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

Macro Reference

constructing and using macros, see Chapter 14, “Help Macros.”

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

Syntax

About command on the Help menu.

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

Syntax

key(s).

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.”

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

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 (;).

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:

Comments

```
AddAccelerator(0x79, 6, "JumpID(`index.hlp', `cont_idx')")
```

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 macro’s “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

Syntax

the Annotate command on the Edit menu.

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

Syntax

Appends a menu item to the end of a menu created with the **InsertMenu** macro.

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 macro's 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

Comments

accelerator key for this menu item.

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.

See Also

Windows Help ignores this macro if it is executed in a secondary window.

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

Syntax

Windows Help.)

Back()

Parameter	**Description**
none	

Comments

If the **Back** macro is executed when the Back list is empty, Windows Help takes no action.

See Also

Windows Help ignores this macro if it is executed in a secondary window.

History

BookmarkDefine

Displays the Bookmark Define dialog box. Executing this macro is the same as

Syntax

choosing the Define command on the Bookmark menu.

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

Syntax

defines more than nine bookmarks.

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

Syntax

Adds browse buttons (<< and >>) to the button bar in Windows Help.

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 it’s used, the **BrowseButtons** macro may interfere with the **DisableButton** macro. See **DisableButton** for details.

See Also

Windows Help ignores this macro if it is executed in a secondary window.

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

Syntax

the **CreateButton** macro.

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.

Parameter	Description
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:

Comments

ChangeButtonBinding("btn_contents", "JumpId(`dict.hlp', `dict_contents')")

See Also

Windows Help ignores this macro if it is executed in a secondary window.

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

Syntax

(and only one) standard Help menu item: How To Use Help.

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:

Comments

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

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.

See Also

Windows Help ignores this macro if it is executed in a secondary window.

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

Syntax

command: on or off, show or hide, and so on.

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.

See Also

Windows Help ignores this macro if it is executed in a secondary window.

AppendItem, **ChangeItemBinding**, **DeleteItem**, **DisableItem**, **EnableItem**, **InsertItem**, **InsertMenu**, **UncheckItem**

CloseWindow

Syntax

Closes either the main Help window or a secondary window.

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:

Comments

```
CloseWindow("index")
```

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

Syntax

of the Help project file.

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.

See Also

Windows Help ignores this macro if it is executed in a secondary window.

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

Syntax

this macro is the same as choosing the Copy command on the Edit menu.

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

Syntax

this macro is the same as pressing CTRL+INS in the main Help window.

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)

Syntax

Creates a new button and adds it to the Windows Help button bar.

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 button's accelerator key. The button name is case sensitive and can have as many as 29 characters–after 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')")
```

Comments

Notice that the letter *I* serves as the button's accelerator key.

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()")
```

See Also

Windows Help ignores this macro if it is executed in a secondary window.

BrowseButtons, **ChangeButtonBinding**, **DestroyButton**, **DisableButton**, **EnableButton**

DeleteItem

Syntax

Removes a menu item added with the **AppendItem** macro.

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:

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

Syntax

Removes a text marker added with the **SaveMark** macro.

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:

Comments

DeleteMark("Managing Memory")

If the marker does not exist when the **DeleteMark** macro is executed, Windows

See Also

Help displays a “Topic not found” error message.

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:

Comments

DestroyButton("btn_ideas")

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.)

See Also

Windows Help ignores this macro if it is executed in a secondary window.

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:

Comments

```
DisableButton("btn_ideas")
```

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 user's 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.

See Also

Windows Help ignores this macro if it is executed in a secondary window.

BrowseButtons, **ChangeButtonBinding**, **CreateButton**, **DestroyButton**, **EnableButton**

DisableItem (or DI)

Syntax

Disables and greys out a menu item added with the **AppendItem** macro.

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:

Comments

DisableItem("mnu_example")

You cannot use this macro to disable a menu item in a topic until the item has been enabled using the **EnableItem** macro.

See Also

Windows Help ignores this macro if it is executed in a secondary window.

AppendItem, **ChangeItemBinding**, **CheckItem**, **DeleteItem**, **EnableItem**, **InsertItem**, **InsertMenu**, **UncheckItem**

EnableButton (or EB)

Syntax

Re-enables a button disabled with the **DisableButton** macro.

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:

Comments

EnableButton("btn_ideas")

If you use this macro to enable a standard Windows Help button (Contents, Search, Back, or History), the user's 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.

See Also

Windows Help ignores this macro if it is executed in a secondary window.

ChangeButtonBinding, **CreateButton**, **DestroyButton**, **DisableButton**

EnableItem (or EI)

Syntax

Re-enables a menu item disabled with the **DisableItem** macro.

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:

Comments

EnableItem("mnu_example")

See Also

Windows Help ignores this macro if it is executed in a secondary window.

AppendItem, **ChangeItemBinding**, **CheckItem**, **DeleteItem**, **DisableItem**, **InsertItem**, **InsertMenu**, **UncheckItem**

ExecProgram (or EP)

Syntax

Starts an application.

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:

Comments

```
ExecProgram("clock.exe", 0)
```

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 user’s 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 application's 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

Syntax

choosing the Exit command on the File menu.

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

Syntax

Open command on the File menu.

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:

Comments

FocusWindow("keys")

See Also

If the window does not exist, Windows Help ignores the macro.

CloseWindow, **PositionWindow**

GotoMark

Syntax

Jumps to a marker set with the **SaveMark** macro.

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:

Comments

GotoMark("Managing Memory")

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

See Also

message.

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

Syntax

menu.

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

See Also

custom version of How To Use Help.

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

Syntax

the Help menu.

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

Syntax

button.

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

Syntax

make the test. You can also use a DLL function as a condition for this macro.

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 don’t 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')",

See Also

"JumpContents(`trb.hlp')")

DeleteMark, **GotoMark**, **IfThen**, **IsMark**, **Not**, **SaveMark**

InsertItem

Inserts a menu item at a given position on an existing menu. The menu can be one

Syntax

you create with the **InsertMenu** macro or a standard Windows Help menu.

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 **Identifier**

File	mnu_file
Edit	mnu_edit
Bookmark	mnu_bookmark
Help	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 item's 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 item's

Comments

accelerator key.

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.

See Also

Windows Help ignores this macro if it is executed in a secondary window.

AppendItem, **ChangeItemBinding**, **CheckItem**, **DeleteItem**, **DisableItem**, **EnableItem**, **InsertMenu**, **UncheckItem**

InsertMenu

Syntax

Adds a new menu to the Windows Help menu bar.

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 menu’s 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

Comments

Bookmark and Help menus. The user presses ALT+u to open the menu.

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.

See Also

Windows Help ignores this macro if it is executed in a secondary window.

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:

Comments

IfThen(IsMark("Managing Memory"), "JI(`trb.hlp', `man_mem')")

See Also

Use the **Not** macro to reverse the results of the **IsMark** macro.

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

Syntax

project file.

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:

Comments

JumpContents("progman.hlp")

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.

See Also

Windows Help ignores this macro if it is executed in a secondary window.

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 file’s project file.

Example

The following macro jumps to the topic mapped to the “801” context ID number in the PIFEDIT.HLP file:

Comments

JumpContext("pifedit.hlp", 801)

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

See Also

“[MAP] Section” in Chapter 16, “The Help Project File.”)

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

Syntax

Help project file.

JumpHelpOn()

Parameter	Description
none	

Comments

If Windows Help cannot find the specified Help file, it displays an error message

See Also

and does not perform the jump.

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:

Comments

```
JumpId("progman.hlp", "groups_how_pm")
```

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

See Also

file and you want to save disk space).

JumpContext, **PopupId**

JumpKeyword (or JK)

Opens the indicated Help file, searches through the K keyword table, and

Syntax

displays the first topic containing the keyword specified in the macro.

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:

Comments

JumpKeyword("clock.hlp", "hands")

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”

See Also

message and displays the Contents topic of the destination Help file.

Search

Next

Displays the next topic in the browse sequence for the Help file. Executing this

Syntax

macro is the same as choosing the Browse next button (>>).

Next()

Parameter	**Description**
none	

Comments

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

See Also

Windows Help ignores this macro if it is executed in a secondary window.

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

Syntax

IfThenElse.

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:

See Also

```
IfThen(Not(IsMark("Managing Memory")), "JI(`trb.hlp', `exp_mem')")
```

DeleteMark, **GotoMark**, **IfThen**, **IfThenElse**, **IsMark**, **SaveMark**

PopupContext (or PC)

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

Syntax

entry in the [MAP] section of the Help project file identifies the context.

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 file's 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:

Comments

PopupContext("pifedit.hlp", 801)

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

Syntax

string (as defined in the # footnote) as the second parameter.

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:

Comments

PopupId("progman.hlp", "019_eg_pm")

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

See Also

context strings, see "Inserting Context Strings" in Chapter 6, "Creating Topics.")

JumpId

PositionWindow (or PW)

Sets the size and position of either the main Help window or a secondary

Syntax

window.

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 Help's 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 window’s 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 system’s 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:

Comments

```
PositionWindow(100, 100, 500, 500, 5, "Samples")
```

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

Syntax

this macro is the same as choosing the Browse previous button (<<).

Prev()

Parameter	**Description**
none	

Comments

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

See Also

Windows Help ignores this macro if it is executed in a secondary window.

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

Syntax

the Print Setup command on the File menu.

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 function’s 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

Example

Windows version 3.1 Software Development Kit.

The following DLL call registers a routine named PlayAudio in the DLL named HELPLIB.DLL:

Comments

RegisterRoutine("helplib", "PlayAudio", "SIU")

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 anywhere–even on a drive not on the user's machine–and 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

Syntax

Help macro.

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:

Comments

RemoveAccelerator(0x79, 6)

Windows Help does not display an error message if the author attempts to

See Also

remove an unassigned accelerator key.

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

Syntax

this location.

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:

Comments

SaveMark("Managing Memory")

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

See Also

recently entered marker.

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

See Also

Windows Help ignores this macro if it is executed in a secondary window.

JumpKeyword

SetContents

Syntax

Designates a specific topic as the Contents topic in the specified Help file.

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 file’s 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

Comments

topic mapped to 101.

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

See Also

numbers, see “[MAP] Section” in Chapter 16, “The Help Project File.”)

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

Syntax

the user chooses the How To Use Help command or presses F1 in Windows Help.

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:

Comments

```
[CONFIG]
SetHelpOnFile("quikhelp.hlp>main")
```

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

See Also

HelpOn macro will display the custom version of How To Use Help.

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

Syntax

command: on or off, show or hide, and so on.

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”:

Comments

```
UncheckItem("time_item")
```

To check a menu item, use the **CheckItem** macro.

See Also

Windows Help ignores this macro if it is executed in a secondary window.

AppendItem, **ChangeItemBinding**, **CheckItem**, **DeleteItem**, **DisableItem**, **EnableItem**, **InsertItem**, **InsertMenu**