HelpGen

Windows Help File Generator
Version 1.20

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.

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:

(A) Educating past and prospective customers of shareware as to its availability and legal nature;

(B) 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;

(C) 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

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 processor'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).

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 does not accept credit card orders. However, you can order HelpGen by credit card if you are on CompuServe. Simply GO SWREG and register (you can search for a keyword of HelpGen or for the file name HPGN12.ZIP).

Getting Started

Installation

To install HelpGen, create a directory and copy all the HelpGen files into the directory. If you received HelpGen as a .ZIP file, you must use PKUNZIP or some other archive extractor to unarchive the HelpGen files. With the .ZIP file on drive A: and using PKUNZIP, this process is

```
C:\HELPGEN>PKUNZIP A:HPGN12.ZIP
```

You should then create a Windows program group and put the HLPGEN.EXE icon into the group. This will allow you easy access to HelpGen from Windows.

Running HelpGen

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

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.

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.

HelpGen actually adds a step so that most of the other steps can be performed automatically. 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, adding your own topics by using the HelpGen macros.

3. Generate the RTF file from the macro file by using the Build | RTF File function.

4. Generate the HLP file from the RTF file and using the project file, by using the Build | HLP File function.

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 - 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 these items.

Build - Items concerned with creating project files, RTF files and help files. The toolbar includes these items.

Test - This item is concerned with testing a help file that has been generated with HelpGen.

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 toolbar icon shown above 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 toolbar icon shown above 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 toolbar icon shown above 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 toolbar icon shown above 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 toolbar icon shown above will also activate this function.

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 toolbar icon shown above will also activate this function.

Build | RTF File

The Build | RTF File 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. The toolbar icon shown above will also activate this function.

Build | HLP File

The Build | HLP File 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. The toolbar icon shown above will also activate this function.

Test

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

Options | Directories

The Options | Directories function allows you to specify the locations of the ASCII text editor and the help compiler you will be using with HelpGen. These two 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. NOTE: In this shareware evaluation version, the copyright notice cannot be edited. The registered version allows you to edit the copyright notice.

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 toolbar icon shown above will also activate this function.

Help | Register HelpGen

The Help | Register HelpGen function allows you to easily create a registration form so that you can register the program. It presents a dialog box in which you may fill out your mailing details. It then allows you to print the form.

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 marco 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.

Hints and Tips

- HelpGen macros are case-sensitive. Be sure to use lower case when typing a HelpGen macro. In general, make sure that each field in a HelpGen macro matches case with any connecting macros.

- Don't put tab characters in your macro file. If you want help text to be tabbed, use the RTF \tab command to tab.

- There are many more RTF commands than are implemented in the HelpGen macro language or 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.

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.

.start Command

SYNTAX: .start(label,topic,bitmap_file,application_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. *application_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 Demonstration)
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: end_file

.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 *.j* and *.j1* 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: .end, .j, .j1

.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: .end, .p, .p1

.end Command

SYNTAX: .end

EXPLANATION: This macro marks the end of a topic entry that was started with *.ent* or *.pent*.

EXAMPLE: See the examples for *.start, .ent* and *.pent*.

SEE ALSO: .start, .ent, .pent

.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: .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: .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: .ent, .j

.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: .pent, .p1

.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: .pent, .p

.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: .ent, .pent

.bmp Command

SYNTAX: .bmp(pathname)

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(pathname)

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(pathname)

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

.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,renault.bmp,Renault Help)
.rem(Insert table of Contents here)
.end_file
```

SEE ALSO:

.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

.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

.[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

.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
This text will be displayed in a box with a thick border.
\brdrs
And this text will be 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	=>	™	(this one 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.
```

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.

HelpGen generates an RTF file that specifies four basic text fonts; font 0 is MS San Serif, font 1 is Roman, font 2 is Courier and font 3 is Symbol. Font 0 is used for topic headers, font 1 is used for normal text in a topic and font 2 is used inside of the .[and .] macros. Font 3 is not currently used.

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 only four of the nine possible sections - [OPTIONS], [FILES], [BITMAPS] 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).

[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)

A HelpGen Tutorial

Planning Your Help File

Planning the layout and contents of the help file is probably the most important part of creating on-line 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.

1. 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.

2. 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 the HelpGen Help Generator, version 1.20)
.rem(Copyright © 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 include 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:

1. 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.

2. Select the **Build | Project File** menu item. Answer 'Yes' to the question and a project file will be built for you.

3. 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 RTF File

When you have finished editing the macro file, you must create an RTF file from the macro file. Select the **Build | RTF File** menu item to perform this step.

Generating the Help File

When the RTF file has been built, HelpGen will enable the generation of a help file. Select the **Build | HLP File** menu item and HelpGen will 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** 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-generate the RTF file (DON'T FORGET THIS!) and 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.

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.