HelpGen

Windows Help File Generator

Version 1.31

Association of Shareware Professionals Member

Rimrock Software is a member of the Association of Shareware Professionals (ASP). ASP wants to make sure that the shareware principle works for you. If you are unable to resolve a shareware-related problem with an ASP member by contacting the member directly, ASP may be able to help. The ASP Ombudsman can help you resolve a dispute or problem with an ASP member, but does not provide technical support for members' products. Please write to the ASP Ombudsman at 545 Grover Road, Muskegon, MI 49442-9427 or send a CompuServe message via easyplex to ASP Ombudsman 70007,3536.

STAR

Rimrock Software is a member of Shareware Trade Association and Resources (STAR). STAR's purpose is to promote the common interest of its Members in improving business conditions in the shareware industry. Such activities may in particular include promoting the interests of the entire shareware industry by:

• Educating past and prospective customers of shareware as to its availability and legal nature;

• Gathering and providing information to shareware vendors, publishers, authors and distributors, whether or not they are Members of STAR, to enable them to better evaluate and improve their business methods as they deem best;

• Assisting the improvement and evolution of all components of the shareware industry by
fostering open communication of ideas, common problems and news relating to shareware
marketing methods.

Table of Contents

Introduction 1
  Registration 1
Getting Started 2
  Installing HelpGen 2
  Running HelpGen 3
  The Help File Generation Process 3
  The HelpGen Menu and Toolbar 4
    File | Open Macro File 5
    File | Close Macro File 5
    File | Print Macro File 5
    File | Exit 5
    Edit | Macro File 6
    Edit | Project File 6
    Edit | Bitmap 6
    Edit | Hotspot 6
    Build | Project File 6
    Build | RTF File Only 7
    Build | HLP File Only 7
    Build | HLP from MAC 7
    Build | HTML File 7
    Test | HLP file 7
    Test | HTML file 7
    Options | Directories 8
    Options | Project File 8
    Help | Contents 8
    Help | Search for Help on 8
    Help | Help on Help 8
    Help | About HelpGen 9
  Testing Help Files 9
  Internet HTML Support 9
  Hints and Tips 9
  Help File Default Settings 10
The HelpGen Macro Language 11
  Topic Macros 11
    .begin_file Command 11
    .startc Command 12
    .start Command 13
    .entc Command 13
    .ent Command 14
    .pentc Command 15

.pent Command 16
.end Command 17
.top Command 17
Hypertext Navigation Macros 18
.j Command 18
.j1 Command 19
.bj Command 19
.bp Command 20
.p Command 21
.p1 Command 22
.fj Command 22
.bfj Command 23
Graphics Macros 24
.bmp Command 24
.bmpl Command 24
.bmpr Command 25
.box Command 25
.bend Command 26
Documentation Macros 27
.rem Command 27
.com Command 27
Text Formatting Macros 28
.in Command 28
.un Command 29
## Command 29
#b Command 30
#n Command 30
.b Command 31
.i Command 31
.bu Command 32
.bi Command 32
.iu Command 33
.c Command 33
.[Command 34
.] Command 35
.n Command 35
.s Command 36
Help Macro Access Macros 36
.macro Command 36
.bmacro Command 37

Miscellaneous Macros 38
.end_file Command 38
Some Useful RTF Commands 38
\brdrdb Command 38
\brdrs Command 39
\brdrsh Command 39
\brdrth Command 40
\li Command 40
\ri Command 40
\ql Command 41
\qr Command 41
\qj Command 41
\qc Command 42
\tab Command 42
\'hh Command 42
A HelpGen Tutorial 43
Planning Your Help File 43
Creating the Macro File 44
Using HelpGen Macros and RTF Commands 46
Creating the Project File 47
Generating the Help File 47
Testing the Help File 48
Appendix A HelpGen Macro File Structure 49
Appendix B Project File Structure 51
[OPTIONS] Section 52
BMROOT 52
COMPRESS 52
COPYRIGHT 52
ERRORLOG 52
ICON 52
REPORT 53
WARNING 53
TITLE 53
[FILES] Section 53
[BITMAPS] Section 53
[MAP] Section 53
[WINDOWS] Section 54
Appendix C HelpGen Macro File Structure 55
Appendix D Creating Bitmaps with Multiple Jumps to Topics 56
Appendix E Invoking a Help File from a Windows Program 60
General Help Access 60
Searching for a Help Topic 60
Help on Using Help 61

Contextual Help Access 61
Closing the Help File 62
Appendix F Glossary 63

Introduction

One of the most difficult items for Windows programmers to master is the creation of an on-line help facility for their application. This usually requires the use of a word processor such as Word for Windows or Ami Pro, and knowledge of how to use the word rocessor's capabilities to generate a document in the format that the Windows help compiler expects to see.

HelpGen eases the task of creating help files. No word processor is required; a regular ASCII text editor is used (Notepad is the default). The help compiler is still required, but it is executed from within HelpGen (HC31 is the default help compiler). The text editor and help compiler defaults may be easily changed from within HelpGen.

HelpGen uses a macro language to support most of the capabilities used in help files, such as topics, popup topics, bitmaps and keyword searches. Any capabilities not directly supported by the macro language can be inserted directly into the macro file in the form of Rich Text Format (RTF) commands.

Basically, HelpGen helps to make the task of creating help files a relatively painless process.

Registration

This version of HelpGen is a shareware evaluation version. You may use this version for up to 60 days to determine whether the program is of use to you. If you continue to use the program beyond that time period, you must register. Registration of HelpGen is $30. You can create and print a registration form from within HelpGen by selecting **Help | Register HelpGen** and by filling out the blanks, or you may print an order form by copying ORDER.FRM to your printer. When you register, you will receive the following from Rimrock Software:

• A registered version of HelpGen. The registered version is functionally identical to the shareware version, with the following exceptions:

1. All shareware references, including the .WAV file, have been removed.

2. The Copyright notice box in the project file options dialog has been enabled.

3. Branding of help files with a Rimrock Software copyright notice has been removed.

- A printed manual for the HelpGen program.

- Technical support for the HelpGen program. Support is available to registered users only. Call (208) 772-9347 after 6 p.m. Pacific time, or email to CompuServe by addressing Michael Burton, 71211,70. Contact us on internet email by addressing 71211.70@compuserve.com.

Rimrock Software accepts both VISA and Mastercard credit card orders. You can also order HelpGen by credit card if you are on CompuServe. Simply GO SWREG and register product number 3462.

Getting Started

Installing HelpGen

HelpGen comes with an installation program. To install the HelpGen files, do the following:

1. Put the diskette with the registered version in drive A or B

2. Select File | Run from the Program Manager menu.

3. Type A:SETUP.EXE or B:SETUP.EXE in the edit box and click on Ok.

Setup will install the registered program and create a program group (if one does not exist).

Running HelpGen

To run HelpGen, double click on the HelpGen Icon. The program will execute, and the HelpGen main window will be displayed (figure 1).

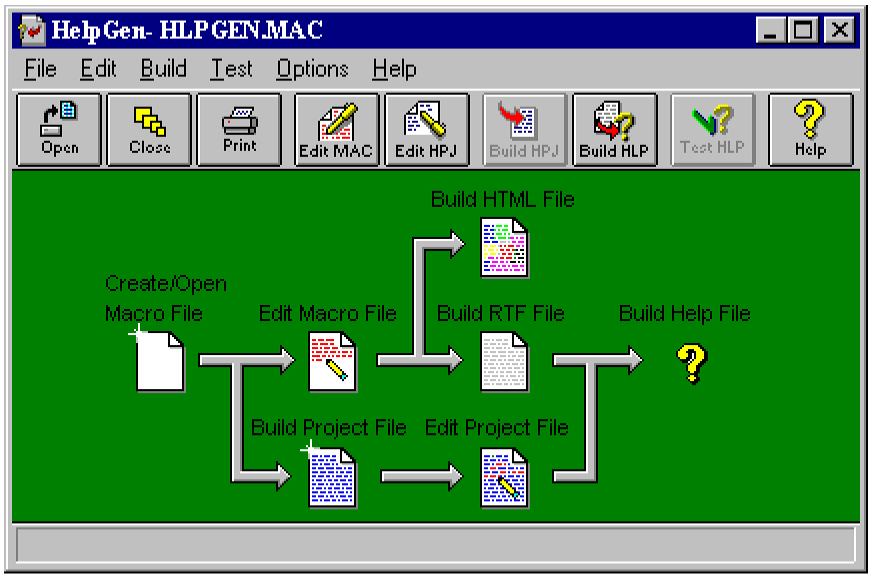


Figure 1. HelpGen

At this point, HelpGen will allow you to do three things: Open or create a macro file, change some of the program options, or activate HelpGen's online help. Note that this situation is reflected in the program's toolbar; only the first (open macro file) and last (help) icons are active. As you perform actions in HelpGen, the status of menu items and toolbar icons change with your actions.

Also note the flow diagram in the HelpGen window. This flow is the basic process you follow when using HelpGen to create a help file or an HTML file.

The Help File Generation Process

If you've ever tried to create help files using a regular word processor, you know what a pain that can be. First, you have to create the topic file in the word processor, using special footnotes, underlined, double-underlined and hidden text. Then you must save this information in Rich Text Format (RTF). Using the RTF file and a project file (that you must also create from scratch), you compile a help file using the DOS-based help compiler.

The basic steps in generating a help file using HelpGen are

1. Create the macro file using the **File | Open Macro File** function of HelpGen. If you don't have a project file, HelpGen can also create one for you.

2. Edit the Macro file using **Edit | Macro File**, adding your own topics by using the HelpGen macros.

3. Generate the HLP file from the MAC file and using the project file, by using the Build | HLP from MAC function. This will create an RTF file from the MAC file, and create an HLP file from the RTF file.

The final step in both instances is to test the generated help file. You can test the newly created help file directly from within HelpGen.

The HelpGen Menu and Toolbar

Actions in HelpGen are performed through the use of menu or toolbar items. There are six major menu sections and there are corresponding toolbar sections for all but one of the menu sections. These sections are File, Edit, Build, Test, Options and Help. All toolbar buttons have labels that help explain what the button does.

File - Items concerned with opening, closing and printing files, as well as exiting the program. The toolbar includes the File Open, File Close and File Print items.

Edit - Items concerned with editing macro and project files. The toolbar includes all these items except the Edit Bitmap and Edit Hotspot items.

Build - Items concerned with creating project files, RTF files and help files. The toolbar includes the Build Project file and Build HLP from MAC items.

Test - This item is concerned with testing help files or HTML files that has been generated with HelpGen. The Test HLP file item is included on the toolbar

Options - Items concerned with configuring the operation of HelpGen. None of these items are included on the HelpGen toolbar.

Help - Items concerned with helping the user to use HelpGen. The Help Contents item is included on the toolbar.

File | Open Macro File

The **File | Open Macro File** function allows you to open a macro file or create a new macro file. If you create a new macro file, HelpGen will use a template to create the file with the minimum macro items required to generate a help file. The Open Macro File toolbar icon will also activate this function.

When you open the macro file, HelpGen will also look for a corresponding project file. If the project file is present, HelpGen will enable the **Edit | Project File** function. If the project file is not present, HelpGen will enable the **Build | Project File** function.

File | Close Macro File

The **File | Close Macro File** function allows you to close the connection between HelpGen and the macro and project files. It does not update any files. It is the user's responsibility to save changes made to the files while they are being edited with Notepad or another text editor. The Close Macro File toolbar icon will also activate this function.

When the macro file is closed, all menu items concerned with using the file are disabled.

File | Print Macro File

The **File | Print Macro File** function allows you to produce a hard copy of the current macro file. The Print Macro File toolbar icon will also activate this function. Note that you can also print the macro and project files from the text editor that is used to update them.

File | Exit

The **File | Exit** function allows you to close all open files and shutdown the HelpGen program.

Edit | Macro File

The **Edit | Macro File** function activates the text editor of your choice, with the macro file already loaded into the editor. You may then make changes to the macro file. Be sure to save the changes in the text editor before you exit back to HelpGen. The Edit Macro File toolbar icon will also activate this function.

Edit | Project File

The **Edit | Project File** function activates the text editor of your choice, with the project file already loaded into the editor. You may then make changes to the project file. Be sure to save the changes in the text editor before you exit back to HelpGen. The Edit Project File toolbar icon will also activate this function.

Edit | Bitmap

The **Edit | Bitmap** function activates the bitmap editor of your choice. You may then make changes to any bitmap you are using with your help file. Be sure to save the changes in the bitmap editor before you exit back to HelpGen. The default bitmap editor is Windows Paintbrush, PBRUSH.EXE.

Edit | Hotspot

The **Edit | Hotspot** function activates the hotspot editor of your choice. You may create or modify a hotspot graphic for use in building your help file. Be sure to save the changes in the hotspot editor before you exit back to HelpGen. The default hotspot editor is Microsoft's SHED.EXE.

Build | Project File

The **Build | Project File** function uses items defined with the **Options | Project File** function to build a project file for the current macro file. The project file tells the help compiler how to compile an RTF file into a help file. The Build Project File toolbar icon will also activate this function.

Build | RTF File Only

The **Build | RTF File Only** function uses the macro file to create a Rich Text Format (RTF) file, which is the file that is used as input to the help compiler. The created file has the same filename as the macro file, and has an extension of .RTF.

Build | HLP File Only

The **Build | HLP File Only** function activates the help compiler. The help compiler uses the project file and the RTF file to generate a help file for the current macro file.

Build | HLP from MAC

The **Build | HLP from MAC** function automates the entire help file creation process into one menu item. The MAC file is processed to create an RTF file, and then the help compiler is activated to build the help file. The Build Help File toolbar icon will also activate this function.

Build | HTML File

The **Build | HTML File** function allows you to create a World Wide Web page from a .MAC file. The generated file will have a .HTM file extension. You can test this file by using any WWW browser program, such as Netscape or Mosaic.

Test | HLP file

The **Test | HLP file** function activates the newly created help file, so you can verify that it was constructed the way you wanted it. The Test toolbar icon will also activate this function.

Test | HTML file

The **Test | HTML file** function allows you to test an HTML file generated by HelpGen. When this menu item is invoked, HelpGen will execute the Web browser you have specified in the **Options | Directories** dialog.

Options | Directories

The **Options | Directories** function allows you to specify the locations of the ASCII text editor, help compiler, graphics editor and hotspot editor you will be using with HelpGen. These entries are saved in HelpGen's .INI file and are reloaded into HelpGen whenever you execute the program.

Options | Project File

The **Options | Project File** function allows you to specify values that will be inserted in any project file that you create with HelpGen. These values include the title, copyright notice, error log, icon file, bitmap root, compression, warnings and report status.

When you exit from HelpGen, some of these values are saved in HelpGen's .INI file, and are reloaded when you execute the program again. The saved values are the copyright notice, the error log, the compression value, the warnings value and the report value.

Help | Contents

The **Help | Contents** function activates HelpGen's on-line help function. The on-line help for HelpGen was generated using HelpGen, and the appropriate files that were used are included with HelpGen. The Help Contents toolbar icon will also activate this function.

Help | Search for Help on

The **Help | Search for Help** on function loads HelpGen's help file and activates the WinHelp Search function. This allows you to search for a particular topic relating to HelpGen.

Help | Help on Help

The **Help | Help on Help** function loads the Windows generic help file that explains how to use help files.

Help | About HelpGen

The **Help | About HelpGen** function displays a dialog box with program version and copyright information in it.

Testing Help Files

It is very important to test the help file that HelpGen has created for you. You must ensure that each topic is layed out the way that you envisioned it, and that each topic is reachable.

One way of doing this testing is to create a printout of the HelpGen macro file, and then use the printout as a test guide. Go through the help file, and for each topic you can access and that looks ok, check off that topic on the printout. If a topic is not displayed correctly or is not reachable, mark it on the printout. When you have finished, you can go back and edit changes to the macro file, re-generate the RTF file and re-compile the help file and start the testing all over again.

Internet HTML Support

HelpGen directly supports the creation of World Wide Web (WWW) home pages for the Internet by allowing you to generate HyperText Markup Language (HTML) files from your macro files. The HTML files may then be viewed using a WWW browser, such as Netscape, Internet Explorer or Mosaic.

Hints and Tips

• Don't put real tab characters in your macro file. If you want help text to be tabbed, use the RTF \tab command to tab. If you are creating an HTML file, you can't use tabs at all.

• There are many more RTF commands than are implemented in the HelpGen macro language or than are shown in the 'Some Useful RTF Commands' section of this manual. You can use any of them in a macro file, but you should be aware that many of the RTF commands have lasting effects. That is, once activated, they must be explicitly deactivated. Some of the macro commands will deactivate many of the RTF commands.

Help File Default Settings

Fonts

Headings - 24 point Arial (True Type font).
.[and .] commands - 10 point Courier.
All other text - 12 point Times New Roman (True Type font).

Font Numbers (TT is a True Type font)

0 - Arial (TT)	5 - MS Serif
1 - Times New Roman (TT)	6 - Symbol
2 - Courier	7 - Helv
3 - MS Sans Serif	8 - Arial MT
4 - Roman (TT)	

Text Colors

0 - Default color (black)	9 - Light Blue
1 - Blue	10 - Light green
2 - Green	11 - Light cyan
3 - Cyan	12 - Light red
4 - Red	13 - Light magenta
5 - Magenta	14 - Yellow
6 - Brown	15 - White
7 - Dark gray	16 - Pool table green
8 - Light gray	

Margins

Left margin - 1/8" (180 twips)
Right margin - 1/8" (180 twips)
.in macro - 1/2" left indent, first line indented 1/4"

Justification

Default is left justified (\ql).

Macro Markup Files

For Windows help files - HELPGEN.MMF
For HTML files - HTML.MMF

The HelpGen Macro Language

This section gives a detailed explanation of each of the HelpGen macro commands. The command syntax, explanation, examples and related commands are included.

Topic Macros

.begin_file Command

SYNTAX:

```
.begin_file
```

EXPLANATION:

Like the **.start** and **.startc** macros, the **.begin_file** is used to start a macro file. It provides the proper heading for the generation of the RTF file for the help compiler. Unlike the other two macros, it does not provide a default topic or a bitmap topic. Since there are no topics, **.begin_file** does not require a corresponding **.end** macro.

.begin_file can be used as the start of a multi-file help project. The first file in the series would have a **.start** ... **.end** or **.startc** ... **.end** combination, and the rest of the files would start with a **.begin_file**.

EXAMPLE:

```
.begin_file

.ent(next_topic,New Topic,My )
.end

.
.
.

.end_file
```

SEE ALSO:

.start, .startc

.startc Command

SYNTAX:

```
.startc(label,topic,bitmap,app,cntxtname,cntxtnum)
```

EXPLANATION:

This macro marks the beginning of a macro file. It produces an attractive header for the Table of Contents topic. *label* and *topic* are as described for the **.ent** macro. *bitmap* is the filename of a .BMP file generated by a bitmap editor such as Windows Paintbrush. *app* is the title that will appear in the header of the Table of Contents topic. You must provide a topic entry labeled 'icon' which will be jumped to when the user clicks on the icon in the Table of Contents header. The **.startc** statement also requires a corresponding **.end** to indicate the end of the table of contents.

cntxtname is the name of the context associated with this topic. *cntxtnum* is the sequence that this topic occupies within the context.

EXAMPLE:

```
.startc(main,Contents,DEMO.BMP,Help Demo,general,001)
This is the table of contents topic. It is ended by the next line.
.end

.ent(icon,Version,Help File )
This is a mandatory topic. We jump to here when the DEMO.BMP icon
is clicked.
.end

.end_file
```

SEE ALSO:

.start, .begin_file, .end_file

.start Command

SYNTAX:

```
.start(label,topic,bitmap_file,app_name)
```

EXPLANATION:

This macro marks the beginning of a macro file. It produces an attractive header for the Table of Contents topic. *label* and *topic* are as described for the **.ent** macro. *bitmap_file* is the filename of a .BMP file that is used as the jump icon. *app_name* is the title that will appear in the header of the Table of Contents topic. You must provide a topic entry labeled 'icon' which will be jumped to when the user clicks on the icon in the Table of Contents header. The **.start** statement also requires a corresponding **.end** to indicate the end of the table of contents.

EXAMPLE:

```
.start(main,Contents,DEMO.BMP,A Help Demo)
This is the table of contents topic. It is ended by the next line.
.end

.ent(icon,Version,Help File )
This is a mandatory topic.  We jump to here when the DEMO.BMP icon
is clicked.
.end

.end_file
```

SEE ALSO:

.startc, .begin_file, .end_file

.entc Command

SYNTAX:

```
.entc(label,topic,prefix,cntxtname,cntxtnum)
```

EXPLANATION:

This macro marks the start of a topic entry. *label* is used to refer to the topic in the hypertext navigation macros described below. *topic* is the name that will appear in the topic header and in the Search dialog box. *prefix* is a string that will appear as a prefix in the header. The prefix can be a single space, to signify an empty string. *cntxtname* is the name of the browse context that this topic will appear in. *cntxtnum* is the order number in which the topic will appear in the browse context.

EXAMPLE:

```
.entc(fileopen,Open File, ,menu,005)
File Open command text goes here.
.end

.entc(fileclose,Close File, ,menu,010)
File Close command text goes here.
.end

.entc(fileprint,Print File, ,menu,015)
Print File command text goes here.
.end
```

In the above example, the Open File topic will be browsed first, followed by the Close File topic and the Print File topic.

SEE ALSO:

.startc, .pentc

.ent Command

SYNTAX:

```
.ent(label,topic,prefix)
```

EXPLANATION:

This macro marks the start of a topic entry. *label* is used to refer to the topic in the jump macros described below, *topic* is the name that will appear in the topic header and in the Search dialog box; and *prefix* is a string that will appear as a prefix in the header (if the topic is "Frogs" and the prefix is "The", the header will be "The Frogs" but the entry in the Search dialog box will be "Frogs"). The prefix can be a single space, to signify an empty string.

EXAMPLE:

```
.ent(file,File Commands, )
.in
##.j(open,Open File).n
.un
.end

.ent(open,Open File, )
This command allows you to open an existing file.
.end
```

SEE ALSO:

.entc, .end

.pentc Command

SYNTAX:

```
.pentc(label,topic,cntxtname,cntxtnum)
```

EXPLANATION:

This macro is identical to the **.entc** macro, except that it creates popup topic boxes instead of topic pages. *label* is used to refer to the topic in the **.p** and **.p1** macros. *topic* should match the topic label in the corresponding **.p** or **.p1** macro. *cntxtname* is the name of the browse context that this topic will appear in. *cntxtnum* is the order number in which the topic will appear in the browse context.

EXAMPLE:

```
.pentc(defstr,String,defs,005)
Definition of a String here.
.end

.pentc(defint,Integer,defs,010)
Definition of an Integer here.
.end

.pentc(defflt,Float,defs,015)
Definition of a Float here.
.end
```

In the above example, the String topic will be browsed first, followed by the Integer topic and the Float topic.

SEE ALSO:

.startc, .entc

.pent Command

SYNTAX:

```
.pent(label,topic)
```

EXPLANATION:

This macro is identical to **.ent**, except that it creates popup topic boxes instead of topic pages. This type of popup box is usually used for definitions. *label* is used to refer to the topic in the **.p** and **.p1** macros described below. *topic* should match the topic label in the corresponding **.p** or **.p1** macro.

EXAMPLE:

```
.ent(size,Font Size,Changing )
This item allows you to change the size of the display font.  The
new size can be anywhere from 6 .p1(points) to 72 points.
.end

.pent(points,Points)
A sizing standard for text.  There are 72 points to the inch.
Therefore, 6 points is 6/72" = 1/12".
.end
```

SEE ALSO:

.pentc, .end, .p, .p1

.end Command

SYNTAX:

```
.end
```

EXPLANATION:

This macro marks the end of a topic entry that was started with **.start**, **.startc**, **.ent**, **.entc**, **.pentc** or **.pent**.

EXAMPLE:

See the examples for **.startc**, **.start**, **.ent**, **.entc**, **.pentc** and **.pent**.

SEE ALSO:

.startc, .start, .ent, .entc, .pentc, .pent

.top Command

SYNTAX:

```
.top(string)
```

EXPLANATION:

This macro is used inside of **.ent** and **.pent** entries to add *string* as another associated topic name to the Search dialog box.

EXAMPLE:

```
.pent(mouse,Pointing Devices)
.top(Trackball)
.top(Graphics Tablet)
.top(Mouse)
.top(Pointing Stick)
.top(Joy Stick)
A device that moves the graphical cursor around the screen and
that allows you to select objects.
.end
```

SEE ALSO:

.entc, .ent, .pentc, .pent

Hypertext Navigation Macros

.j Command

SYNTAX:

```
.j(label,string)
```

EXPLANATION:

This macro creates an underlined string and uses the *string* as a jump area. If the user clicks on a jump area, the topic represented by *label* is shown on the screen.

EXAMPLE:

```
.ent(layout,Page Layout,The )
How to Lay Out
.s
.j(pageno,Page Numbers).n
.j1(Headers).n
.j1(Footers).n
.end

.ent(pageno,Page Numbers, Laying Out )
Text for page number layout goes here.
.end

.ent(Headers,Headers,Laying Out )
Text for page header layout goes here.
.end

.ent(Footers,Footers,Laying Out )
Text for page footer layout goes here.
.end
```

SEE ALSO:

.entc, .ent, .j1

.j1 Command

SYNTAX:

```
.j1(label)
```

EXPLANATION:

This macro is identical to a .j with its *label* and string being identical. This macro is used for brevity.

EXAMPLE:

See the example for the **.j** command.

SEE ALSO:

.entc, .ent, .j

.bj Command

SYNTAX:

```
.bj(label,bitmap_name)
```

EXPLANATION:

This macro creates a graphical jump area using *bitmap_name* as the graphic. If the user clicks on the jump area, the topic represented by *label* is shown on the screen.

EXAMPLE:

```
.ent(filecmds,File Commands, )
.bj(new,new.bmp) New File
.bj(open,open.bmp) Open File
.bj(close,close.bmp) Close File
.end
```

```
.ent(new,New File, )
Text for new file command goes here.
.end

.ent(open,Open File, )
Text for open file command goes here.
.end

.ent(close,Close File, )
Text for close file command goes here.
.end
```

SEE ALSO:

.entc, .ent, .j, .j1, .bp

.bp Command

SYNTAX:

```
.bp(label,bitmap_name)
```

EXPLANATION:

This macro creates a graphical jump area using *bitmap_name* as the graphic. If the user clicks on the jump area, the topic represented by *label* is shown in a popup window.

EXAMPLE:

```
.ent(cmds,Edit Commands, )
.bp(cut,cut.bmp).n
.bp(copy,copy.bmp).n
.bp(paste,paste.bmp).n
.end

.pent(cut,Cut Command)
Cut command definition here.
.end

.pent(copy,Copy Command)
Copy command definition here.
.end

.pent(paste,Paste Command)
Paste command definition here.
.end
```

SEE ALSO:

.pentc, .pent, .p, .p1, .bj

.p Command

SYNTAX:

```
.p(label,string)
```

EXPLANATION:

This macro underlines *string* with a dotted line and uses it as a jump area. If the user clicks on the jump area, the topic represented by *label* is shown in a popup window. This is used to show definitions.

EXAMPLE:

```
.ent(cmds,Keyboard Commands,New )
The keyboard that supports Chicago has three extra keys;
the .p1(LWIN) key, the .p1(RWIN) key and the .p(application,APP)
key.
.end

.pent(application,APP)
When pressed, brings up the context menu at the current select
position.
.end

.pent(LWIN,LWIN)
Sets the focus to the Chicago User Interface. Same functionality
as the RWIN key, but uses a different scan code.
.end

.pent(RWIN,RWIN)
Sets the focus to the Chicago User Interface. Same functionality
as the LWIN key, but uses a different scan code.
.end
```

SEE ALSO:

.pentc, .pent, .p1, .bp

.p1 Command

SYNTAX:

```
.p1(label)
```

EXPLANATION:

This macro is identical to .p with its *label* and string being identical. This macro can be used for brevity.

EXAMPLE:

See the example for the **.p** command.

SEE ALSO:

.pentc, .pent, .p

.fj Command

SYNTAX:

```
.fj(file_name,string)
```

EXPLANATION:

This macro creates an underlined *string* and uses the string as a jump area. Instead of jumping to a topic in the current help (or HTML) file, the jump area jumps to the beginning of a new help (or HTML) file called *file_name*. The current file is closed and the new file is displayed.

EXAMPLE:

Windows Help File:

```
.ent(prgcmds,Programming Commands, )
.fj(c.hlp,C Commands).n
.fj(pascal.hlp,Pascal Commands).n
.fj(winapi.hlp,Windows API).n
.end
```

HTML pages:

```
.ent(places,Favorite Places,My )
.fj(http://www.coriolis.com,Coriolis Group).n
.fj(ftp://ftp.microsoft.com,Microsoft ftp Site).n
.end
```

SEE ALSO:

.bfj

.bfj Command

SYNTAX:

```
.bfj(file_name,bitmap_name)
```

EXPLANATION:

This macro creates a graphical jump area using the *bitmap_name* as the graphic. Instead of jumping to a topic in the current help (or HTML) file, the jump area jumps to the beginning of a new help (or HTML) file called *file_name*. The current file is closed and the new file is displayed.

EXAMPLE:

```
.ent(other,Other Help,Links to )
.bfj(delphi.hlp,delphi.bmp).n
.bfj(bcc.hlp,borcpp.bmp).n
.end
```

SEE ALSO:

.fj

Graphics Macros

.bmp Command

SYNTAX:

```
.bmp(bitmap)
```

EXPLANATION:

This macro inserts a named .BMP *bitmap* file on the current line. The bitmap is positioned as though it were just another character.

EXAMPLE:

```
.ent(open,Open Macro File, )
Use the .bmp(openrtf.bmp) Open Macro File button to open the file.
.end
```

SEE ALSO:

.bmpl, .bmpr

.bmpl Command

SYNTAX:

```
.bmpl(bitmap)
```

EXPLANATION:

This macro inserts a named .BMP *bitmap* file at the far left side of the current line. Any text following the .bmpl entry is wrapped around the bitmap.

EXAMPLE:

```
.ent(close,Close Macro File, )
.bmpl(closertf.bmp)Close the macro file and all associated files.
Disable any menu items relating to open files.
.end
```

SEE ALSO:

.bmp, .bmpr

.bmpr Command

SYNTAX:

```
.bmpr(bitmap)
```

EXPLANATION:

This macro inserts a named .BMP *bitmap* file at the far right side of the current line. Any text following the .bmpr entry is wrapped around the bitmap.

EXAMPLE:

```
.ent(keydisp,Key Information,Display )
.bmpr(keydisp.bmp)The key display area shows the scan codes that
will be sent for this particular key - unshifted, shifted, control
and alternate.
.end
```

SEE ALSO:

.bmp, .bmpl

.box Command

SYNTAX:

```
.box
```

EXPLANATION:

This macro draws a box around the current paragraph. It applies to all subsequent paragraphs up to the **.bend** command. Box drawing should only be done in normal text areas in **.ent** or **.pent** areas.

EXAMPLE:

```
.ent(about,About HelpGen, )
Display a box containing version information and copyright
information.
.n
.box
.b(NOTE:) The integer portion of the version number indicates a
major revision and the decimal portion indicates a minor revision.
.bend
.s
All dialog boxes use icon-style pushbuttons and graphical
elements.
.end
```

SEE ALSO:

.bend

.bend Command

SYNTAX:

```
.bend
```

EXPLANATION:

This macro completes the box drawing that was started with a **.box** command.

EXAMPLE:

See the **.box** example.

SEE ALSO:

.box

Documentation Macros

.rem Command

SYNTAX:

```
.rem(text)
```

EXPLANATION:

This macro allows comments to be inserted into the macro file.

EXAMPLE:

```
.rem(--------------------------------------)
.rem( Macro file for RENAULT.EXE help)
.rem(--------------------------------------)
.start(main,Contents,rnault.bmp,Renault Help)
.rem(Insert table of Contents here)
.end_file
```

SEE ALSO:

.com

.com Command

SYNTAX:

```
.com(text)
```

EXPLANATION:

This macro allows comments to be inserted into the .MAC file and to be copied into the generated .RTF file. IMPORTANT: Do not use this macro at the very beginning of your macro file, before the **.start**, **.startc** or **.begin_file** macros. The Microsoft help compiler doesn't like comments in front of the initial RTF information, so you should use the **.rem** macro at the top of your file.

EXAMPLE:

```
.com(+========================+)
.com(| The Open File menu item |)
.com(+========================+)
.ent(fileopen,Open File, )
Open File description goes here.
.end
```

SEE ALSO:

.rem

Text Formatting Macros

.in Command

SYNTAX:

```
.in
```

EXPLANATION:

This macro starts an indented text section, which usually contains a list of items. The indented section is a hanging indent, which means that the first line is indented less than the following lines. Each beginning line will look good if it begins with either a number or a bullet.

EXAMPLE:

```
.ent(sellcars,Sell Cars,How to )
.in
#n(1)Find a customer..n
#n(2)Convince them this is the car of their dreams, no matter how
bad it is..n
#n(3)Close the sale.  Get their John Hancock on the bottom line..n
.un
.end
```

SEE ALSO:

.un, ##, #b, #n

.un Command

SYNTAX:

```
.un
```

EXPLANATION:

This macro ends ('undents') an indented text section.

EXAMPLE:

See the example for the **.in** command.

SEE ALSO:

.in

## Command

SYNTAX:

```
##
```

EXPLANATION:

This macro represents a bullet for a list item. It should be followed immediately by text.

EXAMPLE:

```
.ent(advantages,Quilt Advantages, )
.s
##Uses no electricity..n
##All natural construction..n
##Reduces number of blankets needed..n
.end
```

SEE ALSO:

.un, #b, #n

#b Command

SYNTAX:

```
#b(text)
```

EXPLANATION:

This macro represents a list item that starts with a bullet and the given *text* in boldface. It should be followed immediately by text.

EXAMPLE:

```
.ent(glossary,Glossary, Keyboard )
.s
#b(Autorepeat)If held down a key will begin to send its code
repeatedly..n
#b(Mode key)Modifier keys, used in conjunction with normal keys.
Includes shift, control and alternate..n
#b(Scan code)A byte value or values sent to the computer..n
.end
```

SEE ALSO:

.un, ##, #n

#n Command

SYNTAX:

```
#n(1)
```

EXPLANATION:

This macro represents a numbered list item. It should be followed immediately by text.

EXAMPLE:

See the example for the **.in** command.

SEE ALSO:

.b Command

SYNTAX:

```
.b(string)
```

EXPLANATION:

This macro presents the given *string* in bold face type.

EXAMPLE:

```
.ent(startup,Engine Startup, )
Always ensure there is no gas fume build-up in the bilge. .b(THIS
IS VERY IMPORTANT.).n
Switch on the ignition and press the start button..n
.end
```

SEE ALSO:

.i, .bi, .bu, .iu

.i Command

SYNTAX:

```
.i(string)
```

EXPLANATION:

This macro presents the given *string* in italic type.

EXAMPLE:

```
.ent(chars,Characters,Game )
Your opponents in the game consist of .i(trolls), rabid .i(dogs)
and defenseless .i(kittens).
.end
```

SEE ALSO:

.b, .bi, .bu, .iu

.bu Command

SYNTAX:

```
.bu(string)
```

EXPLANATION:

This macro presents the given *string* in bold face, underlined type.

EXAMPLE:

```
.ent(remov,Removing the Lightbulb, )
Steps in removing a bulb:.s
#n(1).bu(UNPLUG THE LAMP).n
#n(2)Remove the shade.n
#n(3)Unscrew the bulb.n
.end
```

SEE ALSO:

.i, .b, .bi, .iu

.bi Command

SYNTAX:

```
.bi(string)
```

EXPLANATION:

This macro presents the given *string* in bold face, italic type.

EXAMPLE:

```
.ent(refs,References, )
.bi(Anatomy of a Murder), Gibbons, Euell,.iu(Harper & Row), 1963.n
.end
```

SEE ALSO:

.b, .i, .bu, .iu

.iu Command

SYNTAX:

```
.iu(string)
```

EXPLANATION:

This macro presents the given *string* in italic, underlined type.

EXAMPLE:

See the example for the **.bi** macro.

SEE ALSO:

.b, .i, .bu, .bi

.c Command

SYNTAX:

```
.c(color_number,string)
```

EXPLANATION:

This macro presents the given *string* in the *color_number* color. The color numbers are

0 - Default color (usually black)	9 - Light blue
1 - Blue	10 - Light green
2 - Green	11 - Light cyan
3 - Cyan	12 - Light red
4 - Red	13 - Light magenta
5 - Magenta	14 - Yellow
6 - Brown	15 - White
7 - Dark gray	16 - Pool table green
8 - Light gray	

NOTE: If the color you are using for text is too close to the help file background color, Winhelp will substitute the default color for that color.

EXAMPLE:

```
This is .c(12,extremely) important..n
```

SEE ALSO:

.[Command

SYNTAX:

```
.[
```

EXPLANATION:

This macro marks the beginning of a region of text that will appear in a bold, fixed-width font. This is suitable for showing program examples or tables. NOTE: Since RTF files use the open and close curly braces as part of their syntax, if you want to display them in your help file, you must 'escape' them by putting a backslash in front of them. This is shown in the example below.

EXAMPLE:

```
.ent(progctl,Control,Program )
The program is controlled with a message
router:
.s
.in
.[
switch(wParam).n
\{.n
\tab case ID_OPEN: . . . .n
\tab \tab break;.n
\tab case ID_CLOSE: . . . .n
\tab \tab break;.n
\}.n
.]
.un
.end
```

SEE ALSO:

.]

.] Command

SYNTAX:

```
.]
```

EXPLANATION:

This macro marks the end of a region that was started with **.[**. There should be a **.]** for every **.[**.

EXAMPLE:

See the example for the **.[** command.

SEE ALSO:

.[

.n Command

SYNTAX:

```
.n
```

EXPLANATION:

This macro is used at the end of a line, or on a line of its own, to indicate the actual end of the line. When **.n** is not used, separate lines are simply run together and used to fill whatever WinHelp window width the user has chosen.

EXAMPLE:

See the example for the **.b** command.

SEE ALSO:

.s

.s Command

SYNTAX:

```
.s
```

EXPLANATION:

This macro is used between lines to skip a space. If **.s** is used, then don't use **.n**.

EXAMPLE:

See the example for the **.start** command.

SEE ALSO:

.n

Help Macro Access Macros

.macro Command

SYNTAX:

```
.macro(macro,string)
```

EXPLANATION:

This macro underlines the *string* text as if a jump is there, but executes a WinHelp *macro* instead of jumping. Any WinHelp macro with no arguments may be executed. The WinHelp macros include:

About - Display the WinHelp About box.
Annotate - Display the WinHelp Annotation dialog box.
Back - Display the previously viewed topic.
BookmarkDefine - Display the Bookmark Define dialog box.
BookmarkMore - Display the Bookmark dialog box.
BrowseButtons - Display the browse buttons on the WinHelp button bar.
Contents - Display the Contents topic in the current help file.
CopyDialog - Display the Copy dialog box.
CopyTopic - Copies the currently displayed topic to the Windows Clipboard.
Exit - Exit from WinHelp and the current help file.

FileOpen - Display the File Open dialog box.
FloatingMenu - Display the floating menu, if it is defined.
HelpOn - Display the Help On Help file.
HelpOnTop - Toggle the WinHelp Always On Top menu item.
History - Display the History Window.
JumpHelpOn - Displays the Help On Help file.
Next - Display the next topic in the current browse context.
Prev - Display the previous topic in the current browse context.
Print - Print the currently displayed topic on the printer.
PrinterSetup - Display the Printer Setup dialog box.
ResetMenu - Reset the WinHelp menu to its default.
Search - Display the Search dialog box.

EXAMPLE:

```
.ent(b213,Bearings, Roller)
The roller bearings are an important part of the system..s
.macro(Print,Print) this topic..n
.end
```

SEE ALSO:

.bmacro

.bmacro Command

SYNTAX:

```
.bmacro(macro,bitmap_name)
```

EXPLANATION:

This macro creates a graphical jump area using *bitmap_name* as the graphic. When the user clicks on the graphic, the WinHelp *macro* will execute, instead of performing a jump. See the **.macro** command for a list of executable WinHelp macros.

EXAMPLE:

```
.ent(rm41,Bearings,More on )
This is the More on Bearings    topic..s
.bmacro(Prev,previous.bmp) Previous subject..n
.end
```

SEE ALSO:

.macro

Miscellaneous Macros

.end_file Command

SYNTAX:

```
.end_file
```

EXPLANATION:

This macro marks the end of the macro file.

EXAMPLE:

See the example for the **.start** command.

SEE ALSO:

.start

Some Useful RTF Commands

The RTF commands described below can be used directly inside of a macro file. They help with help file formatting. NOTE: the \brdrxx commands described below all help modify the box command, but you can only use one of them at a time - they are mutually exclusive.

\brdrdb Command

EXPLANATION:

This command changes the border line style to a double line. It is used as a modifier to the **.box** command.

EXAMPLE:

```
.box
\brdrdb
This text will be displayed in a box with a double line border.
.bend
```

\brdrs Command

EXPLANATION:

This command changes the border line style to a single line (this is the default). It is used as a modifier to the **.box** command.

EXAMPLE:

```
.box
\brdrth
Text is displayed in a box with a hick border.
\brdrs
Text is displayed in a another box with a single line border.
.bend
```

\brdrsh Command

EXPLANATION:

This command changes the border line style to shaded style. It is used as a modifier to the **.box** command.

EXAMPLE:

```
.box
\brdrsh
This text will be displayed in a box with a shaded border.
.bend
```

\brdrth Command

EXPLANATION:

This command changes the border line style to a thick line. It is used as a modifier to the **.box** command.

EXAMPLE:

See the example for the **\brdrs** command.

\li Command

EXPLANATION:

This command changes the left margin indentation to the value that immediately follows the command. The value is expressed in twips (1440 twips to the inch). To reset the margin to its default, use \li180.

EXAMPLE:

```
.[
\li720
\ri720
This sentence will be indented on the left
and right margins by 1/2".
.]
\li0
\ri0
```

\ri Command

EXPLANATION:

This command changes the right margin indentation to the value that immediately follows the command. The value is expressed in twips (1440 twips to the inch). To reset the margin to its default, use \ri180.

EXAMPLE:

See the example for the **\li** command.

\ql Command

EXPLANATION:

This command changes the paragraph style to left justified. This is the default paragraph style.

EXAMPLE:

```
\qc
This paragraph will have every line centered in the help window.
.s
\ql
Now we are back to the default left justified paragraph.
```

\qr Command

EXPLANATION:

This command changes the paragraph style to right justified.

EXAMPLE:

```
This paragraph is normally left justified.
.s
\qr
And this paragraph is right justified.
.s
```

\qj Command

EXPLANATION:

This command changes to the paragraph style to fully justified.

EXAMPLE:

```
\qj
The quick brown fox normally jumps over the lazy dog's back in a
fully justified fashion.
.s
```

\qc Command

EXPLANATION:

This command changes the paragraph style to centered lines.

EXAMPLE:

See the example for the **\ql** command.

\tab Command

EXPLANATION:

This command places a tab character in the macro and in the generated RTF file.

EXAMPLE:

See the example for the **.[** macro language command.

\'hh Command

EXPLANATION:

This command allows you to put extended characters into a help file. Some commonly used extended characters are

\'a9	=>	©
\'ae	=>	®
\'bc	=>	¼
\'bd	=>	½
\'be	=>	¾
\'99	=>	™

NOTE: the trademark character may not work correctly, depending on the help compiler.

EXAMPLE:

```
\'a9 1994 Rimrock Software.  All rights reserved..n
Distributed on 3\'bd double density floppy disk.
```

A HelpGen Tutorial

Planning Your Help File

Planning the layout and contents of the help file is probably the most important part of creating online help. A poorly conceived and implemented help file is almost worthless to a user. They may not be able to find anything they are looking for in the help file, and if they do find the correct item, it may not help with their problems.

This part of help file generation doesn't have much to do with the actual mechanics of creating the help file, but it is important, nevertheless. Some basic steps in planning a help file are detailed below.

Outline the Help Subjects: Plan to have a few major sections, and many minor sections. For instance, one way to organize is to use the basic outline of your user manual. An example of another way is shown below:

```
Introduction
How To
Menu Items
      File
            Open
            Close
            Exit
      Edit
            Cut
            Copy
            Paste
```

This outline can go down as many levels as you want, but anything more than about 4 levels is too complicated for a normal user to navigate. Once you have this basic outline, you essentially have each of the topics that will be discussed in the help file. Each level of the outline except the lowest level corresponds to a single display page in the help file. Each topic on the lowest level of the outline corresponds to a single display page in the help file.

Continuing with our example above, then, the Table of Contents page would contain the
Introduction, How To and Menu Items topics. If a user clicked on Menu Items, they would move to a page titled Menu Items, which contains File and Edit topics. Clicking on the File item, they would move to a page containing the Open, Close and Exit topics. And finally, clicking on Open would move to a page that explains the File | Open menu item.

Create Topic Tags: Each item on each level of the outline should have a single word 'tag' that can be used to reference it to. Go through the outline and create a tag for each outline item. We will use these when we add the topics to the macro file. NOTE: These tags should be unique, so that the help compiler won't get confused about where a specific jump should actually jump to.

```
Introduction - intro
How To - howto
Menu Items - menuitems
      File - file
            Open - fileopen
            Close - fileclose
            Exit - fileexit
      Edit - edit
            Cut - editcut
            Copy - editcopy
            Paste - editpaste
```

Flesh Out the Outline: What you are really doing is writing a user manual that will be read on-line in a non-linear (hypertext) fashion. Much of what you would normally say in a manual will be included in the on-line help.

Write explanations for each of the items on the lowest level of your outline. Use graphics (.BMP files) and/or examples liberally to explain the subject (a picture really is worth 1000 words).

Mark Definitions: When you have completely written all the text for your help file, go back and circle the terms that you think the user will have trouble with, and that need to be defined. These terms will form the basis of your glossary. The first time such a term is encountered in any topic, it should be marked with a **.p** or **.p1** and should be explained in a **.pent** entry. Create a topic tag for each of the terms.

Create the Macro File: You are now ready to create and edit a macro file for your on-line help.

Creating the Macro File

Now that you have your help topics outlined and fleshed out, you can create and edit a macro file for your help.

Continuing with the example we started in Planning Your Help File, let's create an initial macro file. Click on **File | Open Macro File** and type in 'MYHELP' for a file name. HelpGen will ask if you wish to create this file. Click on the 'Yes' button and MYHELP will be created.

Now let's do some editing on this file. Click on **Edit | Macro File**. The text editor should appear, with the newly created MYHELP.MAC displayed in it. Let's use the DEMO.BMP file for the icon, change the table of contents header from xxxx to Table of Contents and add the first level of our outline to the table of contents topic:

```
.rem(=======================================)
.rem(MYHELP.MAC)
.rem( )
.rem(Created by HelpGen Help Generator v1.20)
.rem(Copyright c 1994 by Rimrock Software)
.rem(All rights reserved.)
.rem(======================================)
.start(main,Contents,DEMO.BMP,Table of Contents)
.top(Help)
.top(Table of Contents)

.s
.in
##.j(intro,Introduction)
##.j(howto,How To)
##.j(menuitems,Menu Items)
.un
.end

.ent(icon,Information,Icon )
This icon topic .b(MUST) be present.  It is
activated by clicking on the icon in the
Table of Contents.  You may place your own
text here.
.end

.end_file
```

Note that we dress up the topics by indenting them (**.in** and **.un**) and by putting a bullet in front of them (##). We shouldn't compile this until we add the entries for the three jumps we just added:

```
.start(main,Contents,DEMO.BMP,Table of Contents)
.top(Help)
.top(Table of Contents)

.s
.in
##.j(intro,Introduction)
##.j(howto,How To)
##.j(menuitems,Menu Items)
.un
.end
```

```
.ent(icon,Information,Icon )
This icon topic .b(MUST) be present.  It is activated by clicking
on the icon in the Table of Contents.  You may place your own text
here.
.end

.ent(intro,Introduction, )
.end

.ent(howto,How To, )
.end

.ent(menuitems,Menu Items, )
.end

.end_file
```

Now we can compile it if we want to, but let's continue. Next, we add the second level of the outline. In this case, it means adding to the Menu Items topic, and adding new topics for the third outline level:

```
.ent(menuitems,Menu Items, )
.in
##.j(file,File)
##.j(edit,Edit)
.un
.end

.ent(file,File Menu,The )
.end

.ent(edit,Edit Menu,The )
.end

.end_file
```

We continue to add new jumps and new topics until we reach the last level of the outline, where all the text will be entered.

Using HelpGen Macros and RTF Commands

We have already used many of the basic HelpGen macros in creating our basic macro file. The remaining macros and RTF commands are used to primarily dress up the help file, to make it more interesting and easier to use for the user. A typical example of this is shown for the File | Open part of our example:

```
.ent(file,File Menu,The )
.in
##.j(fileopen,Open)
##.j(fileclose,Close)
##.j(fileexit,Exit)
.un
.end

.ent(fileopen,Open File, )
The .b(File | Open) menu item allows you to open an existing
file..n.n
.bmpl(OPENRTF.BMP)  You may also use the Open File button in
the .p1(toolbar) to select this item..n
.box
\brdrsh
.b(NOTE:) You may only have one open file at a time.
.bend
.end
```

In this topic, note that we have emphasized the menu item by putting its name in bold type (**.b** command). We have included a graphic of the pushbutton connected with this item (**.bmpl** command) at the left margin of the help file. We also have included a note that is displayed in a shaded box (**.box**, **\brdrsh**, **.bend** commands).

Creating the Project File

When we are ready to compile our help file, we will need a project file to tell the compiler how to compile it. Follow these steps when creating a project file:

Select the **Options | Project File** menu item. Make sure that all the listed options are set the way you want them. Keep in mind that in the shareware version of HelpGen, you can't change the copyright notice.

Select the **Build | Project File** menu item. Answer 'Yes' to the question and a project file will be built for you. The **Build | Project File** menu item will now be disabled, and the **Edit | Project File** menu item will be enabled. You may now edit the project file, if there are any changes that need to be made (there shouldn't be any).

Generating the Help File

Now select the **Build | HLP** from MAC menu item and HelpGen create an RTF file from the MAC file and will then invoke the help compiler that you have told it to use (see **Options | Directories**)).

Testing the Help File

After the help compiler generates a help file for you (assuming that it did so with no errors), you can test the file. To do this, select the **Test | HLP** file menu item. HelpGen will invoke the WinHelp help engine, using your help file. You can then proceed with the testing.

To thoroughly test your help file, you should adopt a structured approach. Use the outline you developed and start at the top of it. Select each topic and verify that you jump to the proper page, or that the proper popup box appears. For each page, verify that the page is layed out exactly the way you want it. Mark any discrepancies on your outline. When you have finished, re-edit the macro file to make any necessary changes to it. Then re-make the help file. Test again to verify all corrections were performed.

Testing is an iterative process, so don't be impatient. Just keep plugging away until everything is fixed.

Appendix A HelpGen Macro File Structure

HelpGen Macro Files consist of distinct blocks of text. These blocks are marked by HelpGen macros. The first block is the **.start**/**.end**/**.end_file** block:

```
.start(......)
      Table of Contents stuff goes here....
.end

      Rest of macro file goes here....

.end_file
```

The rest of the macro file is broken up into **.ent**/**.end** and **.pent**/**.end** blocks:

```
.start(......)
      Table of Contents stuff goes here....
.end

.ent(.....)
      Topic stuff goes here....
.end

.ent(.....)
      Topic stuff goes here....
.end

.ent(.....)
      Topic stuff goes here....
.end

.pent(.....)
      Popup stuff goes here....
.end

.pent(.....)
      Popup stuff goes here....
.end

.end_file
```

Note the use of white space to separate the various blocks. You may also separate the blocks with remarks:

```
.rem(===== Beginning of topic 1 =====)
.ent(.....)
      Topic stuff goes here....
.end
.rem(===== Beginning of topic 2 =====)
.ent(.....)
      Topic stuff goes here....
.end
```

In order to display each of these blocks, you need to have jumps (.j or .j1 for the .ent/.end blocks and **.p** or **.p1** for the **.pent**/**.end** blocks) somewhere in your macro file. Since the help file begins at the Table of Contents, there should definitely be some jumps in that topic.

All the rest of the HelpGen macros are basically window dressing for the topic blocks. See the various macro commands for examples of how to use them.

Appendix B Project File Structure

The Help Project File is a text file that contains information the help compiler needs to build a Help file. The project file instructs the help compiler on what files to build, where they are located, and how you want the finished help file to appear. It is your interface to the help compiler.

Project files can contain up to nine sections. Each section consists of related settings. The nine section headings are each enclosed in square brackets, like the sections of a Windows initialization file. Comments in the project file start with a semicolon at the beginning of the line. The nine sections are:

[OPTIONS] - Specifies various program options that control how the help file will be built. This section must be the first section in the project file, if it is present.

[FILES] - This section specifies the topic files (RTF files) that will be included in the help file. This is a required section.

[BUILDTAGS] - This optional section specifies valid build tags. Build tags can help you to conditionally compile a help file, or build different versions of the same help file.

[CONFIG] - This section specifies any customized menus and buttons in the help file. It also registers any .DLL functions that are used as macros in the help file. This section is only required if you use any of these features.

[BITMAPS] - This section specifies any bitmap files that are included in the help file. This section is required unless you have specified a path for any bitmap files using the ROOT or BMROOT options.

[MAP] - This optional section associates context strings with context numbers from a Windows application. This helps you to create context sensitive links between your application and the help file.

[ALIAS] - This optional section assigns one or more context strings to the same topic.

[WINDOWS] - This section controls the appearance of the help file window that displays your help. This section is required if you want to customize the appearance of the help file window, or if your help system uses secondary windows.

[BAGGAGE] - This optional section lists files (usually multimedia files) that you want to place within your help file.

HelpGen can generate a project file for you. The generated file contains five of the nine possible sections - [OPTIONS], [FILES], [BITMAPS], [MAP] and [WINDOWS]. The settings that are included in a HelpGen generated project file are detailed below.

[OPTIONS] Section

BMROOT

This item indicates the directory that is used to store all the bitmap files used in building the help file, if the files are not in the current directory. This item is commented out in a normal HelpGen project file. Syntax for this item is BMROOT=D:\PATH.

COMPRESS

This item specifies how the help compiler should compress the help file as it is built. The syntax for this entry is COMPRESS=0 (for no compression), COMPRESS=MEDIUM (for 40% compression) and COMPRESS=HIGH (for 50% compression).

COPYRIGHT

This item contains a string that is added to the help file's About dialog box. It can use up to 35 characters. Using the default for this item, the syntax is COPYRIGHT=© 1994 Rimrock Software.

ERRORLOG

This item specifies a file where all the error messages generated by the help compiler will be placed. You can use a text editor to check this file for errors. The syntax for this item is ERRORLOG=D:\PATH\FILENAME.EXT. The default is ERRORLOG=ERRORLOG.TXT.

ICON

This item specifies the icon that will be used when your help file is minimized. This entry overrides the normal question mark icon for a help file. HelpGen produces a project file that has this item commented out.

REPORT

This item determines the type of error messages that will be displayed. Default for this item is REPORT=ON.

WARNING

This item determines the level of error messages that will be displayed by the help compiler. WARNING=1 means only display the most severe errors, WARNING=2 means display a medium amount of messages, and WARNING=3 means display all messages.

TITLE

This item tells the help compiler what text should be displayed on the help files caption bar. The syntax is TITLE=Text to Place on Caption Bar.

[FILES] Section

The [FILES] section contains a list of topic files (RTF files) that will be included in the help file. When HelpGen generates a project file, there will be only one entry in the list - a file with the same name as the project file, and an extension of .RTF. The entries can be relative (e.g., MYHELP.RTF) or they can be absolute (e.g., E:\DEV\MYPROJ\MYHELP.RTF).

[BITMAPS] Section

The [BITMAPS] section contains a list of the graphics files used in the help file. This section is not required if the files are in the directory contained in the BMROOT directory, or if they are in the current directory (HelpGen assumes the latter). The entries can be relative (e.g., FILERTF.BMP) or they can be absolute (e.g., E:\DEV\MYPROJ\FILERTF.BMP).

[MAP] Section

The [MAP] section contains a list of jump tags that are to be used for contextual help selection. Each tag has a corresponding unique number connected to it. The help file uses the tag, and your program uses the number to access that tag (see Appendix E, Invoking a Help File from a Windows Program). A typical [MAP] section might look like this:

```
[MAP]
#define fileopen        1000
#define fileclose       1010
#define fileprint       1020
#define progexit        1030
#define editcut         2000
#define editcopy        2010
#define editpaste       2020
```

[WINDOWS] Section

The [WINDOWS] section allows you to control the appearance of the help file window. You can control the placement, size and color of both the main help file window and any secondary windows. The main help file window is controlled with the following statement:

```
main="Caption",(h-pos,v-pos,width,hite),state,bkgnd-color,nonscrl-
color,on-top-state
```

The color entries are sets of RGB entries. For example, a bkgnd-color of green would be (0,128,64).

If one of the entries is missing, the comma must still be present. The default HelpGen project file entry is

```
main=,,0,,(0,128,64)
```

Appendix C HelpGen Macro File Structure

The HelpGen macros have been separated from the program and placed in a separate ASCII text file. A similar file has been generated for HTML files. These files are named HELPGEN.MMF and HTML.MMF.

These .MMF files have a specific arrangement. Macros are defined in the file with the longest macro names first, down to the shortest last. This allows macros whose names differ by a single character to be distinguished by the parser.

Macros are separated by a single tilde (~), so you can't use a tilde in any macro definition. You can have up to 255 macros in the .MMF file, and they can be any length.

The basic format of a macro is

```
macro_name=macro_definition~
```

The macro definition can be placed on as many lines as are necessary. Look at the **.start** macro for a typical multiline definition.

Within a macro definition, you can have up to 9 replacement arguments. They are designated as %1 through %9. These are the comma-separated fields that most macros use. When the macro definition replaces the macro name, these fields are also replaced with the real values you supply when you invoke the macro. For instance, the jump macro definition is

```
.j={\uldb %2}{\v %1}~
```

Note that the replacement arguments are %1 and %2. And this is exactly what you must supply in your .MAC file, as in

```
.j(filopen,File Open)
```

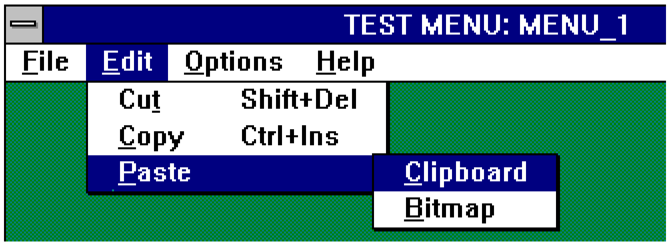
If you look closely at the .MMF files, you will see that HelpGen removes much of the complexity from .RTF file generation.

NOTE: You can change the .MMF files however you want. You should keep a copy of the original somewhere for when you mess up, though. Rimrock Software takes no responsibility for user-created or modified macros.

Appendix D Creating Bitmaps with Multiple Jumps to Topics

If you don't have the Microsoft hot spot editor to help you create jump topics on a bitmap, you can use HelpGen's **.bmp** and **.bj** commands to create an area in your help file that will look like one big bitmap, but is in actuality composed of many pieces. This is useful when you want to explain pulldown menus or program areas to a user. You can display a bitmap of the program area and allow the user to point at the item that they want help on.

For an example of how to do this, let's look at a pulldown menu. The pulldown menu looks like this in the program:



We would like to display this menu in our help file and allow the user to point at the 'Cut', 'Copy', 'Clipboard' and 'Bitmap' items.

The first step in this process is to capture the menu. Start up your program, then select the menu item(s). When everything you want to display in your help file is showing, press the Print Screen key on your keyboard. This will copy the entire display contents into the clipboard.

Next, bring up Windows Paintbrush. Maximize its window, so your menu item has a good chance of being displayed. Make sure that the maximum picture for your display is selected by choosing the Paintbrush **Options | Image Attributes** menu item, selecting 'pels', selecting 'Default' and then 'Ok'.

Now we can get the menu. **Select Edit | Paste**, and the captured display will appear in the paintbrush screen.

To prepare for clipping out the menu from the captured display, we should select **View | Cursor Position**, so we can do some calculations about how big an image we have. Then select the 'Pick' drawing tool (scissors and box icon).

Now we can clip out the menu. Move to the upper left hand corner of the part you want to clip. Write down the coordinates for this corner. Then press and hold the left mouse button and move to the lower right corner of the part you want to clip. Note the coordinates and release the mouse button. Then use the **Edit | Copy** menu item to copy this piece to the clipboard.

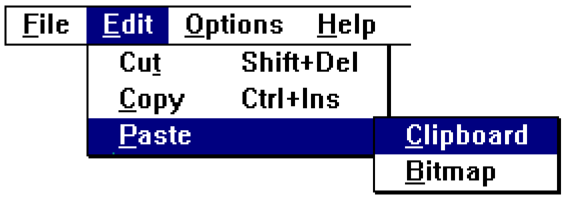
We will now use the coordinates that you wrote down to calculate how big a bitmap we need to just fit the piece in the clipboard. To do this, subtract the final coordinates from the initial coordinates and add 1. For instance, if you started at 46,19 and ended at 155,174, then

155 - 46 + 1 = 110
174 - 19 + 1 = 156

The new bitmap in this example will be 110 by 156.

After you have calculated the bitmap size, go back to **Options | Image Attributes** and enter the size in the dialog box. When you click on Ok, Paintbrush will ask you a couple of questions. We don't care about the current image in Paintbrush (the piece we care about is on the clipboard), so answer 'Ok' to the first question and 'No' to the second. Paintbrush will clear its drawing area and resize it.

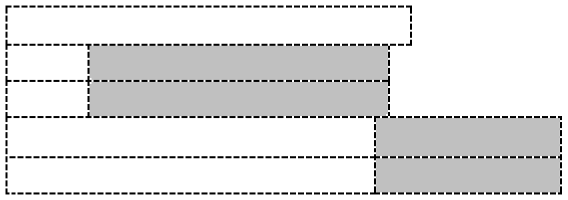
Now paste the clipboard back into Paintbrush. In our example, we will end up with:



Note that we also cleaned up around the menu, eliminating all the background clutter that the menu lays over. This is what we want the bitmap to look like in the help file.

The critical part of the operation comes next. In order to get several distinct topic jump areas, we have to split up this bitmap into many pieces. When we display the pieces in the proper order, the image will look as if it were not split at all.

We will split our example into 9 pieces (compare the picture below with the last picture):



The shaded pieces (topic jumps) will be displayed using the **.bj** macro and the unshaded pieces will be displayed with the **.bmp** macro.

It is important to note why we split it up like this. Since bitmaps are displayed in a help file just like text, we have layed out our split-up menu just like text - that is, it is left justified (ragged right margin) and each bitmap on the same line is the same height.

Now we have to cut up the menu and create 9 separate bitmap files with the 9 pieces. The easiest way to do this is to use two instances of the Paintbrush program.

Go into the Paintbrush display and pick the top piece of the menu. Be sure to note the start and end coordinates as we did before. Use the **Edit | Copy** item to copy the piece to the clipboard.

Now start up a second copy of Paintbrush. Change the image size to be the same as for the piece you just clipped (final - initial + 1) by using **Options | Image Attributes**, then paste the clipped image into Paintbrush. Now all you have to do is save the piece to a file using the **File | Save** as command.

Flip back to the first instance of Paintbrush and pick the next menu piece. Make sure that you don't overlap with the image you just saved. For example, if our last image was copied from 0, 0 to 167, 19, then the next image will start at 0,20. Copy the piece to the clipboard and then paste it into the second Paintbrush instance. Save it to a file. Continue this process until all the pieces are saved into separate files.

Now that we have the menu split up, we will want to add the display macros to our macro file. If the files for this example are named EDIT1.BMP through EDIT9.BMP, then the macro file might look something like this:

```
.ent(edit,Edit Menu, )
.in
.bmp(edit1.bmp).n
.bmp(edit2.bmp).bj(cut,edit3.bmp).n
.bmp(edit4.bmp).bj(copy,edit5.bmp).n
.bmp(edit6.bmp).bj(clip,edit7.bmp).n
.bmp(edit8.bmp).bj(bitm,edit9.bmp).n
.un
Point to the item you want help with and click the left mouse
button.
.end
```

You must try this to appreciate how nicely it works. It is very intuitive for the user, since they point at an actual object instead of a description of an object.

To summarize,

1. Capture the image by using the Print Screen key, then paste it into Paintbrush.

2. Clean up around the piece of the image you will be using in your help file.

3. Divide your image up into pieces ('lines' and 'words').

4. Pick an image piece, noting its coordinates. Copy it to the clipboard.

5. Using a second instance of Paintbrush, resize the image area and paste the clipboard into the image area.

6. Save the image piece to a file.

7. Go back to step 4, until all the image pieces are saved.

8. Add the images to your macro file. Rebuild the macro file, recompile the help file and enjoy the fruits of your labor.

Appendix E Invoking a Help File from a Windows Program

After you have created your help file, you need to marry it to your Windows program. There are several ways to access the help file:

- General access (help file Table of Contents)
- Search for a topic and display it
- Accessing help on using a help file
- Specific (contextual) topic access
- Close the help file when your application closes

This appendix covers all these of these access methods, and provides examples in both C and Delphi (Pascal). For the Delphi examples, you should hook the help file up to your Delphi program using Delphi's **Options | Project** menu item and selecting the Application notebook tab.

General Help Access

This is the easiest way to access a help file. For the C example, you need to have a string containing the name (and optionally, the path) of the help file and a handle to your current window. You then use the following to access the file:

```
WinHelp(hwnd,"helpfile.hlp",HELP_INDEX,NULL);
```

For Delphi, the access is:

```
Application.HelpCommand(HELP_CONTENTS,0);
```

Searching for a Help Topic

To invoke the Help Search function in C, use the following:

```
WinHelp(hwnd,"helpfile.hlp",HELP_PARTIALKEY,(LPARAM)"");
```

To invoke the Help Search function in Delphi, use the following:

```
StrCopy(s,'');
Application.HelpCommand(Help_PartialKey,LongInt(@s));
```

Help on Using Help

To access the Using Help file in C, you would use the following:

```
WinHelp(hwnd,"helpfile.hlp",HELP_HELPONHELP,NULL);
```

To access the Using Help file in Delphi, you would use the following:

```
Application.HelpCommand(HELP_HELPONHELP,0);
```

Contextual Help Access

There are several things you need to do to add contextual help to your program. First, you must add a [MAP] section to the .HPJ file for the help project. Place the [MAP] section just before the [WINDOWS] section in the .HPJ file that HelpGen creates.

Let's say you want context sensitive help for a file open function. In the .HPJ file you would add a define for this:

```
[MAP]
#define fileopen 10
```

In the .MAC file, you would create a help page for this function:

```
.ent(fileopen,File Open Function, )
...
.end
```

Finally, to activate the contextual help, you would place the following line in your C code:

```
WinHelp(hWnd,"helpfile.hlp",HELP_CONTEXT,10);
```

Or add the following line to your Delphi code:

```
Application.HelpCommand(HELP_CONTEXT,10);
```

Note that the number is the same as that defined for fileopen.

Closing the Help File

When you shut down your program, you will also want to shut down the help file. In C, you would add the following code:

```
WinHelp(hwnd,"helpfile.hlp",HELP_QUIT,NULL);
```

In Delphi, add the following code:

```
Application.HelpCommand(HELP_QUIT,0);
```

Appendix F Glossary

help compiler - The program that creates a help (.HLP) file from the .RTF file, using the rules detailed in the .HPJ file.

macro - The replacement of a long series of RTF commands with a single command word and (optional) series of arguments.

popup - A small window that appears when you select a topic with a dotted underline. The window will remain until you click the left mouse button again.

project - A file with the extension .HPJ, that contains options that specifies how a help file is to be built. Required by the help compiler.

RTF - Rich Text Format, a standard text file formatting standard. An RTF file consists of RTF commands and normal text.

RTF commands - Special commands in an RTF file that describe how text is to be displayed. These commands start with a backslash (\) character.

shareware - A software distribution method. Software is distributed for trial use, and if users find it useful, they are expected to pay a registration fee ('try before you buy').

template - A series of text lines that constitute the minimal macro file that will convert to an RTF file and compile to a help file.

title bar - The area at the top of the HelpGen program window that contains the program name, file name, system menu button and window size controls.

toolbar - The line of graphical push buttons that are located directly below HelpGen's menu bar.

topic - A block of text in a help file that will be displayed as a single display page.

twips - A 20th of a point. There are 72 points in an inch, so there are 1440 twips in an inch.