Building the Help File

After you create the topic file(s) and Help project file, you can build the Help file. This chapter describes how to build, debug, and test a Help file.

The Windows Help

To create a Help file from your topic files, you use the Windows Help compiler. The Help compiler is an application that converts RTF source files into a binary Help file that can be displayed by the Windows Help application. Windows Help version 3.1 offers several versions of the Help compiler. The following table lists each version and describes its function.

Compiler	Description
HC30.EXE	The Help compiler that ships with the Microsoft Windows Version 3.0 Software Development Kit. Use this compiler to create Help files compatible with Windows Help 3.0 or 3.1.
HC31.EXE	The Help compiler that ships with the Microsoft Windows Version 3.0 Software Development Kit. Use this compiler to create Help files compatible with Windows Help 3.1.
HCP.EXE	The MS-DOS extended version of the Help compiler. Use this compiler to create large Help files or to make better use of memory. This Help compiler should be able to compile in MS-DOS any file that can be compiled in OS/2.

Choosing the Correct Version for your

Windows version 3.1 includes Windows Help version 3.1 as the system Help application. The Microsoft Windows

Version 3.1 Software Development Kit and the Microsoft Developer Network CD-ROM include a batch file (HC.BAT) that gives applications the choice of creating Help files either in version 3.0 or 3.1 format (3.0 being the default). Applications can also choose to ship Windows Help version 3.1 and replace a user's system-wide Help application when installing their product on the user's computer. All Help files created using the 3.0 Help compiler will display properly in Windows Help version 3.1. However, Help files created using the 3.1 Help compiler cannot be displayed with Windows Help version 3.0. Therefore, when deciding which version of the Help compiler to use to build your Help files, follow these guidelines:

- Your application requires Microsoft Windows version 3.1.

If your application requires Windows version 3.1 and will not run in Windows version 3.0, you should use the 3.1 Help compiler to build your Help files. Your application does not need to include the version 3.1 Windows Help application on the product disk(s) nor should it install the Help application on the user's computer. In this scenario, you can use any 3.1 feature in your Help file.

- Your application is designed to work with Microsoft Windows version 3.0 and 3.1 and does not ship the version 3.1 Windows Help application on the product disk(s).

In this case, you must use the 3.0 Help compiler to build your Help files to ensure that they will display properly on both versions of Windows Help. This also means that you cannot use any feature specific to Windows Help version 3.1 in your Help file.

- Your application is designed to work with Microsoft Windows versions 3.0 and 3.1, and does ship the version 3.1 Windows Help application on the product disk (s).

In this case, you should use the 3.1 Help compiler to build your Help files. When installing your product, the setup program should detect the version number of the user's Help application and replace it if it is version 3.0. In this scenario, you can use any 3.1 feature in your Help file.

Building a Help

Building the Help file is quite simple.

To build a Help file

1. Edit the Help project file and enter the following lines in the [OPTIONS] section so you can monitor the build process on your screen:

```
WARNING=3
REPORT=ON
```

When the **WARNING** option is set as above, the compiler reports all warnings and errors it encounters during the build. When the **REPORT** option is set, the compiler displays its messages as it performs the different phases of the build, including compiling the file, resolving jumps, and verifying browse sequences.

2. Change to the directory that holds the Help project file.
3. Type the **HC** command using the following syntax:

HC *projectfile*

The *projectfile* is the name of your Help project file. The compiler assumes that your Help project file has an .HPJ file extension. If it does, or if it has no extension, you can type just the filename, as in the following example:

```
hc myhelp
```

The compiler displays periods on the screen to show its progress and displays warnings and error messages when it encounters problems. For more information, see Chapter 18, “Help Error Messages.”

If everything goes smoothly, the Help compiler creates a Help file in the project directory and then returns to the MS-DOS prompt. The Help file has the same name, but with an .HLP extension, as the Help project file. For example, the command in the previous example would produce MYHELP.HLP.

Building Large Help Files

The number of topic files affects the amount of time required to complete the build. Help files consisting of a small number of files compile quickly; Help files with large numbers of files can require several hours to compile. For large Help files, you might want to start the build process at the end of the day and let it compile at night. (You can also start a build at the MS-DOS prompt within Windows, and then change its settings to run in the background while you use other applications. For more information, see the *Microsoft Windows User’s Guide*.)

If your Help file is too large to build under MS-DOS, you can compile under OS/2 or use the DOS Protected Mode Interface (DPMI), or extended, version of the Help compiler (HCP.EXE). The extended version of the compiler uses the same syntax as HC.EXE, but it makes better use of memeory. All instructions for building under MS-DOS apply to these other versions unless otherwise indicated.

Getting Around Memory Problems

Depending on the size of your Help file and the computer you are using to build on, you may encounter out-of-memory problems during compilation. Although there is no single solution that addresses all out-of-memory conditions, the following list offers some suggestions you might try to ensure that your Help file will build successfully:

- If possible, compile without compression.

The Help file will build fastest when no compression is used. Compressing the Help file can take up to ten times as long to build as an uncompressed file, depending on the content. In most cases, you can build with no compression while you're developing the Help file, and use compression for only the final builds of the product.

- Compile in a Windows MS-DOS box rather than from the MS-DOS prompt.

In most instances, you are better off running the Help compiler in a Windows MS-DOS box because HC31.EXE uses extended memory, which is not always available in regular MS-DOS but is available in an MS-DOS box under enhanced mode Windows. To verify that you are using the enhanced version of the compiler, you can run the compiler and look at the copyright message. It should include the word "extended" after the version number.

- Close other applications that are also running.

Frequently you want to have Word for Windows running so that you can edit your RTF source files and then do a quick compile. However, if you have a limited amount of memory in your computer, you may have to close Word for Windows or all other running applications to free enough memory to complete the build.

- Compile the Help file under OS/2 or use HCP.EXE under MS-DOS.

OS/2 does not have the MS-DOS 640K memory limitation and manages

available memory much more efficiently than does MS-DOS. If you do not have an OS/2 machine, you can use the DPMI version of the Help compiler (HCP.EXE), which should produce similar results under MS-DOS.

- Be sure that you have sufficient free disk space.

Sometimes the Help compiler can run out of memory if the disk runs out of space. Compare the amount of free disk space you have with the size of your topic files. If you are compiling with compression, you may need as much free disk space as five times the size of your source files. This requirement can be more or less (usually less), depending on the Help file. Compiling without compression also reduces the amount of disk space the compiler requires.

- Replace any inline bitmaps in the topic files with bitmap references.

The 3.1 Help compiler fragments memory when compiling bitmaps under
MS-DOS, which often causes out-of-memory problems. Therefore, if you have pasted any bitmaps directly into the topic files, you should replace them with bitmap references. If the pasted-in bitmaps are larger than 64K, the compiler will always run out of memory during the build. If you have severe memory problems, you may have to limit the number of bitmaps you include in the Help file.

- If you are building with compression, perform partial builds until you achieve a successful build of the Help file.

To perform a partial build, you edit the Help project file and comment out specific lines (topic files) in the [FILES] section. Commenting out topic files will produce an incomplete Help file with unresolved jumps and other problems; however, the compiler may be able to create a .PH file with compression information. After you have a phrase file, you can build the Help file again and include all the topic files. (Remember to set **OLDKEYPHRASE=yes** in the Help project file before building the Help file.) The compiler will create a working Help file with some level of compression. If necessary, you can use the phrase file with several topic file subsets to compress the complete Help file.

- If you are building a version 3.0 Help file with compression, use the 3.1 Help compiler to create the phrase file.

The 3.0 Help compiler often runs out of memory when creating a phrase

file (.PH) for large Help files, even when compiling under OS/2. That is because creating the phrase file is a very memory intensive process, and phrase files over 32K cause the build to fail. However, this is not usually a problem when using the 3.1 Help compiler. So, if you are having trouble compiling a file with the 3.0 Help compiler, you can delete the 3.0-generated phrase file and use the 3.1 compiler to generate a phrase file. After you have a phrase file, you can build the Help file again with the 3.0 compiler. (Remember to set **OLDKEYPHRASE=yes** in the Help project file before building the Help file so the 3.0 compiler will not try to create a phrase file.)

Speeding Up the Build

When compiling large Help files, or when compiling many Help files using a batch file or make file, the build can take several hours. This is often true even if you have a fast machine, large hard disk, and plenty of memory. Although you cannot completely avoid slow compile times with large Help projects, you can do a few things to achieve faster builds.

- If possible, compile without compression or use medium compression.

The Help file will build fastest when no compression is used. Using the **Compress=high** option can require 4 to 10 times as long to build as uncompressed files, depending on the content. Another option is to use medium compression. The difference in size between high and medium compression is small, but the build is a lot faster with medium compression.

- Compile the Help file on an OS/2 machine with HPFS.

OS/2's high-performance file system (HPFS) is much faster than the MS-DOS file allocation table (FAT). If you have a machine that you can configure with OS/2, you might try doing some performance tests to see if your Help file builds faster under OS/2.

- Create a RAM drive and use it to build the Help file.

If your Help file is not too large and you have sufficient memory in your build machine, you can use a RAM drive to build your Help file. For this method to work, the RAM drive should be roughly two and

one-half times larger than the Help file you are building. For example, if the Help file is about 2 MB, the RAM drive should be at least 5 MB.

- Divide the Help file into separate topic files, and then build and test these files independently.

The easiest way to compile individual files separately is by editing the Help project file and commenting out lines in the [FILES] section for topic files you don't want included. To comment out a line, insert a semicolon (;) at the beginning of the line. Remember, however, that if you comment out particular topic files that contain hot spots referencing excluded topics, the compiler will display warnings as the Help file builds. You can also use the View Topic, View Selection, and View File commands in the Help Authoring Templates (WHAT30.DOT or WHAT31.DOT) to perform partial builds.

Compiling a 3.0 Help File Under 3.1 Help

You need to make some changes to get your 3.0 Help file to compile under Windows Help version 3.1.

To prepare a 3.0 source file for compiling under version 3.1

1. Change **INDEX=** to **CONTENTS=** in the Help project file.
2. Change side-by-side paragraphs (if any) to tables.
3. Check to make sure there are no hidden paragraph marks or hidden page breaks.

For more information about the **CONTENTS** option, see Chapter 16, "The Help Project File." For more information about creating tables, see Chapter 7, "Formatting Topics."

Performing Simultaneous Builds

Generally, it is not a good idea to run simultaneous builds on the same machine, and we do not recommend it. However, if you want to attempt a simultaneous build, you can do so at your own risk. As a minimum, take the following precaution: set the TMP environment variable to a different directory for each MS-DOS or OS/2 screen running the Help compiler. Otherwise, temporary

filenames that the Help compiler creates during the build may collide, producing very bad results.

Building Identical Help Files

Sometimes (when performing quality assurance tests, for example) it may be important to compare the binary versions of a Help file compiled from the same source files. However, if the Help compiler displays any error messages on the screen (other than progress periods), subsequent builds of the Help file will not compare to previous ones, even though they are using exactly the same source files. (The Help files work the same; they just aren't identical.)

This happens because Help files include a time stamp indicating when they were compiled. In addition, the Help compiler writes uninitialized parts of memory buffers to disk every time a screen message is printed. Therefore, Help files will not be identical when compiled from the same source.

If binary compares are important to you, you can try this workaround (which is not guaranteed to work).

To compile a Help file that may be recreated exactly

1. Create a batch file that:
   - Zeros the machine's memory.
   - Resets the clock.
   - Compiles the Help file.
2. Run the batch file to create the identical version.

Building Files Saved in Word for the Macintosh version 4.00

When the Help Compiler processes RTF files created by Microsoft Word for the Macintosh, colon (:) characters may disappear from the built Help file. That is because version 4.00 of Word for the Macintosh prefixes each colon with a backslash character (\), which makes each colon a special RTF control word. Because the Help Compiler ignores these RTF statements, the colons drop out of the compiled Help file.

There are two methods to correct this problem:

- Contact Microsoft Customer Service to update your copy of Word for the Macintosh to version 4.00a or later.
- Perform the following procedure to remove the backslash characters from the RTF file.

To remove backslash characters from the RTF file

1. Open the file in Word for the Macintosh version 4.00.
2. When asked if Word should interpret RTF text, choose No.
3. Use the Change feature to find each occurrence of **\:** (backslash, colon) and replace it with **:** (a plain colon).
4. Save the file as Text Only and close the document.
5. Recompile the Help file.

Building Help Projects with Multiple

In some situations, you may want to create a Help project that has multiple Help files. A multi-file Help project is one in which several .HLP files work together to provide users complete information about a product. Even though each .HLP file is separate, users can jump from one Help file to another just as they can within a single Help file.

The process for building a multi-file Help project is slightly different from the process for building a single Help file.

To build a multi-file Help project

1. Set up a directory structure appropriate for your Help project.
2. Create a Help project file (.HPJ) for each Help file in the project.
3. Create a batch file to build all the files in the project.

The following sections examine each of these steps in more detail.

Setting Up the Project Directories

First, create a directory structure to fit the organization of your Help project. Figure 17.1 shows one possible structure.

Graphic

This sample Help project consists of three Help files: CPCD.HLP, FMCD.HLP, and PMCD.HLP. As shown in the illustration, the HYPER project directory is a subdirectory of the directory where you installed Windows Help. The HYPER subdirectory contains the three .HPJ files for the three Help files in the project and a batch file to build the entire project.

HYPER also has a subdirectory for each Help file in the project, a subdirectory for error files, and a subdirectory for archiving built files. In turn, each Help file subdirectory has two subdirectories for the different file types used in the Help file: RTF for topic files and ART for bitmap files.

Creating Help Project Files for Each Help file

After you have set up the directory structure for a multi-file Help project, you create a Help project file for each Help file in the project. The exact contents of the Help project file will vary depending on the type of Help system you are creating, but at a minimum it should include an [OPTIONS] section and a [FILES] section. The Help project file for one of the Help files in this sample project might look like this:

```
[OPTIONS]
ERRORLOG=PROGMAN.BUG
ROOT=C:\HYPER\PROGMAN
BMROOT=.\PROGMAN\ART
CONTENTS=cont_progman
TITLE=Program Manager
COMPRESS=OFF
REPORT=ON
WARNING=3

[FILES]
.\RTF\PROGMAN.IDX
.\RTF\PROGMAN.CON
.\RTF\PROGMAN.CMD
.\RTF\PROGMAN.HOW
.\RTF\PROGMAN.KBD
.\RTF\PROGMAN.QCK
```

Notice the following points:

- The **ERRORLOG** option appears first so that all error messages will

be recorded in the error file.

- The **ROOT** option gives a full path to specify the project root directory. In this example, the root directory is the PROGMAN subdirectory of the \HYPER project directory.
- The **BMROOT** option gives a path relative to the project directory. The full path for the bitmap root would be \HYPER\PROGMAN\ART.
- Similarly, the relative paths in the [FILES] section refer to the project root directory.
- For the first few builds of a new Help project, it is usually more important to build quickly than to get good compression in the built file, so compression is turned off.
- To make debugging the Help files easier, the [OPTIONS] section includes the **REPORT** and **WARNING** options.

Building the Individual Help files

If you set up your Help project files as described in the previous section, you should be able to build each Help file just as you would for a single-file Help project. Simply change to the project directory and enter the HC command for the Help file you want to build.

For example, to build PMCD.HLP, you would do the following:

1. Change to the \HYPER project directory using the following command:

   cd \HYPER

2. Enter an HC command to build the Help file:

   hc pmcd.hpj

To build the other Help files, just repeat the same process for each of the three .HPJ files in the project. All the .HLP files you build will be created in the project directory (in this example, in the \HYPER directory).

Creating a Batch File

You can also create a batch file to build all the Help files in the project. Using a batch file is optional, but it simplifies the build process. The following example shows what a simple batch file for this Help project might look like:

```
@echo off
echo delete files from last build
del *.hlp
del *.ph
del *.bug
del hlperr.all

echo building CONTROL PANEL
hc cpcd.hpj
copy control.bug \hyper\error
copy cpcd.hlp \hyper\archive

echo building FILE MANAGER
hc fmcd.hpj
copy winfile.bug \hyper\error
copy fmcd.hlp \hyper\archive

echo building PROGRAM MANAGER
hc pmcd.hpj
copy progman.bug \hyper\error
copy pmcd.hlp \hyper\archive

echo Building composite error file
copy *.bug hlperr.all
```

The batch file begins by deleting all the files produced during the previous build. This step is optional, but it ensures that the project directory is cleaned up before each build. The next three sections build each individual Help file and place a copy of the resulting error file in the \ERROR subdirectory and a copy of the built Help file in the \ARCHIVE subdirectory. The final section of the batch file creates a single error file from the three individual error files so that all the error messages from the build can be viewed or printed more easily.

Depending on the Help system, you may want to create a master Contents topic that accesses topics in each of the separate Help files. Or you may simply want to create interfile jumps that allow users to move through the information in each of the files. How you interconnect the Help files in a multi-file Help system is up to you.

Debugging the Help

This section examines ways you can investigate problems that might arise during the build process.

The Help compiler displays messages when it encounters errors in building the Help file. (Chapter

18, “Help Error Messages,” contains a list of messages that might appear during the build process.) Whenever possible, the compiler displays the name of the file that contains the error and the number that identifies the specific line of the Help project file or the topic that produced the error.

Because topics are not actually numbered, the topic number given with an error message refers to that topic’s sequential position in the RTF file (first topic, second topic, and so on). These numbers may be identical to the page numbers shown by your word processor, depending on the number of lines you have assigned to the hypothetical printed page. Remember that topics are separated by hard page breaks, even though there is no such thing as a “page” in the Help system.

A single error condition in the topic file can cause many error messages. For example, an incorrectly identified topic causes an error every time a jump term refers to the correct topic identifier. Such a mistake is easily fixed by correcting the footnote that contains the wrong context string. In general, if you get an error (for example, an unresolved jump), correct the error(s) and recompile.

Note

An easy way to avoid build errors is to make sure that the compiler can access all the RTF files and all the graphic files needed to build the Help file.

Displaying Error Messages on the Screen

The **REPORT** option tells the compiler to display messages on the screen while it is building. These messages indicate when the compiler is performing the different phases of the build, including compiling the file, resolving jumps, and verifying browse sequences.

The following option turns on the message display:

REPORT=ON

Warning Message Reporting

You can specify the level of warnings reported by the compiler during the build process. Warning messages alert you to conditions that are not serious enough to stop the build but that might cause problems in your Help file.

The **WARNING** option in your Help project file controls the amount of warning information that the compiler displays. You set this in the [OPTIONS] section, using the following command:

WARNING=*level*

The following table describes the three reporting levels for the build process.

Level	Information reported
1	Only the most severe warnings
2	Severe and less serious warnings
3	All warnings

The following example specifies the intermediate level of error reporting:

```
[OPTIONS]
WARNING=2
```

Redirecting Errors to a File

By default, the Help compiler displays all errors on your screen. However, you can redirect errors to a file so that you can examine the errors more carefully and fix problems in your Help file. You can examine the redirected file using any ASCII text editor. Use the standard MS-DOS redirection syntax to redirect error and status messages to a file; for example:

```
hc myhelp.hpj > errors.txt
```

This command places all the messages sent during the build process into the ERRORS.TXT file. You can also enter the following in the [OPTIONS] section of your Help project file:

ERRORLOG=*error-file*

The Help compiler places compilation errors in the file given by *error-file*. The *error-file* can be an absolute or relative path if the file is in a directory other than the Help project root directory. If you use the **ERRORLOG** option, it should be the first line in the [OPTIONS] section.

Using **ERRORLOG** has two advantages over redirecting error messages:

- Using redirection, the Help compiler puts unnecessary characters (including the periods it displays to indicate the progress of the build) and status messages along with error messages in the output file. Using **ERRORLOG**, you get only the error messages.
- Using redirection, the Help compiler doesn't display error messages on the screen, so you can't monitor the messages during the build. Using **ERRORLOG**, the Help compiler displays the error messages on your screen and writes them to the file.

Testing the Built Help

After you've successfully built the Help file, you'll want to look at the finished product to see if there are problems in displaying the topics.

To test a built Help file

1. From within Windows, start Windows Help by clicking the Windows Help icon or using the Run command from the Program Manager.
2. From the File menu, choose Open and open the .HLP file you've built.
3. Test all components to ensure that text and graphics have been formatted correctly, no typographical errors appear, and that elements such as jumps, keywords, browse sequences, pop-up windows, and macros work correctly.

Using the Keyboard Debug Option

If you want to use a keyboard option to debug your Help file, you can add a switch to the WIN.INI file that lets you display every topic in a Help file in sequential order, whether or not you include browse sequences. The order is determined by the physical location of each topic in the compiled Help file.

To implement the keyboard debug option

1. Add the following line to the [Windows Help] section of your WIN.INI file:

   ```
   SeqTopicKeys=1
   ```

2. Restart Windows and open the Help file you want to test.
3. Press SHIFT+CTRL+HOME to display the first topic in the Help file.
4. Press SHIFT+CTRL+RIGHT ARROW to display each successive topic in the Help file.

   Note

   If the Help file contains a blank topic, it means that one of your RTF topic files ends with a hard page break.