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.

Defining Menus and Buttons

This chapter explains how to customize your Help file.

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, it’s 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 won't 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 don't 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.

Graphic

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	None[1]	None[1]
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	None[1]	None[1]
Edit Copy	mnu_copy	**CopyDialog**	Displays the Copy dialog box.
Edit Annotate	mnu_annotate	**Annotate**	Displays the Annotate dialog box.
Bookmark	mnu_bookmark	None[1]	None[1]
Bookmark Define	mnu_bkdefine	**BookmarkDefine**	Displays the Bookmark Define dialog box.
Bookmark list	None[2]	None[2]	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	None[1]	None[1]

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 don't 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, you'd 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).

Graphic

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 you’ve 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');
```


Defining Buttons

```
ChangeItemBinding("item_showindex", "JumpID(`index.hlp', `sub_cont')")'), 0)
```

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 don’t 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.:

Graphic

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 can't 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:

Graphic

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:

{bmc CALENDAR.BMP}**!ExecProgram("calendar.exe", 0)**

The application appears on top of the Help window, as shown in Figure 13.5.

Graphic

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.

Graphic

For more information abuot project file options, see Chapter 16, “The Help Project File.”

Creating a Custom Icon for Your Help

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.

Graphic

For more information abuot project file options, see Chapter 16, “The Help Project File.”

Creating a Custom How To Use Help

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 you're 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 doesn't overwrite the default Help file when you install your product on a user's 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
```