx statement applies to the leftmost cell, the last to the rightmost cell. For each \cellx statement, the specified position applies to the corresponding cell in each subsequent row of the table up to the next \trowd statement.

Parameter

n	Specifies the position of the cells right edge, in twips. The position is relative to the left edge of the Help window. It is not affected by the current indents.

Comments

A table consists of a grid of cells in columns and rows. Each cell has an explicitly defined right edge; the position of a cells left edge is the same as the position of the right edge of the adjacent cell. For the left-most cell in a row, the left edge position is equal to the Help windows left margin position. Each cell has a left and right margin between which Windows Help aligns and wraps text. By default, the margin positions are equal to the left and right edges. The \trgaph and \trleft statements can be used to set different margins for all cells in a row.

This statement was first supported in the Microsoft Help compiler version 3.1.

Example

The following example creates a three-column table having two rows. The positions of the right edges of the three cells are 2, 4, and 6 inches, respectively:

```
\cellx2880\cellx5760\cellx8640
\intbl
Row 1 Cell 1\cell
Row 1 Cell 2\cell
Row 1 Cell 3\cell
\row
\intbl
Row 2 Cell 1\cell
Row 2 Cell 2\cell
Row 2 Cell 3\cell
\row \pard
```

See Also

\cell, \intbl, \row, \trgaph, \trleft, \trowd

\cb

\cbn

The \cb statement sets the background color. The new color applies to all subsequent text up to the next \ plain or \cb statement.

Parameter

n	Specifies the color number to set as background. The number must be an integer number in the range 1 to the maximum number of colors specified in the color table for the Help file. If an invalid color number is specified, Windows Help uses the default background color.

Comments

No \plain or \cb statement is required if the \cb statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to the enclosed text only.

If the \cb statement is not given, the default background color is the text color set by Control Panel.

Example

The following example creates a light grey background color:

```
{\colortbl;\red128\green128\blue128;}
{\cb1 This background is light grey.}
```

See Also

\cf, \colortbl

\cf

\cfn

The \cf statement sets the foreground color. The new color applies to all subsequent text up to the next \ plain or \cf statement.

Parameter

n	Specifies the color number to set as foreground. The number must be an integer number in the range 1 to the maximum number of colors specified in the color table for the Help file. If an invalid color number is specified, Windows Help uses the default foreground color.

Comments

No \plain or \cf statement is required if the \cf statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to the enclosed text only.

If the \cf statement is not given, the default foreground color is the text color set by Control Panel.

Example

The following example displays green text:

```
{\colortbl;\red0\green255\blue0;}
{\cf1 This text is green.}
```

See Also

\cb, \colortbl

\chftn

\chftnn

The \chftn statement sets the footnote reference character for subsequent \footnote statements. The Microsoft Help compiler ignores this statement.

Parameters

n Specifies the footnote reference character.

See Also

\footnote

\clmgf

\clmgf

The \clmgf statement specifies the first cell in a range of cells to be merged.

The Microsoft Help compiler ignores this statement.

Comments

All cells between the \clmgf statement and a subsequent \clmrg statement are combined into a single cell. The left edge of the new cell is the same as that of the leftmost cell to be merged; the rightedge is the same as that of the rightmost cell.

This statement was first supported in the Microsoft Help compiler version 3.1.

See Also

\clmrg

\clmrg

\clmrg

The \clmrg statement merges the current cell with the preceding cell.

The Microsoft Help compiler ignores this statement.

Comments

All cells between the \clmgf statement and a subsequent \clmrg statement are combined into a single cell. The left edge of the new cell is the same as that of the leftmost cell to be merged; the rightedge is the same as that of the rightmost cell.

This statement was first supported in the Microsoft Help compiler version 3.1.

See Also

\clmgf

\colortbl

{\colortbl\redr\greeng\blueb; . . .}

The \colortbl statement creates a color table for the Help file. The color table consists of one or more color definitions. Each color definition consists of one \red, \green, and \blue statement specifying the amount of primary color to use to generate the final color. Each color definition must end with a semicolon (;).

Parameters

r	Specifies the intensity of red in the color. It must be an integer in the range 0 through 255.
g	Specifies the intensity of green in the color. It must be an integer in the range 0 through 255.
b	Specifies the intensity of blue in the color. It must be an integer in the range 0 through 255.

Comments

Color definitions are implicitly numbered starting at zero. A color definitions implicit number can be used in the \cf statement to set the foreground color.

The default colors are the window text and window background colors set by Control Panel. To override the default colors, a \colortbl statement and \cf and \cb statements must be given.

Example

The following example creates a color table containing two color definitions. The first color definition is empty (only the semicolon is given), so color number 0 always represents the default color. The second definition specifies green; color number 1 can be used to display green text:

```
{\colortbl;\red0\green255\blue0;}
```

See Also

\cb, \cf

\deff

\deffn

The \deff statement sets the default font number. Windows Help uses the number to set the default font whenever a \plain statement is given or an invalid font number is given in a \f statement.

Parameters

n	Specifies the number of the font to be used as the default font. This parameter must be a valid font number as specified by the \fonttbl statement for the Help file.

Comments

If the \deff statement is not given, the default font number is zero.

See Also

\f, \fonttbl, \plain

\ewc

\{ewc DLL-name, window-class, author-data\}

The ewc statement displays information (bitmap, multimedia element, and so on.) under the control of a dynamic-link library in the current line of text. The statement positions the object as if it were the next character in the line, aligning it on the base line and applying the current paragraph properties.

Parameters

DLL-name	Specifies the name of the DLL that controls the embedded window. The file ame should not include an extension or be fully qualified, but it can include a relative path. Windows Help assumes .DLL or .EXE to be the default extension.
window-class	Specifies the name of the embedded window class as defined in the source code for the DLL.
author-data	Specifies an arbitrary string, which Windows Help passes to the embedded window when it creates the window. This string can be one or more substrings separated by any punctuation mark except a comma. The DLL is responsible for parsing this string.

Comments

Since the ewc statement is not a standard RTF statement, the Microsoft Help compiler relies on the opening and closing braces, including the backslashes (\), to distinguish the statement from regular text.

Because the embedded window is treated as text, paragraph formatting properties assigned to the paragraph also apply to the window. Text coming before or after the window does not wrap around the window. Help adjusts the height of the line containing the embedded window to allow enough space for the embedded window.

If you use the \sl statement in conjunction with an ewc statement, dont specify a negative value. If you do, the embedded window might appear on top of the succeeding paragraph when Windows Help displays the topic.

Example

The following example displays a 256-color bitmap in an embedded window within a paragraph:

```
\par
preceding text \ following text...
\par
```

See Also

ewl, ewr

\ewl

\{ewl DLL-name, window-class, author-data\}

The ewl statement displays information (bitmap, multimedia element, and so on.) under the control of a dynamic-link library at the left margin of the Help window. The first line of subsequent text aligns with the upper-right corner of the embedded window and subsequent lines wrap along the right edge of the window.

Parameters

DLL-name	Specifies the name of the DLL that controls the embedded window. The filename should not include an extension or be fully qualified, but it can include a relative path. Windows Help assumes .DLL or .EXE to be the default extension.
window-class	Specifies the name of the embedded window class as defined in the source code for the DLL.
author-data	Specifies an arbitrary string, which Windows Help passes to the embedded window when it creates the window. This string can be one or more substrings separated by any punctuation mark except a comma. The DLL is responsible for parsing this string.

Comments

Since the ewl statement is not a standard RTF statement, the Microsoft Help compiler relies on the opening and closing braces, including the backslashes (\), to distinguish the statement from regular text.

Do not put any space characters between the ewl statement and the paragraph text unless you want the first line of text indented from the rest of the text that wraps along the right edge of the window. To be sure that text wraps correctly around an embedded window, insert a soft carriage return at the end of each line of text.

Example

The following example places an embedded window at the left margin. The subsequent paragraph wraps around the embedded window:

```
\par
\
The map at the left shows the easiest route to the school.
Although many people use Highway 125, there are fewer stops
and less traffic if you use Ames Road.
```

See Also

ewc, ewr

\ewr

\{ewr DLL-name, window-class, author-data\}

The ewr statement displays information (bitmap, multimedia element, and so on.) under the control of a dynamic-link library at the right margin of the Help window. The first line of subsequent text aligns with the upper-left corner of the embedded window and subsequent lines wrap along the left edge of the window.

Parameters

DLL-name	Specifies the name of the DLL that controls the embedded window. The filename should not include an extension or be fully qualified, but it can include a relative path. Windows Help assumes .DLL or .EXE to be the default extension.
window-class	Specifies the name of the embedded window class as defined in the source code for the DLL.
author-data	Specifies an arbitrary string, which Windows Help passes to the embedded window when it creates the window. This string can be one or more substrings separated by any punctuation mark except a comma. The DLL is responsible for parsing this string.

Comments

Since the ewr statement is not a standard RTF statement, the Microsoft Help compiler relies on the opening and closing braces, including the backslashes (\), to distinguish the statement from regular text.

Do not put any space characters between the ewr statement and the paragraph text unless you want the first line of text indented from the rest of the text that wraps from the left topic margin. To be sure that text wraps correctly around an embedded window, insert a soft carriage return at the end of each line of text.

Example

The following example places an embedded window at the right margin. The subsequent paragraph wraps around the embedded window:

```
\par
\
The map at the right shows the easiest route to the school.
Although many people use Highway 125, there are fewer stops
and less traffic if you use Ames Road.
```

See Also

ewc, ewl

\f

\fn

The \f statement sets the font. The new font applies to all subsequent text up to the next \plain or \f statement.

Parameters

n	Specifies the font number. This parameter must be one of the integer font numbers defined in the font table for the Help file.

Comments

The \f statement does not set the point size of the font; use the \fs statement instead.

No \plain or \f statement is required if the \f statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to just the enclosed text.

If the \f statement is not given, the default font is defined by the \deff statement (or is zero if no \deff statement is given).

Example

The following example uses the Arial font to display text:

```
{\fonttbl {\f0\fswiss Arial;}}
\par
{\f0
This text illustrates the Arial font.}
\par
```

See Also

\deff, \fonttbl, \fs, \plain

\fi

\fin

The \fi statement sets the first-line indent for the paragraph. The new indent applies to the first line of each subsequent paragraph up to the next \pard statement or \fi statement. The first-line indent is always relative to the current left indent.

Parameters

n

Specifies the indent, in twips. This parameter can be either a positive or negative number.

Comments

If the \fi statement is not given, the first-line indent is zero by default.

Example

The following example uses the first-line indent and a tab stop to make a numbered list:

```
\tx360\li360\fi-360
1
\tab
Insert the disk in drive A.
\par
2
\tab
Type a:setup and press the ENTER key.
\par
3
\tab
Follow the instructions on the screen.
\par \pard
```

See Also

\li, \pard

\field

\field

The \field statement defines a field.

Comments

The Microsoft Help compiler ignores this statement and all related field statements except the \fldrslt statement.

See Also

\fldrslt

\fldrslt

\fldrslt

The \fldrslt statement specifies the most recently calculated result of a field. The Microsoft Help compiler interprets the result as text and formats it using the current character and paragraph properties.

Comments

The Help compiler ignores all field statements except the \fldrslt statement. Any text associated with other field statements is ignored.

See Also

\field

\fonttbl

\fonttbl {\fn\family font-name; . . .}

The \fonttbl statement creates a font table for the Help file. The font table consists of one or more font definitions. Each definition consists of a font number, a font family, and a font name.

Parameters

n — Specifies the font number. This parameter must be an integer. This number can be used in subsequent \f statements to set the current font to the specified font. In the font table, font numbers should start at zero and increase by one for each new font definition.

family — Specifies the font family. This parameter must be one of the following.

Value	Meaning
fnil	Unknown or default fonts (default)
froman	Roman, proportionally spaced serif fonts (for example, MS Serif and Palatino)
fswiss	Swiss, proportionally spaced sans serif fonts (for example, Swiss)
fmodern	Fixed-pitch serif and sans serif fonts (for example, Courier, Elite, and Pica)
fscript	Script fonts (for example, Cursive)
fdecor	Decorative fonts (for example, Old English and ITC Zapf Chancery)
ftech	Technical, symbol, and mathematical fonts (for example, Symbol)

font-name — Specifies the name of the font. This parameter should specify an available Windows font.

Comments

If a font with the specified name is not available, Windows Help chooses a font from the specified family. If no font from the given family exists, Windows Help chooses a font having the same character set as specified for the Help file.

The \deff statement sets the default font number for the Help file. The default font is set whenever the \ pard statement is given.

See Also

\deff, \f, \fs, \pard

\footnote

n{\footnote text}

The \footnote statement defines topic-specific information, such as the topics build tags, context string, title, browse number, keywords, and execution macros. Every topic must, at least, have a context string to give the user access to the topic through links.

Parameters

n — Specifies the footnote character. It can be one of the following.

Value	Meaning
*	Specifies a build tag. The Microsoft Help compiler uses build tags to determine whether it should include the topic in the Help file. The text parameter can be any combination of characters but must not contain spaces. Uppercase and lowercase characters are treated as equivalent characters (case insensitive). If a topic has build-tag statements, they must be the first statements in the topic. The Microsoft Help compiler checks a topic for build tags if the project file specifies a build expression using the BUILD option.
#	Specifies a context string. The text parameter can be any combination of letters and digits but must not contain spaces. Uppercase and lowercase characters are treated as equivalent characters (case insensitive). The context string can be used with the \v statement in other topics to create links to this topic.
$	Specifies a topic title. Windows Help uses the topic title to identify the topic in the Search and History dialog boxes. The text parameter can be any combination of characters including spaces.
+	Specifies the browse-sequence identifier. Windows Help adds topics having an identifier to the browse sequence and allows users to view the topics by using the browse buttons. The text parameter can be a combination of letters and digits. Windows Help determines the order of topics in the browse sequence by sorting the identifier alphabetically. If two topics have the same identifier, Windows Help assumes that the topic that was compiled first is to be displayed first. Windows Help uses the browse sequence identifier only if the browse buttons have been enabled by using the BrowseButtons macro.
K	Specifies a keyword. Windows Help displays all keywords in the Help file in the Search dialog box and allows a user to choose a topic to view by choosing a keyword. The text parameter can be any combination of characters including spaces. If the first character is the letter K; it must be preceded with an extra space or a semicolon. More than one keyword can be given by separating the keywords with semicolons (;). A topic cannot contain keywords unless it also has a topic title.
!	Specifies a Help macro. Windows Help executes the macro when the topic is displayed. The text parameter can be any Help macro.
@	Specifies an author-defined comment about the Help topic. This footnote is provided for authoring purposes only. The Help compiler ignores this footnote when building the Help file. The text parameter can be any combination of letters and digits, including accented characters and spaces. If n is any letter (other than K), the footnote specifies an alternative keyword. Windows applications can search for topics having alternative keywords by using the HELP_MULTIKEY command with the WinHelp function.
text	Specifies the build tag, context string, topic title, browse-sequence

number, keyword, or macro associated with the footnote. This parameter depends on the footnote type as specified by the n parameter.

Comments

A topic can have more than one build-tag, context-string, keyword, and help-macro statement, but must not have more than one topic-title or browse-sequence-number statement.

In print-based documents, the \footnote statement creates a footnote and the footnote is anchored to the character immediately preceding the \footnote statement.

Example

The following example defines a topic titled Short Topic. The context string topic1 can be used to create links to the topic. The keywords example topic and short topic appear in the Search dialog box and can be used to choose the topic for viewing. Or the user can find the topic by browsing since the topic is included in the short browse sequence. The topic also includes an entry macro that starts the Clock application when Help displays the topic:

```
${\footnote Short Topic}
#{\footnote topic1}
K{\footnote example topic;short topic}
+{\footnote short:015}
!{\footnote ExecProgram(`clock.exe', 0)}
This topic has a title, context string, and two keywords.
\par
\page
```

See Also

\chftn, \v

\fs

\fsn

The \fs statement sets the size of the font. The new font size applies to all subsequent text up to the next \ plain or \fs statement.

Parameters

n Specifies the size of the font, in half points.

Comments

The \fs statement does not set the font face; use the \f statement instead.

No \plain or \fs statement is required if the \fs statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to just the enclosed text.

If the \fs statement is not given, the default font size is 24.

Example

The following example sets the size of the font to 10 points:

```
{\fs20 This line is in 10 point type.}
\par
```

See Also

\f, \plain

\'

\hh

The \ statement converts the specified hexadecimal number into a character value and inserts the value into the Help file. The appearance of the character when displayed depends on the character set specified for the Help file.

Parameters

hh Specifies a two-digit hexadecimal value.

Comments

Since the Microsoft Help compiler does not accept character values greater than 127, the \ statement is the only method to insert such character values into the Help file.

Example

The following example inserts a trademark in a Help file that uses the \windows statement to set the character set:

```
ABC\99 is a trademark of the ABC Product Corporation.
```

See Also

\ansi, \pc, \pca, \windows

\i

\i

The \i statement starts italic text. The statement applies to all subsequent text up to the next \plain or \i0 statement.

Comments

No \plain or \i0 statement is required if the \i statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to just the enclosed text.

Example

The following example sets not to italic:

```
You must {\i not} save the file without first setting the
Auto option.
```

See Also

\b, \plain, \scaps

\intbl

\intbl

The \intbl statement marks subsequent paragraphs as part of a table. The statement applies to all subsequent paragraphs up to the next \row statement.

Comments

This statement was first supported in Microsoft Help compiler version 3.1.

Example

The following example creates a three-column table having two rows:

```
\cellx1440\cellx2880\cellx4320
\intbl
Row 1 Column 1\cell
Row 1 Column 2\cell
Row 1 Column 3\cell \row
\intbl
Row 2 Column 1\cell
Row 2 Column 2\cell
Row 2 Column 3\cell \row \pard
```

See Also

\cell, \cellx, \row, \trgaph, \trleft, \trowd

\keep

\keep

The \keep statement prevents Windows Help from wrapping text to fit the Help window. The statement applies to all subsequent paragraphs up to the next \pard statement.

Comments

If the text in a paragraph exceeds the width of the Help window, Help displays a horizontal scroll bar.

In print-based documents, the \keep statement keeps paragraphs intact.

Example

The following example makes the paragraph beginning Cardfile has two viewsCard and List nonwrapping text:

```
\par\pard\keep
Cardfile has two views--Card and List.
```

See Also

\line

\keepn

\keepn

The \keepn statement creates a nonscrolling region at the top of the Help window for the given topic. The \keepn statement applies to all subsequent paragraphs up to the next \pard statement. All paragraphs with this paragraph property are placed in the 6nonscrolling region.

Comments

If a \keepn statement is used in a topic, it must be applied to the first paragraph in the topic (and subsequent paragraphs as needed). The Help compiler displays an error message and does not create a nonscrolling region if paragraphs are given before the \keepn statement. Only one nonscrolling region per topic is allowed.

Windows Help formats, aligns, and wraps text in the nonscrolling region just as it does in the rest of the topic. It separates the nonscrolling region from the rest of the Help window with a horizontal bar. Windows Help sets the height of the nonscrolling region so that all paragraphs in the region can be viewed if the Help window is large enough. If the window is smaller than the nonscrolling region, the user will be unable to view the rest of the topic. For this reason, the nonscrolling region is typically reserved for a single line of text specifying the name or title of the topic.

In print-based documents, the \keepn statement keeps the subsequent paragraph with the paragraph that follows it.

Example

The following example places the first paragraph of the topic beginning Cardfile has two viewsCard and List into a nonscrolling region:

```
\par\pard\keepn
Cardfile has two views--Card and List.
\par\pard
```

See Also

\page

\li

\lin

The \li statement sets the left indent for the paragraph. The indent applies to all subsequent paragraphs up to the next \pard or \li statement.

Parameters

n Specifies the indent, in twips. The value can be either positive or negative.

Comments

If the \li statement is not given, the left indent is zero by default. Windows Help automatically provides a small left margin so that if no indent is specified the text does not start immediately at the left edge of the Help window.

Specifying a negative left indent moves the starting point for a line of text to the left of the default left margin. If the negative indent is large enough, the start of the text may be clipped by the left edge of the Help window.

Example

The following example uses the left indent and a tab stop to make a bulleted list. In this example, font number 0 is assumed to be the Symbol font:

```
Use the Auto command to:
\par
\tx360\li360\fi-360
{\f0\B7}
\tab
Save files automatically
\par
{\f0\B7}
\tab
Prevent overwriting existing files
\par
{\f0\B7}
\tab
Create automatic backup files
\par \pard
```

See Also

\fi, \pard, \ri

\line

\line

The \line statement breaks the current line without ending the paragraph. Subsequent text starts on the next line and is aligned and indented according to the current paragraph properties.

See Also

\par

\mac

\mac

The \mac statement sets the Apple Macintosh character set.

See Also

\windows

\page

\page

The \page statement marks the end of a topic.

Comments

In a print-based document, the \page statement creates a page break.

Example

The following example shows a complete topic:

```
${\footnote Short Topic}
#{\footnote short_topic}
Most topics in a topic file consist of topic-title and
context-string statements followed by the topic text. Every
topic ends with a {\b \\page} statement.
\par
\page
```

See Also

\par

\par

\par

The \par statement marks the end of a paragraph. The statement ends the current line of text and moves the current position to the left margin and down by the current line-spacing and space-after-paragraph values.

Comments

The first line of text after a \par, \page, or \sect statement marks the start of a paragraph. When a paragraph starts, the current position is moved down by the current space-before-paragraph value. Subsequent text is formatted using the current text alignment, line spacing, and left, right, and first-line indents.

Example

The following example has three paragraphs:

```
\ql
This paragraph is left-aligned.
\par \pard
\qc
This paragraph is centered.
\par \pard
\qr
This paragraph is right-aligned.
\par
```

See Also

\line, \pard, \sect

\pard

\pard

The \pard statement restores all paragraph properties to default values.

Comments

If the \pard statement appears anywhere before the end of a paragraph (that is, before the \par statement), the default properties apply to the entire paragraph.

The default paragraph properties are as follows.

Property	Default
Alignment	Left-aligned
First-line indent	0
Left indent	0
Right indent	0
Space before	0
Space after	0
Line spacing	Tallest character
Tab stops	None
Borders	None
Border style	Single-width

See Also

\par

\pc

\pc

The \pc statement sets the OEM character set (also known as code page 437).

See Also

\windows

\pca

\pca

The \pca statement sets the International English character set (also known as code page 850).

See Also

\windows

\pich

\pichn

The \pich statement specifies the height of the picture. This statement must be used in conjunction with a \pict statement.

Parameters

n	Specifies the height of the picture, in twips or pixels, depending on the picture type. If the picture is a metafile, the width is in twips; otherwise, the width is in pixels.

See Also

\pict, \picw

\pichgoal

\pichgoaln

The \pichgoal statement specifies the desired height of a picture. If necessary, Windows Help stretches or compresses the picture to match the requested height. This statement must be used in conjunction with a \pict statement.

Parameters

n Specifies the desired height, in twips.

See Also

\pict, \picwgoal

\picscalex

\picscalexn

The \picscalex statement specifies the horizontal scaling value. This statement must be used in conjunction with a \pict statement.

Parameters

n	Specifies the scaling value. The parameter must be a value in the range 0 through 100, representing a percentage.

Comments

If the \picscalex statement is not given, the default scaling value is 100.

See Also

\picscaley, \pict

\picscaley

\picscaleyn

The \picscaley statement specifies the vertical scaling value. This statement must be used in conjunction with a \pict statement.

Parameters

n	Specifies the scaling value. The parameter must be an integer in the range 0 through 100, representing a percentage.

Comments

If the \picscaley statement is not given, the default scaling value is 100.

See Also

\picscalex, \pict

\pict

\pict picture-statements picture-data

The \pict statement creates a picture. A picture consists of hexadecimal or binary data representing a bitmap or metafile.

Parameters

picture-statements Specifies one or more statements defining the type of picture, the dimensions of the picture, and the format of the picture data. It can be a combination of the following statements:

Statement	Description
\wbitmap	Specifies a Windows bitmap.
\wmetafile	Specifies a Windows metafile.
\picw	Specifies the picture width.
\pich	Specifies the picture height.
\picwgoal	Specifies the desired picture width.
\pichgoal	Specifies the desired picture height.
\picscalex	Specifies the horizontal scaling value.
\picscaley	Specifies the vertical scaling value.
\wbmbitspixel	Specifies the number of bits per pixel.
\wbmplanes	Specifies the number of planes.
\wbmwidthbytes	Specifies the bitmap width, in bytes.
\bin	Specifies binary picture data.

picture-data Specifies hexadecimal or binary data representing the picture. The picture data follows the last picture statement.

Comments

If a data format is not specified, the default format is hexadecimal.

See Also

\bin, \pich, \pichgoal, \picscalex, \picscaley, \picw, \picwgoal, \wbitmap, \wbmbitspixel, \wbmplanes, \wbmwidthbytes, \wmetafile

\picw

\picwn

The \picw statement specifies the width of the picture. This statement must be used in conjunction with a \pict statement.

Parameters

n	Specifies the width of the picture, in twips or pixels, depending on the picture type. If the picture is a metafile, the width is in twips; otherwise, the width is in pixels.

See Also

\pich, \pict

\picwgoal

\picwgoaln

The \picwgoal statement specifies the desired width of the picture, in twips. If necessary, Windows Help stretches or compresses the picture to match the requested height. This statement must be used in conjunction with a \pict statement.

Parameters

n Specifies the desired width, in twips.

See Also

\pichgoal, \pict

\plain

\plain

The \plain statement restores the character properties to default values.

Comments

The default character properties are as follows.

Property	Default
Bold	Off
Italic	Off
Small caps	Off
Font	0
Font size	24

See Also

\b, \f, \fs, \i, \scaps

\qc

\qc

The \qc statement centers text between the current left and right indents. The statement applies to subsequent paragraphs up to the next \pard statement or text-alignment statement.

Comments

If a \ql, \qr, \qc, or \qj statement is not given, the text is left-aligned by default.

See Also

\qj, \ql, \qr, \pard

\qj

\qj

The \qj statement justifies text between the current left and right indents. The statement applies to subsequent paragraphs up to the next \pard statement or text-alignment statement.

Comments

If a \ql, \qr, \qc, or \qj statement is not given, the text is left-aligned by default.

See Also

\qc, \ql, \qr, \pard

\ql

\ql

The \ql statement aligns text along the left indent. The statement applies to subsequent paragraphs up to the next \pard statement or text-alignment statement.

Comments

If a \ql, \qr, \qc, or \qj statement is not given, the text is left-aligned by default.

See Also

\qc, \qj, \qr, \pard

\qr

\qr

The \qr statement aligns text along the right indent. The statement applies to subsequent paragraphs up to the next \pard statement or text-alignment statement.

Comments

If a \ql, \qr, \qc, or \qj statement is not given, the text is left-aligned by default.

See Also

\qc, \qj, \ql, \pard

\ri

\rin

The \ri statement sets the right indent for the paragraph. The indent applies to all subsequent paragraphs up to the next \pard or \ri statement.

Parameters

n Specifies the right indent, in twips. It can be a positive or negative value.

Comments

If the \ri statement is not given, the right indent is zero by default. Windows Help automatically provides a small right margin so that when no right indent is specified, the text does not end abruptly at the right edge of the Help window.

Windows Help never displays less than one word for each line in a paragraph even if the right indent is greater than the width of the window.

Example

In the following example, the right and left indents are set to one inch and the subsequent text is centered between the indents:

```
\li1440\ri1440\qc
Microsoft Windows Help\line
Sample File\line
```

See Also

\li, \pard

\row

\row

The \row statement marks the end of a table row. The statement ends the current row and begins a new row by moving down pass the end of the longest cell in the row. The next \cell statement specifies the text of the leftmost cell in the next row.

Comments

This statement was first supported in the Microsoft Help compiler version 3.1.

Example

The following example creates a table having four rows and two columns:

```
\cellx2880\cellx5760
\intbl
Row 1, Column 1\cell
Row 1, Column 2\cell \row
\intbl
Row 2, Column 1\cell
Row 2, Column 2\cell \row
\intbl
Row 3, Column 1\cell
Row 3, Column 2\cell \row
\intbl
Row 4, Column 1\cell
Row 4, Column 2\cell \row
\par \pard
```

See Also

\cell, \cellx, \intbl

\rtf

\rtfn

The \rtf statement identifies the file as a rich-text format (RTF) file and specifies the version of the RTF standard used.

Parameters

n	Specifies the version of the RTF standard used. For the Microsoft Help compiler version 3.1, this parameter must be 1.

Comments

The \rtf statement must follow the first open brace in the Help file. A statement specifying the character set for the file must also follow the \rtf statement.

See Also

\windows

\sa

\san

The \sa statement sets the amount of vertical spacing after a paragraph. The vertical space applies to all subsequent paragraphs up to the next \pard or \sa statement.

Parameters

n Specifies the amount of vertical spacing, in twips.

Comments

If the \sa statement is not given, the vertical spacing after a paragraph is zero by default.

See Also

\pard, \sb

\sb

\sbn

The \sb statement sets the amount of vertical spacing before the paragraph. The vertical space applies to all subsequent paragraphs up to the next \pard or \sb statement.

Parameters

n Specifies the amount of vertical spacing, in twips.

Comments

If the \sb statement is not given, the vertical spacing before the paragraph is zero by default.

See Also

\pard, \sa

\scaps

\scaps

The \scaps statement starts small-capital text. The statement converts all subsequent lowercase letters to uppercase before displaying the text. This statement applies to all subsequent text up to the next \plain or \scaps0 statement.

Comments

No \plain or \scaps0 statement is required if the \scaps statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to just the enclosed text.

The \scaps statement does not reduce the point size of the text. To reduce point size, the \fs statement must be used.

Example

The following example displays the key name ENTER in small capitals:

```
Press the {\scaps ENTER} key to complete the action.
```

See Also

\plain

\sect

\sect

The \sect statement marks the end of a section and paragraph.

See Also

\par

\sl

\sln

The \sl statement sets the amount of vertical space between lines in a paragraph. The vertical space applies to all subsequent paragraphs up to the next \pard or \sl statement.

Parameters

n	Specifies the amount of vertical spacing, in twips. If this parameter is a positive value, Windows Help uses this value if it is greater than the tallest character. Otherwise, Windows Help uses the height of the tallest character as the line spacing. If this parameter is a negative value, Windows Help uses the absolute value of the number even if the tallest character is taller.

Comments

If the \sl statement is not given or the specified amount of spacing is 1000, Windows Help automatically sets the line spacing by using the tallest character in the line.

See Also

\pard

\strike

\strike

The \strike statement creates a hot spot. The statement is used in conjunction with a \v statement to create a link to another topic. When the user chooses a hot spot, Windows Help displays the associated topic in the Help window.

The \strike statement applies to all subsequent text up to the next \plain or \strike0 statement.

Comments

No \plain or \strike0 statement is required if the \strike statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to just the enclosed text.

In print-based documents, or whenever it is not followed by \v, the \strike statement creates strikeout text.

Example

The following example creates a hot spot for a topic. When displayed, the hot-spot text, Hot Spot, is green and has a solid line under it:

```
{\strike Hot Spot}{\v Topic}
```

See Also

\ul, \uldb, \v

\tab

\tab

The \tab statement inserts a tab character (ASCII character code 9).

Comments

The tab character (ASCII character 9) has the same effect as the \tab statement.

See Also

\tb, \tqc, \tqr, \tx

\tb

\tb

The \tb statement advances to the next tab stop.

See Also

\tab, \tqc, \tqr, \tx

\tqc

\tqc

The \tqc statement advances to the next tab stop and centers text.

See Also

\tab, \tb, \tqr, \tx

\tqr

\tqr

The \tqr statement advances to the next tab stop and aligns text to the right.

See Also

\tab, \tb, \tqc, \tx

\trgaph

\trgaphn

The \trgaph statement specifies the amount of space between text in adjacent cells in a table. For each cell in the table, Windows Help uses the space to calculate the cells left and right margins. It then uses the margins to align and wrap the text in the cell. Windows Help applies the same margin widths to each cell ensuring that paragraphs in adjacent cells have the specified space between them.

The \trgaph statement applies to cells in all subsequent rows of a table up to the next \trowd statement.

Parameters

n	Specifies the space, in twips, between text in adjacent cells. If this parameter exceeds the actual width of the cell, the left and right margins are assumed to be at the same position in the cell.

Comments

The width of the left margin in the first cell is always equal to the space specified by this statement. The \trleft statement is typically used to move the left margin to a position similar to the left margins in all other cells.

This statement was first supported in the Microsoft Help compiler version 3.1.

Example

The following example creates a three-column table with one-quarter inch space between the text in the columns:

```
\trgaph360 \cellx1440\cellx2880\cellx4320
\intbl
Row 1 Column 1\cell
Row 1 Column 2\cell
Row 1 Column 3\cell \row
\intbl
Row 2 Column 1\cell
Row 2 Column 2\cell
Row 2 Column 3\cell \row \pard
```

See Also

\cell, \cellx, \intbl, \row, \trleft, \trowd

\trleft

\trleftn

The \trleft statement sets the position of the left margin for the first (leftmost) cell in a row of a table. This statement applies to the first cell in all subsequent rows of the table up to the next \trowd statement.

Parameters

n	Specifies the relative position, in twips, of the left margin. This parameter can be a positive or negative number. The final position of the left margin is the sum of the current position and this value.

Comments

This statement was first supported in the Microsoft Help compiler version 3.1.

Example

The following example creates a three-column table with one-quarter inch space between the text in the columns. The left margin in the first cell is flush with the left margin of the Help window:

```
\trgaph360\trleft-360 \cellx1440\cellx2880\cellx4320
\intbl
Row 1 Column 1\cell
Row 1 Column 2\cell
Row 1 Column 3\cell \row
\intbl
Row 2 Column 1\cell
Row 2 Column 2\cell
Row 2 Column 3\cell \row \pard
```

See Also

\cell, \cellx, \intbl, \row, \trgaph, \trowd

\trowd

\trowd

The \trowd statement sets default margins and cell positions for subsequent rows in a table.

Comments

This statement was first supported in the Microsoft Help compiler version 3.1.

See Also

\cell, \cellx, \intbl, \row, \trgaph, \trleft

\trqc

\trqc

The \trqc statement directs Windows Help to dynamically adjust the width of table columns to fit in the current window.

Comments

In a print-based document, the \trqc statement centers a table row with respect to its containing column.

This statement was first supported in the Microsoft Help compiler version 3.1.

See Also

\trowd, \trql

\trql

\trql

The \trql statement aligns the text in each cell of a table row to the left.

Comments

This statement was first supported in the Microsoft Help compiler version 3.1.

See Also

\trowd, \trqc

\tx

\txn

The \tx statement sets the position of a tab stop. The position is relative to the left margin of the Help window. A tab stop applies to all subsequent paragraphs up to the next \pard statement.

Parameters

n Specifies the tab stop position, in twips.

Comments

If the \tx statement is not given, tab stops are set at every one-half inch by default.

See Also

\tab, \tb, \tqc, \tqr

\ul

\ul

The \ul statement creates a link to a pop-up topic. The statement is used in conjunction with a \v statement to create a link to another topic. When the user chooses the link, Windows Help displays the associated topic in a pop-up window.

The \ul statement applies to all subsequent text up to the next \plain or \ul0 statement.

Comments

No \plain or \ul0 statement is required if the \ul statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to just the enclosed text.

In print-based documents, or whenever it is not followed by \v, the \ul statement creates continuous underline.

Example

The following example creates a pop-up link for a topic. When displayed, the link text, Pop-up Link, is green and has a dotted line under it:

```
{\ul Pop-up Link}{\v PopupTopic}
```

See Also

\strike, \uldb, \v

\uldb

\uldb

The \uldb statement creates a hot spot. This statement is used in conjunction with a \v statement to create a link to another topic. When the user chooses a hot spot, Windows Help displays the associated topic in the Help window.

The \uldb statement applies to all subsequent text up to the next \plain or \uldb0 statement.

Comments

No \plain or \uldb0 statement is required if the \uldb statement and subsequent text are enclosed in braces. Braces limit the scope of a character property statement to just the enclosed text.

In print-based documents, or whenever it is not followed by \v, the \uldb statement creates double underline.

Example

The following example creates a hot spot for a topic. When displayed, the hot-spot text, Hot Spot, is green and has a solid line under it:

```
{\uldb Hot Spot}{\v Topic}
```

See Also

\strike, \ul, \v

\v

{\v context-string}

The \v statement creates a link to the topic having the specified context string. The \v statement is used in conjunction with the \strike, \ul, and \uldb statements to create hot spots and links to topics.

Parameters

context-string Specifies the context string of a topic in the Help file. The string can be any combination of characters, except spaces, and must also be specified in a context-string \footnote statement in some topic in the Help file.

Comments

If the context string is preceded by a percent sign (%), Windows Help displays the associated hot spot or link without applying the standard underline and color.

If the context string is preceded by an asterisk (*), Windows Help displays the associated hot spot or link with an underline but without applying the standard color.

In print-based documents, the \v statement creates hidden text.

For links or hot spots, the syntax of the \v statement is as follows:

```
[%|*]context[>secondary-window][@filename]
```

In this syntax, secondary-window is the name of the secondary window to jump to. When the secondary window is not specified, the jump is to the same window as the current Help topic is using. To jump to the main Help window, specify main for this parameter. This parameter may not be used with pop-up windows.

The filename parameter specifies a jump to a topic in a different Help file.

For a macro hot spot, the syntax of the \v statement is as follows:

```
[%|*]!macro[;macro][;...]
```

Example

The following example creates a hot spot for the topic having the context string Topic. Windows Help applies an underline and the color green to the text Hot Spot when it displays the topic:

```
{\uldb Hot Spot}{\v Topic}
```

See Also

\footnote, \strike, \ul, \uldb

\wbitmap

\wbitmapn

The \wbitmap statement sets the picture type to Windows bitmap. This statement must be used in conjunction with a \pict statement.

Parameters

n Specifies the bitmap type. This parameter is zero for a logical bitmap.

Comments

The \wbitmap statement is optional; if a \wmetafile statement is not specified, the picture is assumed to be a Windows bitmap.

Example

The following example creates a 32-by-8-pixel monochrome bitmap:

```
{
\pict \wbitmap0\wbmbitspixel1\wbmplanes1\wbmwidthbytes4\picw32\pich8
3FFFFFFC
F3FFFFCF
FF3FFCFF
FFF3CFFF
FFFC3FFF
FFCFF3FF
FCFFFF3F
CFFFFFF3
}
```

See Also

bmc, bml, bmr, \pict, \wmetafile

\wbmbitspixel

\wbmbitspixeln

The \wbmbitspixel statement specifies the number of consecutive bits in the bitmap data that represent a single pixel. This statement must be used in conjunction with the \pict statement.

Parameters

n Specifies the number of bits per pixel.

Comments

If the \wbmbitspixel statement is not given, the default bits per pixel value is 1.

See Also

\pict, \wbitmap, \wbmplanes

\wbmplanes

\wbmplanesn

The \wbmplanes statement specifies the number of color planes in the bitmap data. This statement must be used in conjunction with a \pict statement.

Parameters

n Specifies the number of bitmap planes.

Comments

If the \wbmplanes statement is not given, the default number of planes is 1.

See Also

\pict, \wbitmap, \wbmbitspixel

\wbmwidthbytes

\wbmwidthbytesn

The \wbmwidthbytes statement specifies the number of bytes in each scan line of the bitmap data. This statement must be used in conjunction with the \pict statement.

Parameters

n Specifies the width of the bitmap, in bytes.

See Also

\pict, \wbitmap

\windows

\windows

The \windows statement sets the Windows character set.

Comments

If no \windows, \pc, or \pca statement is given in the Help file, the Windows character set is used by default.

See Also

\ansi, \pc, \pca

\wmetafile

\wmetafilen

The \wmetafile statement sets the picture type to a Windows metafile. This statement must be used in conjunction with the \pict statement.

Parameters

n Specifies the metafile type. This parameter must be 8.

Comments

Windows Help expects the hexadecimal data associated with the picture to represent a valid Windows metafile. By default, Windows Help sets the MM_ANISOTROPIC mapping mode prior to displaying the metafile. To ensure that the picture is displayed correctly, the metafile data must either set the window origin and extents by using the SetWindowOrg and SetWindowExt records or set another mapping mode by using the SetMapMode record.

Example

The following example creates a picture using a metafile:

```
{{\pict\wmetafile8\picw2880\pich2880
0100090000034f0000000200090000000000
050000000b0200000000050000000c026400
6400090000001d066200ff00640064000000
000008000000fa020000020000000000000000
040000002d0100000500000014020000000
0500000013026400640005000000014020000
64000500000013026400000008000000fa02
000000000000000000000040000002d010100
04000000f00100000300000000004e0dff00
870020000050000020000000000000000000}
\par }
```

See Also

bmc, bml, bmr, \pict, \wbitmap

Appendix C Baggage Access Functions

To access the data from the Help files internal file system, Windows Help provides specialized source code. This source code can be built into an application or custom DLL so that it can retrieve the appropriate data files listed in the Help files [BAGGAGE] section.

Function Calls

Use the following function calls to access the Help file system:

OpenClose

Use the following function calls to access baggage files in the Help file system:

OpenCloseReadTellSeekCreateErrorWErrorPstrLGetInfoFAPILLFileRCLLInfoFromHFRCLLInfoFrom HFS

The include file for these function calls is provided in the following section.

The Baggage Include File

Use the following include file, IMPDLL.H, to enable your application or DLL to retrieve baggage files from the Help file system.

```
/
*************************************************************************\
*
*  IMPDLL.H
*
*  Copyright (C) Microsoft Corporation 1990.
*  All rights reserved.
*
*************************************************************************
*
*
*  Program Description:Import routine export header for DLLs
*
*************************************************************************
**
*
*  Revision History:Created 10/22/90 by Robert Bunney
*
*
*************************************************************************
**
*
*  Known Bugs:None
*
*
\
*************************************************************************
/
/
*************************************************************************
\
*
*  Defines
*
\
*************************************************************************
/
/* magic number and version number */
/* file mode flags */
#define fFSReadOnly        (BYTE)0x01  /* file (FS) is readonly */
#define fFSOpenReadOnly    (BYTE)0x02  /* file (FS) is opened in readonly
mode */
```

```
#define fFSReadWrite       (BYTE)0x00  /* file (FS) is readwrite */
#define fFSOpenReadWrite   (BYTE)0x00  /* file (FS) is opened in read/write
mode */
/* flags for FlushHfs */
#define fFSCloseFile       (BYTE)0x01
#define fFSFreeBtreeCache  (BYTE)0x02
/* seek origins */
#define wFSSeekSet         0
#define wFSSeekCur         1
#define wFSSeekEnd         2
/* low level info options */
#define wLLSameFid         0
#define wLLDupFid          1
#define wLLNewFid          2
/* Exported functions      */
#define HE_Count           26    /* count of exported functions */
#define HE_Documented      17    /* number documented for Help 3.1 */
#define HE_NotUsed         0
#define HE_HfsOpenSz       1
#define HE_RcCloseHfs      2
#define HE_HfOpenHfs       3
#define HE_RcCloseHf       4
#define HE_LcbReadHf       5
#define HE_LTellHf         6
#define HE_LSeekHf         7
#define HE_FEofHf          8
#define HE_LcbSizeHf       9
#define HE_FAccessHfs      10
#define HE_RcLLInfoFromHf  11
#define HE_RcLLInfoFromHfs 12
#define HE_ErrorW          13
#define HE_ErrorSz         14
#define HE_GetInfo         15
#define HE_API             16
/* Return codes           */
#define rcSuccess          0
#define rcFailure          1
#define rcExists           2
#define rcNoExists         3
#define rcInvalid          4
#define rcBadHandle        5
#define rcBadArg           6
#define rcUnimplemented    7
```

```
#define rcOutOfMemory      8
#define rcNoPermission     9
#define rcBadVersion       10
#define rcDiskFull         11
#define rcInternal         12
#define rcNoFileHandles    13
#define rcFileChange       14
#define rcTooBig           15
/**
* Errors for Error()
**/
/* Errors to generate      */
#define wERRS_OOM          2  /* Out of memory
                              */
#define wERRS_NOHELPPS     3  /* No Help during printer setup  */
#define wERRS_NOHELPPR     4  /* No Help while printing              */
#define wERRS_FNF          1001  /* Cannot find file
                                 */
#define wERRS_NOTOPIC      1002  /* Topic does not exist             */
#define wERRS_NOPRINTER    1003  /* Cannot print
                                 */
#define wERRS_PRINT        1004
#define wERRS_EXPORT       1005  /* Cannot copy to clipboard         */
#define wERRS_BADFILE      1006
#define wERRS_OLDFILE      1007
#define wERRS_Virus        1011  /* Bad .EXE
                                 */
#define wERRS_BADDRIVE     1012  /* Invalid drive
                                 */
#define wERRS_WINCLASS     1014  /* Bad window class
                              */
#define wERRS_BADKEYWORD   3012  /* Invalid keyword
                              */
#define wERRS_BADPATHSPEC  3015  /* Invalid path specification       */
#define wERRS_PATHNOTFOUND 3017  /* Path not found
                              */
#define wERRS_DIALOGBOXOOM 3018  /* Insufficient memory for dialog  */
#define wERRS_DiskFull     5001  /* Disk is full
                                 */
#define wERRS_FSReadWrite  5002  /* File read/write failure
                              */
/**
* Actions for LGetInfo()
**/
#define GI_NOTHING         0  /* Not used
```

```
                                */
#define GI_INSTANCE         1  /* Application instance handle
                               */
#define GI_MAINHWND         2  /* Main window handle
                               */
#define GI_CURRHWND         3  /* Current window handle
                                  */
#define GI_HFS              4  /* Handle to file system in use
                               */
#define GI_FGCOLOR          5  /* Foreground color used by application*/
#define GI_BKCOLOR          6  /* Background color used by application*/
#define GI_TOPICNO          7  /* Topic number
                                  */
#define GI_HPATH            8  /* Handle containing path -- caller
                               */
                                /*  must free           */
/
*****************************************************************************
\
*
*     Types
*
\
*****************************************************************************
/
typedef WORD RC;     /* Error return (return code) */
typedef HANDLE HFS;  /* Handle to a file system          */
typedef HANDLE HF;   /* Handle to a file system file     */
/
*****************************************************************************
\
*
*     Prototypes
*
\
*****************************************************************************
/
VOID FAR PASCAL SetCallbacks(FARPROC FAR *);
/
*****************************************************************************
\
*
*     Public Functions pointers
*
\
*****************************************************************************
/
```

```
/
*****************************************************************************
\
*
* Function:RcGetFSError()
*
* Purpose:return the most recent FS error code
*
* Method:give value of last error that the file system encountered
*
* ASSUMES
*
*  globals IN:rcFSError - current error code; set by most recent FS call
*
* PROMISES
*
*  returns:returns current error in file system
*
\
*****************************************************************************
/
extern RC  (FAR PASCAL *RcGetFSError)(void);
/
*****************************************************************************
\
*
* Function:HfsOpenSz( sz, bFlags )
*
* Purpose:open a file system
*
* ASSUMES
*
*  args IN:sz - path to file system to open
*         bFlags - fFSOpenReadOnly or fFSOpenReadWrite
*
* PROMISES
*
*  returns:handle to file system if opened OK, else hNil
*
\
*****************************************************************************
/
extern HFS (FAR PASCAL *HfsOpenSz)( LPSTR, BYTE );
/
*****************************************************************************
```

```
\
*
* Function:RcCloseHfs( hfs )
*
* Purpose:Close an open file system
*  All files must be closed or changes made will be lost
*
* ASSUMES
*
*  args IN:hfs - handle to an open file system
*
* PROMISES
*
*  returns:standard return code
*
*  globals OUT:rcFSError
*
\
*****************************************************************************
/
extern HFS (FAR PASCAL *RcCloseHfs)( HFS );
/
*****************************************************************************
\
*
* Function:HfOpenHfs( hfs, sz, bFlags )
*
* Purpose:open a file in a file system
*
* ASSUMES
*
*  args IN:hfs - handle to file system
*        sz - name (key) of file to open
*        bFlags -
*
* PROMISES
*
*  returns:handle to open file or hNil on failure
*
* Notes:strlen( qNil ) and strcpy( s, qNil ) don't work as they should
*
\
*****************************************************************************
/
```

```
extern HF  (FAR PASCAL *HfOpenHfs) ( HFS,  LPSTR,  BYTE );
/
*****************************************************************************
\
*
* Function:RcCloseHf( hf )
*
* Purpose:close an open file in a file system
*
* Method:If the file is dirty, copy the scratch file back to the FS file.
*        If this is the first time the file has been closed, enter the
*        name into the FS directory. If this file is the FS directory,
*        store the location in a special place instead. Write the FS
*        directory and header to disk. Do other various hairy stuff.
*
* ASSUMES
*
*  args IN:hf - file handle
*
* PROMISES
*
*  returns:rcSuccess on successful closing
*
\
*****************************************************************************
/
extern RC  (FAR PASCAL *RcCloseHf) ( HF      );
/
*****************************************************************************
\
*
* Function:LcbReadHf()
*
* Purpose:read bytes from a file in a file system
*
* ASSUMES
*
*  args IN:hf - file
*        lcb - number of bytes to read
*
* PROMISES
*
*  returns:number of bytes actually read; -1 on error
*
```

```
*  args OUT:qb - data read from file goes here (must be big enough)
*
*  Notes:These are signed longs we're dealing with. This means
*        behavior is different from read() when < 0.
*
\
*****************************************************************************
/
extern LONG (FAR PASCAL *LcbReadHf     ) ( HF,  LPSTR, LONG );
/
*****************************************************************************
\
*
* Function:LcbWriteHf( hf, qb, lcb )
*
* Purpose:write the contents of buffer into file
*
* Method:If file isn't already dirty, copy data into temporary file.
*        Do the write.
*
* ASSUMES
*
*  args IN:hf - file
*        qb - user's buffer full of stuff to write
*        lcb - number of bytes of qb to write
*
* PROMISES
*
*  returns:number of bytes written if successful, -1L if not
*
*  args OUT:hf - lifCurrent, lcbFile updated; dirty flag set
*
*  globals OUT:rcFSError
*
\
*****************************************************************************
/
extern LONG (FAR PASCAL *LcbWriteHf    ) ( HF,  LPSTR, LONG );
/
*****************************************************************************
\
*
* Function:LTellHf( hf )
*
* Purpose:return current file position
```

```
*
* ASSUMES
*
*  args IN:hf - handle to open file
*
* PROMISES
*
*  returns:file position
*
\
*****************************************************************************
/
extern LONG (FAR PASCAL *LTellHf ) ( HF      );
/
*****************************************************************************
\
*
* Function:LSeekHf( hf, lOffset, wOrigin )
*
* Purpose:set current file pointer
*
* ASSUMES
*
*  args IN:hf- file
*        lOffset - offset from origin
*        wOrigin - origin (wSeekSet, wSeekCur, or wSeekEnd)
*
* PROMISES
*
*  returns:new position offset in bytes from beginning of file if
*        successful, or -1L if not
*
*  state OUT:file pointer is set to new position unless error occurs,
*           in which case it stays where it was
*
\
*****************************************************************************
/
extern LONG (FAR PASCAL *LSeekHf ) ( HF,  LONG, WORD );
/
*****************************************************************************
\
*
* Function:FEofHf()
*
```

```
* Purpose:tell whether file pointer is at end of file
*
* ASSUMES
*
*  args IN:hf
*
* PROMISES
*
*  returns:fTrue if file pointer is at EOF, fFalse otherwise
*
\
*****************************************************************************
/
extern BOOL (FAR PASCAL *FEofHf  ) ( HF      );
/
*****************************************************************************
\
*
* Function:LcbSizeHf( hf )
*
* Purpose:return the size in bytes of specified file
*
* ASSUMES
*
*  args IN:hf - file handle
*
* PROMISES
*
*  returns:size of the file in bytes
*
\
*****************************************************************************
/
extern LONG (FAR PASCAL *LcbSizeHf     ) ( HF      );
/
*****************************************************************************
\
*
* Function:FAccessHfs( hfs, sz, bFlags )
*
* Purpose:determine existence or legal access to an FS file
*
* ASSUMES
*
*  args IN:hfs
```

```
*        sz- file name
*        bFlags - ignored
*
* PROMISES
*
*  returns:fTrue if file exists (is accessible in stated mode),
*        fFalse otherwise
*
* Bugs:access mode part is unimplemented
*
\
*****************************************************************************
/
extern BOOL (FAR PASCAL *FAccessHfs    ) ( HFS,  LPSTR, BYTE );
/*******************
 -
 - Name:ErrorW
 *
 * Purpose:displays an error message
 *
 * Arguments:nError - string identifier. See wERRS_* messages.
 *
 * Returns:nothing
 *
 ******************/
extern VOID (FAR PASCAL *ErrorW  ) ( int );
/
*****************************************************************************
 *
 - Name:ErrorSz
 -
 * Purpose:displays standard Windows Help error message dialog based on
 *       the string passed
 *
 * Arguments:lpstr - string to display
 *
 * Returns:nothing
 *

*****************************************************************************
/
extern VOID (FAR PASCAL *ErrorSz ) ( LPSTR );
/
*****************************************************************************
```

```
 *
 - Name:LGetInfo
 -
 * Purpose:gets global information from the application
 *
 * Arguments:hwnd - window handle of topic to query.
 *          wItem - item to get
 *          GI_INSTANCE - application instance handle
 *          GI_MAINHWND - main window handle
 *          GI_CURRHWND - current window handle
 *          GI_HFS - handle to file system in use
 *          GI_FGCOLOR - foreground color used by application
 *          GI_BKCOLOR - background color used by application
 *          GI_TOPICNO - topic number
 *          GI_HPATH - handle containing path -- caller must free
 *
 *
 * Notes:if the HWND is NULL, then the data will come from the window
 *    that currently has the focus
 *

*************************************************************************
/
extern LONG (FAR PASCAL *LGetInfo) ( WORD, HWND    );
/*******************
 -
 - Name:FAPI
 *
 * Purpose:post a message for Help requests
 *
 * Arguments:qchHelp - path (if not current directory) and file to use for
Help topic
 *          usCommand - command to send to Help
 *          ulData - data associated with command
 *
 *
 * Returns:TRUE iff success
 *
 * Notes:see the Chapter 19 entry for the WinHelp API
 *
 ******************/
extern LONG (FAR PASCAL *FAPI    ) ( LPSTR, WORD, DWORD );
/
*************************************************************************
```

```
\
*
- Function:RcLLInfoFromHf( hf, wOption, qfid, qlBase, qlcb )
-
* Purpose:map an HF into low-level file info
*
* ASSUMES
*  args IN:hf - an open HF
*        qfid, qlBase, qlcb - pointers to user's variables
*        wOption - wLLSameFid, wLLDupFid, or wLLNewFid
*
* PROMISES
*  returns:RcFSError(); rcSuccess on success
*
*  args OUT:qfid - depending on value of wOption, either the same fid
*              used by hf, a dup() of this fid, or a new fid obtained
*              by reopening the file
*
*           qlBase - byte offset of first byte in the file
*           qlcb - size in bytes of the data in the file
*
*  globals OUT:rcFSError
*
* Notes:It is possible to read data outside the range specified by *qlBase
*     and *qlcb. Nothing is guaranteed about what will be found there.
*     If wOption is wLLSameFid or wLLDupFid, and the FS is opened in
*     opened in write mode, this fid will be writable. However, writing
*     is not allowed and may destroy the file system.
*
*     Fids obtained with the options wLLSameFid and wLLDupFid share
*     a file pointer with the hfs. This file pointer may change after any
*     operation on this FS. The fid obtained with the option wLLSameFid
*     may be closed by FS operations. If it is, the fid is invalid.
*
*     NULL can be passed for qfid, qlbase, qlcb and this routine will not
*     pass back the information.
*
* Bugs:wLLDupFid is unimplemented
*
* +++
*
* Method:
*
```

```
* Notes:
*
\
*****************************************************************************
/
extern RC  (FAR PASCAL *RcLLInfoFromHf  ) ( HF,    WORD, int FAR *, LONG
FAR *, LONG FAR * );
/
*****************************************************************************
\
*
- Function:RcLLInfoFromHfsSz( hfs, sz, wOption, qfid, qlBase, qlcb )
-
* Purpose:map an HF into low-level file info
*
* ASSUMES
*  args IN:hfs - an open HFS
*        szName - name of file in FS
*        qfid, qlBase, qlcb - pointers to user's variables
*        wOption - wLLSameFid, wLLDupFid, or wLLNewFid
*
* PROMISES
*  returns:RcFSError(); rcSuccess on success
*
*  args OUT:qfid  - depending on value of wOption, either the same fid
*              used by hf, a dup() of this fid, or a new fid obtained by
*              reopening the file
*
*           qlBase - byte offset of first byte in the file
*           qlcb - size in bytes of the data in the file
*
*  globals OUT:rcFSError
*
* Notes:It is possible to read data outside the range specified by *qlBase
*     and *qlcb. Nothing is guaranteed about what will be found. If wOption
*     is wLLSameFid or wLLDupFid, and the FS is opened in write mode,
*     this fid will be writable. However, writing is not allowed and may
*     destroy the file system.
*
*     Fids obtained with the options wLLSameFid and wLLDupFid share
*     a file pointer with the hfs. This file pointer may change after any
*     operation on this FS. The fid obtained with the option wLLSameFid
*     may be closed by FS operations. If it is, your fid is invalid.
*
```

```
*     NULL can be passed for qfid, qlbase, qlcb and this routine will not
*     pass back the information.
*
* Bugs:wLLDupFid is unimplemented
*
* Method:calls RcLLInfoFromHf()
*
* Notes:
*
\
*****************************************************************************
/
extern RC  (FAR PASCAL *RcLLInfoFromHfs ) (HFS, LPSTR, WORD, int FAR *,
LONG FAR *, LONG FAR * );
```