Module 15: Context-Sensitive Help

Program examples compiled using Visual C++ 6.0 (MFC 6.0) compiler on Windows XP Pro machine with Service Pack 2. Topics and sub topics for this Tutorial are listed below:

Context-Sensitive Help
The Windows WinHelp Program
Rich Text Format
Writing a Simple Help File
An Improved Table of Contents
The Application Framework and WinHelp
Calling WinHelp
Using Search Strings
Calling WinHelp from the Application's Menu
Help Context Aliases
Determining the Help Context
F1 Help
Shift-F1 Help
Message Box Help: The `AfxMessageBox()` Function
Generic Help
A Help Example: No Programming Required
The MAKEHELP Process
Help Command Processing
F1 Processing
Shift-F1 Processing
A Help Command Processing Example: MYMFC22B
Header Requirements
`CStringView` Class
`CHexView` Class
Resource Requirements
Help File Requirements
Testing the MYMFC22B Application

Context-Sensitive Help

Help technology is in a transition phase at the moment. The **Hypertext Markup Language** (HTML) format seems to be replacing **rich text format** (RTF). You can see this in the new Visual C++ online documentation via the MSDN viewer, which uses a new HTML-based help system called HTML Help. Microsoft is developing tools for compiling and indexing HTML files that are not shipped with Visual C++ 6.0. In the meantime, Microsoft Foundation Class (MFC) Library version 6.0 application framework programs are set up to use the WinHelp help engine included with Microsoft Windows. That means you'll be writing RTF files and your programs will be using compiled HLP files.
This module first shows you how to construct and process a simple stand-alone help file that has a table of contents and lets the user jump between topics. Next you'll see how your MFC library program activates WinHelp with help context IDs derived from window and command IDs keyed to an AppWizard-generated help file. Finally you'll learn how to use the MFC library help message routing system to customize the help capability.

The Windows WinHelp Program

If you've used commercial Windows-based applications, you've probably marveled at their sophisticated help screens: graphics, hyperlinks, and popups abound. At some software firms, including Microsoft, help authoring has been elevated to a profession in its own right. This section won't turn you into a help expert, but you can get started by learning to prepare a simple no-frills help file.

Rich Text Format

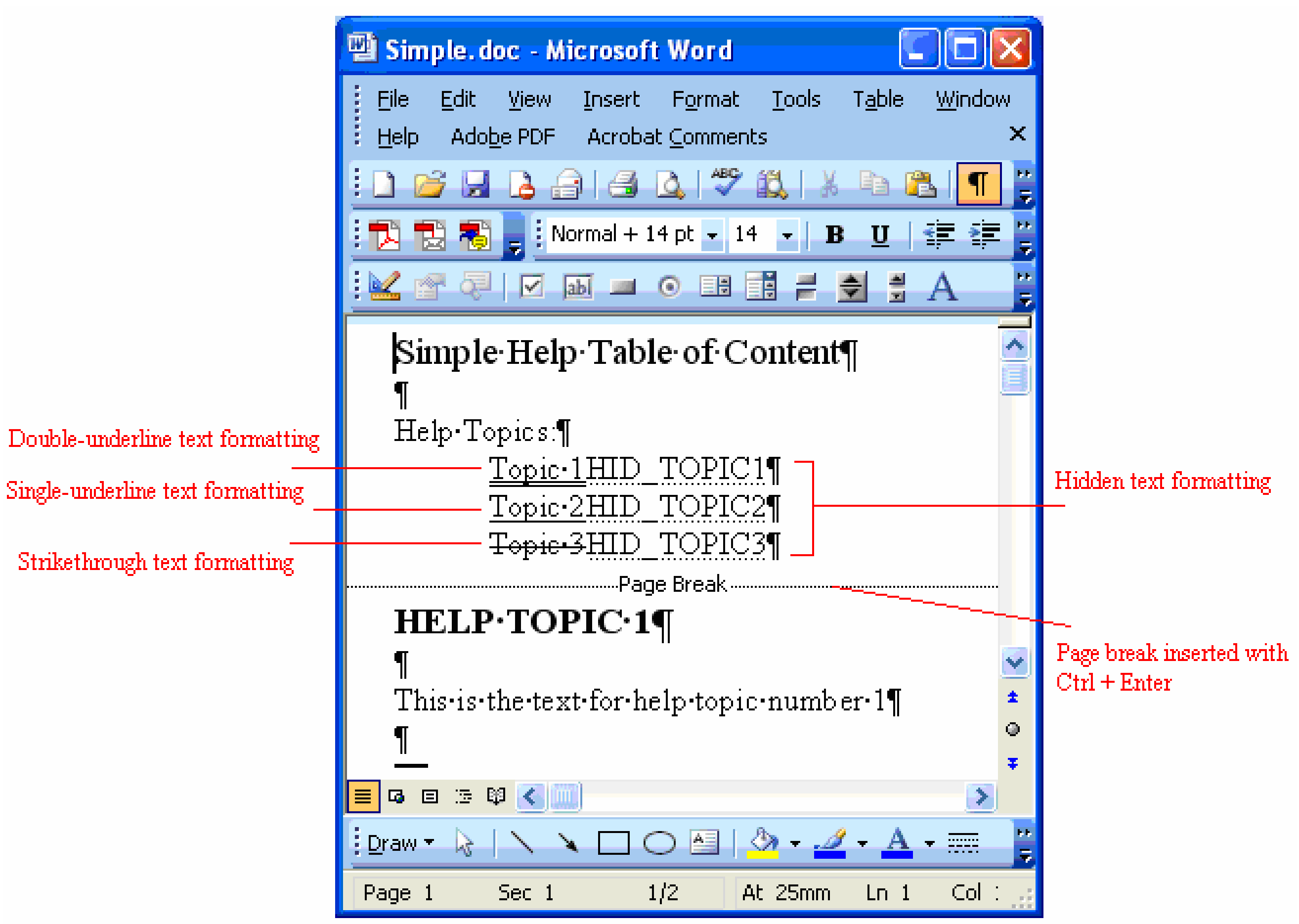
The original Windows SDK documentation showed you how to format help files with the ASCII file format called **rich text format**. We'll be using rich text format too, but we'll be working in WYSIWYG mode, thereby avoiding the direct use of awkward escape sequences. You'll write with the same fonts, sizes, and styles that your user sees on the help screens. You'll definitely need a word processor that handles RTF. I've used Microsoft Word for this book, but many other word processors accommodate the RTF format.
Several commercial Windows help tools are available, including **RoboHELP** from Blue Sky Software and **ForeHelp** from the Forefront Corporation. RoboHELP is a set of templates and macros for Microsoft Word, and ForeHelp is a stand-alone package that simulates **WinHelp**, giving you immediate feedback as you write the help system.

Writing a Simple Help File

We're going to write a simple help file with a table of contents and three topics. This help file is designed to be run directly from **WinHelp** and started from Windows. No C++ programming is involved. Here are the steps:

Create a \mfcproject \mymfc22 directory or any other directory that you have designated for your project.

Write the main help text file. Use Microsoft Word (or another RTF-compatible word processor) to type text as shown here. In this example it is Microsoft Word 2003.


Simple.doc - Microsoft Word
File Edit View Insert Format Tools Table Window
Help Adobe PDF Acrobat Comments
Normal + 14 pt
14
Double-underline text formatting
Single-underline text formatting
Strikethrough text formatting
Hidden text formatting
Page break inserted with Ctrl + Enter
Page Break
Draw
Page 1 Sec 1 1/2 At 25mm Ln 1 Col

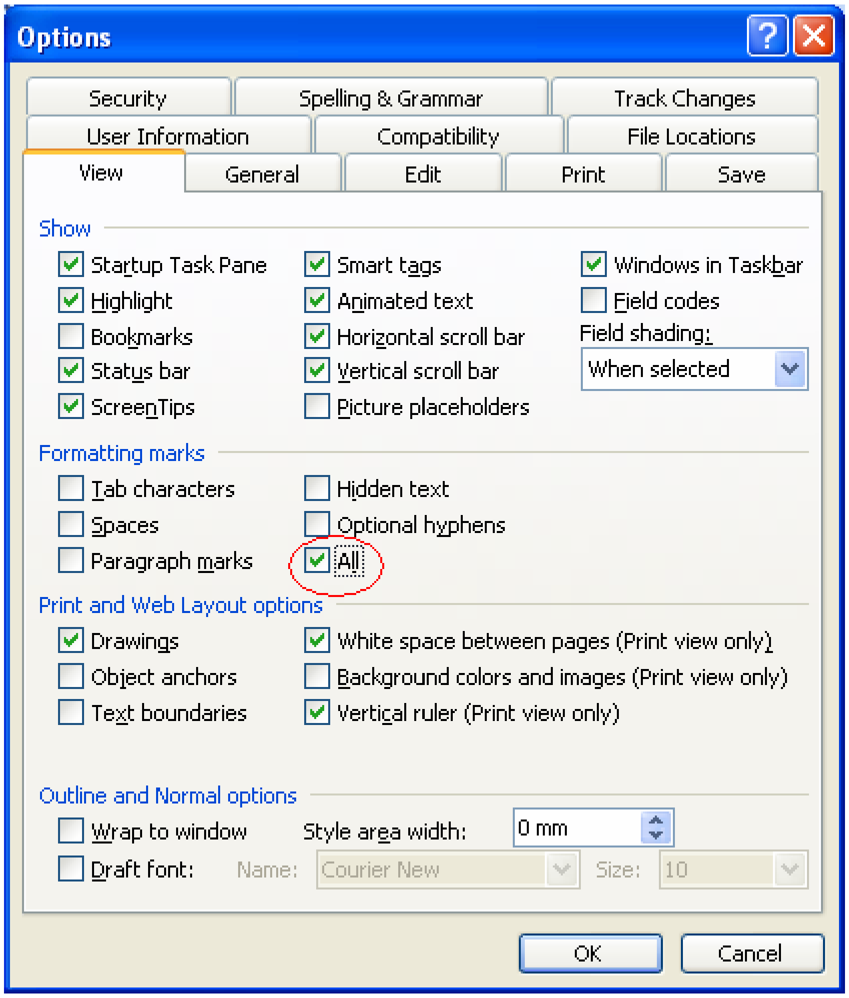
Simple·Help·Table·of·Content¶
¶
Help·Topics:¶
Topic·1HID_TOPIC1¶
Topic·2HID_TOPIC2¶
~~Topic·3~~HID_TOPIC3¶

HELP·TOPIC·1¶
¶
This·is·the·text·for·help·topic·number·1¶
¶

Figure 1: Main help text file saved in RTF format.

Be sure to apply the **double-underline** and **hidden text formatting** correctly and to **insert the page break** at the correct place.

To see hidden text, you must turn on your word processor's hidden text viewing mode. In Word, choose **Options** from the **Tools** menu, click on the **View** tab, and select **All** in the **Formatting marks** (**Nonprinting Characters** – for older version) section or click the Show/Hide (¶)icon.



Show
- [✓] Startup Task Pane
- [✓] Highlight
- [ ] Bookmarks
- [✓] Status bar
- [✓] ScreenTips
- [✓] Smart tags
- [✓] Animated text
- [✓] Horizontal scroll bar
- [✓] Vertical scroll bar
- [ ] Picture placeholders
- [✓] Windows in Taskbar
- [ ] Field codes

Field shading: When selected

Formatting marks
- [ ] Tab characters
- [ ] Spaces
- [ ] Paragraph marks
- [ ] Hidden text
- [ ] Optional hyphens
- [✓] All

Print and Web Layout options
- [✓] Drawings
- [ ] Object anchors
- [ ] Text boundaries
- [✓] White space between pages (Print view only)
- [ ] Background colors and images (Print view only)
- [✓] Vertical ruler (Print view only)

Outline and Normal options
- [ ] Wrap to window
- [ ] Draft font: Name: Courier New Size: 10

Style area width: 0 mm

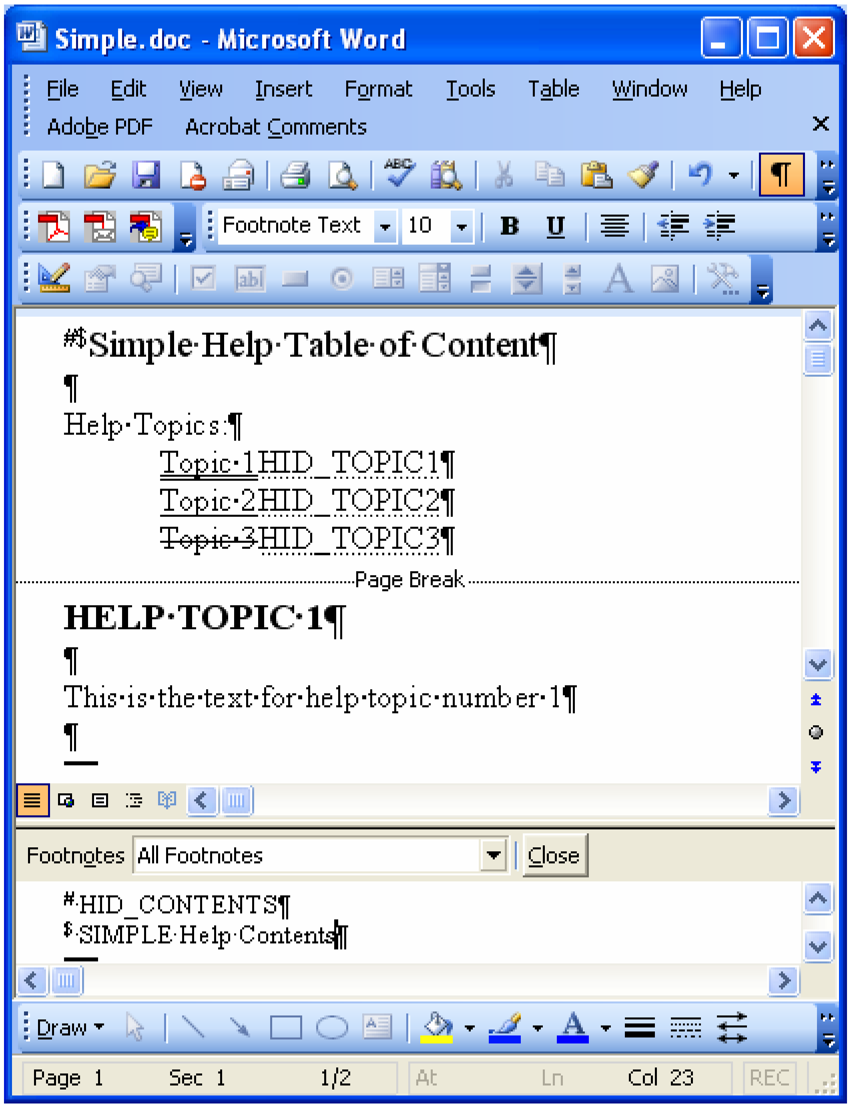
Figure 2: Enabling the Word's Show/Hide view option.

Insert footnotes for the **Table Of Contents** screen. The **Table Of Contents** screen is the first topic screen in this help system. Using the specified custom footnote marks, insert the following footnotes at the beginning of the topic title.

Footnote Mark	**Text**	**Description**
#	`HID_CONTENTS`	Help context ID
$	`SIMPLE Help Contents`	Topic title

Table 1

When you're finished with this step, the document should look like this.


Simple.doc - Microsoft Word
File Edit View Insert Format Tools Table Window Help
Adobe PDF Acrobat Comments
Footnote Text 10
Page Break
Footnotes All Footnotes Close
Draw
Page 1 Sec 1 1/2 At Ln Col 23 REC

#$Simple·Help·Table·of·Content¶
¶
Help·Topics:¶
Topic·1HID_TOPIC1¶
Topic·2HID_TOPIC2¶
~~Topic·3~~HID_TOPIC3¶

HELP·TOPIC·1¶
¶
This·is·the·text·for·help·topic·number·1¶
¶

#·HID_CONTENTS¶
$·SIMPLE·Help·Contents¶

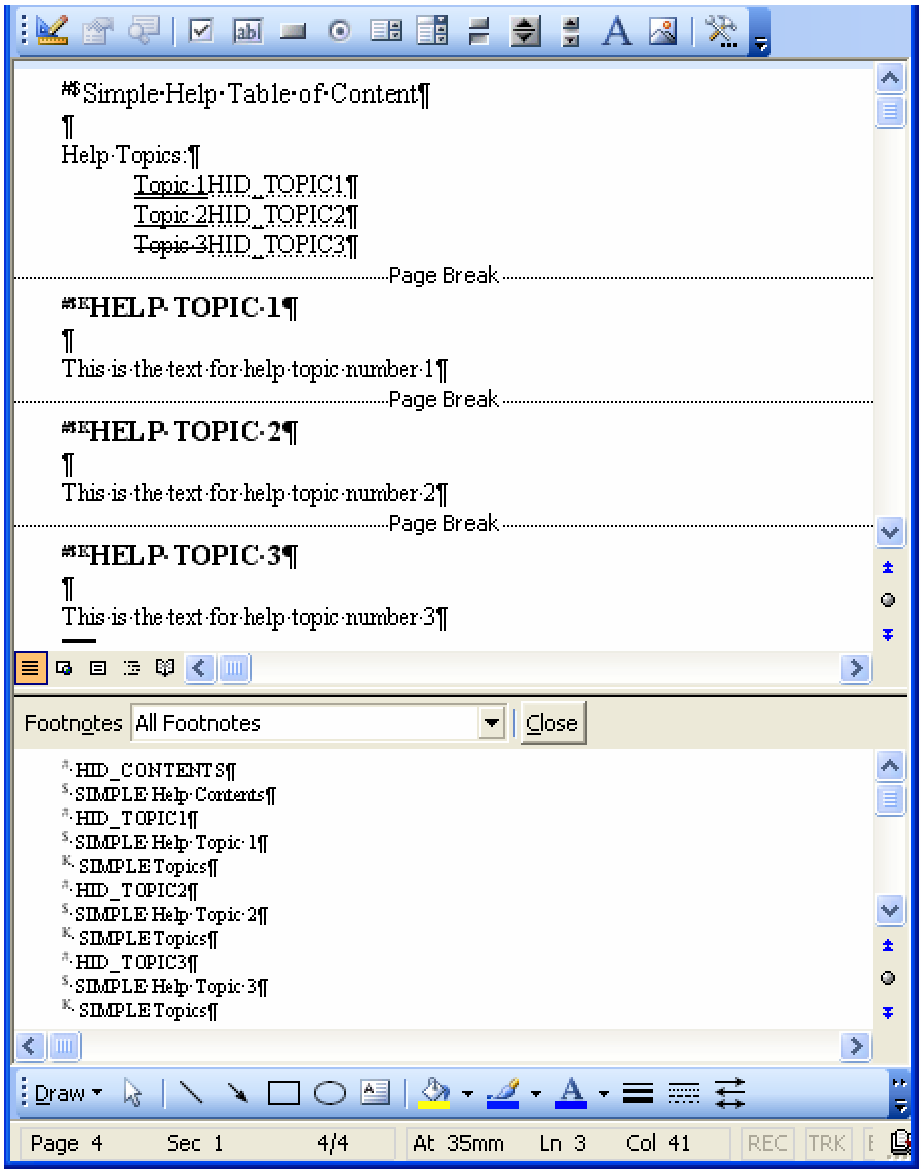
Figure 3: Our completed main help text file.

Insert footnotes for the **Help Topic 1** screen. The **Help Topic 1** screen is the second topic screen in the help system. Using the specified custom footnote marks, insert the footnotes shown here.

Footnote Mark	Text	Description
#	`HID_TOPIC1`	Help context ID
$	`SIMPLE Help Topic 1`	Topic title
K	`SIMPLE Topics`	Keyword text

Table 2.

Clone the **Help Topic 1** screen. Copy the entire Help Topic 1 section of the document, including the page break, to the clipboard, and then paste two copies of the text into the document. The footnotes are copied along with the text. In the first copy, change all occurrences of 1 to 2. In the second copy, change all occurrences of 1 to 3. Don't forget to change the footnotes. With Word, seeing which footnote goes with which topic can be a little difficult, so, be careful. When you're finished with this step, the document text (including footnotes) should look something like this.



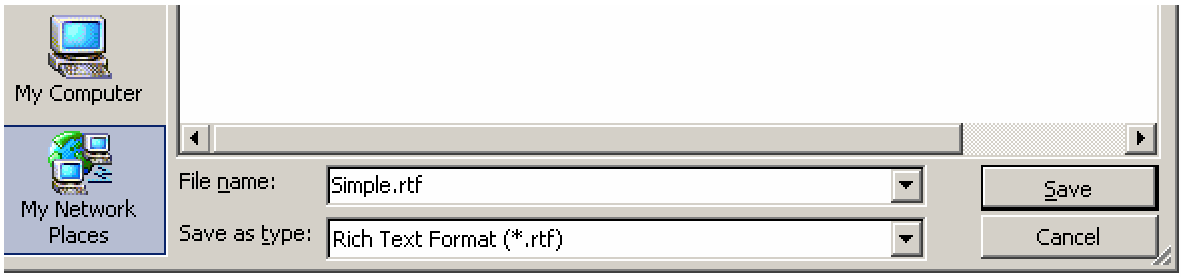
#$Simple·Help·Table·of·Content¶
¶
Help·Topics:¶
Topic·1HID_TOPIC1¶
Topic·2HID_TOPIC2¶
Topic·3HID_TOPIC3¶
Page Break
#$KHELP·TOPIC·1¶
¶
This·is·the·text·for·help·topic·number·1¶
Page Break
#$KHELP·TOPIC·2¶
¶
This·is·the·text·for·help·topic·number·2¶
Page Break
#$KHELP·TOPIC·3¶
¶
This·is·the·text·for·help·topic·number·3¶

Footnotes All Footnotes Close

#·HID_CONTENTS¶
$·SIMPLE·Help·Contents¶
#·HID_TOPIC1¶
$·SIMPLE·Help·Topic·1¶
K·SIMPLE·Topics¶
#·HID_TOPIC2¶
$·SIMPLE·Help·Topic·2¶
K·SIMPLE·Topics¶
#·HID_TOPIC3¶
$·SIMPLE·Help·Topic·3¶
K·SIMPLE·Topics¶

Figure 4: Creating other help topics by copying/cloning.

Save the document. Save the document as \mfcproject\mymfc22\Simple.rtf. Specify **Rich Text Format** as the file type.



File name: Simple.rtf
Save as type: Rich Text Format (*.rtf)

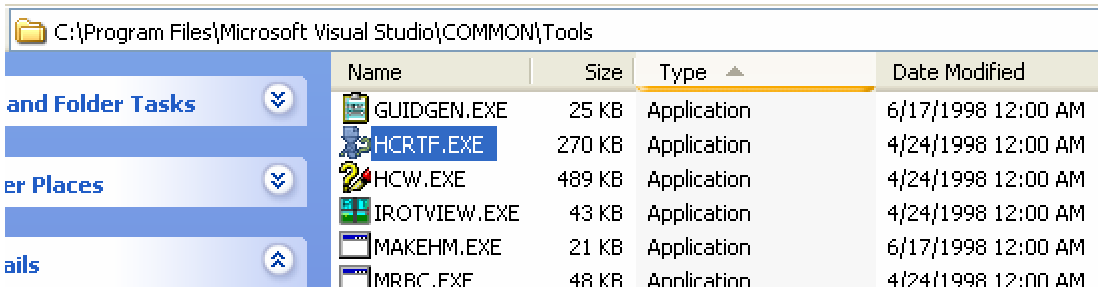
Figure 5: Saving our main help text file in RTF format.

Write a help project file. Using Visual C++ or another text editor, create the file \mfcproject\mymfc22\Simple.hpj, as follows:

```
[OPTIONS]
```

```
CONTENTS=HID_CONTENTS
TITLE=SIMPLE Application Help
COMPRESS=true
WARNING=2

[FILES]
Simple.rtf
```

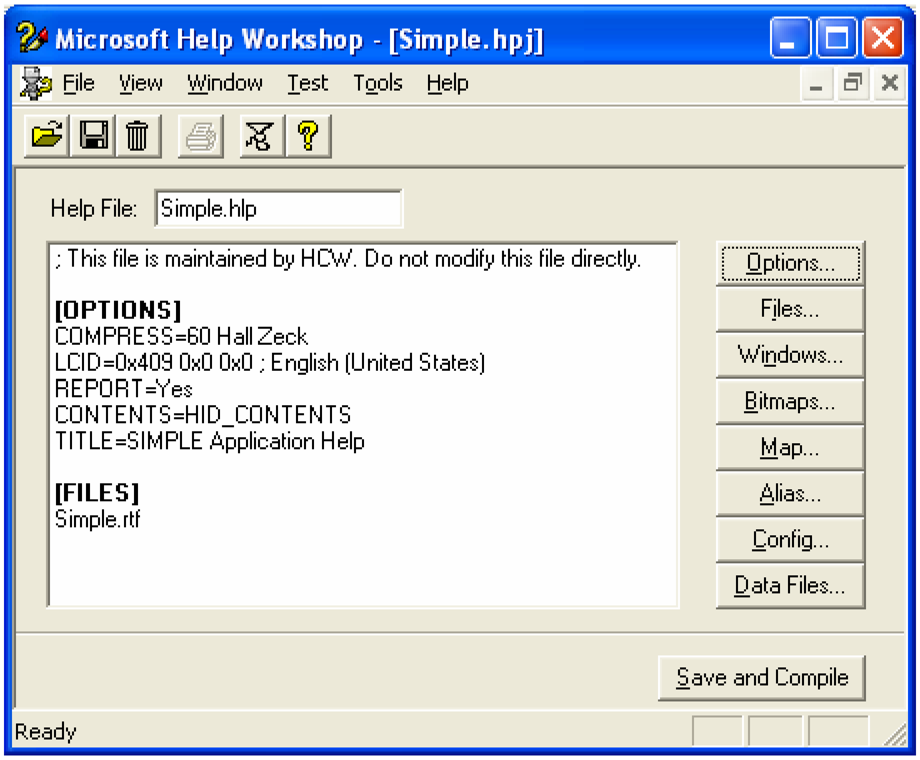
This file specifies the context ID of the **Table Of Contents** screen and the name of the RTF file that contains the help text. Be sure to save the file in text (ASCII) format.
Build the help file. From Windows, run the Microsoft Help Workshop (**HCRTF.EXE**) utility (located by default in **Program Files\Microsoft Visual Studio\Common\Tools**).



```
Name           Size    Type         Date Modified
GUIDGEN.EXE    25 KB   Application  6/17/1998 12:00 AM
HCRTF.EXE      270 KB  Application  4/24/1998 12:00 AM
HCW.EXE        489 KB  Application  4/24/1998 12:00 AM
IROTVIEW.EXE   43 KB   Application  4/24/1998 12:00 AM
MAKEHM.EXE     21 KB   Application  6/17/1998 12:00 AM
MRBC.EXE       48 KB   Application  4/24/1998 12:00 AM
```

Figure 6: The **HCRTF.EXE** utility, just double click to launch it.

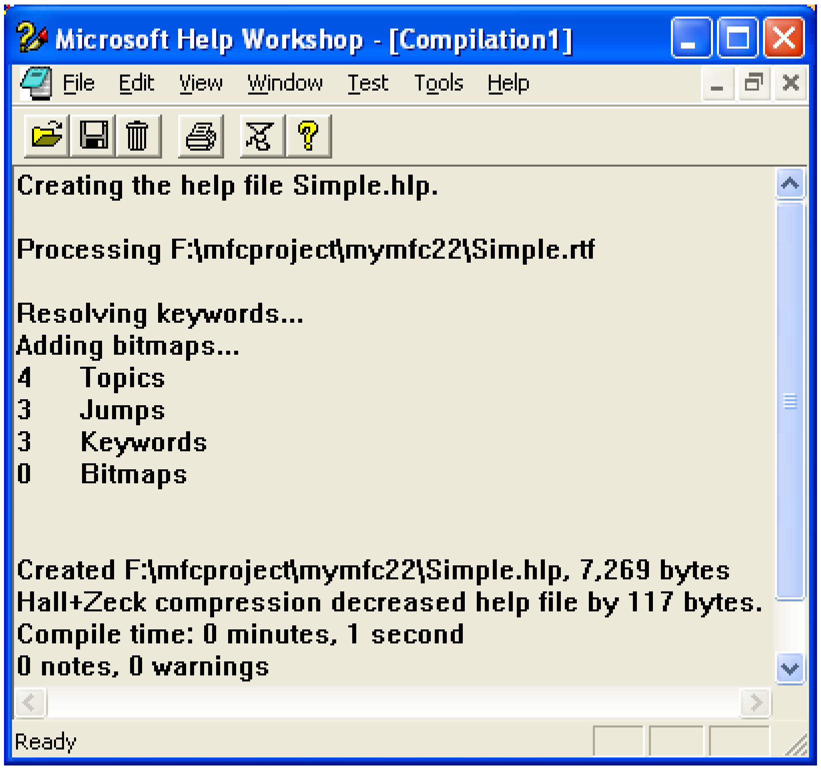
Open the file \mfcproject\mymfc22\Simple.hpj, and then click the **Save And Compile** button.



Help File: Simple.hlp

```
; This file is maintained by HCW. Do not modify this file directly.

[OPTIONS]
COMPRESS=60 Hall Zeck
LCID=0x409 0x0 0x0 ; English (United States)
REPORT=Yes
CONTENTS=HID_CONTENTS
TITLE=SIMPLE Application Help

[FILES]
Simple.rtf
```

Figure 7: **Simple.hpj** file opened in **HCRTF.EXE** program.


Microsoft Help Workshop - [Compilation1]
File Edit View Window Test Tools Help
Ready

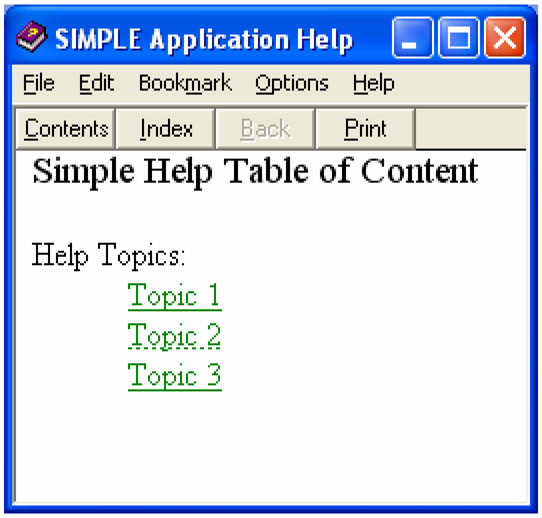
```
Creating the help file Simple.hlp.

Processing F:\mfcproject\mymfc22\Simple.rtf

Resolving keywords...
Adding bitmaps...
4     Topics
3     Jumps
3     Keywords
0     Bitmaps


Created F:\mfcproject\mymfc22\Simple.hlp, 7,269 bytes
Hall+Zeck compression decreased help file by 117 bytes.
Compile time: 0 minutes, 1 second
0 notes, 0 warnings
```

Figure 8: Compiled **Simple.hpj** file generating **Simple.hlp** file.

This step runs the **Windows Help Compiler** with the project file **Simple.hpj**. The output is the help file **Simple.hlp** in the same directory.

If you use Word 97 to create or edit RTF files, make sure you use version 4.02 (or later) of the **HCRTF.EXE** utility. Earlier versions of the HCRTF.EXE cannot process the rich text flags generated by Word 97.

Run **WinHelp** with the new help file. From **Windows Explorer**, double-click the file \mfcproject \mymfc22**Simple.hlp**. The **Table Of Contents** screen should look like this.



Simple Help Table of Content

Help Topics:

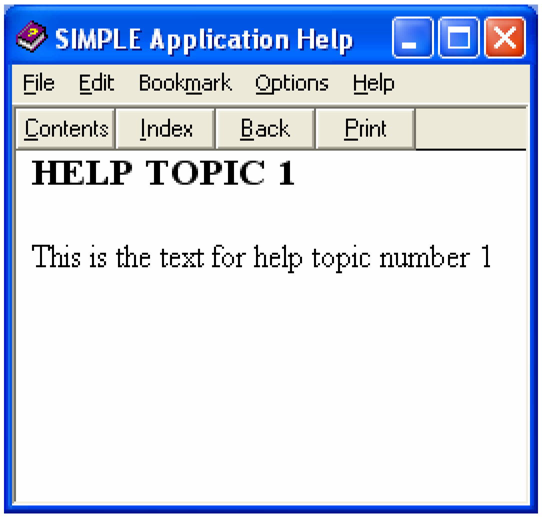
Topic 1

Topic 2

Topic 3

Figure 9: Our **Simple.hlp** program.

Now move the mouse cursor to **Topic 1**. Notice that the cursor changes from an **arrow** to a **pointing hand**. When you press the left mouse button, the **Help Topic 1** screen should appear, as shown here.

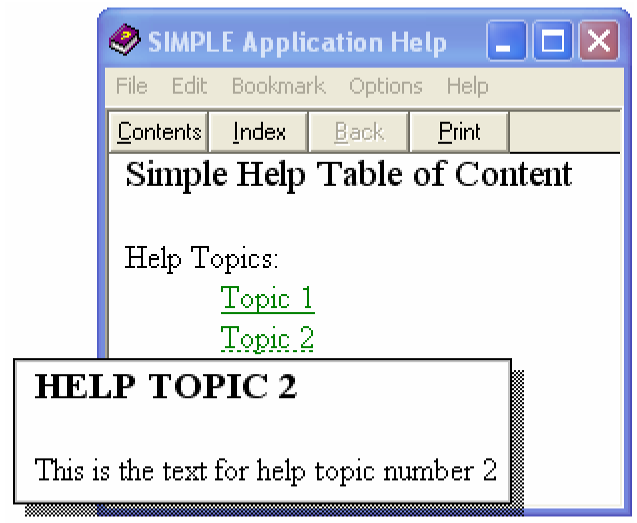


HELP TOPIC 1

This is the text for help topic number 1

Figure 10: Exploring our help file.

The `HID_TOPIC1` text in the **Table Of Contents** screen links to the corresponding context ID (the **#** footnote) in the topic page. This link is known as a jump. The link to **Help Topic 2** is coded as a pop-up jump. When you click on **Topic 2**, here's what you see.



Simple Help Table of Content

Help Topics:
Topic 1
Topic 2

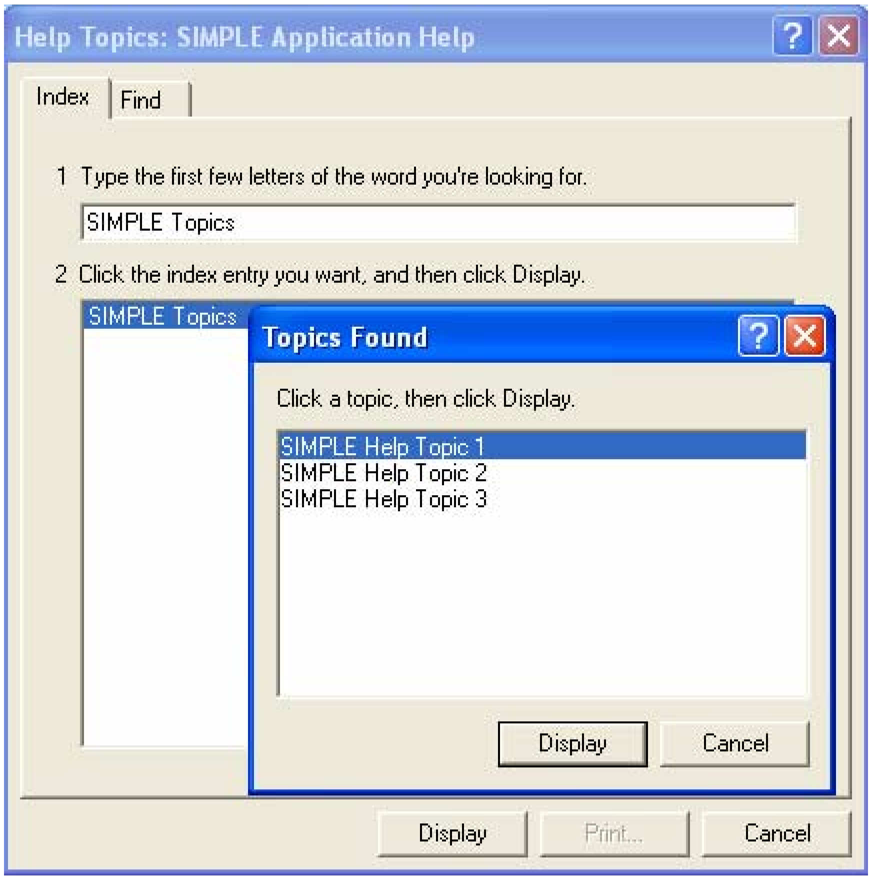
HELP TOPIC 2

This is the text for help topic number 2

Figure 11: Pop-up window for our help file.

Click the **WinHelp Contents** pushbutton. Clicking this button should take you to the Table Of Contents screen, as shown at the beginning of the previous step. WinHelp knows the ID of the Table Of Contents window because you specified it in the HPJ file.

Click the **WinHelp Index** pushbutton. When you click the **Index** button, WinHelp opens its **Index dialog**, which displays the help file's list of keywords. In **Simple.hlp**, all topics (excluding the table of contents) have the same keyword (the **K** footnotes): **SIMPLE Topics**. When you double-click on this keyword, you see all associated topic titles (the **$** footnotes), as shown here.



1 Type the first few letters of the word you're looking for.

SIMPLE Topics

2 Click the index entry you want, and then click Display.

SIMPLE Topics

Click a topic, then click Display.

SIMPLE Help Topic 1
SIMPLE Help Topic 2
SIMPLE Help Topic 3

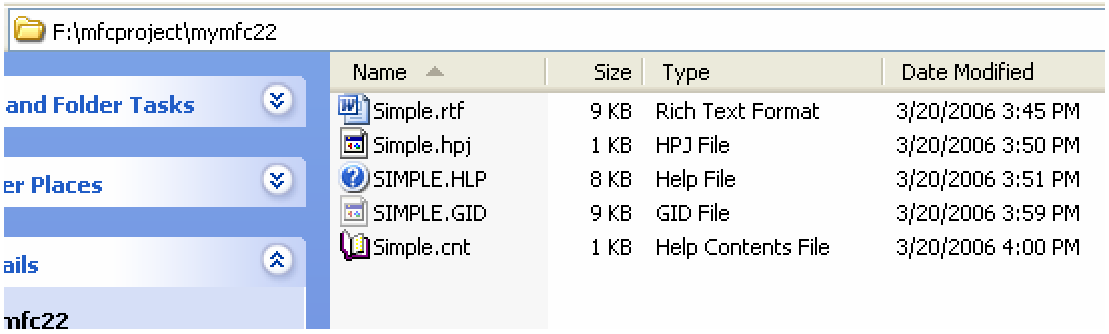
Figure 12: Invoking **WinHelp Index**.

What you have here is a two-level help search hierarchy. The user can type the first few letters of the keyword and then select a topic from a list box. The more carefully you select your keywords and topic titles, the more effective your help system will be.

An Improved Table of Contents

You've been looking at the "old-style" help table of contents. The latest Win32 version of WinHelp can give you a modern tree-view table of contents. All you need is a text file with a **CNT extension**. Add a new file, **Simple.cnt**, in the \mfcproject\mymfc22 directory, containing this text:

```
:Base Simple.hlp
1 Help topics
2 Topic 1=HID_TOPIC1
2 Topic 2=HID_TOPIC2
2 Topic 3=HID_TOPIC3
```



F:\mfcproject\mymfc22

Name	Size	Type	Date Modified
Simple.rtf	9 KB	Rich Text Format	3/20/2006 3:45 PM
Simple.hpj	1 KB	HPJ File	3/20/2006 3:50 PM
SIMPLE.HLP	8 KB	Help File	3/20/2006 3:51 PM
SIMPLE.GID	9 KB	GID File	3/20/2006 3:59 PM
Simple.cnt	1 KB	Help Contents File	3/20/2006 4:00 PM

Figure 13: Creating **Simple.cnt**, a better WinHelp.

Notice the context IDs that match the help file. The next time you run **WinHelp** with the **Simple.hlp** file, you'll see a new contents screen similar to the one shown here.

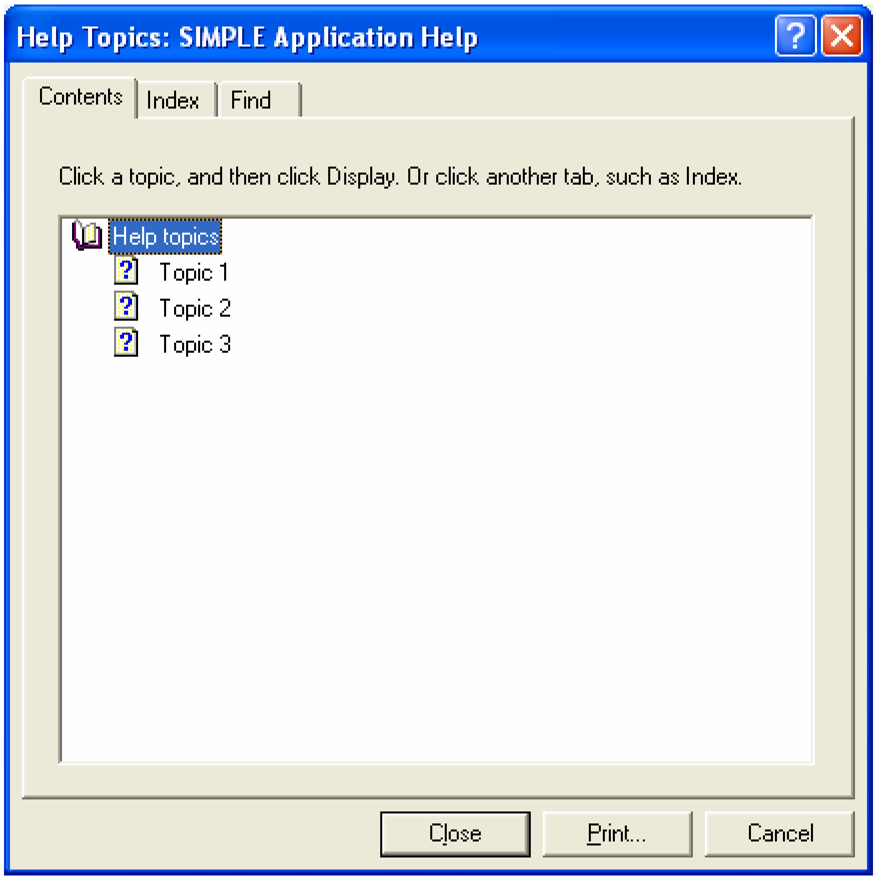

Help Topics: SIMPLE Application Help
Contents
Index
Find
Click a topic, and then click Display. Or click another tab, such as Index.
Help topics
Topic 1
Topic 2
Topic 3
Close
Print...
Cancel

Figure 14: A better WinHelp version.

You can also use **HCRTF.EXE** to edit CNT files. The CNT file is independent of the HPJ file and the RTF files. If you update your RTF files, you must make corresponding changes in your CNT file.

The Application Framework and WinHelp

You've seen WinHelp running as a stand-alone program. The application framework and WinHelp cooperate to give you context-sensitive help. Here are some of the main elements:

1. You select the **Context-Sensitive Help** option when you run AppWizard.
2. AppWizard generates a **Help Topics** item on your application's **Help** menu, and it creates one or more generic RTF files together with an HPJ file and a batch file that runs the **Help Compiler**.
3. AppWizard inserts a keyboard accelerator for the F1 key, and it maps the F1 key and the **Help Topics** menu item to member functions in the main frame window object.
4. When your program runs, it calls **WinHelp** when the user presses F1 or chooses the **Help Topics** menu item, passing a **context ID** that determines which help topic is displayed.

You now need to understand how WinHelp is called from another application and how your application generates context IDs for WinHelp.

Calling WinHelp

The `CWinApp` member function WinHelp activates WinHelp from within your application. If you look up WinHelp in the online documentation, you'll see a long list of actions that the optional second parameter controls. Ignore the second parameter and pretend that WinHelp has only one unsigned long integer parameter, `dwData`. This parameter corresponds to a help topic. Suppose that the SIMPLE help file is available and that your program contains the statement:

```
AfxGetApp()->WinHelp(HID_TOPIC1);
```

When the statement is executed in response to the F1 key or some other event the Help Topic 1 screen appears, as it would if the user had clicked on Topic 1 in the Help Table Of Contents screen.

"Wait a minute," you say. "How does WinHelp know which help file to use?" The name of the help file matches the application name. If the executable program name is **Simple.exe**, the help file is named **Simple.hlp**.

You can force WinHelp to use a different help file by setting the `CWinApp` data member `m_pszHelpFilePath`.

"And how does WinHelp match the program constant `HID_TOPIC1` to the help file's context ID?" you ask. The help project file must contain a MAP section that maps context IDs to numbers. If your application's **resource.h** file defines `HID_TOPIC1` as 101, the **Simple.hpj** MAP section looks like this:

```
[MAP]
HID_TOPIC1        101
```

The program's `#define` constant name doesn't have to match the help context ID; only the numbers must match. Making the names correspond, however, is good practice.

Using Search Strings

For a text-based application, you might need help based on a keyword rather than a numeric context ID. In that case, use the WinHelp `HELP_KEY` or `HELP_PARTIALKEY` option as follows:

```
CString string("find this string");
AfxGetApp()->WinHelp((DWORD) (LPCSTR) string, HELP_KEY);
```

The double cast for string is necessary because the first WinHelp parameter is multipurpose; its meaning depends on the value of the second parameter.

Calling WinHelp from the Application's Menu

AppWizard generates a **Help Topics** option on the **Help** menu, and it maps that option to `CWnd::OnHelpFinder` in the main frame window, which calls WinHelp this way:

```
AfxGetApp()->WinHelp(0L, HELP_FINDER);
```

With this call, WinHelp displays the **Help Table Of Contents** screen, and the user can navigate the help file through jumps and searches. If you want the old-style table of contents, call WinHelp this way instead:

```
AfxGetApp()->WinHelp(0L, HELP_INDEX);
```

And if you want a "help on help" item, make this call:

```
AfxGetApp()->WinHelp(0L, HELP_HELPONHELP);
```

Help Context Aliases

The `ALIAS` section of the HPJ file allows you to equate one context ID with another. Suppose your HPJ file contained the following statements:

```
[ALIAS]
```

```
HID_TOPIC1 = HID_GETTING_STARTED

[MAP]
HID_TOPIC1        101
```

Your RTF files could use HID_TOPIC1 and HID_GETTING_STARTED interchangeably. Both would be mapped to the help context **101** as generated by your application.

Determining the Help Context

You now have enough information to add a simple context-sensitive help system to an MFC program. You define **F1** (the standard MFC library Help key) as a keyboard accelerator, and then you write a command handler that maps the program's help context to a WinHelp parameter. You could invent your own method for mapping the program state to a context ID, but why not take advantage of the system that's already built into the application framework? The application framework determines the help context based on the ID of the active program element. These identified program elements include menu commands, frame windows, dialog windows, message boxes, and control bars. For example, a menu item might be identified as ID_EDIT_CLEAR_ALL. The main frame window usually has the IDR_MAINFRAME identifier. You might expect these identifiers to map directly to help context IDs. IDR_MAINFRAME, for example, would map to a help context ID of the same name. But what if a frame ID and a command ID had the same numeric value? Obviously, you need a way to prevent these overlaps. The application framework solves the overlap problem by defining a new set of help #define constants that are derived from program element IDs. These help constants are the sum of the element ID and a base value, as shown in the following table.

Program Element	Element ID Prefix	Help Context ID Prefix	Base (Hexadecimal)
Menu Item or toolbar button	ID_, IDM_	HID_, HIDM_	10000
Frame or dialog	IDR_, IDD_	HIDR_, HIDD	20000
Error message box	IDP_	HIDP_	30000
Non-client area		H...	40000
Control bar	IDW_	HIDW_	50000
Dispatch error messages	-	-	60000

Table 3.

HID_EDIT_CLEAR_ALL (0x1E121) corresponds to ID_EDIT_CLEAR_ALL (0xE121), and HIDR_MAINFRAME (0x20080) corresponds to IDR_MAINFRAME (0x80).

F1 Help

Two separate context-sensitive help access methods are built into an MFC application and are available if you have selected the AppWizard **Context-Sensitive Help** option. The first is standard **F1** help. The user presses F1; the program makes its best guess about the help context and then calls WinHelp. In this mode, it is possible to determine the currently selected menu item or the currently selected window (frame, view, dialog, or message box).

Shift-F1 Help

The **second context-sensitive help** mode is more powerful than the F1 mode. With **Shift-F1** help, the program can identify the following help contexts:

- A menu item selected with the mouse cursor.
- A toolbar button.
- A frame window.
- A view window.
- A specific graphics element within a view window.
- The status bar.
- Various nonclient elements such as the system menu control.

Shift-F1 help doesn't work with modal dialogs or message boxes. The user activates Shift-F1 help by pressing Shift-F1 or by clicking the **Context Help** toolbar button, shown here.

Figure 15: Context Help button.

In either case, the mouse cursor changes to:
On the next mouse click, the help topic appears, with the position of the mouse cursor determining the context.

Message Box Help: The `AfxMessageBox()` Function

The global function `AfxMessageBox()` displays application framework error messages. This function is similar to the `CWnd::MessageBox` member function except that it has a help context ID as a parameter. The application framework maps this ID to a WinHelp context ID and then calls WinHelp when the user presses F1. If you can use the `AfxMessageBox()` help context parameter, be sure to use prompt IDs that begin with `IDP_`. In your RTF file, use help context IDs that begin with `HIDP_`. There are two versions of `AfxMessageBox()`. In the first version, the prompt string is specified by a character-array pointer parameter. In the second version, the prompt ID parameter specifies a string resource. If you use the second version, your executable program will be more efficient. Both `AfxMessageBox()` versions take a style parameter that makes the message box display an exclamation point, a question mark, or another graphics symbol.

Generic Help

When context-sensitive help is enabled, AppWizard assembles a series of default help topics that are associated with standard MFC library program elements. Following are some of the standard topics:

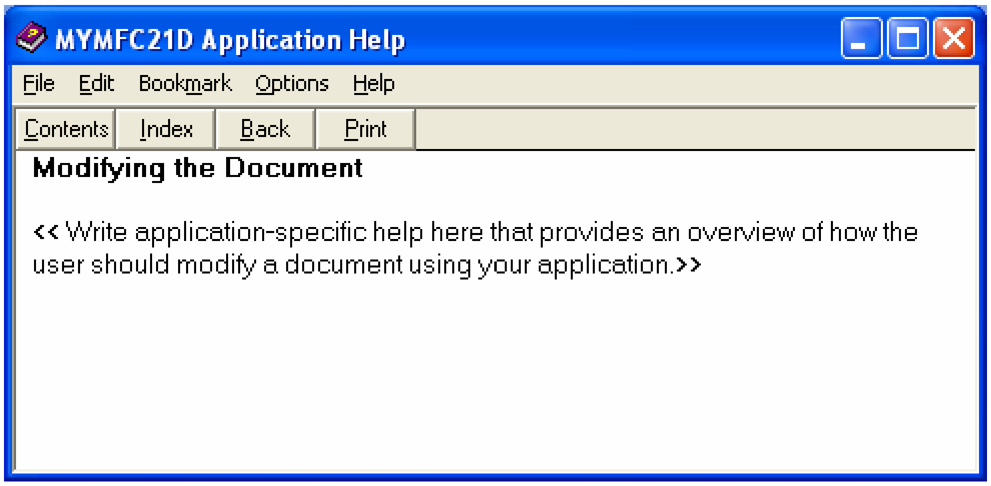
- Menu and toolbar commands (File, Edit, and so forth).
- Nonclient window elements (maximize box, title bar, and so forth).
- Status bar.
- Error message boxes.

These topics are contained in the files **AfxCore.rtf** and **AfxPrint.rtf**, which are copied, along with the associated bitmap files, to the application's **\hlp** subdirectory. Your job is to customize the generic help files. AppWizard generates **AfxPrint.rtf** only if you specify the **Printing And Print Preview** option.

A Help Example: No Programming Required

If you followed the instructions for MYMFC21D in Module 20, you already selected the AppWizard **Context-Sensitive Help** option. We'll now return to that example and explore the application framework's built-in help capability. You'll see how easy it is to link help topics to menu command IDs and frame window resource IDs. You edit RTF files, not CPP files. Here are the steps for customizing the help for MYMFC21D:

Verify that the help file was built correctly. If you have built the MYMFC21D project already, chances are that the help file was created correctly as part of the build process. Check this by running the application and then pressing the F1 key. You should see the generic **Application Help** screen with the title "`Modifying the Document`", something like as shown below.

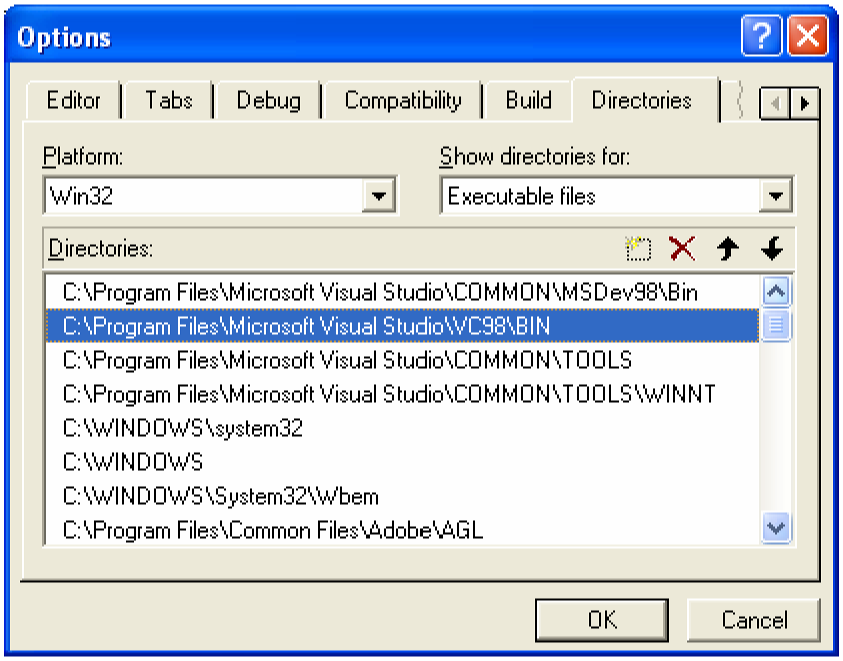

MYMFC21D Application Help
File Edit Bookmark Options Help
Contents Index Back Print

Modifying the Document

<< Write application-specific help here that provides an overview of how the user should modify a document using your application.>>

Figure 16: Verifying that the help file was built correctly in MYMFC21D.

If you do not see this screen, the **MAKEHELP** batch file did not run correctly. First check the last two lines of the **mymfc21D.hpj** file in the **\hlp** subdirectory. Are the paths correct for your Visual C++ installation? Next choose **Options** from the **Tools** menu, and click on the **Directories** tab. Make sure that the **\VC98\bin subdirectory** of your Visual C++ directory is one of the search directories for **Executable Files**.



Platform: Win32

Show directories for: Executable files

Directories:

C:\Program Files\Microsoft Visual Studio\COMMON\MSDev98\Bin
C:\Program Files\Microsoft Visual Studio\VC98\BIN
C:\Program Files\Microsoft Visual Studio\COMMON\TOOLS
C:\Program Files\Microsoft Visual Studio\COMMON\TOOLS\WINNT
C:\WINDOWS\system32
C:\WINDOWS
C:\WINDOWS\System32\Wbem
C:\Program Files\Common Files\Adobe\AGL

Figure 17: Verifying the executable path for visual C++ 6.0.

To generate the help file, highlight the **mymfc21D.hpj** file in the **Workspace FileView** window, and then choose **Compile Mymfc21D.hpj** from the **Build** menu or select the **Compile Mymfc21D.hpj** menu from the mouse right click action.

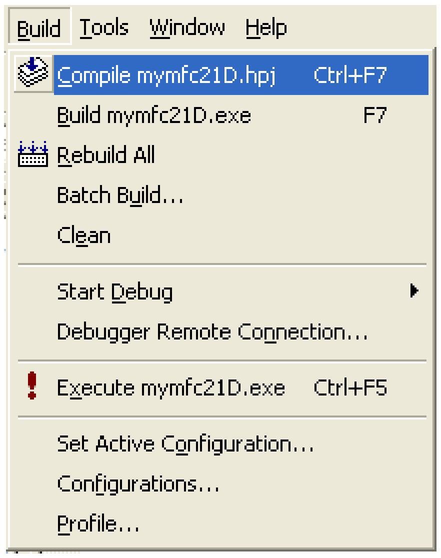

Build
Tools
Window
Help
Compile mymfc21D.hpj Ctrl+F7
Build mymfc21D.exe F7
Rebuild All
Batch Build...
Clean
Start Debug
Debugger Remote Connection...
Execute mymfc21D.exe Ctrl+F5
Set Active Configuration...
Configurations...
Profile...

Figure 18: Compiling the mymfc21D.hpj file to generate WinHelp through **Build** menu.

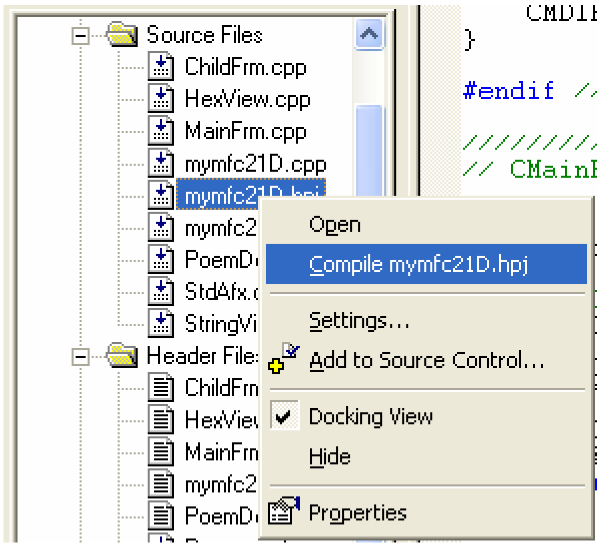
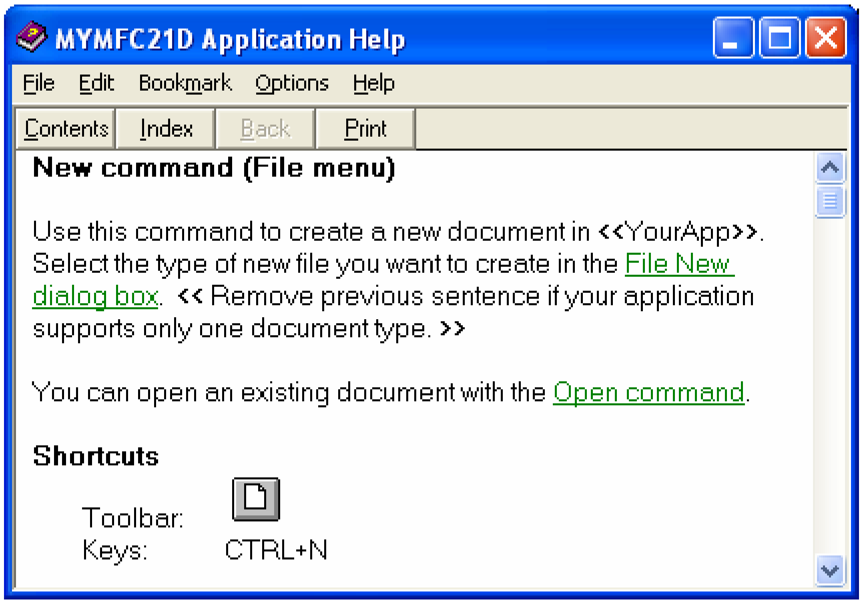


Figure 19: Compiling the **mymfc21D.hpj** file to generate WinHelp through FileView.

This runs the **MAKEHELP** batch file that is in your project directory. You can also run it directly from an MS-DOS prompt. You should observe some "file(s) copied" messages but no error messages. Rerun the MYMFC21D program, and press **F1** again.
The Visual C++ make processor doesn't always detect all the dependencies in your help system. Sometimes you must run the **MAKEHELP** batch file yourself to rebuild the HLP file after making changes.

Test the generic help file. Try the following experiments:

- Close the **Help** dialog, press **Alt-F** and then press **F1**. This should open the help topic for the **File New** command. You can also press **F1** while holding down the mouse button on the **File New** menu item to see the same help topic.


MYMFC21D Application Help

New command (File menu)

Use this command to create a new document in <<YourApp>>. Select the type of new file you want to create in the File New dialog box. << Remove previous sentence if your application supports only one document type. >>

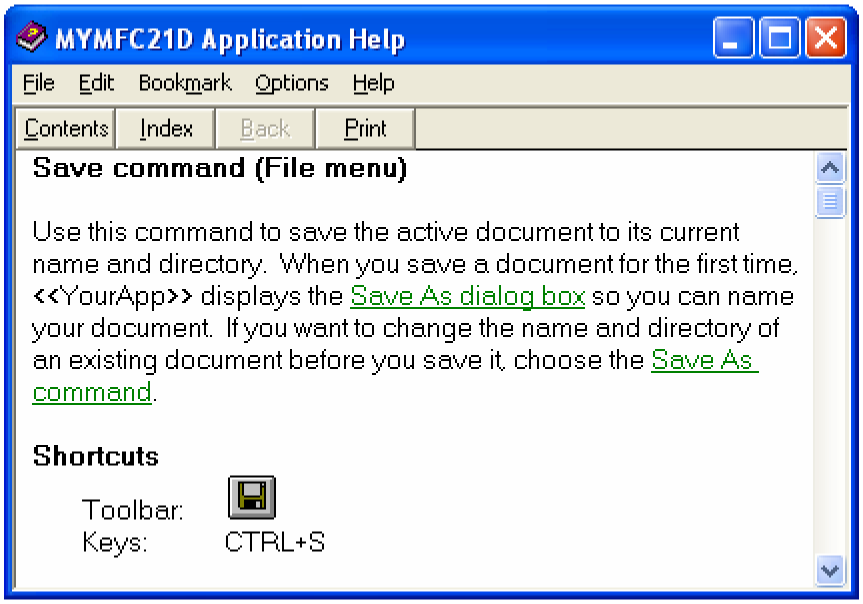
You can open an existing document with the Open command.

Shortcuts

Toolbar:
Keys: CTRL+N

Figure 20: The WinHelp when F1 or Alt-F key is pressed.

- Close the **Help** dialog, click the **Context Help** toolbar button and then choose **Save** from the **File** menu. Do you get the appropriate help topic?


MYMFC21D Application Help

Save command (File menu)

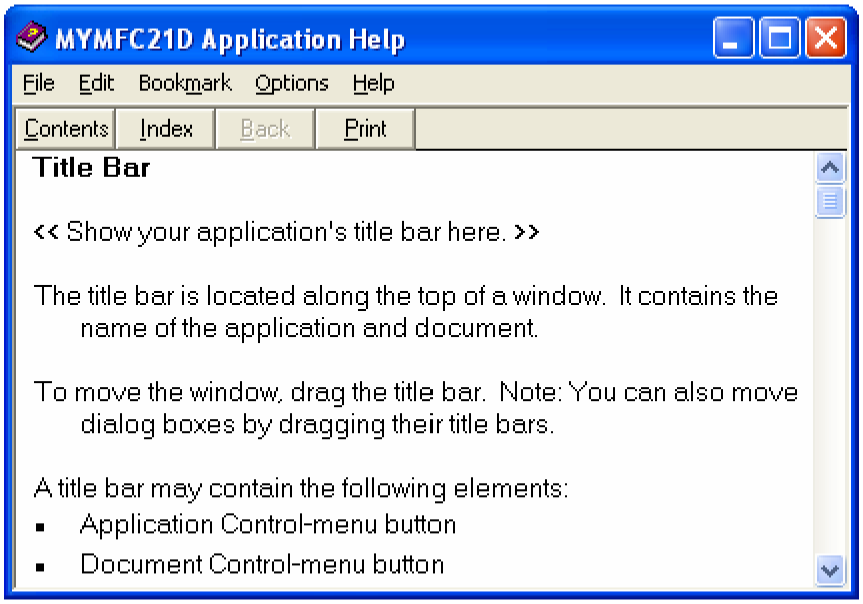
Use this command to save the active document to its current name and directory. When you save a document for the first time, <<YourApp>> displays the Save As dialog box so you can name your document. If you want to change the name and directory of an existing document before you save it, choose the Save As command.

Shortcuts

Toolbar:
Keys: CTRL+S

Figure 21: Using Context help button to invoke the WinHelp.

- Click the **Context Help** toolbar button again, and then select the frame window's title bar. You should get an explanation of a **Windows title bar**.


MYMFC21D Application Help
File Edit Bookmark Options Help
Contents Index Back Print

Title Bar

<< Show your application's title bar here. >>

The title bar is located along the top of a window. It contains the name of the application and document.

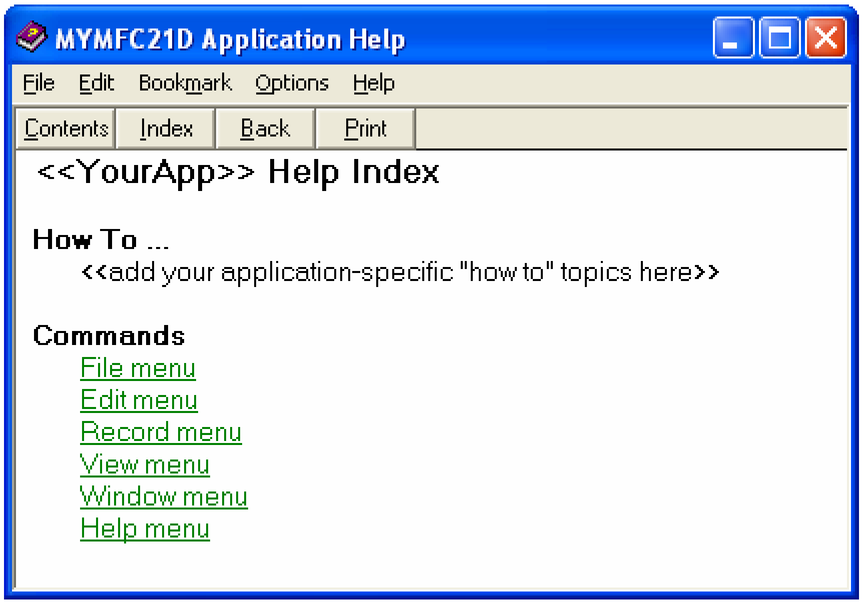
To move the window, drag the title bar. Note: You can also move dialog boxes by dragging their title bars.

A title bar may contain the following elements:

- Application Control-menu button
- Document Control-menu button

Figure 22: Using Context help to invoke the Title bar help.

- Close all child windows and then press **F1**. You should see a main index page that is also an old-style table of contents.


MYMFC21D Application Help
File Edit Bookmark Options Help
Contents Index Back Print

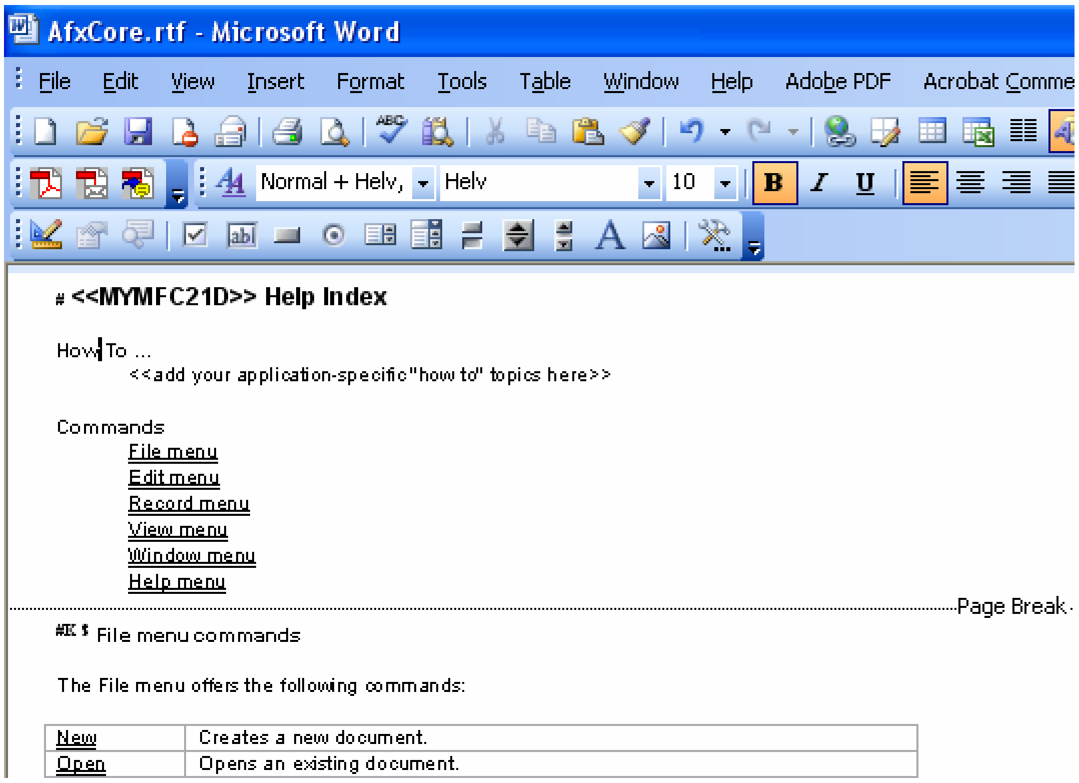
<<YourApp>> Help Index

How To ...
<<add your application-specific "how to" topics here>>

Commands
File menu
Edit menu
Record menu
View menu
Window menu
Help menu

Figure 23: Pressing F1, invoking the help main index page.

Change the application title. The file **AfxCore.rtf**, in the \mfcproject\mymfc21D\hlp directory, contains the string `<<YourApp>>` throughout. Open the **AfxCore.rtf** and replace it globally with **MYMFC21D**.


AfxCore.rtf - Microsoft Word
File Edit View Insert Format Tools Table Window Help Adobe PDF Acrobat Comme
Normal + Helv, Helv 10

<<MYMFC21D>> Help Index

How To ...
<<add your application-specific "how to" topics here>>

Commands
File menu
Edit menu
Record menu
View menu
Window menu
Help menu

Page Break

#K $ File menu commands

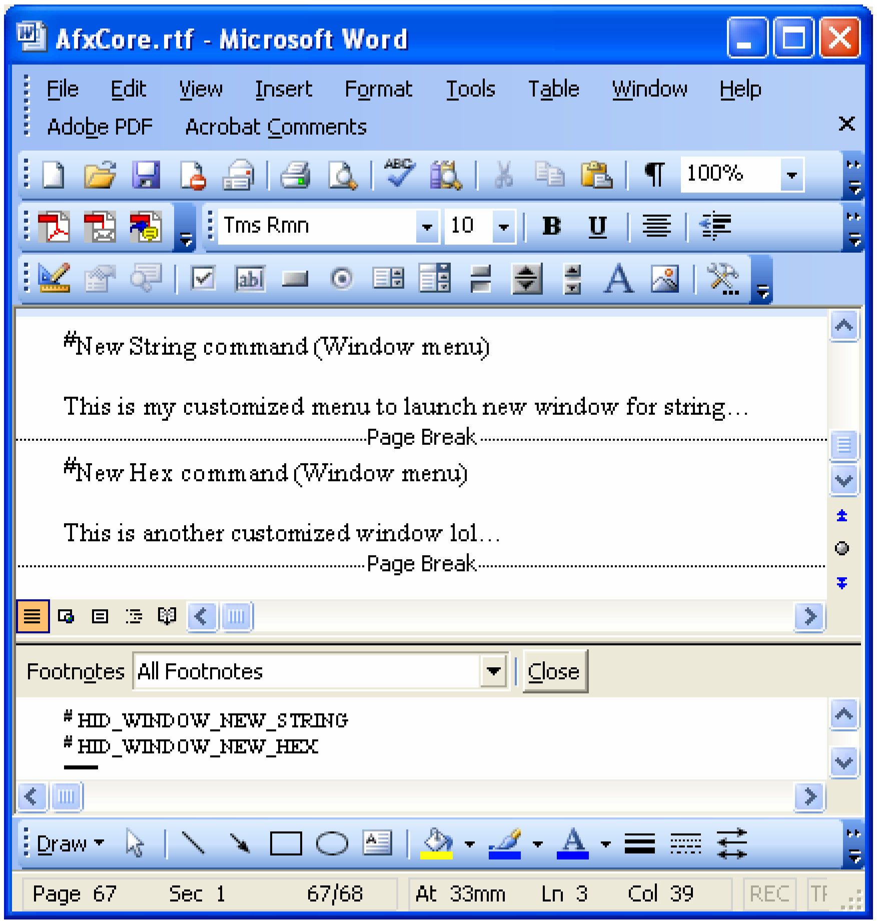
The File menu offers the following commands:

New	Creates a new document.
Open	Opens an existing document.

Figure 24: Modifying the **AfxCore.rtf** to reflect our real WinHelp content.

Change the **Modifying The Document Help** screen. The file **AfxCore.rtf** in the \mfcproject\mymfc21D\hlp directory contains text for the generic **Application Help** screen. Search for **Modifying the Document**, and then change the text to something appropriate for the application. This topic has the help context ID `HIDR_DOC1TYPE`. The generated **mymfc21D.hpj** file provides the alias `HIDR_MYMFC21DTYPE`.

Add a topic for the **Window New String Window** menu item. The **New String Window** menu item was added to MYMFC21D and thus didn't have associated help text. Add a topic to **AfxCore.rtf**, as shown here.


AfxCore.rtf - Microsoft Word
File Edit View Insert Format Tools Table Window Help
Adobe PDF Acrobat Comments
Tms Rmn 10

#New String command (Window menu)

This is my customized menu to launch new window for string…

Page Break

#New Hex command (Window menu)

This is another customized window lol…

Page Break

Footnotes All Footnotes Close

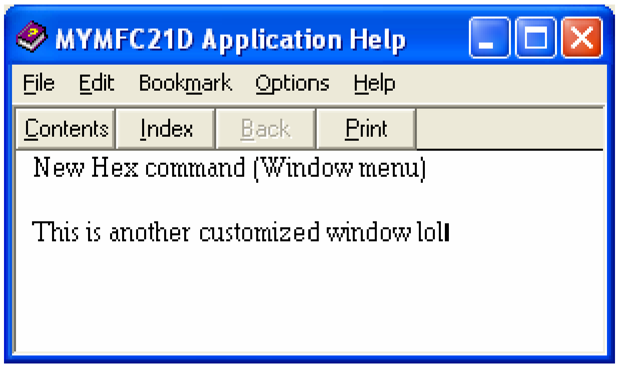
HID_WINDOW_NEW_STRING
HID_WINDOW_NEW_HEX

Page 67 Sec 1 67/68 At 33mm Ln 3 Col 39

Figure 25: Modifying the **AfxCore.rtf** to reflect our real WinHelp content.

Notice the # footnote that links the topic to the context ID `HID_WINDOW_NEW_STRING` as defined in **hlp\mymfc21D.hm**. The program's command ID for the **New String Window** menu item is, of course, `ID_WINDOW_NEW_STRING`.

Rebuild the help file and test the application. Run the **MAKEHELP** batch file again, from the FileView or Build menu, and then rerun the MYMFC21D program. Try the two new help links.


MYMFC21D Application Help
File Edit Bookmark Options Help
Contents Index Back Print

New Hex command (Window menu)

This is another customized window lol!

Figure 26: Our rebuilt WinHelp.

The MAKEHELP Process

The process of building the application's HLP file is complex. Part of the complexity results from the **Help Compiler's** non-acceptance of statements such as:

```
HID_MAINFRAME = ID_MAINFRAME + 0x20000
```

Because of the **Help Compiler's** non-acceptance, a special preprocessing program named **makehm.exe** must read the **resource.h** file to produce a help map file that defines the help context ID values. Below is a diagram of the entire MAKEHELP process.

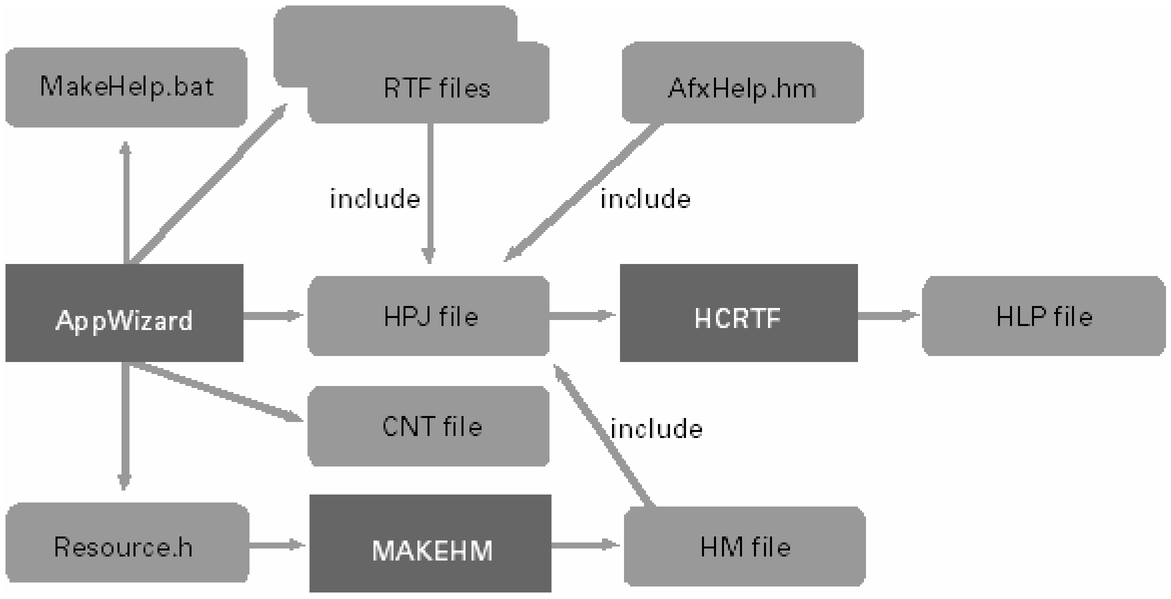


Figure 27: The MAKEHELP process.

AppWizard generates the application's **help project file** (HPJ) and the **help contents file** (CNT). In the project file, the `[FILES]` section brings in RTF files and the `[MAP]` section contains `#include` statements for the generic and the application-specific **help map** (HM) files. The Help Workshop (HCRTF.EXE) processes the project file to produce the help file that WinHelp reads.

Help Command Processing

You've seen the components of a help file, and you've seen the effects of **F1** and **Shift-F1**. You know how the application element IDs are linked to help context IDs. What you haven't seen is the application framework's internal processing of the help requests. Why should you be concerned? Suppose you want to provide help on a specific view window instead of a frame window. What if you need help topics linked to specific graphics items in a view window? These and other needs can be met by mapping the appropriate help messages in the view class. Help command processing depends on whether the help request was an F1 request or a Shift-F1 request. The processing of each help request will be described separately.

F1 Processing

The F1 key is normally handled by a keyboard accelerator entry that AppWizard inserts in the RC file. The accelerator associates the F1 key with an `ID_HELP` command that is sent to the `OnHelp()` member function in the `CFrameWnd` class.

In an active modal dialog or a menu selection in progress, the F1 key is processed by a Windows hook that causes the same `OnHelp()` function to be called. The F1 accelerator key would otherwise be disabled.

The `CFrameWnd::OnHelp` function sends an MFC-defined `WM_COMMANDHELP` message to the innermost window, which is usually the view. If your view class does not map this message or if the handler returns `FALSE`, the framework routes the message to the next outer window, which is either the MDI child frame or the main frame. If you have not mapped `WM_COMMANDHELP` in your derived frame window classes, the message is processed in the MFC `CFrameWnd` class, which displays help for the symbol that AppWizard generates for your application or document type.

If you map the `WM_COMMANDHELP` message in a derived class, your handler must call `CWinApp::WinHelp` with the proper context ID as a parameter.
For any application, AppWizard adds the symbol `IDR_MAINFRAME` to your project and the HM file defines the help context ID `HIDR_MAINFRAME`, which is aliased to `main_index` in the HPJ file. The standard **AfxCore.rtf** file associates the main index with this context ID.
For an MDI application named **SAMPLE**, for example, AppWizard also adds the symbol `IDR_SAMPLETYPE` to your project and the HM file defines the help context ID `HIDR_SAMPLETYPE`, which is aliased to `HIDR_DOC1TYPE` in the HPJ file. The standard **AfxCore.rtf** file associates the topic "`Modifying the Document`" with this context ID.

Shift-F1 Processing

When the user presses **Shift-F1** or clicks the **Context Help** toolbar button, a command message is sent to the `CFrameWnd` function `OnContextHelp()`. When the user presses the mouse button again after positioning the mouse cursor, an MFC-defined `WM_HELPHITTEST` message is sent to the innermost window where the mouse click is detected. From that point on, the routing of this message is identical to that for the `WM_COMMANDHELP` message, described previously in "F1 Processing". The `lParam` parameter of `OnHelpHitTest()` contains the mouse coordinates in device units, relative to the upper-left corner of the window's client area. The **y** value is in the high-order half; the **x** value is in the low-order half. You can use these coordinates to set the help context ID specifically for an item in the view. Your `OnHelpHitTest()` handler should return the correct context ID; the framework will call WinHelp.

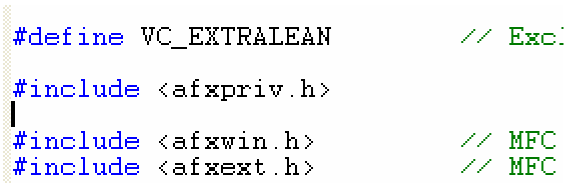
A Help Command Processing Example: MYMFC22B

MYMFC22B is based on example MYMFC21D from Module 21. It's a two-view MDI application with view-specific help added. Each of the two view classes has an `OnCommandHelp()` message handler to process F1 help requests and an `OnHelpHitTest()` message handler to process Shift-F1 help requests. Make a copy of the MYMFC21D project folder as a backup. Use the original for this exercise. All the following codes were copied into those relevant files manually.

Header Requirements

The compiler recognizes help-specific identifiers only if the following `#include` statement is present:

```
#include <afxpriv.h>
```



```
#define VC_EXTRALEAN          // Exc

#include <afxpriv.h>

#include <afxwin.h>           // MFC
#include <afxext.h>           // MFC
```

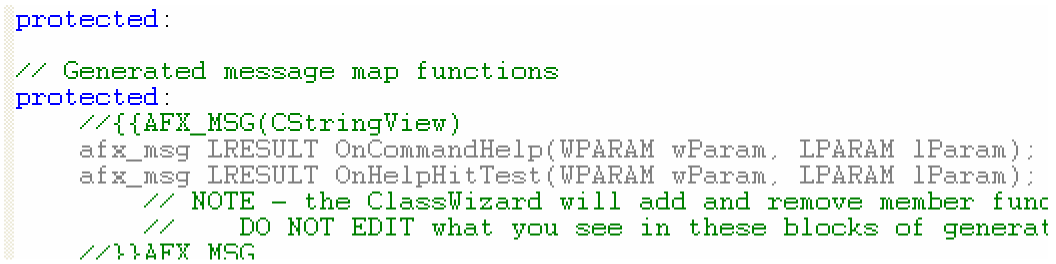
Listing 1.

In MYMFC22B, the statement is in the **StdAfx.h** file.

CStringView Class

The modified string view in **StringView.h** needs message map function prototypes for both F1 help and Shift-F1 help, as shown here:

```
afx_msg LRESULT OnCommandHelp(WPARAM wParam, LPARAM lParam);
afx_msg LRESULT OnHelpHitTest(WPARAM wParam, LPARAM lParam);
```

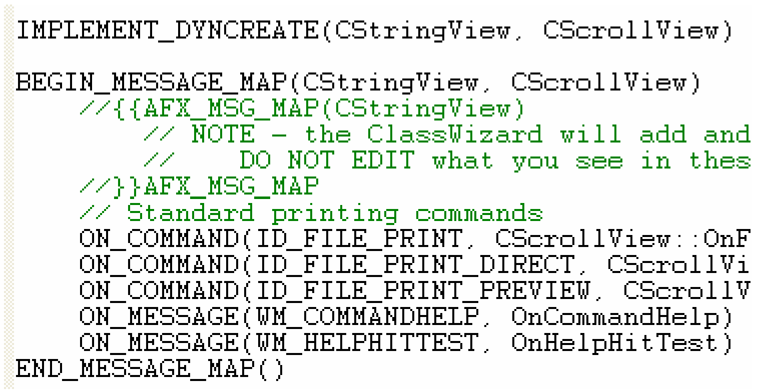


```
protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CStringView)
    afx_msg LRESULT OnCommandHelp(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT OnHelpHitTest(WPARAM wParam, LPARAM lParam);
        // NOTE - the ClassWizard will add and remove member func
        //    DO NOT EDIT what you see in these blocks of generat
    //}}AFX_MSG
```

Listing 2.

Here are the message map entries in **StringView.cpp**:

```
ON_MESSAGE(WM_COMMANDHELP, OnCommandHelp)
ON_MESSAGE(WM_HELPHITTEST, OnHelpHitTest)
```

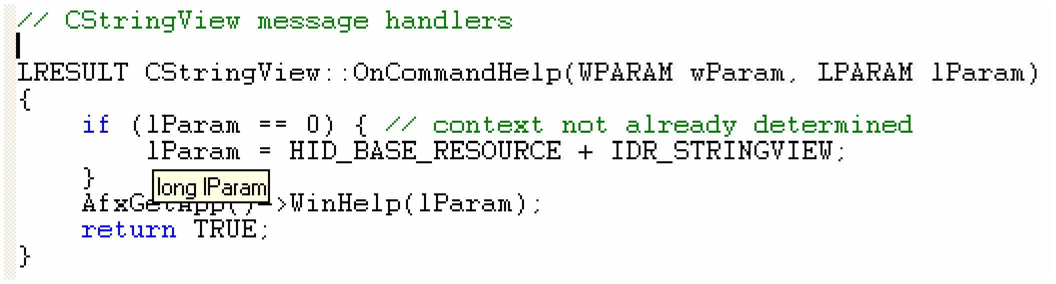


```
IMPLEMENT_DYNCREATE(CStringView, CScrollView)

BEGIN_MESSAGE_MAP(CStringView, CScrollView)
    //{{AFX_MSG_MAP(CStringView)
        // NOTE - the ClassWizard will add and
        //    DO NOT EDIT what you see in thes
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CScrollView::OnF
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CScrollVi
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CScrollV
    ON_MESSAGE(WM_COMMANDHELP, OnCommandHelp)
    ON_MESSAGE(WM_HELPHITTEST, OnHelpHitTest)
END_MESSAGE_MAP()
```

Listing 3.

The `OnCommandHelp()` message handler member function in **StringView.cpp** processes F1 help requests. It responds to the message sent from the MDI main frame and displays the help topic for the string view window, as shown here:

```
LRESULT CStringView::OnCommandHelp(WPARAM wParam, LPARAM lParam)
{
    if (lParam == 0) { // context not already determined
        lParam = HID_BASE_RESOURCE + IDR_STRINGVIEW;
    }
    AfxGetApp()->WinHelp(lParam);
    return TRUE;
}
```



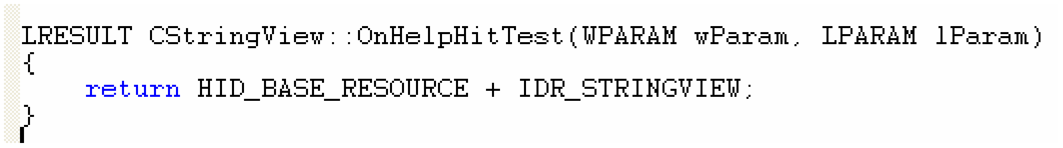
```
// CStringView message handlers

LRESULT CStringView::OnCommandHelp(WPARAM wParam, LPARAM lParam)
{
    if (lParam == 0) { // context not already determined
        lParam = HID_BASE_RESOURCE + IDR_STRINGVIEW;
    }
    AfxGe       ->WinHelp(lParam);
    return TRUE;
}
```

Listing 4.

Finally the `OnHelpHitTest()` member function handles **Shift-F1** help, as shown here:

```
LRESULT CStringView::OnHelpHitTest(WPARAM wParam, LPARAM lParam)
```

```
        {
            return HID_BASE_RESOURCE + IDR_STRINGVIEW;
        }
```



```
LRESULT CStringView::OnHelpHitTest(WPARAM wParam, LPARAM lParam)
{
    return HID_BASE_RESOURCE + IDR_STRINGVIEW;
}
```

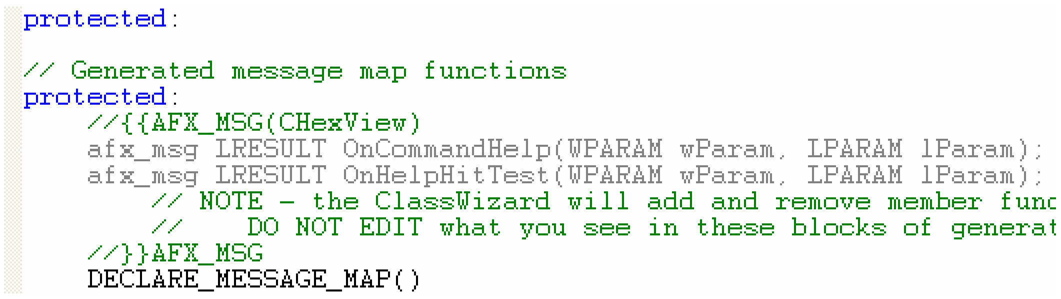
Listing 5.

In a more complex application, you might want `OnHelpHitTest()` to set the help context ID based on the mouse cursor position.

`CHexView` Class

The `CHexView` class processes help requests the same way as the `CStringView` class does. Following is the necessary header code in **HexView.h**:

```
        afx_msg LRESULT OnCommandHelp(WPARAM wParam, LPARAM lParam);
        afx_msg LRESULT OnHelpHitTest(WPARAM wParam, LPARAM lParam);
```

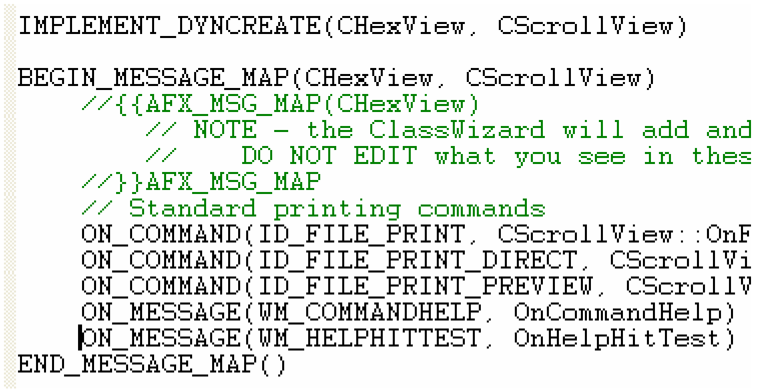


```
protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CHexView)
    afx_msg LRESULT OnCommandHelp(WPARAM wParam, LPARAM lParam);
    afx_msg LRESULT OnHelpHitTest(WPARAM wParam, LPARAM lParam);
        // NOTE - the ClassWizard will add and remove member func
        //    DO NOT EDIT what you see in these blocks of generat
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
```

Listing 6.

Here are the message map entries in **HexView.cpp**:

```
        ON_MESSAGE(WM_COMMANDHELP, OnCommandHelp)
        ON_MESSAGE(WM_HELPHITTEST, OnHelpHitTest)
```



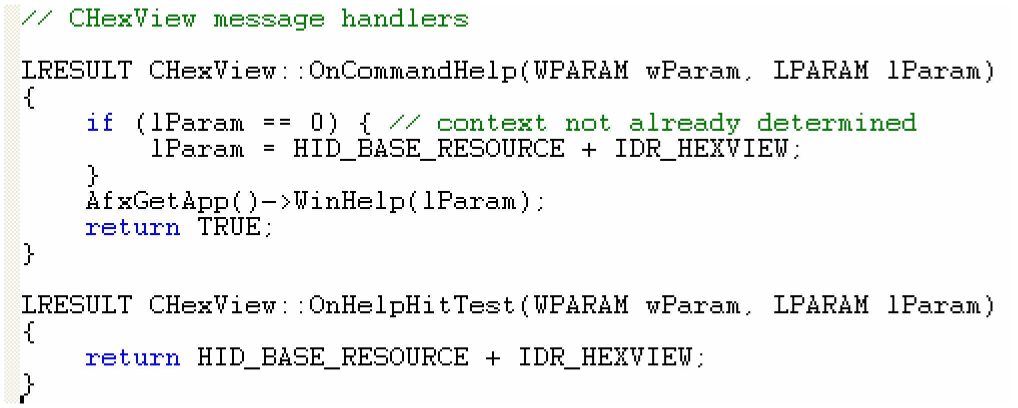
```
IMPLEMENT_DYNCREATE(CHexView, CScrollView)

BEGIN_MESSAGE_MAP(CHexView, CScrollView)
    //{{AFX_MSG_MAP(CHexView)
        // NOTE - the ClassWizard will add and
        //    DO NOT EDIT what you see in thes
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CScrollView::OnF
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CScrollVi
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CScrollV
    ON_MESSAGE(WM_COMMANDHELP, OnCommandHelp)
    ON_MESSAGE(WM_HELPHITTEST, OnHelpHitTest)
END_MESSAGE_MAP()
```

Listing 7.

And here is the implementation code in **HexView.cpp**:

```
        LRESULT CHexView::OnCommandHelp(WPARAM wParam, LPARAM lParam)
        {
```

```
        if (lParam == 0) { // context not already determined
            lParam = HID_BASE_RESOURCE + IDR_HEXVIEW;
        }
        AfxGetApp()->WinHelp(lParam);
        return TRUE;
    }

    LRESULT CHexView::OnHelpHitTest(WPARAM wParam, LPARAM lParam)
    {
        return HID_BASE_RESOURCE + IDR_HEXVIEW;
    }
```



```
// CHexView message handlers

LRESULT CHexView::OnCommandHelp(WPARAM wParam, LPARAM lParam)
{
    if (lParam == 0) { // context not already determined
        lParam = HID_BASE_RESOURCE + IDR_HEXVIEW;
    }
    AfxGetApp()->WinHelp(lParam);
    return TRUE;
}

LRESULT CHexView::OnHelpHitTest(WPARAM wParam, LPARAM lParam)
{
    return HID_BASE_RESOURCE + IDR_HEXVIEW;
}
```

Listing 8.

Resource Requirements

Two new symbols were added to the project's **Resource.h** file. Their values and corresponding help context IDs are shown here.

Symbol	Value	Help Context ID	Value
IDR_STRINGVIEW	101	HIDR_STRINGVIEW	0x20065
IDR_HEXVIEW	102	HIDR_HEXVIEW	0x20066

Table 4

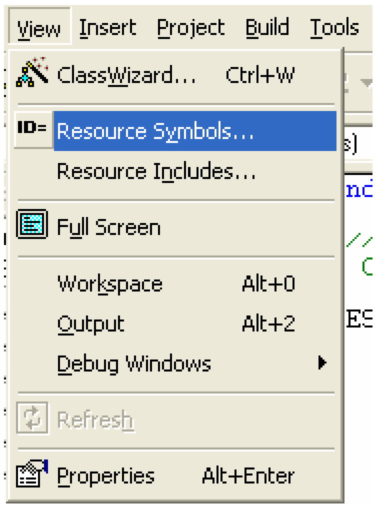
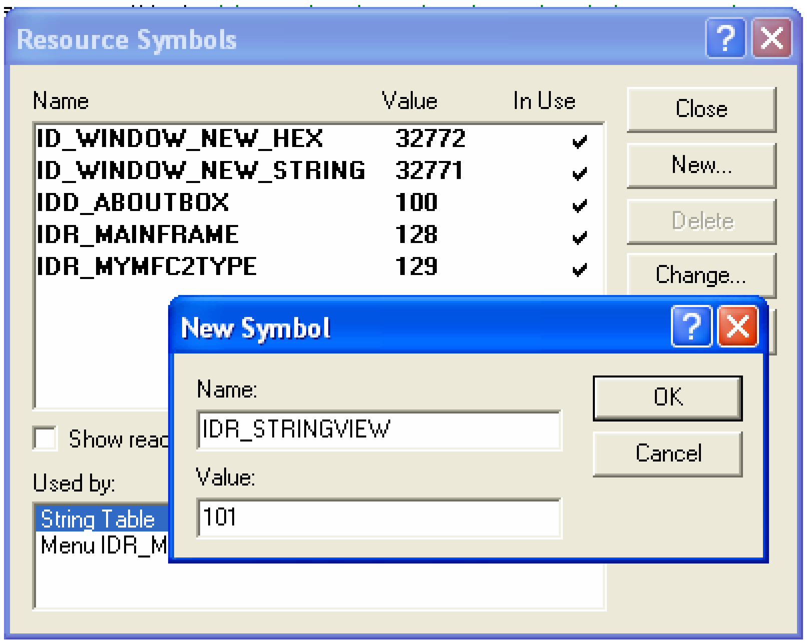


Figure 28: Invoking the resource symbol dialog.



Name	Value	In Use
ID_WINDOW_NEW_HEX	32772	✓
ID_WINDOW_NEW_STRING	32771	✓
IDD_ABOUTBOX	100	✓
IDR_MAINFRAME	128	✓
IDR_MYMFC2TYPE	129	✓

Used by: String Table, Menu IDR_M

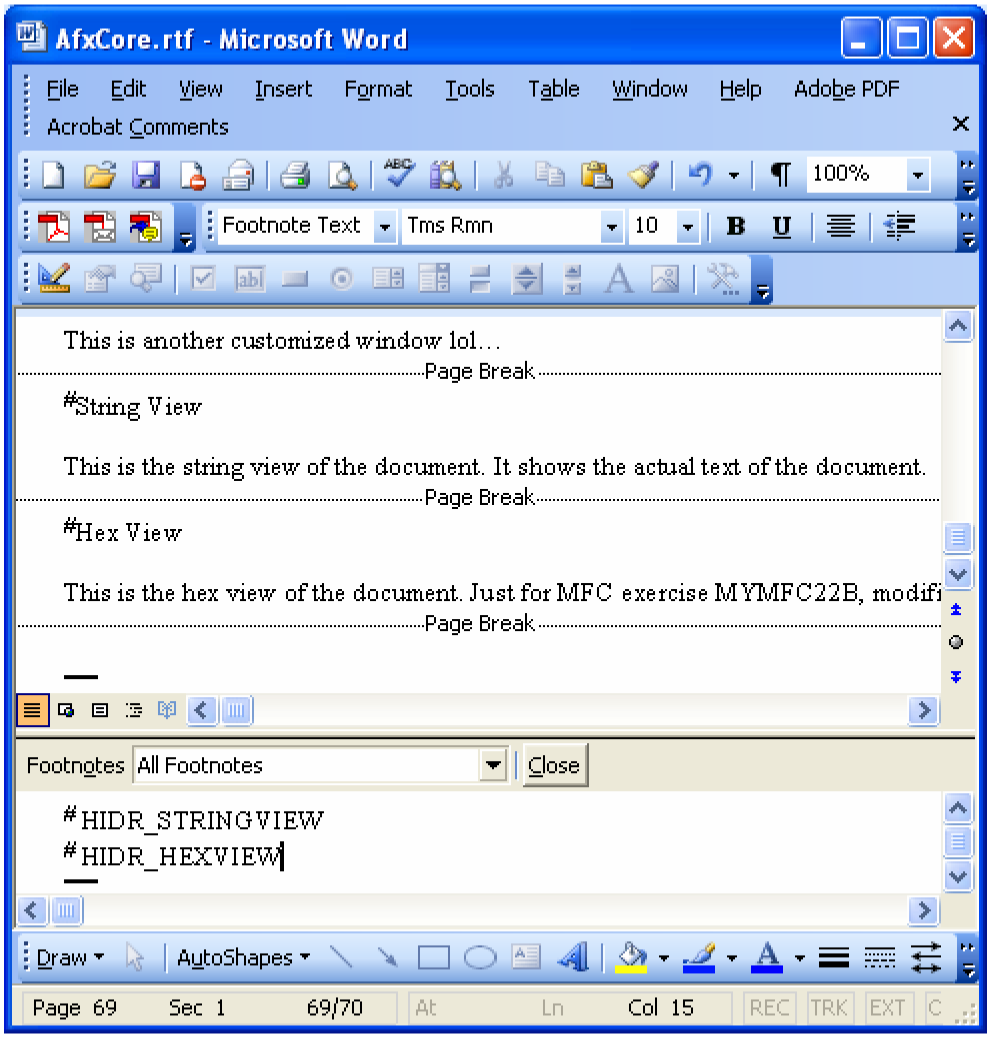
Name: IDR_STRINGVIEW

Value: 101

Figure 29: Adding new resource symbols.

Help File Requirements

Two topics were added to the **AfxCore.rtf** file with the help context IDs `HIDR_STRINGVIEW` and `HIDR_HEXVIEW`, as shown here.


AfxCore.rtf - Microsoft Word
File Edit View Insert Format Tools Table Window Help Adobe PDF
Acrobat Comments
Footnote Text
Tms Rmn
10
100%

This is another customized window lol...

Page Break

#String View

This is the string view of the document. It shows the actual text of the document.

Page Break

#Hex View

This is the hex view of the document. Just for MFC exercise MYMFC22B, modifi

Page Break

Footnotes All Footnotes Close

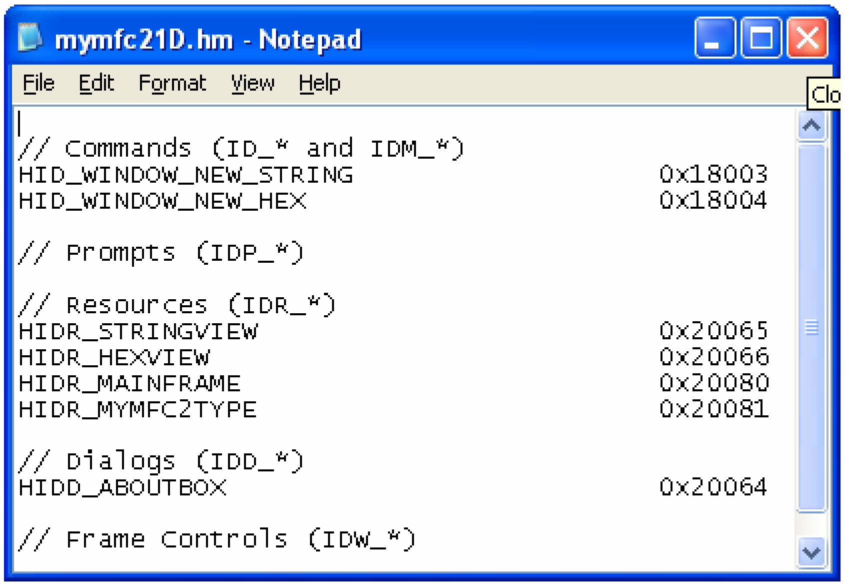
#HIDR_STRINGVIEW
#HIDR_HEXVIEW

Draw AutoShapes
Page 69 Sec 1 69/70 At Ln Col 15 REC TRK EXT

Figure 30: Adding two new help topics to the **AfxCore.rtf** file.

The generated **mymfc22B.hm** file, in the project's **\hlp** subdirectory, should look something like this:

```
// MAKEHELP.BAT generated Help Map file.  Used by MYMFC22B.HPJ.

// Commands (ID_* and IDM_*)
HID_WINDOW_NEW_STRING                   0x18003
HID_WINDOW_NEW_HEX                      0x18005

// Prompts (IDP_*)


// Resources (IDR_*)
HIDR_STRINGVIEW                         0x20065
HIDR_HEXVIEW                            0x20066
HIDR_MAINFRAME                          0x20080
HIDR_MYMFC22BTYPE                       0x20081

// Dialogs (IDD_*)
HIDD_ABOUTBOX                           0x20064

// Frame Controls (IDW_*)
```

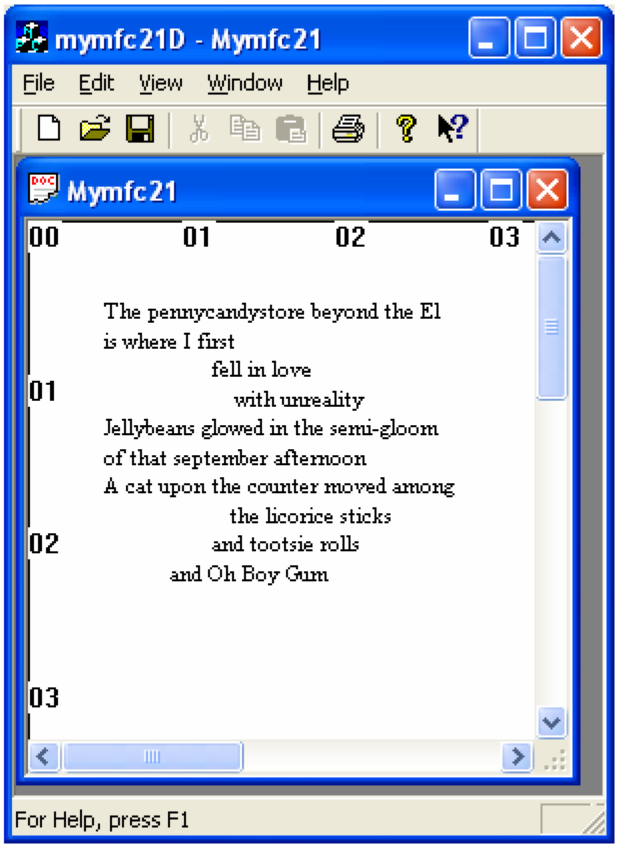


```
// Commands (ID_* and IDM_*)
HID_WINDOW_NEW_STRING                   0x18003
HID_WINDOW_NEW_HEX                      0x18004

// Prompts (IDP_*)

// Resources (IDR_*)
HIDR_STRINGVIEW                         0x20065
HIDR_HEXVIEW                            0x20066
HIDR_MAINFRAME                          0x20080
HIDR_MYMFC2TYPE                         0x20081

// Dialogs (IDD_*)
HIDD_ABOUTBOX                           0x20064

// Frame Controls (IDW_*)
```

Figure 31: **mymfc21D.hm** file content.

Testing the MYMFC22B Application

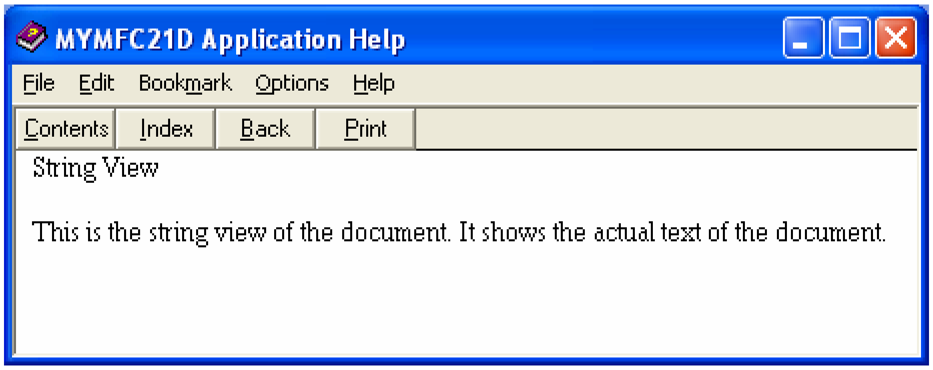
Open a string child window and a hexadecimal child window. Test the action of **F1** help and **Shift-F1** help within those windows. If the help file didn't compile correctly, follow the instructions in the previous step of the help example in "**A Help Example: No Programming Required**".



The pennycandystore beyond the El
is where I first
fell in love
with unreality
Jellybeans glowed in the semi-gloom
of that september afternoon
A cat upon the counter moved among
the licorice sticks
and tootsie rolls
and Oh Boy Gum

Figure 32: The MYMFC22B (MYMFC21D) output used to test the F1 and Shift-F1 help functionalities.

When the **F1** is hit the following help windows launched.

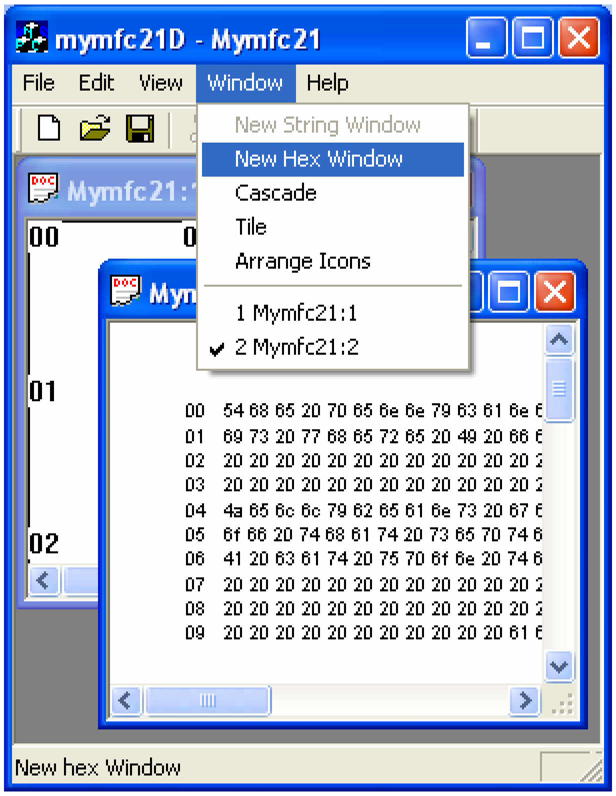

MYMFC21D Application Help
File Edit Bookmark Options Help
Contents Index Back Print

String View

This is the string view of the document. It shows the actual text of the document.

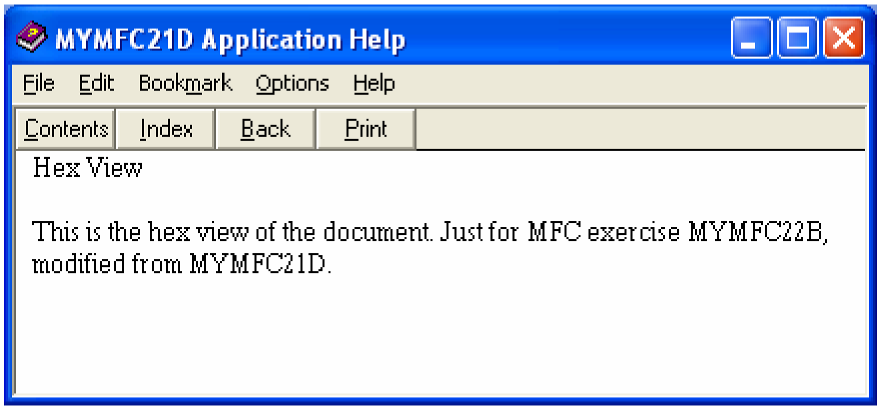
Figure 33: WinHelp when F1 key is pressed for string window.

And another one for the **Hex** view.



```
00   54 68 65 20 70 65 6e 6e 79 63 61 6e
01   69 73 20 77 68 65 72 65 20 49 20 66
02   20 20 20 20 20 20 20 20 20 20 20 20
03   20 20 20 20 20 20 20 20 20 20 20 20
04   4a 65 6c 6c 79 62 65 61 6e 73 20 67
05   6f 66 20 74 68 61 74 20 73 65 70 74
06   41 20 63 61 74 20 75 70 6f 6e 20 74
07   20 20 20 20 20 20 20 20 20 20 20 20
08   20 20 20 20 20 20 20 20 20 20 20 20
09   20 20 20 20 20 20 20 20 20 20 20 61
```

Figure 34: Invoking the Hex window.


MYMFC21D Application Help
File Edit Bookmark Options Help
Contents Index Back Print

Hex View

This is the hex view of the document. Just for MFC exercise MYMFC22B, modified from MYMFC21D.

Figure 35: WinHelp when Shift-F1 key is pressed for Hex window.

Further reading and digging:

1. MSDN MFC 6.0 class library online documentation - used throughout this Tutorial.
2. MSDN MFC 7.0 class library online documentation - used in .Net framework and also backward compatible with 6.0 class library
3. MSDN Library
4. Windows data type.
5. Win32 programming Tutorial.
6. The best of C/C++, MFC, Windows and other related books.
7. Unicode and Multibyte character set: Story and program examples.