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 don't 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.

Graphic

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 item's 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 spots–the 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.

Graphic

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 topic's 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 don’t 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 parameter–a 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 correct–matching parentheses, matching quotation marks, or missing commas
- If the macro has the correct number of parameters
- If the parameter type matches the specified type–numeric 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.

If you want to add a certain feature to your Help file that Windows Help does not

Using DLL Calls as Help Macros

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, you'll 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.

GRAPHIC

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 don't 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 user's system. When preparing to use a .DLL or .EXE file, Help looks in the following locations:

- Help's current directory
- The MS-DOS current directory
- The user's Windows directory
- The Windows SYSTEM directory
- The directory containing WINHELP.EXE
- The directories listed in the user's PATH environment variable
- 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 user's 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 user's 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

Macro Quick Reference

Chapter 20, “Writing DLLs for Windows Help.”

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.