7.0

# BORLAND® PASCAL WITH OBJECTS

## TOOLS AND UTILITIES GUIDE

- TPUMOVER
- WINSIGHT™
- WINSPECTOR
- HELP COMPILER

BORLAND

# *Borland® Pascal with Objects*

Version 7.0

---

# Tools and Utilities Guide

BORLAND INTERNATIONAL, INC. 1800 GREEN HILLS ROAD
P.O. BOX 660001, SCOTTS VALLEY, CA 95067-0001

PRINTED IN THE USA.
10 9 8 7 6 5 4 3 2 1

R1

# C O N T E N T S

# T A B L E S

# F I G U R E S

# INTRODUCTION

Borland Pascal with Objects comes with a host of powerful standalone utilities you can use to ease your DOS and Windows programming.

Read this book to learn how to use the utilities listed in Table 0.1. Read the online file, UTILS.DOC, which the INSTALL utility places in the DOC subdirectory, to find out how to use the rest of the Borland Pascal utilities. If you need help in understanding any error messages you get while using the utilities described in this book, read Appendix A.

Table 0.1
Borland Pascal Utilities

| Name | Description |
|---|---|
| *Documented in this book* | |
| TPUMOVER | Moves units |
| MAKE | Manages programs |
| WinSight | Monitors Windows messages |
| WinSpector | Debugs Windows programs |
| RC | Compiles Windows resources |
| HC | Compiles Windows help files |
| THELP | Displays help screens |
| *Documented in the online document UTILS.DOC* | |
| TOUCH | Updates file date and time |
| GREP | Searches for text in files |
| BINOBJ | Converts a file to an .OBJ file |

C H A P T E R

# 1

# Unit mover

When you write units you use frequently, you want to make them easily available to any programs that you develop. This chapter explains how to insert your units into the .TPL (Turbo Pascal Library) file, Borland Pascal's run-time library, using TPUMOVER. You can also use TPUMOVER to remove seldom-used units from the .TPL file.

## A review of unit files

Ordinarily, when you compile a unit, Borland Pascal puts the resulting symbols and code in a file, which always contains exactly *one* unit. The extension on the file name depends on which platform you are writing applications for and which version of the compiler you are using.

Table 1.1
Unit file extensions for the Borland Pascal compilers

| Unit extension | Target platform | Compiler |
|---|---|---|
| .TPP | DOS protected-mode | BP, BPC |
| .TPU | DOS real-mode | BP, BPC, TPC, TPCX, TURBO, TURBOX |
| .TPW | Windows | BP, BPC, TPW |

A .TPL file, on the other hand, can contain *multiple* units. The particular .TPL file Borland Pascal uses depends on your target

platform. At start-up, Borland Pascal will automatically load one of the following run-time library files into memory:

*TURBO.TPL*

- The units that Borland Pascal uses for DOS real-mode applications are in the file TURBO.TPL. These include the *System*, *Dos*, *Crt*, *Printer*, *Overlay*, and *WinDos* units.

*TPP.TPL*

- The units that Borland Pascal uses for DOS protected-mode applications are in the file TPP.TPL. These include the *System*, *Dos*, *Crt*, *Printer*, *Strings*, *WinDos*, and *WinAPI* units.

*TPW.TPL*

- The units Borland Pascal uses for Windows applications are in the file TPW.TPL. These include the *System*, *WinDos*, *WinCrt*, *WinPrn*, *Strings*, *WinProcs*, and *WinTypes* units.

# Using TPUMOVER

Command-line parameters let you manipulate units quickly. The syntax for these parameters is

```
TPUMOVER filename operations
```

where *filename* is a .TPL file, and operations represents an optional list of one or more of the following commands:

Table 1.2
TPUMOVER commands

| Command | Action |
|---|---|
| +unitname | Adds a unit to the library. |
| *unitname | Extracts a unit from the library. |
| –unitname | Deletes a unit from the library. |

For example, to add the unit titled *Tools* to the TPW.TPL file, type

```
TPUMOVER TPW.TPL +Tools
```

If no operations are specified, TPUMOVER lists the units in the .TPL file along with size and dependency information.

## Adding a unit to the library

Suppose you have a well-written and thoroughly debugged unit you use in many different programs. You can add this unit to the run-time library so that it's automatically loaded into memory when you run Borland Pascal. When you tell TPUMOVER to add a unit to the run-time library, TPUMOVER makes a copy of the specified unit and adds this copy to the .TPL file. Although

adding this unit to the run-time library increases the amount of memory the .TPL file uses, it decreases the time it takes the compiler to access your unit.

If two versions of a unit exist, Borland Pascal will use the one in the run-time library. Therefore, if you add a unit to the run-time library and later make a change to the unit source code, you must recompile the unit and then use TPUMOVER to add the revised unit to the run-time library.

## Extracting a unit from the library

If you need to save memory, you can use TPUMOVER to remove one or all of the units from the run-time library. Extracting a unit from the .TPL file reduces the file's size and the amount of memory it occupies when loaded. Borland Pascal automatically copies the extracted unit into the corresponding .TPU, .TPP, or .TPW file.

You can use the units stored in the run-time library without having to specify their location on the disk. However, after you extract a unit from the run-time library, you need to indicate its location in the directory unit path.

## Deleting a unit from the library

If you are sure you no longer need to use a particular unit, you can delete it from the run-time library. Remember that, unlike the extract command, TPUMOVER does not copy the deleted unit into a corresponding unit file, so make sure you really do not need to use this unit.

C H A P T E R

2

# Project manager

Borland's command-line utility MAKE helps you keep the executable versions of your programs current especially if you link Pascal files with object files or write programs outside the Integrated Development Environment (IDE). You can use MAKE to create real-mode executables compiled by BPC or TPC.

If you write programs in the IDE, the IDE itself automatically compiles source files as needed. This chapter shows you how to create makefiles for use outside the IDE. Read this chapter to learn how to

- Build a simple makefile
- Use MAKE commands and operators
- Write explicit rules
- Write implicit rules
- Define macros
- Control MAKE using directives
- Include command-line options

## Overview of the build process

MAKE updates your programs by performing the following functions:

- Verifying the existence of target files. These target files are usually executables named on the command line or in the makefile used previously by MAKE.
- Building target files if they don't already exist.
- Checking the time and date stamps of files to prevent use of an obsolete executable file or its dependent source and object files.
- Re-creating target files if dependent files are out of date.

## Examining a simple makefile

*This makefile, BGILINK.MAK, is distributed with Borland Pascal and is installed in the BP\EXAMPLES\DOS\BGI directory.*

The following makefile creates an executable file, BGILINK.EXE, from BGIDRIV.TPU and BGIFONT.TPU and then calls TPC to compile BGILINK using the **/M** (build modified units) option.

```
# Build a sample makefile.
BGILINK.EXE: BGIDRIV.TPU BGIFONT.TPU
  TPC BGILINK /M
```

In this file, the pound sign (#) is a comment symbol. The target file, BGILINK.EXE, is typed flush left. Commands, such as TPC BGILINK /M, are indented one or more spaces.

## Using MAKE on a simple file

To use MAKE, type the following command at the DOS prompt:

```
MAKE
```

Unless otherwise instructed, MAKE then looks for a file named MAKEFILE. Let's assume MAKEFILE is the name of our sample program listed above. After finding this makefile, MAKE looks to see if BGILINK.EXE exists. If it does, MAKE compares the time and date stamp of the .TPU files with BGILINK.EXE. If the files are out of date, MAKE calls TPC to create updated files for BGIDRIV.TPU and BGIFONT.TPU and to compile BGILINK.

## Creating a makefile

Creating a makefile is like writing a program, with definitions, commands, and directives. You can create a makefile with any

ASCII text editor, such as the IDE built-in editor, Sprint, or SideKick. Basically, a makefile contains source and target file names and dependent file information. A makefile includes either explicit rules, those that specify complete file names, or implicit rules that specify only file-name extensions to indicate file dependencies. You can add MAKE directives and macros to control and automate the execution of your makefile.

## MAKE syntax rules

The general syntax for MAKE is

```
make [option...] [target...]
```

where *option* is a MAKE option (discussed later), and *target* is the name of a target file to make.

The MAKE syntax rules are the following:

*MAKE stops if any command it has executed is aborted with a Ctrl+Break. Thus, a Ctrl+Break stops the currently executing command and MAKE as well.*

- The word MAKE is followed by a space, then a list of make options.
- Each make option must be separated from its adjacent options by a space. Options can be placed in any order, and any number of these options can be entered (as long as there is room in the command line). All options that do not specify a string (**–s** or **±a**, for example) can have an optional – or + after them. This specifies whether you wish to turn the option off (–) or on (+).
- The list of MAKE options is followed by a space, then an optional list of targets.
- Each target must also be separated from its adjacent targets by a space. MAKE evaluates the target files in the order listed, recompiling their constituents as necessary.

If the command line does not include any target names, MAKE uses the first target file mentioned in an explicit rule. If one or more targets are mentioned on the command line, they are built as necessary.

## Command lists for implicit and explicit rules

This section describes how MAKE processes commands. Commands in a command list take the form

```
[ prefix ... ] command_body
```

Each command line in a command list consists of an optional list of prefixes, followed by a single command body.

## Prefixes

The prefixes allowed in a command modify the treatment of these commands by MAKE. The prefix is either the at-sign (@) or a hyphen (–) followed immediately by a number. Whitespace must follow the modifier.

Table 2.1
MAKE prefixes

| Prefix | What it does |
|---|---|
| @ | Prevents MAKE from displaying the command before executing it. Hides the display even if the **–s** option is not given on the MAKE command line. |
| ***–num*** | Aborts processing commands in the makefile if the exit status exceeds the *num*. In this example, MAKE aborts only if the exit status exceeds 4:<br>`-4 MYPROG SAMPLE.X`<br>If no *–num* prefix is given and the status is nonzero, MAKE stops and deletes the current target file. |
| – | Continues processing commands in the makefile, regardless of the exit code returned. |
| **&** | Expands either the **$**** macro which represents all dependent files, or the **$?** macro which represents all dependent files stamped later than the target. Executes the command for each of the dependent files. See page 23 for more information on these macros. |

*Exit codes are status codes returned by the executed commands.*

## Command body and operators

The command body is treated exactly as if it were entered as a line to the DOS command line, with the exception that pipes (|) are not supported.

The following table describes command-line operators that MAKE uses to redirect files or devices and to create inline files. The operators are entered after the MAKE prefixes.

Table 2.2
MAKE command-line operators

| Prefix | What it does |
|---|---|
| **<** | Takes the input for the command from the file name or device name rather than from standard input. |
| **>** | Sends the output from the command to a file. |
| **<<** | Creates a temporary file and uses its contents as standard input to the command. |
| **>>** | Creates a temporary file and redirects the command's standard input so that it comes from the created file. |

Table 2.2: MAKE command-line operators (continued)

| | |
|---|---|
| **&&** | Creates a temporary file and uses the temporary file's name as standard input to the command. This is useful when you want MAKE to create a file that's going to be used as input to a program. |

## Using command-line operators

If you have the following program, the command

```
MYPROG <<!
This is a test
!
```

would create a temporary file containing the string "This is a test," redirecting it to *myprog*. The exclamation point (!) is a delimiter in this example; you can use any character except # or \ as a delimiter for the file. The first line containing the delimiter character as its first character ends the file. The rest of the line following the delimiter character (in this case, an exclamation point) is considered part of the preceding command.

*The KEEP option for the << operator in compatibility mode tells MAKE not to delete specific temporary files.*

MAKE deletes all temporary files unless you use the **–K** command-line option. Use the **–K** option to "debug" your temporary files if they don't appear to be working correctly.

## Executing commands

MAKE searches for command names using the DOS search algorithm:

1. MAKE first searches for the file in the current directory, then searches each directory in the path.
2. In each directory, MAKE first searches for a file of the specified name with the extension .COM. If it doesn't find it, it searches for the same file name with an .EXE extension. Failing that, MAKE searches for a file by the specified name with a .BAT extension.
3. If MAKE finds a .BAT file, it invokes a copy of COMMAND.COM to execute the batch file.
4. If MAKE can't find a .COM, .EXE, or .BAT file matching the command to be executed, it will invoke a copy of the DOS command processor (COMMAND.COM by default) to execute the command.

If you supply a file-name extension in the command line, MAKE searches only for that extension.

## Comments

Comments begin with a pound sign (#) character; the rest of the line following the # is ignored by MAKE. Comments can be placed anywhere; they don't have to start in a particular column.

A backslash will *not* continue a comment onto the next line; instead, you must use a # on each line. In fact, you cannot use a backslash as a continuation character in a line that has a comment. If the backslash precedes the #, it is no longer the last character on the line; if it follows the #, then it is part of the comment itself.

This is an example of a comment in a makefile:

```
# This file uses BGIFONT.TPU and BGIDRIV.TPU
# to build BGILINK.EXE
```

# Writing explicit rules

An explicit rule is a rule that specifies complete file names explicitly. Explicit rules take the form

*Note that the braces must be included if you use the paths parameter.*

```
target [target ... ] : [{paths}] [dependent ... ]
  [command]
  ⋮
```

where *target* is the file to be updated, *dependent* is a file *target* depends on, *paths* is a list of directories, separated by semicolons and enclosed in braces, in which dependent files might reside in, and *command* is any valid DOS command (including invocation of .BAT files and execution of .COM and .EXE files).

Explicit rules define one or more target names, zero or more dependent files, and an optional list of commands to be performed. Target and dependent file names listed in explicit rules can contain normal DOS drive and directory specifications; they can also contain wildcards.

## Explicit rule syntax

Observe the following syntax requirements when writing explicit rules:

- *target* must be at the start of a line (in column 1).

- The *dependent* file(s) must be preceded by at least one space or tab, after the colon.
- *paths*, if included, must be enclosed in braces.
- Each *command* must be indented, (must be preceded by at least one blank or tab). As mentioned before, the backslash can be used as a continuation character if the list of dependent files or a given command is too long for one line.

Both the dependent files and the commands are optional; it is possible to have an explicit rule consisting only of *target [target ...]* followed by a colon.

The idea behind an explicit rule is that the command or commands listed create or update *target*, usually using the *dependent* files. When MAKE encounters an explicit rule, it first checks to see if any of the *dependent* files are themselves target files elsewhere in the makefile. If so, MAKE evaluates that rule first.

MAKE checks for dependent files in the current directory first. If it can't find them, MAKE then checks each of the directories specified in the path list.

Once all the *dependent* files have been created or updated based on other rules, MAKE checks to see if *target* exists. If not, each *command* is invoked in the order given. If *target* does exist, the time and date of its last modification are compared with the time and date for each *dependent*. If any *dependent* has been modified more recently than *target*, the list of commands is executed.

A given file name can occur on the left side of an explicit rule only once in a given execution of MAKE.

Each command line in an explicit rule begins with whitespace. MAKE considers all lines following an explicit rule to be part of the command list for that rule, up to the next line that begins in column 1 (without any preceding whitespace) or to the end of the file. Blank lines are ignored.

## Special considerations

An explicit rule with no command lines following it is treated a little differently than an explicit rule with command lines.

- If an explicit rule includes commands, the only files that the target depends on are the ones listed in the explicit rule.

- If an explicit rule has no commands, the targets depend on two sets of files: the files given in the explicit rule, and any file that matches an implicit rule for the target(s). This lets you specify a dependency to be handled by an implicit rule. For example in

*The symbol **$<** is a special macro. Macros are discussed starting on page 17. The **$<** macro will be replaced by the full name of the appropriate file name each time the command executes.*

```
.PAS.TPU:
  BPC $<

PROG.TPU:
```

PROG.TPU depends on PROG.PAS; if PROG.TPU is out of date, MAKE executes the command line

```
BPC PROG.PAS
```

### Examples

Here are some examples of explicit rules from a makefile:

1. MYUTIL.OBJ: MYUTIL.ASM
   TASM MYUTIL.ASM,MYUTIL.OBJ;
2. MYAPP.EXE: MYAPP.PAS MYGLOBAL.TPU MYUTIL.TPU
   BPC MYAPP /Tc:\BP\BIN

The first explicit rule states that MYUTIL.OBJ depends upon MYUTIL.ASM, and that MYUTIL.OBJ is created by executing the given TASM command.

The second rule states that MYAPP.EXE depends upon MYAPP.PAS, MYGLOBAL.TPU, and MYUTILS.TPU, and is created by the given BPC command. (The /T plus path name in these examples will be explained later.)

If you reorder the rules so that the one for MYAPP.EXE comes first, followed by the others, MAKE will recompile (or reassemble) only the files that it has to in order to update everything correctly. This is because a MAKE with no target on the command line will try to execute the first explicit rule it finds in the makefile.

## Writing implicit rules

MAKE allows you to define *implicit* rules as well as explicit ones. Implicit rules are generalizations of explicit rules; they apply to all files that have certain identifying extensions.

Here's an example that illustrates the relationship between the two rules. Consider this explicit rule from the previous sample program:

```
MYUTIL.OBJ:MYUTIL.ASM
   TASM MYUTIL.ASM,MYUTIL.OBJ;
```

The rule is typical because it follows a general principle: An .OBJ file is dependent on the .ASM file with the same file name and is created by executing TASM (Turbo Assembler). In fact, you might have a makefile where you have several (or even several dozen) explicit rules following this same format.

By rewriting the explicit rule as an implicit rule, you can eliminate all the explicit rules of the same form. As an implicit rule, it would look like this:

```
.ASM.OBJ:
   TASM $<
```

This rule means "Any file with the extension .ASM can be translated to a file of the same name with the extension .OBJ using this sequence of commands." The .OBJ file is created with the second line of the rule, where **$<** represents the file's name with the source (.PAS) extension.

## Implicit rule syntax

Here's the syntax for an implicit rule:

```
[{source_dir}].source_extension.[{target_dir}]target_extension:
        [command]
      ⋮
```

As before, the commands are optional and must be indented.

*source_dir* (which must be enclosed by braces) tells MAKE to search for source files in the specified directory. *target_dir* tells MAKE where the target files will be placed.

*source_extension* is the extension of the source file; that is, it applies to any file having the format

```
fname.source_extension
```

Likewise, the *target_extension* refers to the file

```
fname.target_extension
```

where *fname* is the same for both files. In other words, this implicit rule replaces all explicit rules having the format

```
fname.target_extension: fname.source_extension
    [command]
  ⋮
```

for any *fname*.

MAKE uses implicit rules if it can't find any explicit rules for a given target, or if an explicit rule with no commands exists for the target.

The extension of the file name in question is used to determine which implicit rule to use. The implicit rule is applied if a file is found with the same name as the target but with the mentioned source extension.

For example, suppose you have a makefile (named MAKEFILE) whose contents are

```
.ASM.OBJ:
   TASM $<
```

If you have an assembly language routine named RATIO.ASM that you wanted to compile to RATIO.OBJ, you can use the command

```
MAKE RATIO.OBJ
```

MAKE would take RATIO.OBJ to be the target. Since there is no explicit rule for creating RATIO.OBJ, MAKE applies the implicit rule and generates the command

```
TASM RATIO.ASM
```

which does the compile step necessary to create RATIO.OBJ.

MAKE also uses implicit rules if you give it an explicit rule with no commands. Suppose you had the following implicit rule at the start of your makefile:

```
.PAS.TPU:
    BPC $<
```

You could then rewrite some explicit rules as follows:

```
MYGLOBAL.TPU:MYGLOBAL.PAS
   MYUTILS.TPU:MYUTILS.PAS MYGLOBAL.TPU MYUTIL.OBJ
```

and it would execute exactly as before.

If you're using Borland Pascal and you enable automatic dependency checking in MAKE, you can remove all explicit dependencies that have .OBJ files as targets.

*Note that with the **-N** compatibility option, the searches go in the **opposite** direction: from the bottom of the makefile up.*

You can write several implicit rules with the same target extension. If more than one implicit rule exists for a given target extension, the rules are checked in the order in which they appear in the makefile, until a match is found for the source extension, or until MAKE has checked all applicable rules.

MAKE uses the first implicit rule that involves a file with the source extension. Even if the commands of that rule fail, no more implicit rules are checked.

All lines following an implicit rule, up to the next line that begins without whitespace or to the end of the file, are considered to be part of the command list for the rule.

## Writing MAKE macros

Often, you'll find yourself using certain commands, file names, or options again and again in your makefile. For instance, if you're writing a Pascal program to be compiled for Windows, all your BPC commands will use the option **/CW**. But suppose you wanted to switch and compile to DOS. You could change all the **/CW** options to **/CD**, or you could define a macro.

A *macro* is a name that represents some string of characters. A macro definition gives a macro name and the expansion text; thereafter, when MAKE encounters the macro name, it replaces the name with the expansion text.

Suppose you defined the following macro at the start of your makefile:

```
TARGET=/CW
```

This line defines the macro **TARGET**, which is now equivalent to the string **/CW**. Using this macro, you could write each command to invoke the Pascal compiler to look something like this:

```
BPC $(TARGET) MYPROG.PAS
```

When you run MAKE, each macro (in this case, `$(TARGET)`) is replaced with its expansion text (here, **/CW**. The command that's actually executed would be

```
BPC /CW MYPROG.PAS
```

Now, changing target platforms is easy. If you change the first line to

```
TARGET=/CD
```

you've changed all the commands to compile for DOS. In fact, if you leave out the first line altogether, you can specify which platform you want each time you run MAKE, using the **–D** (define) command-line option:

```
make -DTARGET=/CD
```

This tells MAKE to treat **TARGET** as a macro with the expansion text **/CD**.

## Defining macros

Macro definitions take the form

```
macro_name = expansion text
```

where *macro_name* is the name of the macro. *macro_name* should be a string of letters and digits with no whitespace in it, although you can have whitespace between *macro_name* and the equal sign (=). The *expansion text* is any arbitrary string containing letters, digits, whitespace, and punctuation; it ends with a newline character.

If *macro_name* has previously been defined, either by a macro definition in the makefile or on the MAKE command line, the new definition replaces the old.

Case is significant in macros; that is, the macro names **target**, **Target**, and **TARGET** are all different.

## Invoking macros

You invoke macros in your makefile using this format:

```
$(macro_name)
```

You need the parentheses for all invocations, except when the macro name is just one character long. This construct—$(*macro_name*)—is known as a *macro invocation*.

When MAKE encounters a macro invocation, it replaces the invocation with the macro's expansion text. If the macro is not defined, MAKE replaces it with the null string.

### Using environment variables as macros

If you invoke a macro where *macro_name* hasn't been defined in the makefile or on the command line, MAKE will try to find *macro_name* as a DOS environment variable. If MAKE does find it in the environment, the expansion text will be the value of the environment variable.

Macros defined in the makefile or on the command line override environment variables of the same name unless the **–e** option was specified.

## Substitution within macros

You can invoke a macro while simultaneously changing some of its text by using a special format of the macro invocation format. Instead of the standard macro invocation form, use

```
$(macro_name:text1=text2)
```

In this form, every occurrence of *text1* in *macro_name* will be replaced with *text2*. *macro_name* can also be one of the predefined macros. This is useful if you'd prefer to keep only a single list of files in a macro. For example, in the following example, the macro **SOURCE** contains a list of all the TASM source files a target depends on. The TLINK command line changes all the .ASM extensions to the matching .OBJ object files and links those.

Note that no extraneous whitespace should appear between the **:** and **=**. If spaces appear after the colon, MAKE will attempt to find the string, including the preceding space.

```
SOURCE = F1.ASM F2.ASM F3.ASM
MYAPP.EXE: $(SOURCE)
  TASM $(SOURCE)
    TLINK $(SOURCE:.ASM=.OBJ)
```

### Special considerations

**Macros in macros:** Macros cannot be invoked on the left side (*macro_name*) of a macro definition. They can be used on the right side (*expansion text*), but they are not expanded until the macro being defined is invoked. In other words, when a macro invocation is expanded, any macros embedded in its expansion text are also expanded.

**Macros in rules:** Macro invocations are expanded immediately in rule lines.

See page 23 for information on directives.

**Macros in directives:** Macro invocations are expanded immediately in **!if** and **!elif** directives. If the macro being invoked in an **!if** or **!elif** directive is not currently defined, it is expanded to the value 0 (FALSE).

**Macros in commands:** Macro invocations in commands are expanded when the command is executed.

## Using predefined macros

MAKE comes with several special macros built in: **$d**, **$***, **$<**, **$:**, **$.**, **$&**, **$@**, **$****, and **$?**. The first is a test to see if a macro name is defined; it's used in the conditional directives **!if** and **!elif**. The others are file-name macros, used in explicit and implicit rules. Finally, MAKE defines several other macros; see Table 2.3.

Table 2.3
MAKE predefined macros

| | |
|---|---|
| **__MSDOS__** | "1" if running MAKE under DOS |
| **__MAKE__** | MAKE's version number in hexadecimal (for this version, "0x0360") |
| **MAKE** | MAKE's executable file name |
| **MAKEFLAGS** | Any options used on the MAKE command line |
| **MAKEDIR** | The directory from which MAKE was run |

Table 2.4
MAKE file-name macros

| Macro | What part of the file name it returns in an implicit rule | explicit rule |
|---|---|---|
| **$*** | Dependent base with path | Target base with path |
| **$<** | Dependent full with path | Target full with path |
| **$:** | Dependent path only | Target path only |
| **$.** | Dependent full without path | Target full without path |
| **$&** | Dependent base without path | Target base without path |
| **$@** | Target full with path | Target full with path |
| **$**** | Dependent full with path | All dependents |
| **$?** | Dependent full with path | All out of date dependents |

### Defined test macro ($d)

The defined test macro (**$d**) expands to 1 if the given macro name is defined, or to 0 if it is not. The content of the macro's expansion text does not matter. This special macro is allowed only in **!if** and **!elif** directives.

For example, suppose you want to modify your makefile so that if you don't specify how code is to be compiled, it will be compiled for DOS. You could put this at the start of your makefile:

```
!if !$d(TARGET)   # if TARGET is not defined
TARGET=/CD          # define it to be DOS
!endif
```

If you then invoke MAKE with the command line

```
MAKE -DTARGET=/CW
```

then **TARGET** is defined as */CW*. If, however, you just invoke MAKE by itself,

```
MAKE
```

then **TARGET** is defined as */CD*, your "default" target platform.

File-name macros

The various file-name macros work in similar ways, expanding to some variation of the full path name of the file being built.

### Base file-name macro ($*)

The base file-name macro is allowed in the commands for an explicit or an implicit rule. This macro (**$***) expands to the file name being built, excluding any extension, like this:

```
File name is A:\P\TESTFILE.PAS
$* expands to A:\P\TESTFILE
```

Suppose you've defined the following macro:

```
TURBO = C:\BP\BIN
```

You could modify the explicit MYAPP.EXE rule to look like this:

```
MYAPP.EXE: MYAPP.PAS MYGLOBAL.TPU MYUTILS.TPU
   BPC $* /T$(TURBO)
```

When the command in this rule is executed, the macro **$*** is replaced by the target file name without an extension and with a path. For implicit rules, this macro is very useful.

### Full file-name macro ($<)

The full file-name macro (**$<**) is also used in the commands for an explicit or implicit rule. In an explicit rule, **$<** expands to the full target file name (including extension), like this:

```
File name is A:\P\TESTFILE.PAS
$< expands to A:\P\TESTFILE.PAS
```

In an implicit rule, **$<** takes on the file name plus the source extension. For example, the implicit rule

```
.PAS.TPU:
  BPC $*.PAS
```

can be rewritten as

```
.PAS.TPU:
  BPC $<
```

### File-name path macro ($:)

This macro expands to the path name (without the file name), like this:

```
File name is A:\P\TESTFILE.pas
$: expands to A:\P\
```

### File-name and extension macro ($.)

This macro expands to the file name, with an extension but without the path name, like this:

```
File name is A:\P\TESTFILE.PAS
$. expands to TESTFILE.PAS
```

### File-name only macro ($&)

This macro expands to the file name only, without path or extension, like this:

```
File name is A:\P\TESTFILE.PAS
$& expands to TESTFILE
```

### Full target name with path macro ($@)

This macro expands to the full target file name with path and extension, like this:

```
File name is A:\P\TESTFILE.PAS
$@ expands to A:\P\TESTFILE.PAS
```

The **$@** macro is similar to the **$<** macro, except that **$@** expands to the full target file name in *both* implicit and explicit rules, which expands to the target in an explicit rule and the dependent in an implicit rule.

### All dependents macro ($**)

In an explicit rule, this macro expands to all the dependents of the target, including the full file name with path and extension. For example, in the following explicit rule, the **$**** will be replaced with `P1.OBJ P2.OBJ`, the two dependents of `ALL.EXE`:

```
ALL.EXE: P1.OBJ P2.OBJ
   TLINK  $**, $*
```

### All out of date dependents macro ($?)

In an explicit rule, this macro expands to all the out of date dependents of the target, including the full file name with path and extension. Out of date dependents are those that have been modified since the target was last made. Use the **&** prefix if you want MAKE to repeat the command for each of the out of date dependents.

## Modifying Macros

If there isn't a predefined file-name macro to give you the parts of a file name you need, you can use *macro modifiers* to extract any part of a file-name macro. The format is:

```
$(macro[D | F | B | R])
```

where *macro* is one of the predefined file-name macros and D, F, B, and R are the modifiers. Note that since the macro is now longer than a single character, parentheses are necessary. The following table describes what each modifier does. The examples assume that **$<** returns `C:\BIN\PROG.OBJ`.

Table 2.5
MAKE macro modifiers

| Modifier | What part of the file name | Example |
|---|---|---|
| D | Drive and directory | $(<D) = C:\BIN\ |
| F | Base and extension | $(<F) = PROG.OBJ |
| B | Base only | $(<B) = PROG |
| R | Drive, directory, and base | $(<R) = C:\BIN\PROG |

# Controlling MAKE with directives

MAKE directives, which perform various control functions, resemble directives used in languages such as C and Pascal. You can use these directives to perform a variety of useful and

powerful actions. Some directives in a makefile begin with an exclamation point (!) as the first character of the line. Others begin with a period. This table lists the MAKE directives.

Table 2.6
MAKE directives

| | |
|---|---|
| **.autodepend** | Turns on autodependency checking. |
| **!elif** | Performs a conditional execution. |
| **!else** | Performs a conditional execution. |
| **!endif** | Performs a conditional execution. |
| **!error** | Stops MAKE and prints an error message. |
| **!if** | Begins a conditional execution. |
| **!ifdef** | Performs a conditional execution. |
| **!ifndef** | Performs a conditional execution. |
| **.ignore** | Tells MAKE to ignore return value of a command. |
| **!include** | Specifies a file to include in the makefile. |
| **.noautodepend** | Turns off autodependency checking. |
| **.noignore** | Turns off **.ignore**. |
| **.nosilent** | Prints commands before executing them. |
| **.noswap** | Tells MAKE not to swap itself in and out of memory. |
| **.path.*ext*** | Search for files in *path* with the extension .EXT. |
| **.precious** | Saves the specified target even if the commands to build the target fail. |
| **.silent** | Executes without printing the commands. |
| **.suffixes** | Determines the implicit rule to use when a target's dependent is ambiguous. |
| **.swap** | Tells MAKE to swap itself in and out of memory. |
| **!undef** | Removes the definition of a specified macro. |

## Dot directives

Each of the following directives has a corresponding command-line option, but takes precedence over that option. For example, if you invoke MAKE like this:

```
MAKE -a
```

but the makefile has a **.noautodepend** directive, then autodependency checking will be off.

**.autodepend** and **.noautodepend** turn on or off autodependency checking. They correspond to the **–a** command-line option.

**.ignore** and **.noignore** tell MAKE whether or not to ignore the return value of a command, much like placing the prefix – in front of it (described earlier). They correspond to the **–i** command-line option.

**.silent** and **.nosilent** tell MAKE whether or not to print commands before executing them. They correspond to the **–s** command-line option.

**.swap** and **.noswap** tell MAKE whether or not to swap itself out of memory. They correspond to the **–S** option.

.precious

The syntax for the **.precious** directive is:

.precious: *target* [...]

where *target* is one or more target files. **.precious** prevents MAKE from deleting the target if the commands building the target fail. This is desirable for certain targets such as libraries. Even if a build fails, you might not want the library deleted.

.path.ext

This directive, placed in a makefile, tells MAKE where to look for files of the given extension. For example, if the following is in a makefile:

```
.PATH.PAS = C:\BP\SOURCE

TEMP.EXE: TEMP.PAS
  BPC TEMP.PAS
```

MAKE will look for TEMP.PAS, the implied source file for TEMP.EXE, in C:\BP instead of the current directory.

The **.path** is also a macro that has the value of the path. The following is an example of the use of **.path**. The source files are contained in one directory, the .TPU files in another, and all the .EXE files in the current directory.

```
.PATH.TPU
.PAS.TPU

   BPC $</U(.PATH.TPU)

TEMP.EXE: TEMP.TPU
```

.suffixes

The **.suffixes** command is useful when multiple source files of the same name exist. In such cases, **.suffixes** determines the searching order for dependent files when MAKE builds a target from an implicit rule. The syntax for **.suffixes** is:

.suffixes: *.ext [.ext] [.ext]...[.ext]*

where *.ext* is a list of the file extensions subject to implicit rules.

In the following example, MYPROG.OBJ can be created from MYPROG.ASM and MYPROG.c:

```
MYPROG.EXE: MYPROG.PAS MYPROG.OBJ
  BPC MYPROG.PAS
```

```
.c.OBJ:
 bcc -c $<

.ASM.OBJ:
TASM/mx $<
```

If we add `.suffixes: .asm .c` to the top of the previous makefile example, MAKE would first look for MYPROG.ASM and then MYPROG.c.

## File-inclusion directive

A file-inclusion directive (**!include**) specifies an external file to be included into the makefile. It takes the following form:

```
!include filename
```

*filename* can be surrounded by quotes (*"filename"*) or angle brackets (*<filename>*). You can nest these directives to any depth. If an **!include** directive attempts to include a file that has already been included in some outer level of nesting, the inner **!include** directive is rejected as an error.

How do you use this directive? Suppose you created the file TARGET.MAC that contains the following:

```
!ifndef TARGET
TARGET=/CD
!endif
```

You could use this conditional macro definition in any makefile by including the directive

```
!include "TARGET.MAC"
```

When MAKE encounters **!include**, it opens the specified file and reads the contents as if they were in the makefile itself.

## Conditional execution directives

Conditional execution directives (**!if**, **!ifdef**, **!ifndef**, **!elif**, **!else**, and **!endif**) give you a measure of flexibility in constructing makefiles. Rules and macros can be made conditional, so that a command-line macro definition (using the **–D** option) can enable or disable sections of the makefile.

The format of these directives parallels those in C, assembly language, and Pascal:

```
!if expression
[lines]
⋮
!endif

!if expression
[lines]
⋮
!else
[lines]
⋮
!endif

!if expression
[lines]
⋮
!elif expression
[lines]
⋮
!endif

!ifdef macro
[lines]
⋮
!endif

!ifndef macro
[lines]
⋮
!endif
```

*[lines]* can be any of the following statement types:

- macro definition
- explicit rule
- implicit rule
- !include directive
- if group
- error directive
- undef directive

The conditional directives form a group, with at least an **!if**, **!ifdef**, or **!ifndef** directive beginning the group and an **!endif** directive closing the group.

- One **!else** directive can appear in the group.
- Multiple **!elif** directives can appear between the **!if** (or **!ifdef** and **!ifndef**) and **!else** directives.
- Rules cannot be split across conditional directives.

■ Conditional directive groups can be nested to any depth.

Any rules, commands, or directives must be complete within a single source file.

All **!if**, **!ifdef**, and **!ifndef** directives must have matching **!endif** directives within the same source file. Thus the following include file is illegal, regardless of what's in any file that might include it, because it doesn't have a matching **!endif** directive:

```
!if $(FILE_COUNT) > 5
   some rules
!else
   other rules
<end-of-file>
```

The **!ifdef** directive is another way of testing whether a macro is defined. **!ifdef** *MACRO* is equivalent to **!if $d**(*MACRO*). The same holds true for **!ifndef**; **!ifndef** *MACRO* is equivalent to **!if ! $d**(*MACRO*).

## Expressions allowed in conditional directives

Expressions allowed in conditional directives can consist of decimal, octal, or hexadecimal constants and the following operators:

Table 2.7 MAKE operators

*See Chapter 6 of the Language Guide for descriptions of these operators.*

| Operator | Operation | Operator | Operation |
|---|---|---|---|
| *Unary operators* | | **&** | Bitwise AND |
| | | **\|** | Bitwise OR |
| **–** | Negation | **^** | Bitwise XOR |
| **~** | Bit complement | | |
| **!** | Logical NOT | **&&** | Logical AND |
| | | **\|\|** | Logical OR |
| *Binary operators* | | **>** | Greater than* |
| | | **<** | Less than* |
| **+** | Addition | **>=** | Greater than or equal* |
| **–** | Subtraction | **<=** | Less than or equal* |
| ***** | Multiplication | **==** | Equality* |
| **/** | Division | **!=** | Inequality* |
| **%** | Remainder | | |
| **>>** | Right shift | *Ternary operator* | |
| **<<** | Left shift | **? :** | Conditional expression |

* This operator works with string expressions also.

These operators have the following characteristics:

■ The operators have the same precedences as they do in Pascal.

■ Parentheses can be used to group operands in an expression.

- MAKE can compare strings using the **==**, **!=**, **>**, **<**, **>=**, and **=>** operators. However, strings and numeric expressions can't be compared.

A string expression may contain spaces, but if it does it must be enclosed in quotation marks.

MAKE evaluates a conditional expression as either a 32-bit signed integer or as a character string.

### Using macros in directives

You can invoke macros within an expression; the special macro **$d()** is recognized. After all macros have been expanded, the expression must have proper syntax.

## Error directive

The error directive (**!error**) causes MAKE to stop and print a fatal diagnostic containing the text after **!error**. It takes the format

```
!error any_text
```

This directive is designed to be included in conditional directives to allow a user-defined error condition to abort MAKE. For example, you could insert the following code in front of the first explicit rule:

```
!if !$d(TARGET)
# if TARGET is not defined
!error TARGET not defined
!endif
```

If **TARGET** isn't defined, then MAKE stops with this error message:

```
Fatal makefile 4: Error directive: TARGET not defined
```

## Macro undefinition directive

The macro "undefinition" directive (**!undef**) causes any definition for the named macro to be forgotten. If the macro is currently undefined, this directive has no effect. The syntax is

```
!undef macro_name
```

## Command-line options

Here's a complete list of MAKE's command-line options. Note that case (upper or lower) *is* significant; for example, the option **–d** is not a valid substitution for **–D**. Also note that you can use either a **–** or a / to introduce the options.

Table 2.8: MAKE options

| Option | What it does |
|---|---|
| **–h** or **–?** | Prints a help message. The default options are displayed with plus signs following. |
| **–a** | Checks dependencies of include files and nested include files associated with .OBJ files. |
| **–B** | Builds all targets regardless of file dates. |
| **–d*directory*** | When used with the **–S** options, tells MAKE to write its swap file in the specified directory. *directory* can include a drive letter. |
| **–D*identifier*** | Defines the named identifier as a single character, causing an expression **!ifdef** *identifier* in the makefile to return true. |
| **[-D]*iden=string*** | Defines the named identifier *iden* as *string*. If the string contains any spaces or tabs, it must be enclosed in quotes. The **–D** option is optional. |
| **–e** | Ignores any attempt to redefine a macro whose name is the same an environment variable. (In other words, causes the environment variable to take precedence.) |
| **–f *filename*** | Uses *filename* as the makefile. If *filename* does not exist and no extension is given, tries *filename*.MAK. The space after the **–f** is optional. |
| **–i** | Ignores the exit status of all programs run and continues the build. This is equivalent to putting '–' in front of all commands in the makefile. |
| **–I*directory*** | Searches for include files in the indicated directory and in the current directory. |
| **–K** | Does not erase temporary files created by MAKE. All temporary files have the form MAKE*nnnn*.$$$, where *nnnn* ranges from 0000 to 9999. See page 11 for more on temporary files. |
| **–m** | Displays the date and time stamp of each file as MAKE processes it. |
| **–n** | Prints the commands but does not actually perform them. This is useful for debugging a makefile. |
| **–N** | Changes the behavior of **<<** for compatibility between MAKE's syntax and the syntax of Microsoft's NMAKE. Instead of redirecting contents of the temporary file created by **<<** to standard input, **<<** resembles **&&** and creates a response file.<br><br>Searches for implicit rules for a target from the bottom instead of from the top of the makefile. |
| **–p** | Displays all macro definitions and implicit rules before executing the makefile. |
| **–r** | Ignores the rules (if any) defined in BUILTINS.MAK. |
| **–s** | Supresses the printing of each command as Make executes it. |

Table 2.8: MAKE options (continued)

| | |
|---|---|
| **–S** | Swaps MAKE out of memory while executing commands. This significantly reduces the memory overhead of MAKE, allowing it to compile very large modules. |
| **–U*identifier*** | Cancels any previous definitions of the named identifier. |
| **–W** | Writes instructions in MAKE.EXE to turn on the entered option or options by default, and doesn't build the program. To turn off a default option, enter it with a trailing minus symbol (-) and the **–W** option. |

## Using the –N option

The **–N** command line option increases compatibility with Microsoft's NMAKE. You should use it only when you need to build a project using makefiles created for NMAKE tools. Running MAKE without the **–N** option is preferred, since **–N** introduces some subtle differences in how makefiles work:

- `$$` expands to a single `$` and a single `$` expands to nothing.
- The caret character `^` causes the character that follows, if a special character, to be treated literally. For example,

```
TEST = this is ^
a test
```

will cause **TEST** to expand to `this is \na test` where `\n` is the C symbol for a new line. It's especially useful when you need to end a line with the line continuation character:

```
SOURCEDIR = C:\BP\BIN^\
```

- If the caret is followed by a normal character (one without a special meaning), the caret will be ignored.
- The **$d** macro won't be the special defined test macro. Use the **!ifdef** and **!ifndef** directives instead.
- Predefined macros that return paths only will *not* end in a trailing backslash. For example, without the **–N** switch **$(<D)** might return C:\BIN\, but with the **–N** switch, the same **$(<D)** macro would return C:\BIN.
- Unless there's a matching **.suffixes** directive, MAKE searches for implicit rules from the bottom of the makefile up.
- The **$*** macro always expands to the target name. (In normal mode, **$*** expands to the dependent in an implicit rule.)

### Compatibility option

If you specified **–N** on the MAKE command line, the **<<** operator changes its behavior to be more like that of the **&&** operator; that is, the temporary file isn't redirected to standard input, it's just created on the fly for use mainly as a response file. This behavior is consistent with Microsoft's NMAKE.

The format for this version of the **<<** operator is

```
   command <<[filename1] ... <<[filenameN]
text
⋮

<<[KEEP | NOKEEP]
text
⋮
<<[KEEP | NOKEEP]
```

*Note that there must be no space after the << and before the KEEP or NOKEEP option.*

The KEEP option tells MAKE not to delete the file after it's been used. If you don't specify anything or specify NOKEEP, MAKE will delete the temporary file (unless you specified the **–K** option to keep temporary files).

## Using the –f option

If you want to use a file with a name other than MAKEFILE or MAKEFILE.MAK, you give MAKE the file ( **–f**) option, like this:

```
MAKE -f MYFILE.MAK
```

If the **–f** option isn't entered, MAKE looks for a file by the name of either MAKEFILE or MAKEFILE.MAK. If it can't find either of these files, and if any targets were entered on the MAKE command line, MAKE tries to build the targets using BUILTINS.MAK.

Make also accepts the DOS wildcards * and **?** in file names.

## Using BUILTINS.MAK

You will often find that there are MAKE macros and rules that you use again and again. There are three ways of handling them.

- First, you can put them in every makefile you create.
- Second, you can put them all in one file and use the **!include** directive in each makefile you create. (See page 23 for more on directives.)

- Third, you can put them all in a BUILTINS.MAK file.

Each time you run MAKE, it looks for a BUILTINS.MAK file; however, there is no requirement that any BUILTINS.MAK files exist. If MAKE finds a BUILTINS.MAK file, it interprets that file first. If MAKE cannot find a BUILTINS.MAK file, it proceeds directly to interpreting MAKEFILE or MAKEFILE.MAK (or whatever makefile you specify with the **–f** option).

The first place MAKE searches for BUILTINS.MAK is the current directory. If it's not there, MAKE searches the directory MAKE.EXE was invoked from. You should place the BUILTINS.MAK file in the same directory as the MAKE.EXE file.

MAKE always searches for the makefile in the current directory only. This file contains the rules for the particular executable program file being built. Both BUILTINS.MAK and the makefile files have identical syntax rules.

MAKE also searches for any **!include** files (see page 26 for more on this MAKE directive) in the current directory. If you use the **–I** (include) option, it will also search in the directory specified with the **–I** option.

## Returning MAKE exit codes

At the end of the search and execute routine, MAKE returns the following codes:

Table 2.9
Exit codes

| Code | Description |
|---|---|
| 0 | No error encountered. |
| 1 | The build couldn't be completed correctly. This code is returned either when MAKE's **–i** option is entered or when the **.Ignore** directive is used. |
| 2 | An error, such as incorrect makefile syntax or a user-break occurred. |
| 4 | Insufficient memory to execute the makefile instructions. |

## Controlling error-checking

You can use the following commands to turn off error checking:

- **.Ignore** turns off error checking for a selected portion of the makefile.
- **–i** MAKE command-line option turns off error checking for the entire makefile.

- **–num** command operator, which is entered as part of a rule, turns o error checking for the related command if the exit code exceeds the specified number.
- **–** command operator turns off error checking for the related command regardless of the exit code.

C H A P T E R

3

# Message watcher

WinSight is a debugging tool that gives you information about windows, window classes, and messages. You can use it to study a Windows application—yours or others—to see how windows and window classes are created and used, and what messages the windows receive.

You can configure WinSight to trace messages by

- Window
- Window class
- Message type
- A combination of these

WinSight is a passive observer: It intercepts and displays information about messages, but it does not keep messages from getting to other applications.

## Getting started

Double-clicking the WinSight icon displays the main window in its default configuration, which is a list of all the windows currently active on the desktop.

You'll probably want to use a mouse to manipulate items in the WinSight window. Table 3.1 summarizes the mouse actions and their keyboard and menu equivalents.

Table 3.1: Mouse and keyboard actions

| Desired action | With the mouse | With the keyboard | With menus |
|---|---|---|---|
| Select an item | Left-click | ↑ or ↓ | |
| Move selection bar | | *Ctrl*+↑ or *Ctrl*+↓ | |
| Toggle highlighted item | *Ctrl*+Left-click | *Spacebar* | |
| Show details of item | Left-Double-click | *Enter* | Spy I Open detail |
| Expand window tree | Left-click <+> | + | Tree I Expand one level |
| Expand a branch | Right-click <+> | * | Tree I Expand branch |
| Collapse window tree | Left-click <-> | – | Tree I Collapse branch |
| Expand tree completely | | *Ctrl*+* | Tree I Expand all |
| Activate following pane | | *Tab* or *F6* | |
| Activate previous pane | | *Shift*+*Tab* or *Shift*+*F6* | |

## Exiting WinSight

When you are finished with WinSight, exit using the Spy I Exit menu command.

# Choosing a view

You can select various views and degrees of information for your message tracing.

## Picking a pane

WinSight offers three panes that can appear within its main window: a Window Tree, a Class List, and a Message Trace. You can choose to look at any or all of the panes. WinSight will automatically tile the panes within the main window, but you can resize them to suit your needs.

- The Window Tree pane displays the hierarchy of windows on the desktop. This is the default display when you start WinSight.
- The Class List pane shows all the currently registered window classes.
- The Message Trace pane displays information about messages received by selected windows or window classes.

You can hide or display panes at any time, using the View menu. Information and selections are not lost when a pane is hidden.

### Arranging the panes

When you have two or more panes displayed in the main window, you can choose to display them either stacked on top of one another or arrayed side by side. The Split Vertical and Split Horizontal commands on the view menu toggle between these configurations.

### Getting more detail

Within the window tree and class list panes, you can get more detailed information about a selected window or class. Choosing Open Detail from the Spy menu will bring up detail windows on selected windows or selected classes, depending on which pane is focused. See the *ObjectWindows Programming Guide* for a description of the TWndClass attributes displayed in the detail windows.

**Window detail** Double-clicking or pressing *Enter* on an item in the window tree brings up a Window Detail window that shows full detailed information on that window, in addition to information about that window's class.

**Class detail** Double-clicking or pressing *Enter* on an item in the class list pane brings up a Class Detail window that shows full detailed information about that window class.

## Using the window tree

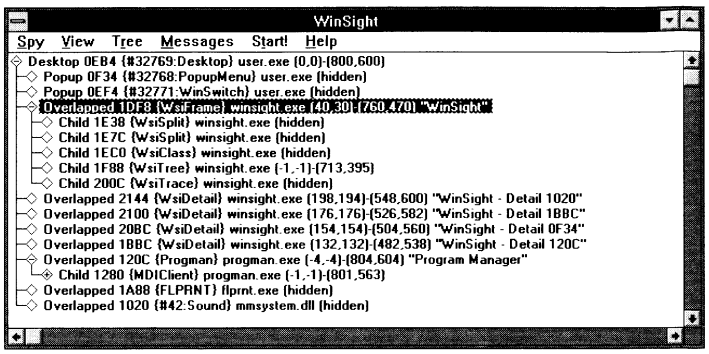
The Window Tree pane shows an outline of the hierarchy of all existing windows. You can use this display in several ways to:

- Determine what windows are actually present
- See the status of windows, including hidden windows
- See which windows are receiving messages
- Select the windows you want to trace messages for

The lines on the left show the tree structure. Each window is connected to its parent, siblings, and children with these lines. The lines are similar to those in the File Manager. The diamond next to each window shows whether the window has any

children. If it is empty, there are no children. If it contains a plus sign, there are children but they are collapsed out in the tree display. If it contains a minus sign, there are children and they are visible in the tree display (at least one level of child windows is visible; further levels may be collapsed). Figure 3.1 shows a typical Window Tree pane.

Figure 3.1
Window Tree pane




## Pruning the tree

Within the Window Tree pane, you can show or hide lists of child windows.

### Showing child windows

*Notice that the plus sign in the diamond changes to a minus sign to show that the window's children are shown.*

To show a window's children, click on the diamond next to the window with the left mouse button or press +. All windows which are children of the window appear. To show *all* the levels of child windows (children of children, etc.), click the right mouse button.

### Hiding child windows

*Notice that the minus sign in the diamond changes back to a plus sign, showing that the window's children are hidden.*

To hide all of a window's child windows, click (or double-click) the diamond next to that window or press –. All child windows (and their child windows, if any) will disappear from the Window Tree.

## Finding a window

WinSight has a special mode for locating windows. It can work in two ways, either identifying the line in the Window Tree that corresponds to a window you point at, or highlighting a window you select in the Window Tree.

***Important!*** *All other applications are suspended while you're in Find Window mode.*

In either case, you enter Find Window mode by choosing Spy | Find Window. In this mode, whenever the mouse passes into the boundaries of a window, a thick border appears around that window, and the window is selected in the Window Tree pane.

Alternatively, once in Find Window mode, you can select windows in the Window Tree with the mouse or cursor keys, and WinSight will put the thick border around the selected window or windows. If you press *Enter*, you will see the Window Detail window for the selected window.

### Leaving Find Window mode

Once you have located the window you want, you can leave Find Window mode by clicking the mouse button or by pressing the *Esc* key. This removes the border from the screen, leaving the current window selected in the Window Tree pane.

### Spying on windows

Once you have selected a window from the Window Tree, you can trace the messages going to that window by choosing Messages | Selected Windows. Changing the selection in the Window Tree immediately changes which windows have messages traced.

You can also choose to spy on all windows, regardless of what is selected in the Class List or the Window Tree, by choosing Messages | All Windows.

Choosing Messages | Selected Windows or Windows | All Windows when the Message Trace pane is hidden causes the Message Trace pane to become visible.

Choosing Messages | Trace off disables message tracing without causing the Message Trace pane to become hidden.

## Working with classes

Sometimes, instead of choosing specific windows to trace messages for, you might want to look at messages for entire classes of windows. WinSight helps you do this using the Class List pane.

A class in WinSight refers to the class name with which the window class was registered with Windows.

### Using the Class List pane

The Class List pane works much the same way as the Window Tree pane, but it's much simpler, since classes are not hierarchical. The Class List pane shows all the currently registered window classes. You can get full details about a class by double-clicking it or pressing *Enter* when it's selected. The diamonds to the left work the same way as they do in the Window Tree pane.

The diamonds have another purpose: Whenever the window receives any messages, they invert color momentarily. This gives you an overview of which windows are currently receiving messages. If a window's children are collapsed in the tree, the diamond for that window inverts to show message activity in the children.

### Spying on classes

Once you have selected a class from the Class List pane, you can choose Messages | Selected Classes to trace only messages going to that particular class. If the Message Trace pane is hidden when you choose Messages | Selected Classes, it will become visible.

Note that tracing messages to a class enables you to see all messages to windows of that class, including creation messages, which would otherwise not be accessible.

Changing the selection in the Class List while tracing messages by class immediately changes which classes have their messages traced.

## Taking time out

Several parts of WinSight can be disabled and reenabled at your control.

### Turning off tracing

Choosing Messages | Trace Off turns off message tracing. The Message Trace pane remains visible, and tracing resumes when you choose another of the message-tracing menu options, Selected Classes, Selected Windows, or All Windows.

### Suspending screen updates

The Stop! command on the main menu bar turns off all real-time updating in WinSight. Normally, all the panes are kept current as classes are registered, windows are created and destroyed, and messages are received. Choosing Stop! suspends all of these, and changes the menu command to Start!. Choosing Start! resumes normal operation.

Using Stop! has two main purposes: It gives you a chance to study a particular situation, and it removes the overhead of having WinSight update itself constantly.

## Choosing messages to trace

WinSight gives you several ways to narrow down the tracing of messages. Watching the messages to specific windows and window classes is described in "Spying on windows" and "Spying on classes," earlier in this chapter.

You might also want to specify which types of messages you want to trace, regardless of which windows you're spying on. This is done with the Message Trace Options dialog box, displayed by choosing the Messages | Options... command.

### Filtering out messages

By default, WinSight traces all messages. If you uncheck the All Messages box, you can then select any or all of ten subgroups of messages. These subgroups are described in Tables 3.2 through 3.14. Checking All Messages again disables all the separate subgroups and will trace all messages.

### Message tracing options

The Message Trace Options dialog box gives two other useful options; one determines the format of the Message Trace pane's display, and the other logs traced messages to a file.

### Formatting message parameters

Normally, the Message Trace pane interprets each message's parameters and displays them in a readable format. You can disable this by checking Hex Only in the Message Trace Options dialog box. When Hex Only is checked, message parameters appear only as hex values of *wParam* and *lParam*.

### Logging traced messages

Information on traced messages usually goes only to the Message Trace pane. By checking Log File in the Message Trace Options dialog box and typing a file name, you can capture the message trace to a log file. If the file already exists, messages are appended to the end. To stop logging message traces to the file, uncheck Log File.

To send logged messages to a printer or other device, type the name of the device instead of a file name. For example, typing PRN for the log file would send output to the printer port.

Table 3.2
Mouse messages

| | |
|---|---|
| wm_HScroll | wm_MouseActivate |
| wm_LButtonDblClk | wm_MouseMove |
| wm_LButtonDown | wm_RButtonDblClk |
| wm_LButtonUp | wm_RButtonDown |
| wm_MButtonDblClk | wm_RButtonUp |
| wm_MButtonDown | wm_SetCursor |
| wm_MButtonUp | wm_VScroll |

Table 3.3
Window messages

| | |
|---|---|
| wm_Activate | wm_Move |
| wm_ActivateApp | wm_Paint |
| wm_CancelMode | wm_PaintIcon |
| wm_ChildActivate | wm_QueryDragIcon |
| wm_Close | wm_QueryEndSession |
| wm_Create | wm_QueryNewPalette |
| wm_CtlColor | wm_QueryOpen |
| wm_Destroy | wm_Quit |
| wm_Enable | wm_SetFocus |
| wm_EndSession | wm_SetFont |
| wm_EraseBkgnd | wm_SetRedraw |
| wm_GetDlgCode | wm_SetText |
| wm_GetMinMaxInfo | wm_ShowWindow |
| wm_GetText | wm_Size |
| wm_GetTextLength | wm_WindowPosChanged |
| wm_IconEraseBkgnd | wm_WindowPosChanging |
| wm_KillFocus | |

Table 3.4
Input messages

| | |
|---|---|
| wm_Char | wm_MenuSelect |
| wm_CharToItem | wm_ParentNotify |
| wm_Command | wm_SysChar |

Table 3.4: Input messages (continued)

| | |
|---|---|
| wm_DeadChar | wm_SysDeadChar |
| wm_KeyDown | wm_SysKeyDown |
| wm_KeyLast | wm_SysKeyUp |
| wm_KeyUp | wm_Timer |
| wm_MenuChar | wm_VKeyToItem |

Table 3.5
System messages

| | |
|---|---|
| wm_Compacting | wm_Power |
| wm_DevModeChange | wm_QueueSynch |
| wm_EnterIdle | wm_SpoolerStatus |
| wm_FontChange | wm_SysColorChange |
| wm_Null | wm_SysCommand |
| wm_PaletteChanged | wm_TimeChange |
| wm_PaletteIsChanging | wm_WinIniChange |

Table 3.6
Initialization messages

| | |
|---|---|
| wm_InitDialog | wm_InitMenuPopup |
| wm_InitMenu | |

Table 3.7
Clipboard messages

| | |
|---|---|
| wm_AskCBFormatName | wm_PaintClipboard |
| wm_ChangeCBChain | wm_Paste |
| wm_Clear | wm_RenderAllFormats |
| wm_Copy | wm_RenderFormat |
| wm_Cut | wm_SizeClipboard |
| wm_DestroyClipboard | wm_Undo |
| wm_DrawClipboard | wm_VScrollClipboard |
| wm_HScrollClipboard | |

Table 3.8
DDE messages

| | |
|---|---|
| wm_dde_Ack | wm_dde_Poke |
| wm_dde_Advise | wm_dde_Request |
| wm_dde_Data | wm_dde_Terminate |
| wm_dde_Execute | wm_dde_Unadvise |
| wm_dde_Initiate | |

Table 3.9
Non-client messages

| | |
|---|---|
| wm_NCActivate | wm_NCMButtonDblClk |
| wm_NCCalcSize | wm_NCMButtonDown |
| wm_NCCreate | wm_NCMButtonUp |
| wm_NCDestroy | wm_NCMouseMove |
| wm_NCHitTest | wm_NCPaint |
| wm_NCLButtonDblClk | wm_NCRButtonDblClk |
| wm_NCLButtonDown | wm_NCRButtonDown |
| wm_NCLButtonUp | wm_NCRButtonUp |

Table 3.10
Control messages

bm_GetCheck
bm_GetState
bm_SetCheck
bm_SetState
bm_SetStyle

bn_Clicked
bn_Disable
bn_DoubleClicked
bn_Hilite
bn_Paint
bn_Unhilite

cb_AddString
cb_DeleteString
cb_Dir
cb_FindString
cb_FindStringExact
cb_GetCount
cbn_CloseUP
cbn_DblClk
cbn_DropDown
cbn_EditChange
cbn_EditUpdate
cbn_KillFocus
cbn_SelChange
cbn_SetFocus

dm_GetDefID
dm_SetDefID

em_CanUndo
em_EmptyUndoBuffer
em_FmtLines
em_GetFirstVisibleLine
em_GetHandle
em_GetLine
em_GetLineCount
em_GetModify
em_GetPasswordChar
em_GetRect
em_GetSel
em_GetThumb
em_GetWordBreakProc
em_LimitText
em_LineFromChar
em_LineIndex
em_LineLength
em_LineScroll
em_MsgMax
em_ReplaceSel

cb_GetCurSel
cb_GetDroppedControlRect
cb_GetDroppedState
cb_GetEditSel
cb_GetExtendedUI
cb_GetItemData
cb_GetItemHeight
cb_GetLBText
cb_GetLBTextLen
cb_InsertString
cb_LimitText
cb_MsgMax
cb_ResetContent
cb_SelectString
cb_SetCurSel
cb_SetEditSel
cb_SetItemData
cb_SetItemHeight
cb_ShowDropDown
en_MaxText
en_SetFocus
en_Update
en_VScroll

lb_AddString
lb_DeleteString
lb_Dir
lb_FindString
lb_FindStringExact
lb_GetCaretIndex
lb_GetCount
lb_GetCurSel
lb_GetHorizontalExtent
lb_GetItemData
lb_GetItemHeight
lb_GetItemRect
lb_GetSel
lb_GetSelCount
lb_GetSelItems
lb_GetText
lb_GetTextLen
lb_GetTopIndex
lb_InsertString
lb_MsgMax
lb_ResetContent
lb_SelectString
lb_SelItemRange
lb_SetCaretIndex
lb_SetColumnWidth
lb_SetCurSel
lb_SetHorizontalExtent

Table 3.10: Control messages (continued)

| | |
|---|---|
| em_Scroll | lb_SetItemData |
| em_SetFont | lb_SetItemHeight |
| em_SetHandle | lb_SetSel |
| em_SetModify | lb_SetTabStops |
| em_SetPasswordChar | lb_SetTopIndex |
| em_SetRect | |
| em_SetRectNP | lbn_DblClk |
| em_SetSel | lbn_KillFocus |
| em_SetTabStops | lbn_SelCancel |
| em_SetWordBreak | lbn_SelChange |
| em_Undo | lbn_SetFocus |
| en_Change | stm_GetIcon |
| en_ErrSpace | stm_SetIcon |
| en_HScroll | |
| en_KillFocus | |

Table 3.11
Pen messages

| | |
|---|---|
| wm_GlobalRcChange | wm_RcResult |
| wm_HeditCtl | wm_Skb |
| wm_HookRcResult | |

Table 3.12
Multimedia messages

| | |
|---|---|
| mm_Joy1ButtonDown | mm_Mim_LongData |
| mm_Joy1ButtonUp | mm_Mim_LongError |
| mm_Joy1Move | mm_Mim_Open |
| mm_Joy1ZMove | mm_Mom_Close |
| mm_Joy2ButtonDown | mm_Mom_Done |
| mm_Joy2ButtonUp | mm_Mom_Open |
| mm_Joy2Move | mm_Wim_Close |
| mm_Joy2ZMove | mm_Wim_Data |
| mm_MCINotify | mm_Wim_Open |
| mm_Mim_Close | mm_Wom_Close |
| mm_Mim_Data | mm_Wom_Done |
| mm_Mim_Error | mm_Wom_Open |

Table 3.13
Other messages

| | |
|---|---|
| wm_CommNotify | wm_MDIGetActive |
| wm_CompareItem | wm_MDIIconArrange |
| wm_DeleteItem | wm_MDIMaximize |
| wm_DrawItem | wm_MDINext |
| wm_DropFiles | wm_MDIRestore |
| wm_GetFont | wm_MDISetMenu |
| wm_MDIActivate | wm_MDITile |
| wm_MDICascade | wm_MeasureItem |
| wm_MDICreate | wm_NextDlgCtl |
| wm_MDIDestroy | wm_SystemError |

Table 3.14
Messages not documented by Microsoft

| | |
|---|---|
| wm_AltTabActive | wm_IsActiveIcon |
| wm_BeginDrag | wm_LBTrackPoint |
| wm_ConvertRequest | wm_NextMenu |
| wm_ConvertResult | wm_QueryDropObject |
| wm_DragLoop | wm_QueryParkIcon |
| wm_DragMove | wm_SetHotkey |
| wm_DragSelect | wm_SetVisible |
| wm_DropObject | wm_SizeWait |
| wm_EnterMenuLoop | wm_SyncPaint |
| wm_EnterSizeMove | wm_SyncTask |
| wm_ExitMenuLoop | wm_SysTimer |
| wm_ExitSizeMove | wm_Testing |
| wm_FileSysChange | wm_Yomichar |
| wm_GetHotkey | |

# WinSight windows

This section describes the various windows and window panes WinSight provides.

## Class List pane

Displays all registered window classes.

### Format

```
Class (Module) Function Styles
```

The diamonds have one more purpose: whenever the window receives any messages, they invert color momentarily. This gives you an overview of which windows are currently receiving messages. If a window's children are collapsed in the tree, the diamond for that window will invert to show message activity in the children.

*Class* is the name of the class. Some predefined Windows classes have numeric names. For example, the Popup menu class uses the number 32768 as its name. These classes are shown with both the number and a name, such as #32768:PopupMenu. The actual class name is only the number itself in the MAKEINTRESOURCE format also used for resource IDs.

*Module* is the name of the executable module (.EXE or .DLL) that registered the class.

*Function* is the address of the class window function.

*Styles* is a list of the cs_ styles for the class. The names are the same as the cs_ definitions in WinTypes, except the cs_ is removed and the name is in mixed case. See the *ObjectWindows Programming Guide* for a detailed description of the window class style attributes.

## Window Tree pane

Displays all windows in existence, showing their parent-child relationships.

### Format

```
Handle {Class} Module Position "Title"
```

The lines on the left show the tree structure. Each window is connected to its parent, siblings, and children with these lines. The lines are in the same fashion as the File Manager. The diamond next to each window shows whether the window has any children. If it is empty, there are no children. If it contains a plus sign, there are children but they are collapsed out in the tree display. If it contains a minus sign, there are children and they are visible in the tree display (at least one level of child windows is visible; further levels may be collapsed).

*Handle* is the window handle as returned by *CreateWindow*.

*Class* is the window class name, as described in the Class List pane.

*Module* is the name of the executable module (.EXE or .DLL) that created the window. Strictly speaking, this is the name of the module owning the data segment passed as the *hInstance* parameter to *CreateWindow*.

*Position* is either hidden if the window is hidden, or indicated by using coordinates in the format

```
xBegin,yBegin - xEnd,yEnd
```

if the window is visible. For top-level windows, these are screen coordinates. For child windows, they are coordinates within the parent window's client area, as used in *CreateWindow* for a child window.

*Title* is the window title or text, as returned by *GetWindowText* or a wm_GETTEXT message. If the title is the null string, the quotes are omitted.

## Message Trace pane

Displays messages received by selected window classes or windows. Messages received via *SendMessage* are shown twice, once when they are sent and again when they return to show the return value. Dispatched messages are shown once only, since their return value is meaningless. The message display is indented to show how messages are nested within other messages.

### Format

```
Handle ["Title" or {Class}] Message Status
```

*Handle* is the window handle receiving the message.

*Title* is the window's title. If the title is the null string, the class name is displayed instead, in curly braces.

*Message* is the message name as defined by Windows. They are displayed in WinSight in all uppercase letters. Known undocumented Windows messages are shown in lowercase. Unknown message numbers (user-defined) are shown as *wm_User+0xXXXX* if they are greater-than or equal to *wm_User*, or *wm_0xXXXX* if they are less than *wm_User*. Registered message numbers (from *RegisterWindowsMessage*) are shown with their registered name in single quotes.

*Status* is one or more of the following:

- *Dispatched* indicates the message was received via *DispatchMessage*.
- *Sent [from XXXX]* indicates the message was received via *SendMessage*. If it was sent from another window, `from XXXX` gives that window's handle. If it was sent from the same window receiving it, this is shown with `from self`. If it was sent from Windows itself, the "from" phrase is omitted.
- *Returns* indicates the message was received via *SendMessage* and is now returning.
- Additional messages might include a numeric return value or text message such as *wm_GetText*. For sent and dispatched messages, WinSight interprets the parameters and gives a readable display. For messages that have associated data structures (*wm_Create*, for example) it grabs those structures and includes them in the display.

C H A P T E R

# 4

# *Windows debugger*

WinSpector and its utilities help you perform a postmortem examination of Windows Unrecoverable Application Errors (UAEs). When a UAE occurs, WinSpector writes a log file to your disk. This log file shows you helpful information about the cause of the exception including

- The call stack which was active when an exception occurred
- Function and procedures names in the call stack
- CPU registers
- A disassembly of the machine instructions where the exception occurred
- Windows information about the program environment

## Getting started

Before using WinSpector, be sure that TOOLHELP.DLL (from Windows 3.1 or later) is in your search path. TOOLHELP.DLL is a Windows DLL that provides ways for utilities to get access to low-level system information. WinSpector uses TOOLHELP.DLL to know when an exception occurs and to obtain the system information it writes to the log file. To be safe, don't have other exception debugging tools, except for Turbo Debugger, running concurrently with WinSpector.

To use WinSpector, you can include it in the "load=" section of your WIN.INI file. Upon starting, WinSpector minimizes and requires no additional interaction. Alternatively, you can double click the WinSpector icon to run WinSpector after you load Windows.

When a UAE occurs, WinSpector automatically writes a log file to your disk and displays a dialog box with a brief exception report. You can click OK to remove the box and then read the log file to discover the cause of the exception. See the section "Specifying options" for details about configuring the log file.

## Using WinSpector when an exception occurs

When you get a UAE (in Windows 3.0) or a fault (in Windows 3.1), WinSpector goes to work, writing the information you've requested to files. The log file WINSPCTR.LOG is a text file you can read. WINSPCTR.BIN is a binary file that the DFA utility analyzes. See the section "Other WinSpector tools" for details about WINSPCTR.BIN.

### WINSPCTR.LOG

The first line of the report(s) in WINSPCTR.LOG gives the date and time when the exception occurred, for example,

```
WinSpector failure report - 6/18/1992 11:04:25
```

The second line of the report(s) lists

- What type of exception occurred
- The module name
- The logical address
- The physical address
- The currently active task at the time of the exception

A second line might look like this:

```
Exception 13 at USER 002A:0429 (079F:0429)  (TASK=BAD)
```

Table 4.1 lists frequently encountered types of exceptions.

Table 4.1
Exception types

| Number | Name | Description |
|---|---|---|
| 0 | Division by zero | Occurs during a DIV or an IDIV interaction if the divisor is 0. |
| 12 | Stack fault | Usually occurs when there is not enough room on the stack to proceed. |
| 13 | General protection fault | All protection errors which don't cause another exception cause an exception 13. |

Exception 13 errors include, but are not limited to, the following errors:

- Invalid selector loaded into a segment register.
- Segment limit exceeded. Although the selector is valid, the offset value is greater than the segment limit (for example, an array index out of bounds error in DS, ES, or other segments).
- Execution is transferred to a non-executable segment, such as, a bad function pointer.
- Accessing DS, ES, FS, or GS registers containing a null selector. (This error can cause a 0 to appear in the segment register of the log file.)

A log file lists both the physical and logical addresses where the exception occurred. These two types of addresses are important to Windows programs for the following reasons. When a program is loaded, Windows allocates space for each logical segment and assigns each segment a unique selector. The selector and its offset are combined form a physical address. When a Windows .EXE file is linked, each segment is placed in a different section of the file, and a segment table is created. A logical address, which is actually a segment's position in the Windows segment table, consists of a module name, a logical segment, and an offset. You can run TDUMP on the file to find out segment size and other information, or you can generate a .MAP file that contains the same kind of information.

If the stack pointer is too small at the time of exception, TOOLHELP.DLL automatically switches the stack. When this happens, the message `Stack Switched` is appended to the end of the second line of the log.

### Disassembly section

The first line of the disassembly section in the log file identifies the assembly language instruction that caused the exception.

This is followed by the next few instructions in the program. These subsequent commands are listed to provide a point of reference for finding the task that caused the exception.

For example, given the following code where ES is the segment register that contains a selector and BX is the offset into the segment,

```
079F:0429   CMP     BYTE  PTR  ES:[BX],FF
079F:042D   JNE     043A
079F:042F   CMP     WORD  PTR  [BP+06],03
079F:0435   MOV     DI, 0001
```

an exception 13 occurred because the value in BX was greater than the segment limit referenced by ES.

### Stack trace section

The first line of the stack trace section of the log identifies the function or procedure that was executing at the time of the exception. Stack Trace information includes the

- Frame number
- Module name
- Name of the closest function before the address of the one that caused the exception, plus a number indicating how far away you were from that function (This information is present only if a .SYM file is present.)
- Logical and physical address for the stack frame
- Location where your program returns after the call

When WinSpector gives function names, it looks in the .SYM file for the closest symbol name that appears before the address in the call stack. Some .SYM files do not contain information for all functions. Thus, the function name appearing in the log file is that of the closest function in the .SYM file with an address preceding the frame address. If the offset field appears to be too high, function names may not be reliable. The following stack trace information shows some of the functions that were executing at the time, BAD, our sample task, caused an exception:

```
Stack Trace:
0   User  <no info>
    CS:IP 002A:0429  (079F:0429)     SS:BP  10FF:18CA
```

```
    C:\WIN31\SYSTEM\USER.EXE
    ⋮
3   BAD   function5(unsigned long, unsigned long,unsigned long) +
0014
    CS:IP 0001:0184 (1107:0184)      SS:BP  10FF:1952
    C:\BIN\BAD.EXE
    ⋮
```

### Register section

The register section of the log file lists the values stored in the standard registers when the exception occurred as the following example shows:

```
Registers:
AX     0037
BX     0000
CX     0008
DX     10EE
SI     0037
DI     0028
```

Limits and access rights are given for the CS, DS, ES, and SS registers.

### Message queue section

The message queue section of the log file gives the last message actually received in the middle of processing. This section also lists any messages that were waiting in the queue at the time of exception. For each message, WinSpector lists the following information.

- The Window handle that identifies the destination window
- The Message ID number that identifies the message
- Two parameters that contain additional message information

Keep in mind that what is recorded in the message queue section might not be the last message the program received. Windows might bypass the message queue, for example, by using the *SendMessage* function.

The following example shows one message received and one waiting in the queue:

```
Message Queue:
Last message received:
   hWnd: 0000   msg: 0001   wParam: 0002   lParam: 00000003
Waiting in queue:
   hWnd: 0000   msg: 0001   wParam: 0002   lParam: 00000003
```

### Tasks section

In the tasks section of the log file, WinSpector lists the programs running in the system when the exception occurred. This information includes the

- Complete path for the executable file
- Module name
- Windows module handle
- Task handle
- The data segment value for the task (the instance handle)

Some of the tasks running when our application, BAD, caused an exception, include

```
C:\WIN31\SYSTEM\NWPOPUP.EXE
   Module: NWPOPUP    hModule: 142F   hTask: 141F   hInstance: 13F6
   ⋮
C:\BIN\WINSPCTR.EXE
   Module: WINSPCTR   hModule: 1397   hTask: 1387   hInstance: 135E
   ⋮
C:\BIN\BAD.EXE
   Module: BAD        hModule: 1467   hTask: 1127   hInstance: 10FE
```

### Modules section

In the modules section of the log, WinSpector lists the modules that were running at the time of the exception. For each module, the following information is given:

- The path for the executable file
- The date stamp of the executable file
- The size of the file
- The name of the module
- The module handle
- The reference count indicating how many times the module is in use

Three of the modules that were running when our sample application, BAD, caused an exception include,

```
C:\WIN31\SYSTEM\KRNL386.EXE     Date: 03/02/1992   Size: 116132
   Module: KERNEL   hModule: 010F  reference count: 21
C:\WIN31\SYSTEM\SYSTEM.DRV      Date: 03/01/1992   Size: 2304
   Module: SYSTEM   hModule: 013F  reference count: 13
   ⋮
C:\C\BIN\WINSPCTR.EXE           Date: 06/02/1992   Size: 46256
   Module: WINSPCTR  hModule: 1397 reference count: 1
   ⋮
```

### USER and GDI heap information

The USER and GDI (graphics device interface) heap information section of the log shows what percentage of the USER and GDI heaps was available at the time of exception. For example,

```
USER      Free   91%
GDI       Free   83%
```

Because Windows has only 64k of internal heap space that all applications must share, it's often helpful to keep track of how much of this space is currently being used. If you find that USER and GDI are taking up a lot of heap space, check to see if you have deallocated resources you are not using.

In the About Program Manager box, Windows lists the lower of these values as the amount of free System Resources.

### System information section

The System Information section of the log file shows the mode and windows version under which your program is running. It also shows the

- Type of CPU
- Largest free block of contiguous linear memory in the system
- Total linear address space represented in pages
- Amount of free memory pages in the linear address space
- Number of pages in the system swap file

For example, for a 486 system running under Windows 3.1 enhanced mode, the amount of memory available is

```
System info: Running in enhanced mode under Windows 3.1 debug version
CPU: 80486
Largest free memory block: 3457024 bytes
Total linear memory space: 19696 K
Free linear memory space:  18212 K
Swap file Pages:           0 (0 K)
```

## Specifying options

This section describes WinSpector's options that you can set either by selecting from a list in the Preferences dialog box or by entering commands directly into the WINSPCTR.INI file. Table 4.2 on page 58 shows you how to use both methods to configure WinSpector.

## Setting log file options

You can select several options from the Preferences dialog box that control the location, contents, and display of the log file. Before you use WinSpector, enter the directory where you want WinSpector to write the log file and the name of the editor you want to use to view the file. After an exception occurs, you can choose View Log on the Latest UAE dialog box or the Preferences dialog box to see the log file. WinSpector then runs the selected editor and passes the WINSPCTR.LOG file as a command-line argument.

### Overwriting or appending reports

The Append New Reports and Overwrite Previous Reports options let you control whether WinSpector appends reports to the previous log file or overwrites the previous log file when a new report is generated.

If you choose to overwrite the previous log file, the first time an exception occurs WinSpector overwrites any previously existing log file. If subsequent exceptions occur during the same Windows session, they are appended to this same log file.

### Adding system information

You can select the System Information option if you want WinSpector to add the Task List, the Module List, and information about the USER and GDI heaps to the log file.

### Writing a report to the AUX device

If you select the AUX Summary option, WinSpector writes an abbreviated form of the report to the AUX device (usually a serial port), in addition to writing the information to the log file. To use this option, you need a terminal connected to AUX or a device driver that redirects the AUX device to a second monitor.

### Writing stack frame data

You can select the Stack Frame Data option if you want WinSpector to add a verbose stack trace display to the log file. For each stack frame that doesn't exceed 256 bytes, WinSpector performs a hex dump, starting at the SS:BP for that frame. If there are more than 256 bytes between 2 successive stack frames, the memory display is omitted for that frame. You can use this data to get the values of the parameters that were passed to the function.

It is usually easier to let DFA do the hard work of figuring out what your parameters are. However, for those cases where Turbo

Debugger information is not available, you might find a verbose trace supplies helpful information.

### Writing a postmortem dump

If you select the PostMortem Dump option, WinSpector will generate a WINSPCTR.BIN file. DFA takes a WINSPCTR.BIN file and Turbo Debugger information (either in the .EXE, .DLL or .TDS files) and translates the raw binary data into a useful form. It then generates a file that contains not only stack trace information similar to the log file but also function names, line numbers, local and global variables.

## Writing user comments

If you want to enter information about what was happening at the time of the exception, select the User Comments option. WinSpector displays a dialog box immediately after the exception log is written. You can enter comments about what was happening at the time of the exception, and WinSpector will append your comments to the log file.

Table 4.2 on page 58 describes the commands you can use to configure WinSpector.

Table 4.2: WinSpector preferences

| To set this option | from the Preferences Menu | type inside WINSPCTR.INI | Default |
|---|---|---|---|
| Specify a directory | Enter the directory name in the Directory input box. | LogDir=[directory] | Windows |
| Specify a viewer | Enter the viewer file name in the Viewer text box. | LogViewer=[viewername] | Windows Notepad |
| Append reports | Set Log File to Append New Reports. | CreateNewLog=0 | Off |
| Overwrite reports | Set Log File to Overwrite Previous Reports. | CreateNewLog=1 | On |
| Include system information in log file | Under Report Information, check System Info. | ShowSystemInfo=1 | On |
| Omit system information in log file | Under Report Information, uncheck System Info. | ShowSystemInfo=0 | Off |
| Send summary to AUX | Under Report Information, check Summary to AUX. | LogToStdAux=1 | Off |
| Omit summary to AUX | Under Report Information, uncheck Summary to AUX. | LogToStdAux=0 | On |
| Write stack frame data to log file | Under Report Information, check Stack Frame Data. | ShowStackInfo=1 | Off |
| Omit stack frame from log file | Under Report Information, uncheck Stack Frame Data. | ShowStackInfo=0 | On |
| Write PostMortem Dump | Under Report Information, check PostMortem Dump. | PostMortemDump=1 | On |
| Omit PostMortem Dump | Under Report Information, uncheck PostMortem Dump. | PostMortemDump=0 | Off |
| Add user comments to log file | Under Report Information, check User Comments. | ShowUserInfo=1 | On |
| Omit User comments from log file | Under Report Information, uncheck User Comments. | ShowUserInfo=0 | Off |

# Processing WinSpector data

DFA post-processes Turbo Debugger information that WinSpector gathered at the time of the exception. If you choose to send report information to a PostMortem dump, WinSpector writes a WINSPCTR.BIN file at the time of the exception. Then, you can

use DFA to translate the binary data in the WINSPCTR.BIN file into usable information stored in the DFA.OUT file.

## DFA output

DFA writes a file only if Turbo Debugger information exists for the file in the stack frame. The DFA output file (DFA.OUT) has a stack trace similar to the one in the WinSpector log file, except that it contains

- Function names
- Line numbers
- Local and global variables
- Data segments and their values (including the stack segment)

Because only one WINSPCTR.BIN file is written per Windows session, make sure you run DFA promptly. For example, if you get three UAE's in succession, WinSpector will write three reports to the log file, but binary data will exist for only the first report. It's best to run DFA immediately after receiving the first UAE. You may then want to rename the DFA.OUT file and delete the WINSPCTR.BIN and WINSPCTR.LOG files before continuing. Table 4.3 shows the relationship between the files created by WinSpector and DFA.

Table 4.3
DFA processing

| WINSPCTR.LOG | WINSPCTR.BIN | DFA.OUT |
|---|---|---|
| Contains Exception 1 Report | Contains binary data for the first exception report | Contains DFA information for the first exception report |
| Exception 2 Report | | |
| Exception 3 Report | | |

## Using DFA with WINSPCTR.LOG

When DFA is used with the WINSPCTR.LOG file alone, it gives minimal stack trace information, such as addresses. If Turbo Debugger information (contained in a .EXE, .DLL, or .TDS file) is present either in the executable file or in a separate file, source file names and line numbers are added to the report.

## Using DFA with WINSPCTR.BIN

When used with the WINSPCTR.BIN file, DFA

- Adds Stack-based variables to the log, including local variables, parameters passed to the function, structures and arrays.
- Lists variable types, values, and addresses by function.

If Turbo Debugger information is present, for each stack frame, DFA reports

- In section one, the
  - Source file
  - Line number
  - Local variables
  - Parameters
- In section two, the
  - Module name for the task with the fault
  - File names
  - Logical segments
  - The segments' selectors
  - Whether the segments are data or code segments
- In section three, the
  - Global variables
  - Static variables
  - The variables' values at the time of the exception

### Syntax

The syntax for DFA is

```
DFA [option] WINSPCTR.LOG [WINSPCTR.BIN]
```

When WINSPCTR.LOG (required) is present, you get source file and line numbers. When WINSPCTR.BIN (optional) is present, you get additional variable information.

Table 4.4 DFA options

| Option | What it does |
|---|---|
| **/O**[outputfile] | Renames the output file from the DFA.OUT default |
| **/D** | Forces DFA to write a hex dump of the saved data segments |

# Other WinSpector tools

EXEMAP, TMAPSYM, and BUILDSYM are three utilities you can use to enhance the information WinSpector provides about an exception.

## Creating a .MAP file from an executable

EXEMAP creates .MAP files for Windows executables. A .MAP file can be used to create a .SYM file, which can then be used by WinSpector to expand the error reporting. If you are using .DLLs or other programs for which you don't have the source code, this information can be especially useful.

Although the resulting .MAP file isn't as complete as one generated by the link phase of the compile process, it does include addresses for exported public functions.

### Syntax

The syntax for using EXEMAP is

```
EXEMAP exefilename [output mapfile]
```

The [output mapfile] is optional. If an [output mapfile] is not specified, EXEMAP uses exefilename.MAP.

## Creating a .SYM file from a .MAP file

TMAPSYM creates .SYM files from existing .MAP files (created either by the compiler or by the EXEMAP utility). The resulting .SYM files make public functions, variable names, and functions in the entry table of the executable available to WinSpector. Constants and line number information, however, are not included in a TMAPSYM-generated .SYM file.

### Syntax

The syntax for using TMAPSYM is

```
TMAPSYM filename[.MAP]
```

The .MAP extension is optional.

Because BUILDSYM overwrites any existing .SYM files, the persistent browsing information stored in the BP.SYM and TPX.SYM files could be inadvertently overwritten when BUILDSYM generates a .SYM file. To be safe, copy existing .SYM files before using BUILDSYM or TMAPSYM.

## Creating .SYM files from executables

BUILDSYM offers a convenient way to create .SYM files for one or more executable programs in a directory. If BUILDSYM is not available, you need to create .SYM files by following this two-step process:

1. Use the EXEMAP utility on the program to make a .MAP file.
2. Then, use the TMAPSYM utility on the .MAP file to make a .SYM file.

If you use BUILDSYM, you only need to enter one command to create .SYM files. When BUILDSYM executes, it automatically runs EXEMAP and TMAPSYM. It also erases .MAP files from your directory after .SYM files are created. Because BUILDSYM supports the use of wildcards, you can create .SYM files for part or all of a directory by entering a single command.

To run BUILDSYM, both EXEMAP and TMAPSYM must be in your search path. BUILDSYM then places any resulting .SYM files in the current directory. For WinSpector to find a .SYM file, the file must be in the same directory as the executable that caused the exception.

BUILDSYM performs the following tasks:

- Verifies that the files are really Windows files, and if they're not, leaves them alone.
- Calls EXEMAP to create .MAP files.
- Verifies that .MAP files were created.
- Calls TMAPSYM, passing the names of the new .MAP files, to create .SYM files.
- Deletes the .MAP files since they are no longer needed.

### Syntax

Use BUILDSYM by typing

```
BUILDSYM filename
```

Conveniently, you can use DOS wildcards in the *filename* portion of the syntax.

Because BUILDSYM overwrites any existing .SYM files, the persistent browsing information stored in the BP.SYM and TPX.SYM files could be inadvertently overwritten when

BUILDSYM generates a .SYM file. To be safe, copy existing .SYM files before using BUILDSYM or TMAPSYM.

## Diagnostic messages

Diagnostic messages you might receive from WinSpector can be easily understood and the causes eliminated. If you attempt to run more than one copy of WinSpector at a time, it will give you the following message,

```
Error: WinSpector is already running
```

Just click OK and continue using the currently running copy.

If WinSpector cannot find the editor you have indicated in the View log option in the Preferences dialog box, it will display the message

```
Error: Unable to execute: [option]
```

where option refers to the name of the file you typed. Check to make sure the editor you indicated exists in the specified directory.

C H A P T E R

# 5

# Resource compiler

Most Windows programs are easy to use because they provide a standard user interface. For example, most Windows programs use menus to let you implement program commands, and they change cursors to let the mouse pointer represent a wide variety of tools, such as arrows or paint brushes.

Menus and cursors are two examples of a Windows program's resources. Resources are data stored in a program's executable (.EXE) file separate from the program's usual data. Resources are designed and specified outside the program code, then added to the program's compiled code to create the executable file.

These are the resources you will create and use most often:

- Menus
- Dialog boxes
- Icons
- Cursors
- Keyboard accelerators
- Bitmaps
- Character strings

## Creating resources

You can create resources using a resource editor or the Resource Compiler. In most cases, it's easier to use a resource editor and visually create your resources. However, it is sometimes

convenient to use the Resource Compiler to compile resource script files that appear in books or magazines.

Regardless of which approach you take, you normally create a resource file (.RES) for each application. This resource file contains binary information for all the menus, dialogs, bitmaps, and other resources used by your application.

The binary resource file (.RES) is added to your executable file (.EXE) using the Resource Compiler as described later in this chapter. You must also write code that loads the resources into memory. Each resource must be loaded into memory separately. This gives you flexibility, since your program will use memory only for the resources that are currently required.

## Adding resources to an executable

Once the resources are stored in binary format in a .RES file, they must be added to the program's executable (.EXE) file. The result is a file that contains the application's compiled code as well as its resources.

You can add resources to an executable file in two ways:

- Use a resource editor (such as Resource Workshop) to copy resources from a .RES file into the program's already-compiled .EXE file.
- Use the Resource Compiler (RC) to copy resources from a .RES file into the program's already-compiled .EXE file.

### Resource compiling from the command line

From the command line, you can compile the resource files you want to use in your Windows application with the Resource Compiler. When you're ready to build the application, use the Resource Compiler to bind the .RES file to the .EXE or .DLL.

### Resource compiling from a makefile

In a makefile, add the .RES file to the list of files in the explicit rule that governs the build of the final .EXE. In that rule, also add the command to invoke the Resource Compiler with the correct .RES file. You can also add a rule to invoke the Resource Compiler on an out-of-date .RES file.

## Including .RES files in the IDE

Although you can't compile a resource file from the Borland Pascal IDE, once you have a .RES file, you can include it in your application with the **$R** compiler directive. When your application is compiled and linked, the specified resource files are processed and each resource in each resource file is copied to the .EXE or .DLL being produced. Borland Pascal's linker searches for .RES files in the current directory and in the Resource directory specified in the IDE.

# Resource Compiler syntax

*See Table 5.1 for a description of the Resource Compiler options.*

This is how you invoke the Resource Compiler from the command line:

RC [*options*] *ResourceFile* [*ModuleFile*]

For example, to compile WHELLO.RC file and add it to WHELLO.EXE, you would type this command:

```
rc whello
```

This simplest form works only if the resource file and the executable file share the same name. If WHELLO.RC was instead named WHELLORS.RC, you would type

```
rc whellors whello
```

To compile only the WHELLO.RC resource file (and not add the resulting WHELLO.RES to WHELLO.EXE), use the **–R** option, like this:

```
rc -r whello
```

You would then have a WHELLO.RES file. To add WHELLO.RES to WHELLO.EXE, type

```
rc whello.res
```

To mark a module as Windows-compatible, but not add any resources to it, simply invoke the Resource Compiler with the module name (note that the file name must have one of these extensions: .EXE, .DLL, or .DRV). For example,

```
rc whello.exe
```

The following table describes the Resource Compiler options. Note that Resource Compiler options are not case sensitive (**-e** is the same as **–E**). Also, options that take no arguments can be combined (for instance, **–kpr** is legal).

Table 5.1: Resource Compiler options

| Option | What it does |
|---|---|
| **–30** | Marks the executable file so it will run with Windows version 3.0 or version 3.1. By default, the Resource Compiler marks the executable file to run only with Windows 3.1. |
| **–31** | Marks the executable file so it will run only with Windows 3.1. This is the default condition. |
| **–?** | Lists help on Resource Compiler options (also **–H**). |
| **–d** *Symbol* | Defines *Symbol* for the preprocessor. |
| **–e** | Changes location of global memory for a DLL to above the EMS bank line. This option has no effect with Windows 3.1. |
| **–fe** *FileName* | Renames the .EXE file to *FileName*. |
| **–fo** *FileName* | Renames the .RES file to *FileName*. |
| **–h** | Lists help on Resource Compiler options (also **–?**). |
| **–i** *Path* | After searching the current directory for include files and resource files, RC searches the directory named in *Path*. The **–i** option can be repeated if you want to specify more than one search path. Also see the description for the **–x** option. |
| **–k** | Turns off load optimization for segments and resources. (Normally, the Resource Compiler preloads all data segments, nondiscardable code segments, and the entry-point code segment, even if the segments were not marked as PRELOAD in the module definition file. In addition, the Resource Compiler normally places all preloaded segments in a contiguous area in the executable file.) |
| **–lim32** | Informs Windows that the application will be using expanded memory, according to the LIM 3.2 specification. This option has no effect with Windows 3.1. |
| **–multinst** | Assigns each instance of a task to a different EMS bank, if the expanded memory under Windows is configured under EMS 4.0. This option has no effect with Windows 3.1. |
| **–p** | Makes a DLL private to one or more instances of a single application, which might result in performance gains. This option has no effect with Windows 3.1. |
| **–r** | Compiles the .RC file into a .RES file, but does not add it to an .EXE. |
| **–t** | Creates application to be run only in protected (standard or 386 enhanced) mode. If the user tries to run in real mode, a message is displayed. This option has no effect with Windows 3.1. |
| **–v** | Displays all compiler progress messages (compile verbose). |
| **–x** | Excludes searching in the directories named in the INCLUDE environment variable. Also see the description for the **–i** option. |
| **–z** | Prevents the compiler from checking for RCINCLUDE statements. When you have not used RCINCLUDE statements, using this option can improve the speed of the Resource Compiler. |

C H A P T E R

6

# Windows Help compilers

A Help system provides users with online information about an application. Creating the system requires the efforts of both a Help writer and a Help programmer. The Help writer plans, writes, codes, builds, and keeps track of Help topic files, which are text files that describe various aspects of the application. The Help programmer ensures that the Help system works properly with the application.

This chapter describes the following topics:

- Providing and creating the Help system
- Planning the Help system
- Creating Help topic files
- Building the Help file
- Help examples and compiler error messages

This section and those that follow assume you are familiar with Windows Help. The sections use examples from sample applications provided on your disks. If you are unfamiliar with Windows Help, take a moment to run the sample application.

## Creating a Help system: the development cycle

The creation of a Help system for a Windows application comprises the following major tasks:

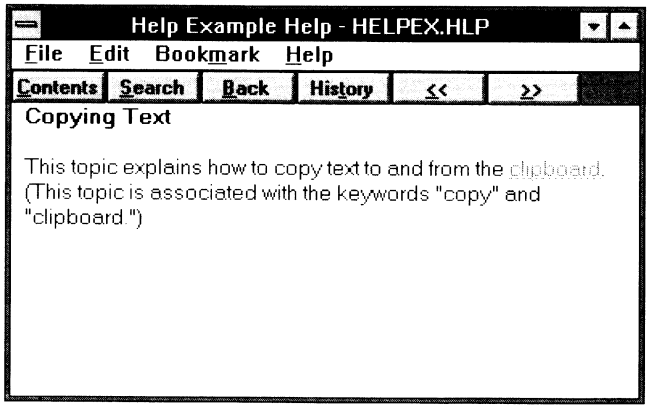
1. Gathering information for the Help topics.
2. Planning the Help system. The section, "Planning the Help system," describes considerations you should keep in mind when planning your Help system.
3. Writing the text for the Help topics.
4. Entering all required control codes into the text files. Control codes determine how the user can move around the Help system. The section titled, "How Help Appears to the Writer," includes an example of several control codes. A later section, "Coding Help topic files," describes the codes in detail.
5. Creating a project file for the build. The Help project file provides information that the Help Compiler needs to build a Help resource file. A later section, "Building the Help project file," describes the Help project file.
6. Building the Help resource file. The Help resource file is a compiled version of the topic files the writer creates. Later in this chapter, the section "Compiling Help files," describes how to compile a Help resource file.
7. Testing and debugging the Help system.
8. Programming the application so that it can access Windows Help.

## How Help appears to the user

To the user, the Help system appears to be part of the application, and is made up of text and graphics displayed in the Help window in front of the application. Figure 6.1 illustrates the Help window that appears when the user asks for help with copying text in Helpex.

The Help window displays one sample Help topic, a partial description of how to perform one task. In Figure 6.1, the first sentence includes a definition of the word "clipboard." By clicking the mouse button when the cursor is on the word (denoted by dotted underlined text), the user can read the definition, which appears in an overlapping box as long as the mouse button is held down.

Figure 6.1
Helpex help window

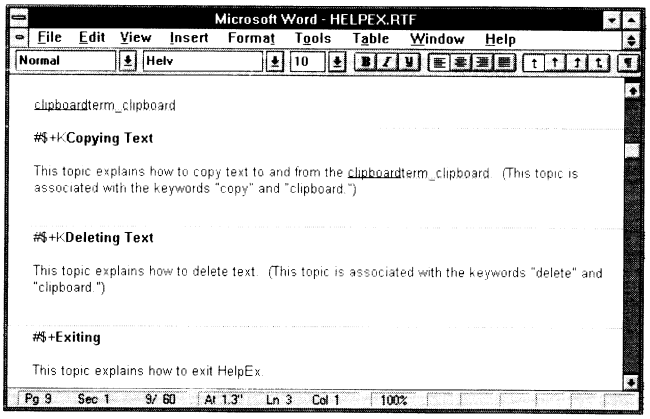



Cross-references to related topics are called jumps. By clicking a jump term for a related topic (denoted by underlined text), the user changes the content of the Help window to a description of the new topic or command. Figure 6.1 includes a look-up to the definition of "clipboard."

## How Help appears to the help writer

To the writer, the Help system is a group of topic files, which are text files that include special codes. Figure 6.2 illustrates the source text that corresponds to the topic shown in Figure 6.1.

Figure 6.2
Topic file




To create this topic, the Help writer describes the task, formats the text, and inserts codes using strikethrough text, underlined text, and footnotes. In place of strikethrough, the writer can use double underlining if the word processor does not support strikethrough

formatting. Footnotes in the text contain linking information required by the Help Compiler. The section "Planning the Help system" discusses formatting considerations. Another section,"Creating the Help topic files," describes how to create topics and enter the special codes that the Help system uses.

### How Help appears to the help programmer

To the programmer, Windows Help is a standalone Windows application the user can run like any other application. Your application can call the *WinHelp* function to ask Windows to run the Help application and specify which topic to display in the Help window.

## Planning the Help system

The initial task for the Help writer is to develop a plan for creating the system. This section discusses planning the Help system for a particular application; it covers these topics:

- Developing a plan
- Determining the topic file structure
- Designing the visual appearance of Help topics

### Developing a plan

Before you begin writing Help topics using the information you have gathered, you and the other members of the Help team should develop a plan that addresses the following issues:

- The audience for your application
- The content of the Help topics
- The structure of topics
- The use of context-sensitive topics

You might want to present your plan in a design document that includes an outline of Help information, a diagram of the structure of topics, and samples of the various kinds of topics your system will include. Keep in mind that context-sensitive Help requires increased development time, especially for the application programmer.

## Defining the audience

The audience you address determines what kind of information you make available in your Help system and how you present the information.

Users of Help systems might be classified as follows:

Table 6.1
Your application audience

| User | Background |
|---|---|
| Computer novice | Completely new to computing. |
| Application novice | Some knowledge of computing, but new to your kind of application. For example, if you are providing Help for a spreadsheet program, the application novice might be familiar only with word-processing packages. |
| Application intermediate | Knowledgeable about your kind of application. |
| Application expert | Experienced extensively with your type of application. |

Keep in mind that one user may have various levels of knowledge. For example, the expert in word processors may have no experience using spreadsheets.

## Planning the contents

You should create topics that are numerous enough and specific enough to provide your users with the help they need.

Novice users need help learning tasks and more definitions of terms. More sophisticated users occasionally seek help with a procedure or term, but most often refresh their memory of commands and functions. The expert user tends to seek help only with command or function syntax, keyboard equivalents, and shortcut keys.

There are no rules for determining the overall content of your Help system. If you are providing Help for all types of users, you will want to document commands, procedures, definitions, features, functions, and other relevant aspects of your application. If you are providing help for expert users only, you might want to omit topics that describe procedures. Let your audience definition guide you when deciding what topics to include.

Keep in mind that the decision to implement context-sensitive Help is an important one. Context-sensitive Help demands a close

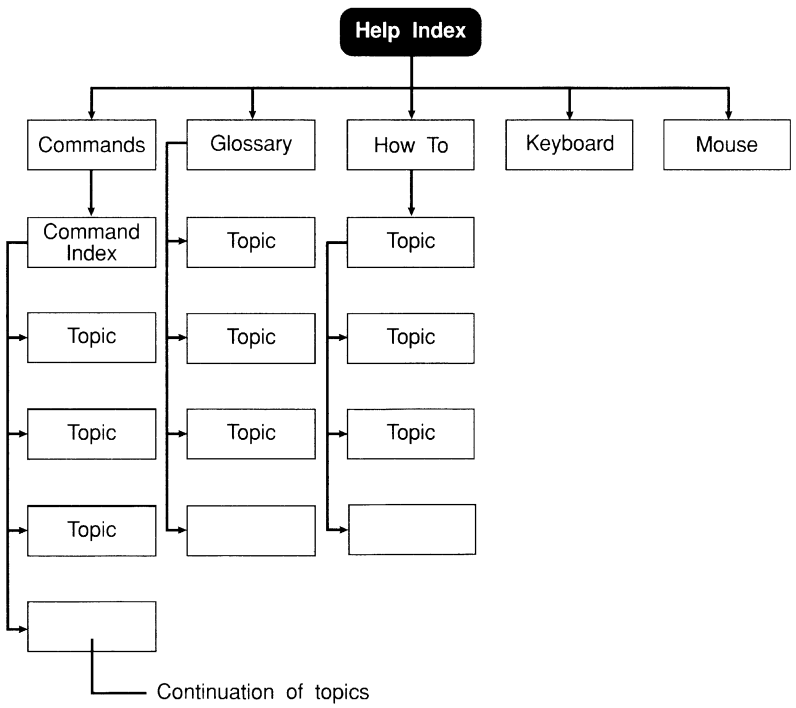
working relationship between the Help writer and the application programmer, and will therefore increase the development time necessary to create a successful Help system.

## Planning the structure

Many Help systems structure topics hierarchically. At the top of the hierarchy is an index or a table of contents, or both. The index and table of contents list individual topics or categories of topics available to the user.

Topics themselves can be related hierarchically. Each successive step takes the user one level down in the hierarchy of the Help system until the user reaches topic information. The hierarchical relationship of Help topics determines in part how the user navigates through the Help system. Figure 6.3 illustrates a possible hierarchy:

Figure 6.3
Example of a help hierarchy




Helpex contains an index that lists several categories of topics. Each category includes a secondary index, which lists topics in the category, and the topics themselves.

Moving from the index to a topic, the user goes from the general to the specific.

The hierarchical structure provides the user a point of reference within Help. Users are not constrained to navigate up and down the hierarchy; they can jump from one topic to another, moving across categories of topics. The effect of jumps is to obscure hierarchical relationships. For example, the Windows Help application contains a search feature that lets the user enter a keyword into a dialog box and search for topics associated with that keyword. The Help application then displays a list of titles to choose from in order to access information that relates to the keyword.

*For more about the search feature, see page 87.*

Because users often know which feature they want help with, they can usually find what they want faster using the search feature than they can by moving through the hierarchical structure.

*For more about browse sequences, see page 81.*

In addition to ordering topics hierarchically, you can order them in a logical sequence that suits your audience. The logical sequence, or "browse sequence," lets the user choose the Browse button to move from topic to topic. Browse sequences are especially important for users who like to read several related topics at once, such as the topics covering the commands in the File menu.

Whichever structure you decide to use, try to minimize the number of lists that users must traverse in order to obtain information. Also, avoid making users move through many levels to reach a topic. Most Help systems function quite well with only two or three levels.

## Displaying context-sensitive Help topics

Windows Help supports context-sensitive Help. When written in conjunction with programming of the application, context-sensitive Help lets the user press *F1* in an open menu to get help with the selected menu item. Alternatively, the user can press *Shift+F1* and then click on a screen region or command to get help on that item.

Developing context-sensitive Help requires coordination between the Help writer and the application programmer so that Help and the application pass the correct information back and forth.

*For information on creating a Help project file, see page 98.*

To plan for context-sensitive Help, the Help writer and the application programmer should agree on a list of context

numbers. Context numbers are arbitrary numbers that correspond to each menu command or screen region in the application, and are used to create the links to the corresponding Help topics. You can then enter these numbers, along with their corresponding context-string identifiers, in the Help project file, which the Help Compiler uses to build a Help resource file.

*For more on assigning context numbers, see page 118.*

The context numbers assigned in the Help project file must correspond to the context numbers that the application sends at run time to invoke a particular topic.

*See page 114 for more on context-sensitive Help.*

If you don't explicitly assign context numbers to topics, the Help Compiler generates default values by converting topic context strings into context numbers.

*Page 96 provides you with more information about using a tracker.*

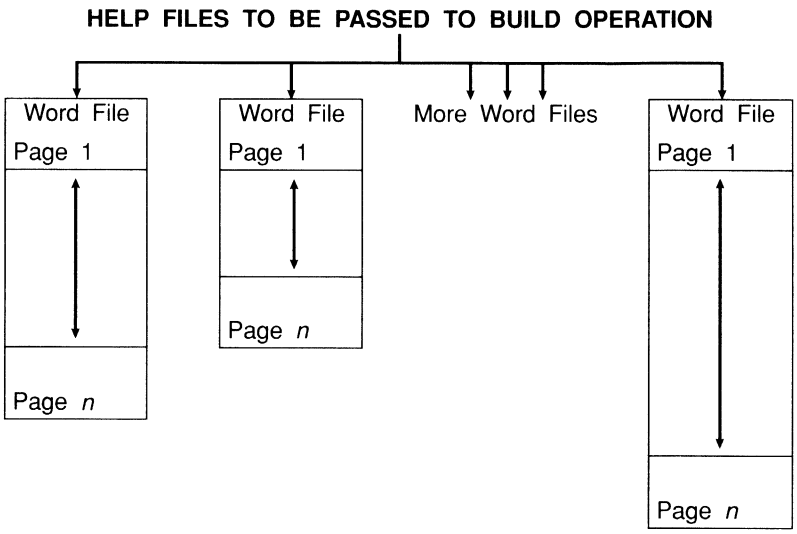
To manage context numbers and file information, you might want to create a Help tracker to list the context numbers for your context-sensitive topics.

## Determining the topic file structure

The Help file structure remains essentially the same for all applications even though the context and number of topic files differ. Topic files are segmented into the different topics by means of page breaks. When you build the Help system, the compiler uses these topic files to create the information displayed for the user in the application's Help window.

Figure 6.4 shows this basic file structure.

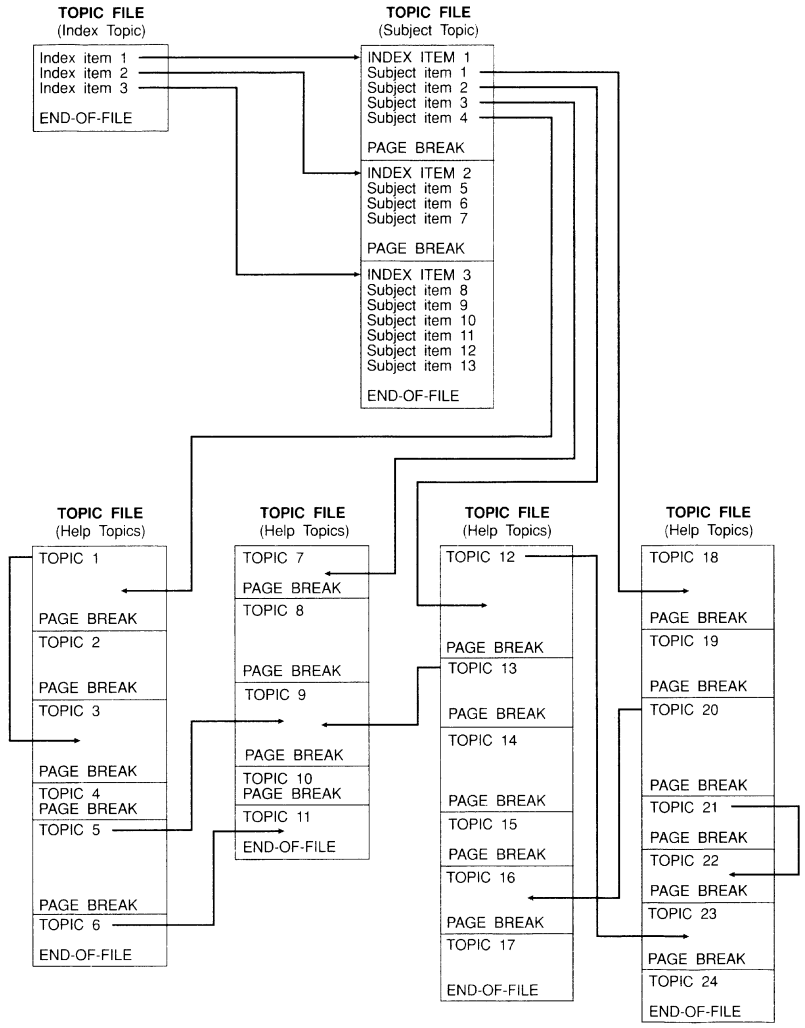
Figure 6.4
Basic help file structure




## Choosing a file structure for your application

When choosing a file structure for your Help system, consider the scope and content of the Help system you are planning. For example, you could place all Help topics in a single large topic file. Or, you could place each Help topic in a separate file. Neither of these file structures is generally acceptable. An enormous single file or too many individual files can present difficulties during the creation of the Help resource file.

Figure 6.5
Help file structure showing hypertext jumps




The number of topics relates to the number of features covered by the Help system. Consequently, you can't make extensive changes to one without making changes to the other. For instance, if a number of additional product features are added to Help, then additional topics must be created to accommodate the new information.

Figure 6.5 illustrates the file structure of a possible Help system. The number of topics and topic files is limited to simplify the diagram and more clearly show the concept of linking the topics together through jumps, shown in the figure as arrows. The figure

is not intended to show the number of files that can be included in the Help file system. Moreover, the figure does not show how topic files are ordered using the browse feature.

## Designing Help topics

How the information in the Help window appears to the user is primarily a function of the layout of the Help topic. The Windows Help application supports a number of text attributes and graphic images you can use to design your Help window.

This section provides general guidelines for designing a window and describes fonts and graphic images that Windows Help supports.

### Layout of the Help text

Help text files are not limited to plain, unformatted text. You can use different fonts and point sizes, include color and graphics to emphasize points, indent paragraphs to present complex information, and use a variety of other visual devices to present your information.

Research on screen format and Help systems has produced general guidelines for presenting information to users. Table 6.2 summarizes the findings of these studies.

Table 6.2: Help design issues

| Design Issue | Guideline |
|---|---|
| Language | *Use language appropriate for the audience you have defined.* Language that is too sophisticated for your audience can frustrate users by requiring them to learn the definition of unfamiliar terms and concepts. |
| Amount of text | *Use a minimum of text.* Studies indicate that reading speed decreases by 30 percent when users read online text rather than printed text. Minimal, concise text helps users compensate for the decreased reading speed. |
| Paragraph length | *Use short paragraphs.* Online users become overloaded with text more easily than do readers of printed material. Breaking your text into short paragraphs helps avoid this problem. |
| Whitespace | *Use it to help group information visually.* Whitespace is important to making online text more readable. Use it liberally, while also considering the overall size that a topic will occupy on the screen. Users tend to think there is more information on a screen than exists. For example, if the ratio of whitespace to text is 50:50, users perceive the ratio to be 40:60. |
| Highlighting | *Use highlighting techniques judiciously.* Windows Help provides a variety of highlighting devices, such as font sizes, font types, and color. Using a few devices can help users find information quickly. Using many devices will decrease the |

Table 6.2: Help design issues (continued)

| | |
|---|---|
| | effectiveness of your visual presentation and frustrate users. As with print-based documentation, use only one or two fonts at a time. |
| Graphics and icons | *Use graphics to support the explanation of visual events.* Windows Help supports the use of bitmapped graphic images. Use appropriate images to help explain the function of icons and screen elements in your application. Remember that graphics will draw the user's eye before the accompanying text. Be sure to crop your images to remove distracting information. Use color images only if you are certain that all your users' systems have color capability. As with context-sensitive Help, consider the additional time necessary to create accurate and meaningful graphic images. |
| Design consistency | *Be rigorously consistent in your design.* Users expect the appearance of Help topics to be the same, regardless of the information presented. Consistent titling, highlighting, fonts, and positioning of text in the window is essential to an effective Help system. |

### Type fonts and sizes

The Windows Help application can display text in any font and size available to the system. When the topic files are passed to the build process, the Help Compiler attempts to use the fonts and sizes found in the topic files. If a font or point size can't be matched exactly when the Help file is displayed by Windows Help, the closest available font and size on the user's system will be used.

Windows ships with only certain fonts in specific font sizes. If you write Help files using these fonts and sizes, the displayed Help file will closely match the printed word-processed file. Because fonts other than those shipped with Windows may not be available on users' machines, you might want to limit your font selection to the shipped Windows fonts.

Since Windows Help supports any Windows font, special symbols such as arrows can be included in your topics by using the Symbol font.

## Graphic images

The Windows Help application allows you to embed graphics in the Help file. Graphics can be placed and displayed anywhere on the page. Text can appear next to the graphic.

Color graphic images can be included, provided you use only the available Windows system colors. If you use graphics tools that support an enhanced color palette to create or capture images, these images may not always display with the intended colors. And since you can't control the color capabilities on a user's

machine, you might want to limit your graphic images to black and white bitmaps.

*For more information on placing graphics into your Help files, see page 93.*

Keep in mind that graphics are most effective when they contribute to the learning process. Graphics not tied to the information are usually distracting rather than helpful and should be avoided.

For additional information about screen design, refer to the following books and journals:


- Bradford, Annette Norris. "Conceptual Differences Between the Display Screen and the Printed Page." *Technical Communication* (Third Quarter 1984): 13–16.
- Galitz, Wilbert O. *Handbook of Screen Format Design.* 3d ed. Wellesley, MA: QED Information Sciences, Inc., 1989.
- Houghton, Raymond C., Jr. "Online Help Systems: A Conspectus." *Communications of the ACM 27*(February 1984): 126–133.
- Queipo, Larry. "User Expectations of Online Information." *IEEE Transactions on Professional Communications* PC 29 (December 1986): 11–15.


# Creating the Help topic files

Probably the most time-consuming task in developing a Help system for your application is creating the Help topic files from which you compile the Help system. Help topic files are text files that define what the user sees when using the Help system. The topic files can define various kinds of information, such as an index to information on the system, a list of commands, or a description of how to perform a task.

Creating topic files entails writing the text that the user sees when using Help, and entering control codes that determine how the user can move from one topic to another. This section describes the following topics:

- Choosing an authoring tool
- Structuring Help topic files
- Coding Help topic files
- Managing Help topic files

## Choosing an authoring tool

To write your text files, you will need a Rich Text Format (RTF) editor, which lets you create the footnotes, underlined text, and strikethrough or double-underlined text that indicate the control codes. These codes are described in the section titled "Coding Help Topic Files" on page 82. RTF capability allows you to insert the coded text required to define Help terms, such as jumps, keywords, and definitions.

## Structuring Help topic files

A Help topic file typically contains multiple Help topics. To identify each topic within a file:

- Topics are separated by hard page breaks.
- Each topic accessible via a hypertext link must have a unique identifier, or context string.
- Each topic can have a title.
- Each topic can have a list of keywords associated with it.
- Each topic can have a build-tag indicator.
- Any topic can have an assigned browse sequence.

For information about inserting page breaks between topics, see the documentation for the editor you are using. For information about assigning context strings and titles to topics, see the following sections.

## Coding Help topic files

The Help system uses control codes for particular purposes:

Table 6.3
Help control codes

| Control Code | Purpose |
|---|---|
| Asterisk (*) footnote | Build tag defines a tag that specifies topics the compiler conditionally builds into the system. Build tags are optional, but they must appear first in a topic when they are used. |
| Pound sign (#) | Context string defines a context string that uniquely identifies a topic. Because hypertext relies on links provided by context strings, topics without context strings can |

Table 6.3: Help control codes (continued)

| | |
|---|---|
| | only be accessed using keywords or browse sequences. |
| Dollar sign ($) footnote | Title defines the title of a topic. Titles are optional. |
| Letter "K" footnote | Keyword defines a keyword the user uses to search for a topic. Keywords are optional. |
| Plus sign (+) footnote | Browse sequence number defines a sequence that determines the order in which the user can browse through topics. Browse sequences are optional. However, if you omit browse sequences, the Help window will still include the Browse buttons, but they will be grayed. |
| Strikethrough or double-underlined text | Cross-reference indicates the text the user can choose to jump to another topic. |
| Underlined text | Definition specifies that a temporary or "look-up" box be displayed when the user clicks the mouse button or presses the *Enter* key. The box can include such information as the definition of a word or phrase, or a hint about a procedure. |
| Hidden text | Cross-reference context string specifies the context string for the topic that will be displayed when the user chooses the text that immediately precedes it. |
| Exclamation mark | Macro specifies a macro to be executed in text or footnotes. This option is new for Windows 3.1. |

If you are using build tags, footnote them at the very beginning of the topic. Place other footnotes in any order you want. For information about assigning specific control codes, see the following sections.

### Assigning build tags

Build tags are strings that you assign to a topic in order to conditionally include or exclude that topic from a build. Each topic can have one or more build tags. Build tags are not a necessary component of your Help system. However, they do provide a means of supporting different versions of a Help system without having to create different source files for each version. Topics without build tags are always included in a build.

*For information about the **BUILD** option, the (BuildTags) section and the Help project file, see "Building the Help file."*

You insert build tags as footnotes using the asterisk (*). When you assign a build tag footnote to a topic, the compiler includes or excludes the topic according to build information specified in the **BUILD** option and [BuildTags] section of the Help project file.

To assign a build tag to a topic:

1. Place the cursor at the beginning of the topic heading line so that it appears before all other footnotes for that topic.
2. Insert the asterisk (*) as a footnote reference mark.

   Note that a superscript asterisk ( * ) appears next to the heading.
3. Type the build tag name as the footnote.

   Be sure to allow only a single space between the asterisk (*) and the build tag.

Build tags can be made up of any alphanumeric characters. The build tag is not case-sensitive. The tag may not contain spaces. You can specify multiple build tags by separating them with a semicolon, as in the following example:

```
* AppVersion1; AppVersion2; Test_Build
```

Including a build tag footnote with a topic is equivalent to setting the tag to true when compared to the value set in the project file. The compiler assumes all other build tags to be false for that topic. After setting the truth value of the build tag footnotes, the compiler evaluates the build expression in the Options section of the Help project file. Note that all build tags must be declared in the project file, regardless of whether a given conditional compilation declares the tags. If the evaluation results in a true state, the compiler includes the topic in the build. Otherwise, the compiler omits the topic.

The compiler includes in all builds topics that don't have a build tag footnote regardless of the build tag expressions defined in the Help project file. For this reason, you may want to use build tags primarily to exclude specific topics from certain builds. If the compiler finds any build tags not declared in the Help project file, it displays an error message.

By allowing conditional inclusion and exclusion of specific topics, you can create multiple builds using the same topic files. This saves time and effort for the Help development team. It also

means that you can develop Help topics that will help you maintain a higher level of consistency across your product lines.

### Assigning context strings

Context strings identify each topic in the Help system. Each context string must be unique. A given context string may be assigned to only one topic within the Help project; it can't be used for any other topic.

*For information about assigning jumps, see page 91; for assigning browse sequences, see page 88; for assigning keywords, see page 87.*

The context string provides the means for creating jumps between topics or for displaying look-up boxes, such as word and phrase definitions. Though not required, most topics in the Help system will have context-string identifiers. Topics without context strings may not be accessed through hypertext jumps. However, topics without context-string identifiers can be accessed through browse sequences or keyword searches, if desired. It is up to the Help writer to justify the authoring of topics that can be accessed only in these manners.

To assign a context string to a Help topic:

1. Place the cursor to the left of the topic heading.
2. Insert the pound sign (#) as the footnote reference mark.

   Note that a superscript pound sign ( # ) appears next to the heading.
3. Type the context string as the footnote.

   Be sure to allow only a single space between the pound sign (#) and the string.

   Context strings are not case-sensitive.

Valid context strings may contain the alphabetic characters A–Z, the numeric characters 0–9, and the period (.) and underscore (_) characters. The following example shows the context string footnote that identifies a topic called "Opening an Existing Text File":

```
#OpeningExistingTextFile
```

Although a context string has a practical limitation of about 255 characters, there is no good reason for approaching this value. Keep the strings sensible and short so that they're easier to enter into the text files.

### Assigning titles

Title footnotes perform the following functions within the Help system:

- They appear on the Bookmark menu.
- They appear in the "Topics found " list that results from a keyword search and in the "History" list. (Topics that don't have titles but are accessible via keywords are listed as `>>>>Untitled Topic<<<<` in the Topics found list.)

*Definition items such as glossary entries or popups are not listed in the "Topics found" and don't appear in the "History" list.*

Although not required, most topics have a title. You might not assign a title to topics containing low-level information that Help's search feature, look-up boxes, and system messages don't access.

To assign a title to a topic,

1. Place the cursor to the left of the topic heading.
2. Insert a footnote with a dollar sign ($) as the footnote reference mark.

   Note that a superscript dollar sign ( $^{\$}$ ) appears next to the heading.
3. Type the title as the footnote.

   Be sure to allow only a single space between the dollar sign ($) and the title.

The following is an example of a footnote that defines the title for a topic:

```
$ Help Keys
```

When adding titles, keep in mind the following restrictions:

Table 6.4
Restrictions of Help titles

| Item | Restrictions |
|---|---|
| Characters | Titles can be up to 128 characters in length. The Help compiler truncates title strings longer than 128 characters. The help system displays titles in a list box when the user searches for a keyword or enters a bookmark. The title is also displayed in the history list. |
| Formatting | Title footnote entries can't be formatted. |

## Assigning keywords

Help allows the user to search for topics with the use of keywords assigned to the topics. When the user searches for a topic by keyword, Help matches the user-entered word to keywords assigned to specific topics. Help then lists matching topics by their titles in the Search dialog box. Because a keyword search is often a fast way for users to access Help topics, you'll probably want to assign keywords to most topics in your Help system.

***Note*** You should specify a keyword footnote only if the topic has a title footnote, since the title of the topic will appear in the search dialog when the user searches for the keyword.

To assign a keyword to a topic,

1. Place the cursor to the left of the topic heading.
2. Insert an uppercase K as the footnote reference mark.

   Note that a superscript K ( $^{K}$ ) appears next to the heading.
3. Type the keyword, or keywords, as the footnote.

   Be sure to allow only a single space between the K and the first keyword.

   If you add more than one keyword, separate each with a semicolon.

The following is an example of keywords for a topic:

```
K open;opening;text file;ASCII;existing;text only;documents;
```

Whenever the user performs a search on any of these keywords, the corresponding titles appear in a list box. More than one topic may have the same keyword.

When adding keywords, keep in mind the following restrictions:

Table 6.5
Help keyword restrictions

| Item | Restrictions |
|---|---|
| Characters | Keywords can include any ANSI character, including accented characters. The maximum length for keywords is 255 characters. |
| | A space embedded in a key phrase is considered to be a character, permitting phrases to be keywords. |
| Phrases | Help searches for any word in the specified phrase. |
| Formatting | Keywords are unformatted. |
| Case sensitivity | Keywords are not case-sensitive. |

Table 6.5: Help keyword restrictions (continued)

| | |
|---|---|
| Punctuation | Except for semicolon delimiters, you can use punctuation. |

### Creating multiple keyword tables

Multiple keyword tables are useful to allow a program to look up topics that are defined in alternate keyword tables. You can use an additional keyword table to allow users familiar with keywords in a different application to discover the matching keywords in your application.

*For information on the **MULTIKEY** option, see page 110.*

Authoring additional keyword tables is a two-part process. First, the **MULTIKEY** option must be placed in the [Options] section of the project file.

Second, the topics to be associated with the additional keyword table must be written and labeled. Footnotes are assigned in the same manner as the normal keyword footnotes, except that the letter specified with the **MULTIKEY** option is used. With this version of the Help Compiler, the keyword footnote used is case-sensitive. Therefore, care should be taken to use the same case, usually uppercase, for your keyword footnote. Be sure to associate only one topic with a keyword. Help does not display the normal search dialog box for a multiple keyword search. Instead it displays the first topic found with the specified keyword. If you want the topics in your additional keyword table to appear in the normal Help keyword table, you must also specify a "K" footnote and the given keyword.

The application you are developing Help for can then display the Help topic associated with a given string in a specified keyword table. Keywords are sorted without regard to case for the keyword table. For information on the parameters passed by the application to call a topic found in alternate keyword table, see page 125.

### Assigning browse sequence numbers

The `Browse >>>>` and `Browse <<<<` buttons on the icon bar in the Help window let users move back and forth between related topics. The order of topics that users follow when moving from topic to topic is called a "browse sequence." A browse sequence is determined by sequence numbers, established by the Help writer.

To build browse sequences into the Help topics, the writer must

1. Decide which topics should be grouped together and what order they should follow when viewed as a group.

   Help supports multiple, discontiguous sequence lists.
2. Code topics to implement the sequence.

In this version of Help, topics defined in browse sequences are accessed using the Browse buttons at the top of the Help window. Future versions of Help won't normally display browse buttons for the user. However, if your Help resource file includes browse sequences written in the format described here, these future versions will support the feature by automatically displaying browse buttons for the user.

### Organizing browse sequences

When organizing browse sequences, the writer must arrange the topics in an order that will make sense to the user. Topics can be arranged in alphabetical order within a subject, in order of difficulty, or in a sensible order that seems natural to the application. The following example illustrates browse sequences for the menu commands used in a given application. The Help writer has subjectively defined the order that makes the most sense from a procedural point of view. You may, of course, choose a different order.

```
SampleApp Commands
     File Menu - commands:005
          New Command - file_menu:005
          Open Command - file_menu:010
          Save Command - file_menu:015
          Save As Command - file_menu:020
          Print Command - file_menu:025
          Printer Setup Command - file_menu:030
          Exit Command - file_menu:035
     Edit Menu - commands:010
          Undo Command - edit_menu:025
          Cut Command - edit_menu:015
          Copy Command - edit_menu:010
          Paste Command - edit_menu:020
          Clear Command - edit_menu:005
          Select All Command - edit_menu:030
          Word Wrap Command - edit_menu:035
          Type Face Command - edit_menu:040
          Point Size Command - edit_menu:045
```

```
Search Menu - commands:015
     Find Command - search_menu:005
     Find Next Command - search_menu:010
     Previous Command - search_menu:015
Window Menu - commands:020
     Tile Command - window_menu:005
     Cascade Command - window_menu:010
     Arrange Icons Command - window_menu:015
     Close All Command - window_menu:020
     Document Names Command - window_menu:025
```

Each line consists of a sequence list name followed by a colon and a sequence number. The sequence list name is optional. If the sequence does not have a list name, as in the following example, the compiler places the topic in a "null" list:

```
Window Menu - 120
```

Note that the numbers used in the browse sequence example begin at 005 and advance in increments of 005. Generally, it is good practice to skip one or more numbers in a sequence so you can add new topics later if necessary. Skipped numbers are of no consequence to the Help Compiler; only their order is significant.

Sequence numbers establish the order of topics within a browse sequence list. Sequence numbers can consist of any alphanumeric characters. During the compiling process, strings are sorted using the ASCII sorting technique, not a numeric sort.

Both the alphabetic and numeric portions of a sequence can be several characters long; however, their lengths should be consistent throughout all topic files. If you use only numbers in the strings make sure the strings are all the same length; otherwise a higher sequence number could appear before a lower one in certain cases. For example, the number 100 is numerically higher than 99, but 100 will appear before 99 in the sort used by Help, because Help is comparing the first two digits in the strings. In order to keep the topics in their correct numeric order, you would have to make 99 a three-digit string: 099.

### Coding browse sequences

After determining how to group and order topics, code the sequence by assigning the appropriate sequence list name and number to each topic, as follows:

1. Place the cursor to the left of the topic heading.

2. Insert the plus sign (+) as the footnote reference mark.

   Note that a superscript plus sign ( + ) appears next to the heading.
3. Type the sequence number using alphanumeric characters.

For example, the following footnote defines the browse sequence number for the Edit menu topic in the previous browse sequence example:

```
+ commands:010
```

While it may be easier to list topics within the file in the same order that they appear in a browse sequence, it is not necessary. The compiler orders the sequence for you.

### Creating cross-references between topics

Cross-references, or "jumps," are specially-coded words or phrases that are linked to other topics. Although you indicate jump terms with strikethrough or double-underlined text in the topic file, they appear underlined in the Help window. In addition, jump terms appear in color on color systems. For example, the strikethrough text (double-underlined in Word for Windows) ~~**New Command**~~ appears as **New Command** in green text to the user.

To code a word or phrase as a jump in the topic file :

1. Place the cursor at the point in the text where you want to enter the jump term.
2. Select the strikethrough (or double-underline) feature of your editor.
3. Type the jump word or words in strikethrough mode.
4. Turn off strikethrough and select the editor's hidden text feature.
5. Type the context string assigned to the topic that is the target of the jump.

When coding jumps, keep in mind that:

- No spaces can occur between the strikethrough (or double-underlined) text and the hidden text.
- Coded spaces before or after the jump term are not permitted.
- Paragraph marks must be entered as plain text.

## Defining terms

Most topic files contain words or phrases that require further definition. To get the definition of a word or phrase, the user first selects the word and then clicks the mouse button or presses the *Enter* key, causing the definition to appear in a box within the Help window. The Help writer decides which words to define, considering the audience that will be using the application and which terms might already be familiar.

The look-up feature need not be limited to definitions. With the capability of temporarily displaying information in a box, you might want to show a hint about a procedure or other suitable information for the user.

Defining a term requires that you

- Create a topic that defines the term. The definition topic must include a context string. See the section titled "Assigning Context Strings" on page 85.
- Provide a cross-reference for the definition topic whenever the term occurs.

  You don't need to define the same word multiple times in the same topic, just its first occurrence. Also, consider the amount of colored text you are creating in the Help window. See the following "Coding definitions" section.

### Creating definition topics

You can organize definition topics any way you want. For example, you can include each definition topic in the topic file that mentions the term. Or you can organize all definitions in one topic file and provide the user with direct access to it. Helpex uses the latter method, with all definitions residing in the TERMS.RTF file. Organizing definition topics into one file provides you with a glossary and lets you make changes easily.

### Coding definitions

After creating definition topics, code the terms as they occur, as follows:

1. Place the insertion point where you want to place the term that requires definition.
2. Select the underline feature of your editor.

3. Type the term.
4. Turn off the underline feature, and select the editor's hidden-text feature.
5. Type the context string assigned to the topic that contains the definition of the term.

## Inserting graphic images

Bitmapped graphic images can be placed in Help topics using either of two methods. If your word processor supports the placement of Windows graphics directly into a document, you can simply paste your bitmaps into each topic file. Alternatively, you can save each bitmap in a separate file and specify the file by name where you want it to appear in the Help topic file. The latter method of placing graphics is referred to as "bitmaps by reference." The following sections describe the process of placing bitmaps directly or by reference into your Help topics.

### Creating and capturing bitmaps

You can create your bitmaps using any graphical tools, as long as the resulting images can be displayed in the Windows environment. Each graphic image can then be copied to the Windows clipboard. Once on the clipboard, a graphic can be pasted into a graphics editor such as Paint, and modified or cleaned up as needed.

Windows Help supports color bitmaps. However, for future compatibility, you might want to limit graphics to monochrome format. If you are producing monochrome images, you might have to adjust manually the elements of your source graphic that were originally different colors to either black, white, or a pattern of black and white pixels.

When you are satisfied with the appearance of your bitmap, you can either save it as a file, to be used as a bitmap by reference, or you can copy it onto the clipboard and paste it into your word processor. If you save the graphic as a file, be sure to specify its size in your graphics editor first, so that only the area of interest is saved for display in the Help window. The tighter you crop your images, the more closely you will be able to position text next to the image. Always save (or convert and save if necessary) graphics in Windows' .BMP format.

Bitmap images should be created in the same screen mode that you intend Help to use when topics are displayed. If your Help

files will be displayed in a different mode, bitmaps might not retain the same aspect ratio or information as their source images.

### Placing bitmaps using a graphical word processor

The easiest way to place bitmaps precisely into Help topics is to use a graphical word processor. Microsoft Word for Windows supports the direct importation of bitmaps from the clipboard. Simply paste the graphic image where you want it to appear in the Help topic. You can format your text so that it is positioned below or alongside the bitmap. When you save your Help topic file in RTF, the pasted-in bitmap is converted as well and will automatically be included in the Help resource file.

### Placing bitmaps by reference

If your word processor can't import and display bitmaps directly, you can specify the location of a bitmap that you have saved as a file. To insert a bitmap reference in the Help topic file, insert one the following statements where you want the bitmap to appear in the topic:

{**bmc** *filename*.bmp}
{**bml** *filename*.bmp}
{**bmr** *filename*.bmp}

Don't specify a full path for *filename*. If you need to direct the compiler to a bitmap in a location other than the root directory for the build, specify the absolute path for the bitmap in the [Bitmaps] section of the project file.

The argument **bmc** stands for "bitmap character," indicating that the bitmap referred to will be treated the same as a character placed in the topic file at the same location on a line. Text can precede or follow the bitmap on the same line, and line spacing will be determined based upon the size of the characters (including the bitmap character) on the line. Don't specify negative line spacing for a paragraph with a bitmap image, or the image may inadvertently overwrite text above it when it's displayed in Help. When you use the argument **bmc**, there is no automatic text wrapping around the graphic image. Text will follow the bitmap, positioned at the baseline.

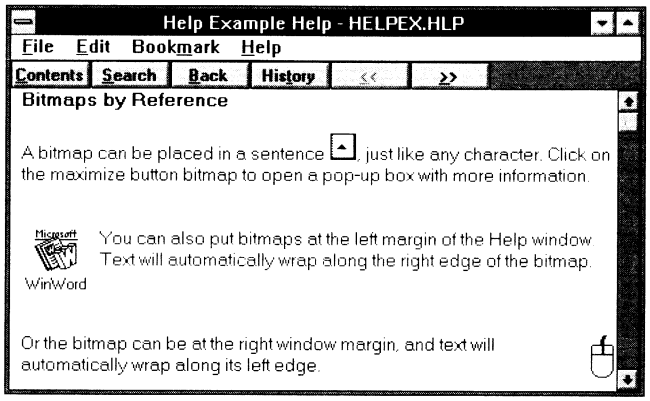
The argument **bml** specifies that the bitmap appear at the left margin, with text wrapping automatically along the right edge of the image. The argument **bmr** specifies that the bitmap appear at the right margin, with text to its left. Bitmap file names must be the same as those listed in the [Bitmaps] section of the Help

project file. The [Bitmaps] section is described in the section "Building the Help file."

Multiple references to a bitmap of the same name refer to the same bitmap when the Help file is displayed. This means that bitmap references can be repeated in your Help system without markedly increasing the size of the Help resource file.

Figure 6.6 shows the placement of three bitmaps with related text in a topic as displayed in Help.

Figure 6.6
Help topic display showing bitmaps by reference




## Managing topic files

Help topic files can be saved in the default word-processor format or in RTF. If you always save your files in RTF, and later need to make a change, the word processor may take additional time to interpret the format as it reloads the file. If you anticipate making numerous changes during Help development, you might want to minimize this delay by saving topic files in both default and RTF formats, with different file extensions to distinguish them. The compiler needs only the RTF files, and you will have faster access to the source files for changes. On a large project, this practice can save a considerable amount of development time.

### Keeping track of files and topics

It is important to keep track of all topic files for the following reasons:

- To ensure that no topics are left out of the build process
- To ensure that each topic has been assigned a unique context string

- To double-check browse sequencing within general and specific lists
- To show keyword and title matches
- To allow writers to see where the text for each of the topics is located
- To keep track of changes to files and the current status
- To track any other aspect of the Help development process that you think essential

At a minimum, writers must keep track of their own topic files, and must pass the file names to the person who is responsible for creating the Help project file.

### Creating a help tracker

While it is important that you track topic files throughout the development cycle, the tracking tool can be anything that suits your needs. You can maintain a current list of topics in an ASCII text file, in a Quattro Pro spreadsheet, or in another format.

When you or another writer creates or revises a topic, you should update the Help tracking file to reflect the change. The contents of the tracking file are not rigidly defined, but should contain entries for file name, context string, title, browse sequence, and keywords. If your application makes use of the context-sensitive feature of Help, you may want to keep track of the context-sensitive information as well. This entry is necessary only if you are assigning context numbers to topics in the Help project file. You can also include optional information, such as date created, date modified, status, and writer, if you want to keep track of all aspects of the Help development process. How you organize this information is entirely up to you.

The following sample text file and worksheet illustrate how the tracker might be organized for the Help system topics. The examples show both Help menu and context-sensitive Help entries for the topic files. Typically, the same topics that the user accesses when choosing commands from the Help menus can be accessed by the context-sensitive Help feature. The topics with entries in the context ID column are used for context-sensitive help as well as for the Help menus. Notice that some topics have more than one context-sensitive help number. This enables the topic to be displayed when the user clicks on different regions of the screen. Of course, you're free to keep track of your topic files in any manner you choose.

Figure 6.7
Help tracker text file example

| Ctx. String | Title | Browse Seq. | Key Words | Ctx. No. | Filename | Modified | Status |
|---|---|---|---|---|---|---|---|
| hlpidx_id_mp | Multipad Help Index | | | 0xFFFF | helpex.idx | 5/16/89 | Done |
| mc_cmd_mp | Multipad Commands | commands:0001 | commands; menus | 0x1000 | helpex.cmd | 5/16/89 | Done |
| fm_cmd_mp | File Menu | commands:0004 | commands; menus; files; documents | 0x1001 | helpex.cmd | 5/16/89 | Done |
| nc_cmd_mp | New Command | commands:0008 | commands; new files; new documents | 0x1002 | helpex.cmd | 5/16/89 | Done |
| oc_cmd_mp | Open Command | commands:0012 | commands; file; open; documents; read only | 0x1003 | helpex.cmd | 5/16/89 | Test |
| sac_cmd_mp | Save Command | commands:0016 | commands; file; save; save as; documents; files | 0x1004 | helpex.cmd | 5/16/89 | Done |
| sasc_cmd_mp | Save As Command | commands:0020 | commands; file; save as; save; documents; files | 0x1005 | helpex.cmd | 5/16/89 | Done |
| ptc_cmd_mp | Print Command | commands:0024 | commands; file; print; documents; files | 0x1006 | helpex.cmd | 5/16/89 | Done |
| psc_cmd_mp | Print Setup Command | commands:0028 | commands; file; printer setup; print | 0x1007 | helpex.cmd | 5/16/89 | Debug |
| ec_cmd_mp | Exit Command | commands:0032 | commands; file; exit; exiting; close; closing; quit; quitting | 0x1008 | helpex.cmd | 5/16/89 | Done |
| em_cmd_mp | Edit Menu | commands:0036 | commands; menus; editing; documents | 0x1009 | helpex.cmd | 5/16/89 | Done |
| uc_cmd_mp | Undo Command | commands:0040 | commands; edit; editing; undo; typing | 0x100A | helpex.cmd | 5/16/89 | Done |
| ctc_cmd_mp | Cut Command | commands:0044 | commands; edit; editing; cut; cutting; text; Clipboard | 0x100B | helpex.cmd | 5/16/89 | Done |
| cyc_cmd_mp | Copy Command | commands:0048 | commands ; edit; editing; copy; copying; text; Clipboard | 0x100C | helpex.cmd | 5/16/89 | Done |
| pec_cmd_mp | Paste Command | commands:0052 | commands; edit; editing; paste; insert; inserting; text; Clipboard | 0x100D | helpex.cmd | 5/16/89 | Done |
| crc_cmd_mp | Clear Command | commands:0056 | commands; edit; editing; clear; text | 0x100E | helpex.cmd | 5/16/89 | Done |
| salc_cmd_mp | Select All Command | commands:0060 | commands; edit; editing; select all; select; selecting; text | 0x100F | helpex.cmd | 5/16/89 | Done |
| wwc_cmd_mp | Word Wrap Command | commands:0064 | commands; edit; editing; word wrap; wrapping; text; format | 0x1010 | helpex.cmd | 5/16/89 | Done |
| tfc_cmd_mp | Type Face Command | commands:0068 | commands; edit; editing; type face; font; text; format | 0x1011 | helpex.cmd | 5/16/89 | Test |
| ptsc_cmd_mp | Point Size Command | commands:0072 | commands; edit; editing; point size; text; format | 0x1012 | helpex.cmd | 5/16/89 | Test |

Figure 6.8
Help tracker worksheet example

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | Ctx. String | Text | Browse Seq. | Key Words | Ctx. No. | Filename | Modified | Status |
| 2 | hlpidx_id_mp | Multipad Help Index | | | 0xFFFF | helpex.idx | 5/16/89 | Done |
| 3 | mc_cmd_mp | Multipad Commands | commands:0001 | commands; menus | 0x1000 | helpex.cmd | 5/16/89 | Done |
| 4 | fm_cmd_mp | File Menu | commands:0004 | commands; menus; files; documents | 0x1001 | helpex.cmd | 5/16/89 | Done |
| 5 | | | | | | | | |
| 6 | nc_cmd_mp | New Command | commands:0008 | commands; new files; new documents | 0x1002 | helpex.cmd | 5/16/89 | Done |
| 7 | | | | | | | | |
| 8 | oc_cmd_mp | Open Command | commands:0012 | commands; file; open; documents; read only | 0x1003 | helpex.cmd | 5/16/89 | Test |
| 9 | | | | | | | | |
| 10 | sac_cmd_mp | Save Command | commands:0016 | commands; file; save; save as; documents; files | 0x1004 | helpex.cmd | 5/16/89 | Done |
| 11 | | | | | | | | |
| 12 | sasc_cmd_mp | Save As Command | commands:0020 | commands; file; save as; save; documents; files | 0x1005 | helpex.cmd | 5/16/89 | Done |
| 13 | | | | | | | | |
| 14 | ptc_cmd_mp | Print Command | commands:0024 | commands; file; print; documents; files | 0x1006 | helpex.cmd | 5/16/89 | Done |
| 15 | | | | | | | | |
| 16 | psc_cmd_mp | Print Setup Command | commands:0028 | commands; file; printer setup; print | 0x1007 | helpex.cmd | 5/16/89 | Debug |
| 17 | | | | | | | | |
| 18 | ec_cmd_mp | Exit Command | commands:0032 | commands; file; exit; exiting; close; closing; quit; quitting | 0x1008 | helpex.cmd | 5/16/89 | Done |
| 19 | | | | | | | | |
| 20 | em_cmd_mp | Edit Menu | commands:0036 | commands; menus; editing; documents | 0x1009 | helpex.cmd | 5/16/89 | Done |
| 21 | | | | | | | | |
| 22 | uc_cmd_mp | Undo Command | commands:0040 | commands; edit; editing; undo; typing | 0x100A | helpex.cmd | 5/16/89 | Done |
| 23 | | | | | | | | |
| 24 | ctc_cmd_mp | Cut Command | commands:0044 | commands; edit; editing; cut; cutting; text; Clipboard | 0x100B | helpex.cmd | 5/16/89 | Done |
| 25 | | | | | | | | |
| 26 | cyc_cmd_mp | Copy Command | commands:0048 | commands ; edit; editing; copy; copying; text; Clipboard | 0x100C | helpex.cmd | 5/16/89 | Done |
| 27 | | | | | | | | |
| 28 | pec_cmd_mp | Paste Command | commands:0052 | commands; edit; editing; paste; insert; inserting; text; Clipboard | 0x100D | helpex.cmd | 5/16/89 | Done |
| 29 | | | | | | | | |
| 30 | crc_cmd_mp | Clear Command | commands:0056 | commands; edit; editing; clear; text | 0x100E | helpex.cmd | 5/16/89 | Done |
| 31 | | | | | | | | |
| 32 | salc_cmd_mp | Select All Command | commands:0060 | commands; edit; editing; select all; select; selecting; text | 0x100F | helpex.cmd | 5/16/89 | Done |
| 33 | | | | | | | | |
| 34 | wwc_cmd_mp | Word Wrap Command | commands:0064 | commands; edit; editing; word wrap; wrapping; text; format | 0x1010 | helpex.cmd | 5/16/89 | Done |
| 35 | | | | | | | | |
| 36 | tfc_cmd_mp | Type Face Command | commands:0068 | commands; edit; editing; type face; font; text; format | 0x1011 | helpex.cmd | 5/16/89 | Test |
| 37 | | | | | | | | |
| 38 | ptsc_cmd_mp | Point Size Command | commands:0072 | commands; edit; editing; point size; text; format | 0x1012 | helpex.cmd | 5/16/89 | Test |
| 39 | | | | | | | | |

# Building the Help file

After the topic files for your Help system have been written, you are ready to create a Help project file and run a build to test the Help file. The Help project file contains all information the compiler needs to convert help topic files into a binary Help resource file.

You use the Help project file to tell the Help Compiler which topic files to include in the build process. Information in the Help project file also enables the compiler to map specific topics to context numbers (for the context-sensitive portion of Help).

After you have compiled your Help file, the development team programs the application so the user can access it.

This section describes the following:

- Creating a Help project file
- Compiling the Help file
- Programming the application to access Help

## Creating the Help project file

You use the Help project file to control how the Help Compiler builds your topic files. This section describes both the HC30 and HC31 help compilers. Options that apply to the HC31 compiler only are specified as new to HC31 or designated as available for the "HC31 Help Compiler only." Help project files can contain the following sections:

Table 6.6
Help project file sections

| Section | Function |
|---|---|
| [Files] | Specifies topic files to be included in the build. This section is mandatory. |
| [Options] | Specifies the level of error reporting, topics to be included in the build, the directory in which to find the files, and the location of your Help index. This section is optional. |
| [BuildTags] | Specifies valid build tags. This section is optional. |
| [Config] | Specifies Help macros that define nonstandard menus, buttons, and macros used in the Help file. This section is required if your Help file uses any of these features. This section is new for HC31. |
| [Alias] | Assigns one or more context strings to the same topic. This section is optional. |
| [Map] | Associates context strings with context numbers. This section is optional. |
| [Bitmaps] | Specifies bitmap files to be included in the build. This section is optional. |
| [Windows] | Defines the characteristics of the primary Help window and the secondary-window types used in the Help file. This section is required if the Help file uses secondary windows. This section is new for HC31. |

Table 6.6: Help project file sections (continued)

| | |
|---|---|
| [Baggage] | Lists files that are to be placed within the help file that contains its own file system. This section is optional. |

You can use any ASCII text editor to create your Help project file. The extension of a Help project file is .HPJ. If you don't use the extension .HPJ on the **HC** command line, the compiler looks for a project file with this extension before loading the file. The .HLP output file will have the same name as the .HPJ file.

The order of the sections within the Help project file is arbitrary, except that an [Alias] section must always precede the [Map] section (if an [Alias] section is used).

Section names are placed within square brackets using the following syntax:

[*section-name*]

You can use a semicolon to indicate a comment in the project file. The compiler ignores all text from the semicolon to the end of the line on which it occurs.

## Specifying topic files

Use the [Files] section of the Help project file to list all topic files that the Help Compiler should process to produce a Help resource file. A Help project file must have a [Files] section.

The following sample shows the format of the [Files] section:

```
[FILES]
HELPEX.RTF  ;Main topics for HelpEx application
KEYS.RTF    ;Keyboard topics for HelpEx application
TERMS.RTF   ;Lookup terms for HelpEx application
```

*For more information about the **ROOT** option, see the section "Specifying the root directory: The Root option."*

Using the path defined in the **ROOT** option, the Help Compiler finds and processes all files listed in this section of the Help project file. If the file is not on the defined path and can't be found, the compiler generates an error.

You can include files in the build process using the C **#include** directive command. The **#include** directive uses this syntax:

#include <*filename*>

You must include the angle brackets around the file name. The pound sign ( # ) must be the first character in the line. The file

name must specify a complete path, either the path defined by the **ROOT** option or an absolute directory path to the file.

You may find it easier to create a text file that lists all files in the Help project and include that file in the build, as in this example:

```
[FILES]
#include <hlpfiles.inc>
```

## Specifying macros

*HC31 Help Compiler only*

Use the [Config] section of the Help project file to add macros that Windows should execute when it first opens the Help file. If there is more than one macro listed in the [Config] section, Windows executes them in the order in which they are listed. You can add macros that create new menus, menu items, or buttons. These macros remain in effect until the user quits Help or opens a new help file.

You can also develop your own dynamic-link libraries (DLLs) and define Help macros that call function in the libraries. To do this, you must register each function and its corresponding library by using the *RegisterRoutine* macro in the [Config] section of the project file.

### Macro syntax

When Windows loads Help or displays a topic, it follows the actions specified in a macro. A Help macro consists of a name and parameters enclosed in parentheses.

Macro names specify the action to take, such as creating buttons or inserting menu items. Because the names are not case-sensitive, you can use any combination of uppercase and lowercase letters.

Macro parameters specify the files, buttons, menus, or topics that carry out the indicated actions. Parameters need to be enclosed in parentheses and separated by spaces. In addition, if the parameter contains spaces, it must also be enclosed in quotation marks. If a quotation character is needed as part of the parameter, enclose the parameter in single quotation marks.

You can also use macros as parameters to other macros. In most cases, if macros are embedded, they must be enclosed in quotation marks.

Although an individual macro and its parameters must not exceed 512 characters, you can combine Help macros into macro strings by separating the macros with semicolons. The Help

Compiler treats the macro string as a unit, executing the individual macro sequentially.

Table 6.7 lists the HC31 Help compiler macros. For more detailed information about macros, look at BPW.HLP, the Windows Help file shipped with Borland Pascal.

Table 6.7
Help compiler macros

| Name | Function |
|---|---|
| About | Displays the Windows Help's About dialog box. |
| AddAccelerator | Assigns a Help macro to an accelerator key (or key combination) so that the macro is carried out when the user presses the accelerator key(s). |
| Annotate | Displays the Annotation dialog box. |
| AppendItem | Appends a menu item to the end of a menu created with the *InsertMenu* macro. |
| Back | Displays the previous topic in the history list. |
| BookmarkDefine | Displays the Define dialog from the Bookmark menu. |
| BookmarkMore | Displays the More dialog from the Bookmark menu. |
| BrowseButtons | Adds browse buttons to the button bar. |
| ChangeButtonBinding | Assigns a Help macro to a Help button. |
| ChangeItemBinding | Assigns a Help macro to an item previously added to a Windows Help menu using the *AppendItem* macro. |
| CheckItem | Places a check-mark beside a menu item. |
| CloseWindow | Closes either a secondary window or the main Help window. |
| Contents | Displays the Contents topic in the current Help file. |
| CopyDialog | Displays the Copy dialog from the Edit menu. |
| CopyTopic | Copies all the text in the currently displayed topic to the Clipboard. |
| CreateButton | Adds a new button to the button bar. |
| DeleteItem | Removes a menu item that was added with the *AppendItem* macro. |
| DeleteMark | Removes a text marker added with the *SaveMark* macro. |

Table 6.7: Help compiler macros (continued)

| | |
|---|---|
| DestroyButton | Removes a button added with the *CreateButton* macro. |
| DisableButton | Dims a button added with the *CreateButton* macro. |
| DisableItem | Dims a menu item added with the *AppendItem* macro. |
| EnableButton | Re-enables a button disabled with the *DisableButton* macro. |
| EnableItem | Enables a menu item disabled with the *DisableItem* macro. |
| ExecProgram | Executes a Windows application. |
| Exit | Exits the Windows Help application. |
| FileOpen | Displays the Open dialog box from the File menu. |
| FocusWindow | Changes the focus to the specified window, either the main Help window or a secondary window. |
| GoToMark | Jumps to a marker set with the *SaveMark* macro. |
| HelpOn | Displays the Help file for the Windows Help application. |
| HelpOnTop | Toggles the on-top state of Windows Help. |
| History | Displays the history list which shows the last 40 topics the user has viewed since opening a Help file in Windows Help. |
| IfThen | Executes a Help macro if a given marker exists. |
| IfThenElse | Executes one of two Help macros depending on whether or not a marker exists. |
| InsertItem | Inserts a menu item at a given position on an existing menu. |
| InsertMenu | Inserts a new menu in the Windows Help menu bar. |
| IsMark | Tests whether or not a marker set by SaveMark exists. |
| JumpContents | Jumps to the Contents topic of a specified file in the Help file. |
| JumpContext | Specifies the name of the destination file for the jump. |

Table 6.7: Help compiler macros (continued)

| | |
|---|---|
| JumpHelpOn | Jumps to the Contents topic of the How to Use Help file. |
| JumpId | Jumps to the topic with the specified context string in the Help file. |
| JumpKeyword | Loads the indicated Help file, searches through the K keyword table, and displays the first topic containing the index keyword specified in the macro. |
| Next | Displays the next topic in the browse sequence for the Help file. |
| Not | Reverses the result (nonzero or zero) returned by IsMark. |
| PopupContext | Displays in a pop-up window the topic identified by a specific context number. |
| PopupId | Displays a topic from a specified file in a pop-up window. |
| PositionWindow | Sets the size and position of a window. |
| Prev | Displays the previous topic in the browse sequence for the Help file. |
| Print | Sends the currently displayed topic to the printer. |
| PrinterSetup | Displays the Printer Setup dialog box from the File menu. |
| RegisterRoutine | Registers function within a dynamic-link library (DLL). |
| SaveMark | Saves the location of the currently displayed topic and file and associates a text marker with that location. |
| Search | Displays the dialog for the Search button, which allows users to search for topics using keywords defined by the K footnote character. |
| SetContents | Designates a specific topic as the Contents topic in the specified Help file. |
| SetHelpOnFile | Specifies the name of the replacement How to Use Help file. |
| UncheckItem | Removes the checkmark from a menu item. |

## Specifying build tags

If you code build tags in your topic files, use the [BuildTags] section of the Help project file to define all the valid build tags for a particular Help project. The [BuildTags] section is optional.

The following example shows the format of the [BuildTags] section in a sample Help project file:

```
[BUILDTAGS]
WINENV              ;topics to include in Windows build
DEBUGBUILD          ;topics to include in debugging build
TESTBUILD           ;topics to include in a mini-build for testing
```

*For information about coding build tags in topic files, see page 83.*

The [BuildTags] section can include up to 30 build tags. The build tags are not case-sensitive and may not contain space characters. Only one build tag is allowed per line in this section. The compiler will generate an error message if anything other than a comment is listed after a build tag in the [BuildTags] section.

## Specifying options

Use the [Options] section of the Help project file to specify the following options:

Table 6.8 The Help (Options) options

| Option | Meaning |
|---|---|
| BMROOT | Specifies the directory containing the bitmap files named in the **bmc, bml,** and **bmr** topic files. This is new for Windows 3.1. |
| BUILD | Determines what topics the compiler includes in the build. |
| COMPRESS | Specifies compression of the Help resource file. |
| CONTENTS | Specifies the context string for the Contents topic of an HC31 Help file. This is new for Windows 3.1. |
| COPYRIGHT | Adds a unique copyright message for the Help file to the About dialog box. This is new for Windows 3.1. |
| ERRORLOG | Puts compilation errors in a file during the build. This is new for Windows 3.1. |
| FORCEFONT | Specifies the creation of a Help resource file using only one font. |
| ICON | Specifies the icon file to be displayed when the Help file is minimized. This is new for Windows 3.1. |
| INDEX | Specifies the context string for an HC30 Help index. |

Table 6.8: The Help (Options) options (continued)

| | |
|---|---|
| LANGUAGE | Specifies a different sorting order for Help files written in a Scandanavian language. This is new for Windows 3.1. |
| MAPFONT SIZE | Determines the mapping of specified font sizes to different sizes. |
| MULTIKEY | Specifies alternate keyword mapping for topics. |
| OLDKEYPHRASE | Specifies whether the compiler should use the existing keyphrase table or create a new one during the build. This is new for Windows 3.1. |
| OPTCDROM | Optimizes the Help file for CD-ROM use. This is new for Windows 3.1. |
| REPORT | Controls the display of messages during the build process. This is new for Windows 3.1. |
| ROOT | Designates the directory to be used for the Help build. |
| TITLE | Specifies the title shown for the Help system. |
| WARNING | Indicates the kind of error message the compiler reports. |

These options can appear in any order within the [Options] section. The [Options] section is not required.

Detailed explanations of the available options follow.

### Specifying build topics

If you have included build tags in your topic files, use the **BUILD** option to specify which topics to conditionally include in the build. If your topic files have no build tags, omit the **BUILD** option from the [Options] section.

All build tags must be listed in the [BuildTags] section of the project file, regardless whether or not a given conditional compilation declares the tags.

*See "Assigning build tags" on page 81 for information on assigning build tags to topics in the Help topic files.*

The **BUILD** option line uses the following syntax:

BUILD = *expression*

Build expressions can't exceed 255 characters in length and must be entered on only one line. Build expressions use Boolean logic to specify which topics within the specified Help topic files the compiler will include in the build. The compiler evaluates all build expressions from left to right. The tokens of the language (listed in order of precedence from highest to lowest) are:

Table 6.9
Build tag order of precedence

| Token | Description |
|---|---|
| *<tag>* | Build tag |
| ( ) | Parentheses |
| ~ | NOT operator |
| & | AND operator |
| \| | OR operator |

For example, if you coded build tags called WINENV, APP1, and TEST_BUILD in your topic files, you could include one of the following build expressions in the [Options] section:

Table 6.10: Build expression examples

| Build expression | Topics built |
|---|---|
| BUILD = WINENV | Only topics that have the WINENV tag |
| BUILD = WINENV & APP1 | Topics that have both the WINENV and APP1 tags |
| BUILD = WINENV \| APP1 | Topics that have either the WINENV tag or the APP1 tag |
| BUILD = (WINENV \| APP1) & TESTBUILD | Topics that have either the WINENV or the APP1 tags and that also have the TESTBUILD tag |
| BUILD =~ APP1 | Topics that don't have an APP1 tag |

## Specifying the bitmap file directory

*HC31 only*

The **BMROOT** option lets you specify the directory containing the bitmap files specified in the **bmc, bml,** and **bmr** statements. If the project file has a **BMROOT** option, you don't need to list the bitmap files in the [BITMAPS] section.

If the project file does not have a **ROOT** option or if the **ROOT** option does not specify the directory containing the bitmap files, the file name for each bitmap must be specified in the [BITMAPS] section.

The **BMROOT** option line uses the following syntax:

BMROOT = *path[,path]...*

## Compressing the file

You can use the **COMPRESS** option to reduce the size of the Help resource file created by the compiler. The amount of file compression realized varies according to the number, size and complexity of topics that are compiled. In general, the larger the Help files, the more they can be compressed.

In HC30 the **COMPRESS** option uses the following syntax:

COMPRESS = TRUE | FALSE

In HC31 the **COMPRESS** option accepts the following additional values:

| Value | Meaning |
|---|---|
| 0 | No compression |
| 1 | High compression |
| False | No compression |
| High | High compression |
| Medium | Medium compression |
| No | No compression |
| True | High compression |
| Yes | High compression |

Because the Help application can load compressed files quickly, there is a clear advantage in creating and shipping compressed Help files with your application. Compiling with compression turned on, however, may increase the compile time, because of the additional time required to assemble and sort a key-phrase table. Thus, you may want to compile without compression in the early stages of a project.

The **COMPRESS** option causes the compiler to compress the system by combining repeated phrases found within the source file(s). The compiler creates a phrase-table file with the .PH extension if it does not find one already in existence. If the compiler finds a file with the .PH extension, it will use the file for the current compilation. This is in order to speed compression when not a lot of text has changed since the last compilation.

Deleting the key-phrase file before each compilation will prevent the compiler from using the previous file. Maximum compression will result only by forcing the compiler to create a new phrase table.

### Specifying the contents
*HC31 only*

Use the **CONTENTS** option to identify the context-string of the highest-level or Contents topic. This topic is usually a table of contents or an index within a Help file. The context-string can be any combination of characters, except spaces, and must also be specified in a context-string **footnote** statement in some topic in the Help file.

The **CONTENTS** option line uses the following syntax:

CONTENTS = *context-string*

If the [OPTIONS] section does not include a **CONTENTS** options, the compiler assumes that the Contents topic is the first topic encountered in the first listed topic file in the [FILES] section of the project file.

### Specifying a copyright notice

*HC31 only*

You can place a customized copyright notice in the About dialog box with the **COPYRIGHT** option. The notice can be any combination of characters and range in length from 35 through 75 characters.

The **COPYRIGHT** option line uses the following syntax:

COPYRIGHT = *copyright-notice*

### Specifying an error file

*HC31 only*

You can use the **ERRORLOG** option to tell the Help Compiler to write all error messages to a specified file. The compiler also displays the error messages on the screen.

The **ERRORLOG** option line uses the following syntax:

ERRORLOG = *error-filename*

The *error-filename* parameter, which specifies the name of the file to receive the error messages, should include a full or partial path if the error file is written to a directory other than the project root directory.

### Converting fonts

You can use the **FORCEFONT** option to create a Help resource file that is made up of only one typeface or font. This is useful if you must compile a Help system using topic files that include fonts not supported by your users' systems.

The **FORCEFONT** option uses the following syntax:

FORCEFONT = *fontname*

The *fontname* parameter is any Windows system font. Note that the *fontname* used in the **FORCEFONT** option can't contain spaces. Therefore, Times Roman font can't be used with **FORCEFONT**.

Font names must be spelled the same as they are in the Fonts dialog box in Control Panel. Font names don't exceed 20 characters in length. If you designate a font that is not recognized by the compiler, an error message is generated and the compilation continues using the default font. In HC30, the default font is Helvetica (Helv) font; in HC31, the default font is MS Sans Serif.

### Specifying an icon file

*HC31 only*

You can use the **ICON** option to identify the icon file to display when the user minimizes Windows Help. The icon file must have the standard Windows icon-file format.

The **ICON** option line uses the following syntax:

ICON = *icon-file*

### Specifying the index

Use the **INDEX** option to identify the context string of the HC30 Help index. (For HC31, use the **CONTENTS** option.) Because the Index button gives the user access to the index from anywhere in the Help system, you will probably not want to write terms to jump to the index. Users access this general index either from the Help menu of the application or by choosing the Index button from the Help window.

Assigning a context string to the index topic in the [Options] section lets the compiler know the location of the main index of Help topics for the application's Help file. If you don't include the **INDEX** option in the [Options] section, the compiler assumes that the first topic it encounters is the index.

The **INDEX** option uses the following syntax:

INDEX = *context-string*

*For information on assigning context strings, see page 85.*

The context string specified must match the context string you assigned to the Help index topic. In the following example, the writer informs the compiler that the context string of the Help index is "main_index":

```
[OPTIONS]
INDEX=main_index
```

### Specifying a language sort order

*HC31 only*

The **LANGUAGE** option sets the sort order for keywords in the Search dialog box. You can specify either the English-language order, the default, or the Scandanavian-language order.

The **LANGUAGE** option line uses the following syntax:

LANGUAGE = *language-name*

### Changing font sizes

The font sizes specified in your topic files can be mapped to different sizes using the **MAPFONTSIZE** option. In this manner, you can create and edit text in a size chosen for easier viewing in the topic files and have them sized by the compiler for the actual Help display. This may be useful if there is a large size difference between your authoring monitor and your intended display monitor.

The **MAPFONTSIZE** option uses the following syntax:

MAPFONTSIZE = *m*[–*n*]:*p*

The *m* parameter is the size of the source font, and the *p* parameter is the size of the desired font for the Help resource file. All fonts in the topic files that are size *m* are changed to size *p*. The optional parameter *n* lets you specify a font range to be mapped. All fonts in the topic files falling between *m* and *n*, inclusive, are changed to size *p*. The following examples illustrate the use of the **MAPFONTSIZE** option:

```
MAPFONTSIZE=12-24:16 ;make fonts from 12 to 24 come out 16.
MAPFONTSIZE=8:12  ;make all size 8 fonts come out size 12.
```

Note that you can map only one font size or range with each **MAPFONTSIZE** statement used in the Options section. If you use more than one **MAPFONTSIZE** statement, the source font size or range specified in subsequent statements can't overlap previous mappings. For instance, the following mappings would generate an error when the compiler encountered the second statement:

```
MAPFONTSIZE=12-24:16 MAPFONTSIZE=14:20
```

Because the second mapping shown in the first example contains a size already mapped in the preceding statement, the compiler will ignore the line. There is a maximum of five font ranges that can be specified in the project file.

### Multiple keyword tables

The **MULTIKEY** option specifies a character to be used for an additional keyword table.

The **MULTIKEY** option uses the following syntax:

MULTIKEY = *footnote-character*

The *footnote-character* parameter is the case-sensitive letter to be used for the keyword footnote. The following example illustrates the enabling of the letter *L* for a keyword-table footnote:

```
MULTIKEY=L
```

You must be sure to limit your keyword-table footnotes to one case, usually uppercase. In the previous example, topics with the footnote *L* would have their keywords incorporated into the additional keyword table, whereas those assigned the letter *l* would not.

You may use any alphanumeric character for a keyword table except *K* and *k*, which are reserved for Help's normal keyword table. There is an absolute limit of five keyword tables, including the normal table. However, depending upon system configuration and the structure of your Help system, a practical limit of only two or three may actually be the case. If the compiler can't create an additional table, the excess table is ignored in the build.

### Specifying a key-phrase file
*HC31 only*

You can use the **OLDKEYPHRASE** to specify whether or not an existing key-phrase file should be used to build the Help file. The *onoff* parameter can include one of the following options:

Table 6.11
Key-phrase file values

| Value | Meaning |
|---|---|
| 0 | Recreate the file |
| 1 | Use the existing file |
| FALSE | Recreate the file |
| NO | Recreate the file |
| OFF | Recreate the file |
| ON | Use the existing file |
| TRUE | Use the existing file |
| YES | Use the existing file |

The **OLDKEYPHRASE** option line uses the following syntax:

OLDKEYPHRASE = *onoff*

### Optimizing a help file for CD-ROM
*HC31 only*

Use the **OPTCDROM** option if you want to optimize a help file for display on CD-ROM by aligning topic files on predefined block boundaries.

The **OPTCDROM** option line uses the following syntax:

OPTCDROM = *yesvalue*

The *yesvalue* parameter can be any one of the following values:

```
YES             1
TRUE            ON
```

### Displaying build messages
*HC31 only*

Use this option to display messages on the screen during the build. These messages indicate when the Help compiler is performing the different phases of the build, such as compiling the file, resolving jumps, and verifying browse sequences.

The **REPORT** option line uses the following syntax:

REPORT = ON

### Specifying the root directory

Use the **ROOT** option to designate the root directory of the Help project. The compiler searches for files in the specified root directory.

The **ROOT** option uses the following syntax:

ROOT = *pathname*

For example, the following root option specifies that the root directory is \BUILD\TEST on drive D:

```
[OPTIONS]
ROOT=D:\BUILD\TEST
```

The **ROOT** option lets you refer to all relative paths off the root directory of the Help project. For example, the following entry in the [Files] section refers to a relative path off the root directory:

```
TOPICS\FILE.RTF
```

To refer to a file in a fixed location, independent of the project root, you must specify a fully qualified or "absolute" path, including a drive letter, if necessary, as in the following line:

```
D:\HELPTEST\TESTFILE.RTF
```

If you don't include the **ROOT** option in your Help project file, all paths are relative to the current DOS directory.

### Assigning a title to the Help system

You can assign a title to your Help system with the **TITLE** option. The title appears in the title bar of the Help window with the word "Help" automatically appended, followed by the DOS file name of the Help resource file.

The **TITLE** option uses the following syntax:

TITLE = *Help-system-title-name*

Titles are limited to 32 characters in length. If you don't specify a title using the **TITLE** option, only the word Help followed by the Help system file name will be displayed in the title bar. Because the compiler always inserts the word Help, don't duplicate it in your title.

### Specifying error reporting

Use the **WARNING** option to specify the amount of debugging information that the compiler reports. The **WARNING** option has the following syntax:

WARNING = *level*

You can set the **WARNING** option to any of the following levels:

Table 6.12
WARNING levels

| Level | Information Reported |
|---|---|
| 1 | Only the most severe warnings. |
| 2 | An intermediate level of warnings. |
| 3 | All warnings. This is the default level if no **WARNING** option is specified. |

The following example specifies an intermediate level of error reporting:

```
[OPTIONS]
WARNING-2
```

*Use the DOS Ctrl+PrtSc accelerator key before you begin your compilation to echo errors which appear on the screen to your printer. Type Ctrl+PrtSc again to stop sending information to the printer.*

The compiler reports errors to the standard output file, typically the screen. You may want to redirect the errors to a disk file so that you can browse it when you are debugging the Help system. The following example shows the redirection of compiler screen output to a file.

```
HC HELPEX >> errors.out
```

## Specifying alternate context strings

Use the [Alias] section to assign one or more context strings to the same topic alias. Because context strings must be unique for each topic and can't be used for any other topic in the Help project, the [Alias] section provides a way to delete or combine Help topics without recoding your files. The [Alias] section is optional.

For example, if you create a topic that replaces the information in three other topics, and you delete the three, you will have to search through your files for invalid cross-references to the deleted topics. You can avoid this problem by using the [Alias]

section to assign the name of the new topic to the deleted topics. You can also use the [Alias] section when your application program has multiple context identifiers for which you have only one topic. This can be the case with context-sensitive Help.

Each expression in the [Alias] section has the following format:

*context_string=alias*

In the alias expression, the *alias* parameter is the alternate string, or alias name, and the *context_string* parameter is the context string identifying a particular topic. An alias string has the same format and follows the same conventions as the topic context string. That is, it is not case-sensitive and may contain the alphabetic characters A–Z, numeric characters 0–9, and the period and underscore characters.

The following example illustrates an [Alias] section:

```
[ALIAS]
sm_key=key_shrtcuts
cc_key=key_shrtcuts
st_key=key_shrtcuts;combined into keyboard shortcuts topic
clskey=us_dlog_bxs
maakey=us_dlog_bxs;covered in using dialog boxes topic
chk_key=dlogprts
drp_key=dlogprts
lst_key=dlogprts
opt_key=dlogprts
tbx_key=dlogprts;combined into parts of dialog box topic
frmtxt=edittxt
wrptxt=edittxt
seltxt=edittxt;covered in editing text topic
```

You can use alias names in the [Map] section of the Help project file. If you do, however, the [Alias] section must precede the [Map] section.

## Mapping context-sensitive topics

*For more information on context-sensitive Help, see page 75.*

If your Help system supports context-sensitive Help, use the [Map] section to associate either context strings or aliases to context numbers. The context number corresponds to a value the parent application passes to the Help application in order to display a particular topic. This section is optional.

When writing the [Map] section, you can do the following:

- Use either decimal or hexadecimal numbers formatted in standard C notation to specify context numbers.
- Assign no more than one context number to a context string or alias. Assigning the same number to more than one context string generates a compiler error.
- Separate context numbers and context strings by an arbitrary amount of whitespace using either space characters or tabs.

You can use the C **#include** directive to include other files in the mapping. In addition, the Map section supports an extended format that lets you include C files with the .H extension directly. Entries using this format must begin with the **#define** directive and may contain comments in C format, as in this example:

```
#define   context_string context_number   /* comment */
```

The following example illustrates several formats you can use in the [Map] section:

```
[MAP]

Edit_Window             0x0001
Control_Menu            0x0002
Maximize_Icon           0x0003
Minimize_Icon           0x0004
Split_Bar               0x0005
Scroll_Bar              0x0006
Title_Bar               0x0007
Window_Border           0x0008

dcmb_scr                30; Document Control-menu Icon
dmxi_scr                31; Document Maximize Icon
dmni_scr                32; Document Minimize Icon
dri_scr                 33; Document Restore Icon
dtb_scr                 34; Document Title Bar

#define vscroll   0x010A   /* Vertical Scroll Bar */
#define hscroll   0x010E   /* Horizontal Scroll Bar */
#define scrollthm 0x0111   /* Scroll Thumb */
#define upscroll  0x0112   /* Up Scroll Arrow */
#define dnscroll  0x0113   /* Down Scroll Arrow */

#include <sample.h>
```

*These eight entries give hexadecimal equivalents for the context numbers.*

*These five entries show decimal context numbers.*

*These five entries show how you might include topics defined in a C include file.*

*This entry shows a C **#include** directive for some generic topics.*

If context numbers use the **#define** directive, and the file containing the **#define** statements is included in both the application code and the Help file, then updates made to the context numbers by the application programmers will automatically be reflected in the next Help build.

You can define the context strings listed in the [Map] section either in a Help topic or in the [Alias] section. The compiler generates a warning message if a context string appearing in the [Map] section is not defined in any of the topic files or in the [Alias] section.

If you use an alias name, the [Alias] section must precede the [Map] section in the Help project file.

## Including bitmaps by reference

If your Help system uses bitmaps by reference, the file names of each of the bitmaps must be listed in the [Bitmaps] section of the project file. The following example illustrates the format of the [Bitmaps] section.

```
[BITMAPS]
DUMP01.BMP
DUMP02.BMP
DUMP03.BMP
c:\PROJECT\HELP\BITMAPS\DUMP04.BMP
```

The [Bitmaps] section uses the same rules as the [Files] section for locating bitmap files.

## Compiling Help files

After you have created a Help project file, you are ready to build a Help file using the Help Compiler. The compiler generates the binary Help resource file from the topic files listed in the Help project file. When the build process is complete, your application can access the Help resource file that results.

Your distribution disks include two Help compilers – HC30 and HC31. HC30, the Windows 3.0 Help compiler, has fewer options than the HC31 Help compiler, but HC30 help files can be used by both *WinHelp 3.0* and *WinHelp 3.1*. HC31, the Windows 3.1 version of the Help compiler, produces help files that can be used only with *WinHelp 3.1*. Throughout this chapter, only the name *WinHelp* is used if the function described applies to either *WinHelp 3.0* or *WinHelp 3.1*. The following table illustrates which compilers are compatible with which versions of Windows and *WinHelp*.

Table 6.13
Help compiler compatibility

| Help compiler | Windows 3.0 compatible | Windows 3.1 compatible |
|---|---|---|
| HC30 Help files | Yes (use WinHelp 3.0) | Yes (use WinHelp 3.1) |
| HC31 Help files | No (Yes, if WinHelp 3.1 is shipped with the application) | Yes (use WinHelp 3.1) |

Before initiating a build operation to create the Help file, consider the locations of the following files:

- The Help Compiler, HC30.EXE or HC31.EXE. The compiler must be in a directory from which it can be executed. This could be the current working directory, on the path set with the PATH environment variable, or a directory specified by a full pathname, as follows:

```
C:\BIN\HC HELPEX.HPJ
```

- The Help project file, *filename*.HPJ. The project file can be located either in the current directory or specified by a path, as follows:

```
C:\BIN\HC D:\MYPROJ\HELPEX.HPJ
```

- The topic files named in the Help project file, saved as RTF. The topic files may be located in the current working directory, a subdirectory of the current working directory specified in the [Files] section, or the location specified in the Root option.
- Files included with the **#include** directive in the Help project file. Since the **#include** directive can take path names, then any number of places will work for these files.
- All bitmap files listed by reference in the topic files.

You must also place any files named in an **#include** directive in the path of the project root directory or specify their path using the **ROOT** option. The compiler searches only the directories specified in the Help project file. For information about the **ROOT** option, see the section titled "Specifying the root directory," on page 112.

If you use a RAM drive for temporary files (set with the DOS environment variable TMP), it must be large enough to hold the compiled Help resource file. If your Help file is larger than the size of the available RAM drive, the compiler will generate an error message and abort the compilation.

### Using the Help Compilers

Your distribution disks include a batch file named HC.BAT. If you use the command **HC** to compile your help file, you will run HC.BAT, which automatically runs the HC30 Help compiler. If you want to run the HC31 Help compiler, type **HC –n**. You can also just type **HC31** or **HC30** to run either compiler. Use the following syntax:

```
HC filename.HPJ                          Runs the HC30 compiler
```

or

```
HC -n filename.HPJ                       Runs the HC31 Help compiler
```

As the compiler program runs, it displays sequential periods on the screen, indicating its progress in the process. Error messages are displayed when each error condition is encountered. When the Help Compiler has finished compiling, it writes a Help resource file with an .HLP extension in the current directory and returns to the DOS prompt. The Help resource file that results from the build has the same name as does the Help project file.

Compiler errors and status messages can be redirected to a file using standard DOS redirection syntax. This is useful for a lengthy build where you may not be monitoring the entire process. The redirected file is saved as a text file that can be viewed with any ASCII editor.

## Programming the application to access Help

You need to program the application so that the user can access both the Windows Help application and your Help file. The Help application is a standalone Windows application, and your application can ask Windows to run the Help application and specify the topic that Help is to show the user. To the user, Help appears to be part of your application, but it acts like any other Windows application and exists as a resource that Windows applications can share.

### Calling WinHelp from an application

An application makes a Help system available to the user by calling the *WinHelp* function.

The *WinHelp* function uses the following syntax:

```
function WinHelp(Wnd: HWnd; HelpFile: PChar; Command: Word;
  Data: LongInt): Bool;
```

The *Wnd* parameter identifies the window requesting Help. The Windows Help application uses this identifier to keep track of which applications have requested Help.

The *HelpFile* parameter specifies the name (with optional directory path) of the Help file containing the desired topic.

In Windows 3.1, the *HelpFile* parameter points to a null-terminated string containing the path, if necessary, and the name of the help file that the Help application is to display. The file name can be followed by an angle bracket (>) and the name of a secondary window if the topic is to be displayed in a secondary window.

The *Command* parameter specifies either the type of search that the Windows Help application uses to locate the specified topic or indicates that the application no longer requires Help. Values for the *Command* parameter for *WinHelp 3.0* are defined in WINTYPES.PAS and for *WinHelp 3.1* are defined in WIN31.PAS. *Command* may be set to any one of the following values:

Table 6.14
Command values

| Value | Meaning |
|---|---|
| HELP_COMMAND | Executes a Help macro. Defined in WIN31.PAS. |
| HELP_CONTEXT | Displays Help for a particular topic identified by a context number. |
| HELP_CONTENTS | Displays the Help contents topic as defined by the Contents option in the [OPTIONS] section of the .HPJ file. Defined in WIN31.PAS. |
| HELP_CONTEXTPUPUP | Displays in a pop-up window a particular Help topic identified by a context number that has been defined in the [MAP] section of the .HPJ file. Defined in WIN31.PAS. |
| HELP_FORCEFILE | Ensures that *WinHelp* displays the correct Help file. Defined in WIN31.PAS. |
| HELP_HELPONHELP | Displays the Using Help index topic. |
| HELP_INDEX | Displays the main Help index topic. |
| HELP_KEY | Displays Help for a topic identified by a keyword. |
| HELP_MULTIKEY | Displays Help for a topic identified by a keyword in an alternate keyword table. |
| HELP_PARTIALKEY | Displays the topic found in the keyword list that matches the keyword passed in the *Data* parameter if there is an exact match. If there |

Table 6.14: Command values (continued)

| | |
|---|---|
| | is more than one match, displays the Search dialog box with the topics found listed in the Go To list box. If there is no match, displays the Search dialog box. Defined in WIN31.PAS. |
| HELP_QUIT | Informs the Help application that Help is no longer needed. If no other applications have asked for Help, Windows closes the Help application. |
| HELP_SETCONTENTS | Determines which Contents topic Help should display when a user presses the F1 key. Defined in WIN31.PAS. |
| HELP_SETINDEX | Displays a designated Help index topic. |
| HELP_SETWINPOS | Displays the Help window if it is minimized or in memory. Defined in WIN31.PAS. |

The *Data* parameter specifies the topic for which the application is requesting Help. The format of *Data* depends upon the value of *Command* passed when your application calls *WinHelp*. The following list describes the format of *Data* for each value of *Command*.

Table 6.15
Data formats

| ***Command* value** | ***Data* format** |
|---|---|
| HELP_COMMAND | A *PChar* that contains a Help macro to be executed. |
| HELP_CONTENTS | Ignored. |
| HELP_CONTEXT | A long integer containing the context number for the topic. Instead of using HELP_INDEX, HELP_CONTEXT can use the value –1. |
| HELP_CONTEXTPOPUP | A long integer containing the context number for a topic. |
| HELP_FORCEFILE | Ignored. |
| HELP_HELPONHELP | Ignored. |
| HELP_INDEX | Ignored. |
| HELP_KEY | A *PChar* that contains a keyword for the desired topic. |
| HELP_MULTIKEY | A pointer to the **TMultiKeyHelp** structure, as defined in WINTYPES.PAS. This structure specifies the table footnote character and the keyword. |

Table 6.15: Data formats (continued)

| | |
|---|---|
| HELP_PARTIALKEY | A *PChar* that contains a keyword for the desired topic. |
| HELP_QUIT | Ignored. |
| HELP_SETCONTENTS | A long integer containing the context number for the topic the application wants to designate as the Contents topic. |
| HELP_SETINDEX | A long integer containing the context number for the topic. |
| HELP_SETWINPOS | A pointer to the **THelpWinInfo** record defined in WIN31.PAS. This structure specifies the size and position of the primary Help window or a secondary window. |

Because it can specify either a context number or a keyword, *WinHelp* supports both context-sensitive and topical searches of the Help file.

To ensure that the correct index remains set, the application should call *WinHelp* with *Command* set to HELP_SETINDEX (with *Data* specifying the corresponding context identifier) following each call to *WinHelp* with a command set to HELP_CONTEXT. HELP_INDEX should never be used with HELP_SETINDEX.

## Getting context-sensitive Help

Context-sensitive Help should be made available when a user wants to learn about the purpose of a particular window or control. For example, the user might pull down the File menu, select the Open command (by using the arrow keys), and then press *F1* to get Help for the command.

Implementing certain types of context-sensitive help requires advanced programming techniques. The Helpex sample application illustrates the use of two techniques. These techniques are described in the following sections.

### Shift+F1 support

To implement a *Shift+F1* mode, Helpex responds to the *Shift+F1* hot key by calling **SetCursor** to change the shape of the cursor to an arrow pointer supplemented by a question mark.

```
WM_KEYDOWN:
  if WParam = vk_F1 then

    { If Shift-F1, turn help mode on and set help cursor }
```

```
    if GetKeyState(VK_Shift) < 0 then
    begin
      Help := True;
      SetCursor(HelpCursor);
      MainWndProc := DefWindowProc(Wnd, Message, WParam, LParam);
    end

    { If F1 without shift, call up help main index topic }

    else
      WinHelp(Wnd, HelpFileName, Help_Index, 0)

    { Escape during help mode: turn help mode off }

  else
    if (WParam = vk_Escape) and Help then
    begin
      Help := False;
      SetCursor(hCursor(GetClassWord(Wnd, GCW_HCursor)));
    end;
```

As long as the user is in Help mode (that is, until the user clicks the mouse or presses *Esc*), Helpex responds to WM_SETCURSOR messages by resetting the cursor to the arrow and question mark combination.

```
WM_SETCURSOR:

  { In help mode it is necessary to reset the cursor
    in response to every WM_SETCURSOR message.  Otherwise,
    by default, Windows will reset the cursor to that
    of the window class.  }

  if Help then
    SetCursor(HelpCursor)
  else
    MainWndProc := DefWindowProc(Wnd, Message, WParam, LParam);

WM_INITMENU:
  if Help then
    SetCursor(HelpCursor)
  else
    MainWndProc := 1;
```

When the user is in *Shift+F1* Help mode and clicks the mouse button, Helpex receives a WM_NCLBUTTONDOWN message if the click is in a nonclient area of the application window. By examining the *wParam* value of this message, the program can determine which context ID to pass to *WinHelp*.

```
WM_NCLBUTTONDOWN:

  { If we are in help mode (Shift+F1) then display
```

```
  context sensitive help for non-client area.  }
if Help then
begin
  case WParam of
    HtCaption: HelpContextId := HelpIdTitleBar;
    HtReduce: HelpContextId := HelpIdMinimizeIcon;
    HtZoom: HelpContextId := HelpIdMaximizeIcon;
    HtSysMenu: HelpContextId := HelpIdSystemMenu;
    HtBottom: HelpContextId := HelpIdSizingBorder;
    HtBottomLeft: HelpContextId := HelpIdSizingBorder;
    HtBottomRight: HelpContextId := HelpIdSizingBorder;
    HtTop: HelpContextId := HelpIdSizingBorder;
    HtLeft: HelpContextId := HelpIdSizingBorder;
    HtRight: HelpContextId := HelpIdSizingBorder;
    HtTopLeft: HelpContextId := HelpIdSizingBorder;
    HtTopRight: HelpContextId := HelpIdSizingBorder;
  else
    HelpContextId := 0;
  end;
  if HelpContextId = 0 then
    MainWndProc := DefWindowProc(Wnd, Message, WParam, LParam)
  else
  begin
    Help := False;
    WinHelp(Wnd, HelpFileName, Help_Context, HelpContextId);
  end
end
else
  MainWndProc := DefWindowProc(Wnd, Message, WParam, LParam);
```

## F1 support

Context-sensitive *F1* support for menus is relatively easy to implement in your application. If a menu is open and the user presses *F1* while one of the menu items is highlighted, Windows sends a WM_ENTERIDLE message to the application to indicate that the system is going back into an idle state after having determined that *F1* was not a valid keystroke for choosing a menu item. You can take advantage of this idle state by looking at the keyboard state at the time of the WM_ENTERIDLE message.

If the *F1* key is down, then you can simulate the user's pressing *Enter* by posting a WM_KEYDOWN message using VK_RETURN. You don't really want your application to execute the menu command. What you do is set a flag (Help=TRUE) so that when you get the WM_COMMAND message for the menu item, you

don't execute the command. Instead, the topic for the menu item is displayed by Windows Help.

The following code samples illustrate *F1* sensing for menu items.

```
WM_ENTERIDLE:
  if ((WParam = msgf_Menu) and ((GetKeyState(VK_F1) and $8000) <> 0))
  then
     begin
        Help := True;
        PostMessage(Wnd, WM_KEYDOWN, VK_RETURN, 0);
  end;

WM_COMMAND:
      { Was F1 just pressed in a menu, or are we in help mode
        Shift+F1?  }
      if Help then
      begin
        case WParam of
          IdmNew: HelpContextId := HelpIdFileNew;
          IdmOpen: HelpContextId := HelpIdFileOpen;
          IdmSave: HelpContextId := HelpIdFileSave;
          IdmSaveAs: HelpContextId := HelpIdFileSaveAs;
          IdmPrint: HelpContextId := HelpIdFilePrint;
          IdmExit: HelpContextId := HelpIdFileExit;
          IdmUndo: HelpContextId := HelpIdEditUndo;
          IdmCut: HelpContextId := HelpIdEditCut;
          IdmClear: HelpContextId := HelpIdEditClear;
          IdmCopy: HelpContextId := HelpIdEditCopy;
          IdmPaste: HelpContextId := HelpIdEditPaste;
        else
          HelpContextId := 0;
        end;

        if HelpContextId = 0 then
        begin
          MessageBox(Wnd, 'Help not available for Help Menu Item',
            'Help Example', Mb_Ok);
          MainWndProc := DefWindowProc(Wnd, Message, WParam, LParam);
        end
        else
        begin
          Help := False;
          WinHelp(Wnd, HelpFileName, Help_Context, HelpContextId);
        end
```

Detecting *F1* in dialog boxes is somewhat more difficult than in menus. You must install a message filter, using the WH_MSGFILTER option of the **SetWindowsHook** function. Your message filter function responds to WM_KEYDOWN and

WM_KEYUP messages for VK_F1 when they are sent to a dialog box, as indicated by the MSGF_DIALOGBOX code. By examining the message structure passed to the filter, you can determine the context of the *F1* help—what the dialog box is, and the specific option or item. You should not call *WinHelp* while processing the filtered message, but rather post an application-defined message to your application to call *WinHelp* at the first available opportunity.

### Getting help on items on the Help menu

Sometimes users may want information about a general concept in the application rather than about a particular control or window. In these cases, the application should provide Help for a particular topic that is identified by a keyword rather than a context identifier.

For example, if the Help file for your application contains a topic that describes how the keyboard is used, you could place a "Keyboard" item in your Help menu. Then, when the user selects that item, your application calls *WinHelp* and requests that topic:

```
IdmHelpKeyBoard:
  WinHelp(Wnd, HelpFileName, Help_Key, LongInt(PChar('keys')));
```

### Accessing additional keyword tables

Your application may have commands or terms that correspond to terms in a similar, but different, application. Given a keyword, the application can call *WinHelp* and look up topics defined in an alternate keyword table. This multikey functionality is accessed through the *WinHelp* hook with the *Command* parameter set to HELP_MULTIKEY.

You specify the footnote character for the alternate keyword table, and the keyword or phrase, via a **TMultiKeyHelp** record which is passed as the *Data* parameter in the call to *WinHelp*. This record is defined in WINTYPES.PAS as:

```
type
  TMultiKeyHelp = record
    mkSize: Word;
    mkKeyList: Byte;
    szKeyPhrase: array[0..0] of Byte;
  end;
```

The following table lists the format of the fields of the **TMultiKeyHelp** record:

Table 6.16
TMultiKeyHelp record format

| Parameter | Format |
|---|---|
| **mkSize** | The size of the structure, including the keyword (or phrase) and the associated key-table letter. |
| **mkKeyList** | A single character that defines the footnote character for the alternate keyword table to be searched. |
| **szKeyPhrase** | A null-terminated keyword or phrase to be looked up in the alternate keyword table. |

The following example illustrates a keyword search for the word "frame" in the alternate keyword table designated with the footnote character "L":

```
uses Wintypes;
var pmk: ^TMultikeyHelp;
    Keyword: PChar;
begin
  Keyword:= 'frame';
  GetMem(PMK, Sizeof(TMultiKeyHelp) + StrLen(KeyWord));
  with PMK^ do
  begin
    mkSize := Sizeof(TMultiKeyHelp) + StrLen(KeyWord));
    mkKeyList := ord('L');
    StrCopy(@szKeyPhrase, KeyWord);
  end;
  WinHelp(hWindow, HelpFileName, Help_MultiKey, PMK)
  FreeMem(PMK, PMK^.mkSize);
end.
```

## Canceling Help

The Windows Help application is a shared resource that is available to all Windows applications. In addition, since it is a standalone application, the user can execute it like any other application. As a result, your application has limited control over the Help application. While your application can't directly close the Help application window, your application can inform the Help application that Help is no longer needed. Before closing its main window, your application should call *WinHelp* with the *Command* parameter set to HELP_QUIT, as shown in the following example, to inform the Help application that your application won't need it again.

```
WM_DESTROY:
  begin
    WinHelp(Wnd, HelpFileName, HELP_QUIT, 0);
    PostQuitMessage(0);
  end
```

An application that has called *WinHelp* at some point during its execution must call *WinHelp* with the *Command* parameter set to HELP_QUIT before the application exits from WinMain (typically during the WM_DESTROY message processing).

If an application opens more than one Help file, then it must call *WinHelp* to quit help for each file.

If an application or DLL has opened a Help file but no longer wants the associated instance of the Help engine (WINHELP.EXE) to remain active, then the application or DLL should call *WinHelp* with the *Command* parameter set to HELP_QUIT to destroy the instance of the Help engine.

Under no circumstances should an application or DLL terminate without calling *WinHelp* for any of the opened Help files. A Help file is opened if any other *WinHelp* call has been previously made using the Help file name.

The Windows Help application does not exit until all windows that have called *WinHelp* have called it with *Command* set to HELP_QUIT. If an application fails to do so, then the Help application continues running after all applications that requested Help have terminated.

# Help examples

This section contains some examples of Help source files and their corresponding topics as displayed in Help. Each example shows a topic (or part of a topic) as it appears to the Help writer in the RTF-capable word processor and as it appears to the user in the Help window. You can use these examples as guides when creating your own topic files. The examples should help you predict how a particular topic file created in a word processor will appear to the user.

Figure 6.9
Word for Windows topic

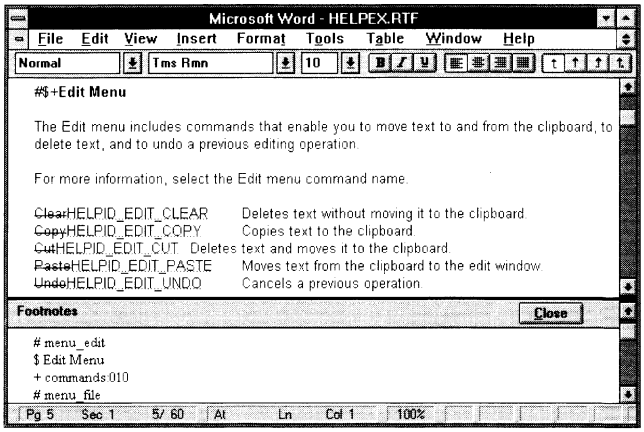



Figure 6.10
Help topic display

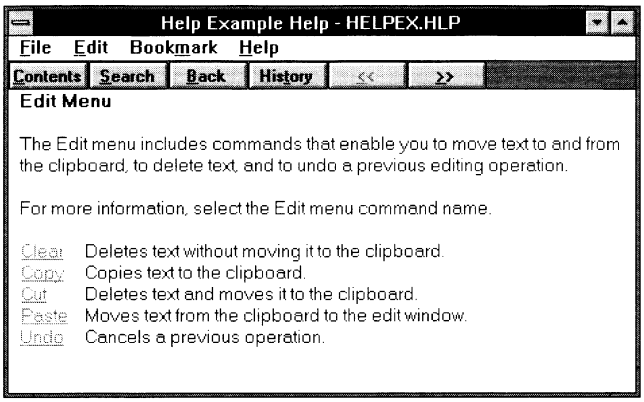



Figure 6.11
Bitmap by reference in topic

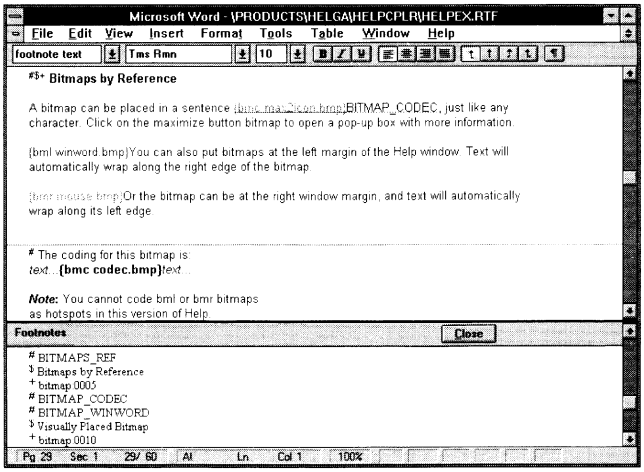
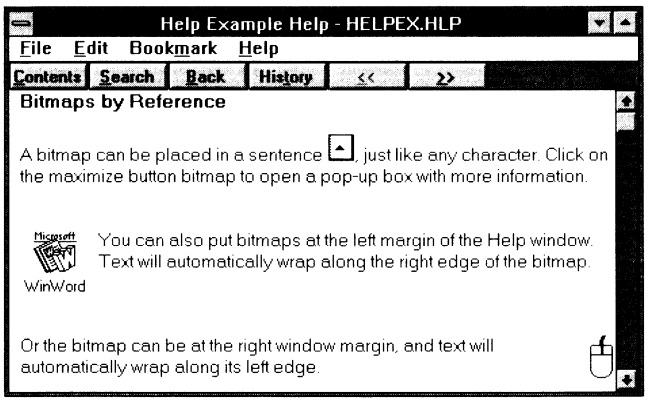


Figure 6.12
Help topic display




## The Helpex project file

The following is the Helpex (sample Help) project file. If you compile this file with the HC31 Help Compiler, change the statement,

```
INDEX=main_index
```

to

```
CONTENT=main_index
```

If you don't change this statement, HC31 displays the error message,

```
2511 unrecognized option 'Index' in [OPTIONS] section
```

Although the file compiles in spite of this warning, the first screen in the first file is displayed as your content screen.

```
[OPTIONS]
INDEX=main_index
TITLE=Help Example
COMPRESS=true
WARNING=1  ; If you remove this line, then HC will report the
following
           ; error message:
           ;
           ;    R1023: Keyword(s) defined without title in page 59
           ;           of file helpex.rtf
           ;
           ; This message indicates that a topic will be listed in
           ; the Search Topics Found box as ">>Untitled Topic<<"

[FILES]
```

```
helpex.rtf ; main topics
keys.rtf   ; keyboard topics
terms.rtf  ; look-up terms

[BITMAPS]
bullet.bmp
chkboff.bmp
chkbon.bmp
codec.bmp
continue.bmp
done.bmp
helpicon.bmp
max2icon.bmp
mouse.bmp
winword.bmp

[MAP]
#include <HELPIDS.H>
```

The following is the helpids.h file:

```
#define HELPID_EDIT_CLEAR     100
#define HELPID_EDIT_COPY      101
#define HELPID_EDIT_CUT       102
#define HELPID_EDIT_PASTE     103
#define HELPID_EDIT_UNDO      104
#define HELPID_FILE_EXIT      200
#define HELPID_FILE_NEW       201
#define HELPID_FILE_OPEN      202
#define HELPID_FILE_PRINT     203
#define HELPID_FILE_SAVE      204
#define HELPID_FILE_SAVE_AS   205
#define HELPID_EDIT_WINDOW    300
#define HELPID_MAXIMIZE_ICON  301
#define HELPID_MINIMIZE_ICON  302
#define HELPID_SYSTEM_MENU    305
#define HELPID_TITLE_BAR      306
#define HELPID_SIZING_BORDER  307
```

C H A P T E R

# 7

# Turbo Help viewer

THELP.COM is a RAM-resident (TSR) utility that accesses Borland Pascal's online Help information for you when you aren't using the IDE. If you are using an editor other than the one in the IDE, or you are using the command-line version of Borland Pascal, or if you are using another product, such as Turbo Debugger, you can use THELP to obtain online help with the press of a single key. THELP requires about 21K bytes of memory.

## Loading and invoking THELP

***Warning!*** *If you are going to have THELP resident in memory at the same time as SideKick 1.x or SideKick Plus, make sure you load THELP before you load SideKick*

You need to first load THELP in order to use it from within another program or from the command line. Make sure that the .TPH file containing the online help information is in the current directory. If you want to keep the .TPH file in another directory, use the **/F** command-line option to tell THELP where to find the .TPH file. The Borland Pascal INSTALL program copies THELP into the \BIN directory and creates a configuration file that contains the correct path information. To load THELP, just type

```
THELP [options]
```

at the DOS command line before you go into your application. This needs to be done only once when you first boot up.

Once you are in the other application, you can activate THELP at any time. Just position the cursor under the item you want information on, then press the THELP hot key. The default hot

key is 5 on the numeric keypad (that is, scan code 4ch, shift state 00h). See page 135 for information on how to reassign this hot key.

# Navigating THELP

Table 7.1 lists the keys you can use to navigate through the screens THELP displays on your monitor:

Table 7.1
THelp keys

| Use these keys | To do this |
|---|---|
| Arrow keys | Moves the highlight from keyword to keyword within the current Help screen. |
| *Shift*+Arrow key | Moves the cursor while marking a block. |
| *Home* and *End* | Moves to the beginning and end of a line, respectively. |
| *Tab* and *Shift+Tab* | Moves to the next or previous keyword. |
| *PgUp/PgDn* | Moves from screen to screen if additional screens are available. |
| *Enter* | Selects a Help entry for the item highlighted in the current Help screen. |
| *Esc* | Ends Help session. |
| *F1* | Displays the Help Table of Contents screen. |
| *Shift+F1* | Displays the Help Index. You can search for a specific keyword incrementally. For example, you can find print by typing p r i. With each letter you type, the list jumps to the keyword that starts with p, then to pr, then to pri, and so on. |
| *Alt+F1* | Pressing *Alt+F1* repeatedly takes you in reverse order through the last 20 screens you have reviewed. |
| *Alt+F* | Selects a new Help file if you have specified more than one help file in the THELP.CFG file or on the command line. |
| *Ctrl+P* | Pastes the marked block or example text into your current application. |

# THELP options

Table 7.2 summarizes the THELP command-line options. If you use more than one option, separate them with spaces. You can create a configuration file that contains the default command-line options you want to be loaded each time you activate THELP. This configuration file, called THELP.CFG for convenience, must be located in the same directory as THELP.COM. Each option in the configuration file must be placed on its own line and begin in the leftmost column.

Table 7.2
THelp options

| Use this option | To do this |
|---|---|
| /**C**#*xx* | Select color where<br># = color number and<br>*xx* = hex color values |
| /**F***name* | Specify full path and file name of Help file |
| /**H**, /?, ? | Display help screen |
| /**K***xxyy* | Change hot key where<br>*xx* = shift state (hex) and<br>*yy* = scan code (hex) |
| /**U** | Remove THELP from memory |
| /**W***x,y,w,h* | Set the window size and location |

## /C#xx (select color)

This option lets you customize the background and foreground colors of various elements in a help screen. Follow the /**C** option with the number of the color you want plus the hex color values for background and foreground, respectively.

For example, the command

```
THELP /C075
```

sets a border with a gray background and magenta text.

The twelve possible colors are numbered as follows:

Table 7.3
Color options

| Number | Element |
|---|---|
| 0 | Color border attribute |
| 1 | Monochrome border attribute |
| 2 | Color text attribute |
| 3 | Monochrome text attribute |
| 4 | Color keyword attribute |
| 5 | Monochrome keyword attribute |
| 6 | Color selected keyword word attribute |
| 7 | Monochrome selected keyword word |

Table 7.3: Color options (continued)

| | |
|---|---|
| 8 | Color example text attribute |
| 9 | Monochrome example text attribute |
| A | Color marked block attribute |
| B | Monochrome marked block attribute |

**OR**ing the color value with 0x80 produces a blinking color unless blinking has been disabled. The color values for a standard IBM-compatible color display are as follows:

Table 7.4
Color values

| **First digit (background)** | **Second digit (foreground)** |
|---|---|
| 0 Black | 0 Black |
| 1 Blue | 1 Blue |
| 2 Green | 2 Green |
| 3 Cyan | 3 Cyan |
| 4 Red | 4 Red |
| 5 Magenta | 5 Magenta |
| 6 Brown | 6 Brown |
| 7 Gray | 7 Gray |
| | 8 Intense black |
| | 9 Intense blue |
| | A Intense green |
| | B Intense cyan |
| | C Intense red |
| | D Intense magenta |
| | E Intense brown (yellow) |
| | F Intense gray (white) |

On monochrome monitors, the attribute values can differ widely, so you may need to experiment.

## /Fname (full path name)

The name that follows the /**F** option should be the full drive/directory path and name of the Help file to use. For example, the full path and name for TURBO.TPH is

```
THELP /FC:\BP\BIN\TURBO.TPH
```

You can specify up to eight help files on the command-line or in the THELP.CFG file. Even if you use a configuration file to load your help file, you can still use a command-line option to load another help file. This help file will be loaded into memory along with the previously loaded help files. For example to load both TURBO.TPH and TVISION.TPH, type

```
THELP /FC:\BP\BIN\TURBO.TPH /FC:\BP\BIN\TVISION.TPH
```

THELP, however, does not merge the indexes for the multiple help files. THELP displays the index only for the active help file. To view a different index, you need to press *Alt+F* and select a new help file.

## /Kxxyy (reassign hot key)

This option allows you to reassign a function to a new hot key. The option must be followed by the shift state (xx) and the scan code (yy) of the new key. You can select virtually any shift state/scan code combination. For example, to change the THELP hotkey from 5 on the numeric keypad to *Alt+A* or *Alt+a*, type

```
THELP /K081E
```

Table 7.5 lists some of the common shift states. These keys can be **OR**ed together.

Table 7.5
Shift states

| Keys | Code |
|---|---|
| *Shift+→* | 01h |
| *Shift+←* | 02h |
| *Ctrl* | 04h |
| *Alt* | 08h |

Table 7.6 lists the scan codes you can use with THelp.

Table 7.6
Scan codes

| Scan codes: | | | | | | | |
|---|---|---|---|---|---|---|---|
| A | 1eh | N | 31h | 0 | 0bh | F1 | 3bh |
| B | 30h | O | 18h | 1 | 02h | F2 | 3ch |
| C | 2eh | P | 19h | 2 | 03h | F3 | 3dh |
| D | 20h | Q | 10h | 3 | 04h | F4 | 3eh |
| E | 12h | R | 13h | 4 | 05h | F5 | 3fh |
| F | 21h | S | 1fh | 5 | 06h | F6 | 40h |
| G | 22h | T | 14h | 6 | 07h | F7 | 41h |
| H | 23h | U | 16h | 7 | 08h | F8 | 42h |
| I | 17h | V | 2fh | 8 | 09h | F9 | 43h |
| J | 24h | W | 11h | 9 | 0ah | F10 | 44h |
| K | 25h | X | 2dh | | | F11 | 57h* |
| L | 26h | Y | 15h | | | F12 | 58h* |
| M | 32h | Z | 2ch | | | | |

* These codes work only on enhanced keyboards and may not work with all computers or keyboards.

### /H, /?, and ? (display help screen)

Any of these options displays a summary of THELP's command-line options.

### /U (remove THELP from memory)

This option removes THELP from memory. If other TSRs have been loaded after THELP, make sure to remove them before removing THELP.

### /W (set window size and location)

You can use this option (/Wx,y,w,h) to set the size and the location of the window where

x = window column location (zero based)
y = window row location
w = window width
h = window height

For example, to create a full-screen help window, type

```
THELP /W0,0,80,25
```

### Diagnostic messages

If you enter an unrecognized command-line option or an invalid line in the configuration file, THELP displays the message,

```
Unrecognized command-line/configuration-file option[option]
```

where *option* represents the incorrectly typed option. After the message, THELP displays the command-line options so you can check the syntax of the command you entered. One of the more common errors is forgetting to type **/F** and the full path name before each help file listed on the command line.

A P P E N D I X

# A

# Error messages

*For a list of Borland Pascal compiler, run-time library, and DPMI error messages, refer to Chapter 4, "Error messages," in the Programmer's Reference.*

This appendix lists the error messages you might encounter when you are using Borland Pascal utilities. Table A.1 lists the utilities that are documented in this chapter. Whenever possible, the explanation includes a possible cause and a remedy for the error or warning message.

Table A.1
Utility error messages

| Utility | Page |
|---|---|
| TPUMOVER | See page 137. |
| MAKE | See page 139. |
| HC30 | See page 144. |
| HC31 | See page 153. |

## TPUMOVER messages

TPUMOVER displays the following kinds of error messages:

- Unit file format. Unit file format error messages occur if your version of TPUMOVER is incompatible with the run-time library you are using or if the unit is damaged.
- File creation. File-creation error messages occur if TPUMOVER cannot locate the run-time library or the units you want to add to, extract, or delete from the run-time library.

- System. System error messages occur if there is not enough space on the disk or in memory for TPUMOVER to manipulate the run-time library units.

TPUMOVER displays error messages in one of the following formats:

```
Disk full error

Unit file format error (xxxx)
```

where xxxx represents the name of the specified unit file.

**Disk full error**
There is not enough disk space for TPUMOVER to update or extract the run-time library units.

**File creation error (xxxx)**
TPUMOVER cannot add the specified unit to the run-time library file. The file might not exist or the run-time library file might have changed.

**File not found (xxxx)**
TPUMOVER cannot locate the run-time library or specified unit. Make sure that the run-time library file is on your path or in the same directory as TPUMOVER.

**File open error (xxxx)**
The specified run-time library file cannot be opened. If you are running Borland Pascal on a network, make sure that the run-time library file can be opened in write mode.

**Out of memory**
There isn't enough memory to run TPUMOVER. Remove any memory-resident applications that are not absolutely necessary.

**Unit file format error (xxxx)**
TPUMOVER cannot read the unit file. This error can occur if the version of TPUMOVER is incompatible with the run-time library you are using or if the unit has been damaged. As a general rule, if the compiler can use a unit file, the file is valid.

**Unit not found (xxxx)**
TPUMOVER cannot locate the unit to be removed from the run-time library. Make sure that the unit exists and that the path or unit name is typed correctly.

# MAKE messages

MAKE diagnostic messages fall into two classes: errors and fatal errors.

- Errors indicate some sort of syntax or semantic error in the source makefile.
- Fatal errors prevent processing of the makefile. You must take appropriate action and then restart MAKE.

The following generic names and values appear in the messages listed in this section. When you get an error message, the appropriate name or value is substituted.

Table A.2
MAKE error message variables

| What you'll see in the manual | What you'll see on your screen |
|---|---|
| *argument(s)* | An argument (command-line or other) |
| *expression* | An expression |
| *filename* | A file name (with or without extension) |
| *line number* | A line number |
| *message* | A message string |
| *target* | A receiver |

**')' missing in macro invocation in command *command***
A left parenthesis is required to invoke a macro.

**Bad file name format in include statement**
Include file names must be surrounded by quotes or angle brackets. The file name was missing the opening quote or angle bracket.

**Bad macro output translator**
Invalid syntax for substitution within macros. For example:

```
$(MODEL:=s)  or  $(MODEL:)  or  $(MODEL:s)
```

**Bad undef statement syntax**
An **!undef** statement must contain a single identifier and nothing else as the body of the statement.

**Cannot have multiple paths for implicit rule**
You can have only a single path for each of the extensions in an implicit rule. Multiple path lists are allowed only for dependents in an explicit rule. For example:

```
{path1;path2}.asm.obj:    # Invalid
{path}.asm.obj            # Valid
```

**Cannot have path list for target**

You can only specify a path list for dependents of an explicit rule. For example:

```
{path1;path2}prog.exe: prog.obj   # Invalid
prog.exe: {path1;path2}prog.obj   # Valid
```

**Circular dependency exists in makefile**

The makefile indicates that a file needs to be up-to-date *before* it can be built. Take, for example, the explicit rules:

```
filea:  fileb
fileb:  filec
filec:  filea
```

This implies that file*a* depends on file*b*, which depends on file*c*, and file*c* depends on file*a*. This is illegal, since a file cannot depend on itself, indirectly or directly.

**Colon expected**

You have forgotten to put the colon at the end of your implicit rule.

```
.pas.tpu:     # Correct
.pas.tpu      # Incorrect
```

**Command arguments too long**

The arguments to a command were more than the 127-character limit imposed by DOS.

**Command syntax error**

This message occurs if

- The first rule line of the makefile contained any leading whitespace.
- An implicit rule did not consist of *.ext.ext:*.
- An explicit rule did not contain a name before the **:** character.
- A macro definition did not contain a name before the **=** character.

**Command too long**

The length of a command has exceeded 128 characters. You might want to use a response file.

**Division by zero**

A division or remainder operator in an **!if** statement has a zero divisor.

***filename* does not exist – don't know how to make it**
There's a nonexistent file name in the build sequence, and no rule exists that would allow the file name to be built.

**Error directive: *message***
MAKE has processed an **#error** directive in the source file, and the text of the directive is displayed in the message.

**Expression syntax error in !if statement**
The expression in an **!if** statement is badly formed—it contains a mismatched parenthesis, an extra or missing operator, or a missing or extra constant.

**File name too long**
The path name in an **!include** directive overflowed MAKE's internal buffer (512 bytes).

**If statement too long**
**Ifdef statement too long**
**Ifndef statement too long**
An **If**, **Ifdef**, or **Ifndef** statement has exceeded 4,096 characters.

**Illegal character in constant expression *expression***
MAKE encountered a character not allowed in a constant expression. If the character is a letter, this probably indicates a misspelled identifier.

**Int and string types compared**
You have tried to compare an integer operand with a string operand in an **!if** or **!elif** expression.

**Macro expansion too long**
A macro cannot expand to more than 4,096 characters. This error often occurs if a macro recursively expands itself. A macro cannot legally expand to itself.

**Macro substitute text *string* is too long**
**Macro replace text *string* is too long**
The macro substitution or replacement text *string* overflowed MAKE's internal buffer of 512 bytes.

**Misplaced elif statement**
An **!elif** directive is missing a matching **!if** directive.

**Misplaced else statement**
There's an **!else** directive without any matching **!if** directive.

**Misplaced endif statement**
There's an **!endif** directive without any matching **!if** directive.

**No closing quote**
There is no closing quote for a string expression in a **!if** or **!elif** expression.

**No file name ending**
The file name in an **!include** statement is missing the correct closing quote or angle bracket.

**No macro before =**
You must give a macro a name before you can assign it a value.

**No match found for wildcard *expression***
There are no files matching the wildcard *expression* for MAKE to expand. For example, if you write

```
prog.exe: *.obj
```

MAKE sends this error message if there are no files with the extension .OBJ in the current directory.

**No terminator specified for in-line file operator**
The makefile contains either the **&&** or **<<** command-line operators to start an in-line file, but the file is not terminated.

**Not enough memory**
All your working storage has been exhausted. Try simplifying the source file or using a machine with additional memory.

**Only <<KEEP or <<NOKEEP**
You have specified something besides KEEP or NOKEEP when closing a temporary inline file.

**Redefinition of target *filename***
The named file occurs on the left side of more than one explicit rule.

**Rule line too long**
An implicit or explicit rule was longer than 4,096 characters.

**String type not allowed with this operand**
You have tried to use an operand that is not allowed for comparing string types. Valid operands are ==, !=, <, >, <=, and >=.

**Too many suffixes in .SUFFIXES list**
You have exceeded the 255 allowable suffixes in the suffixes list.

**Unable to execute command**
A command failed to execute. This could be because the command file was not be found, the name was misspelled, there was no disk space left in the specified swap directory, or the swap directory does not exist. A less likely possibility is that the command exists but has been corrupted.

**Unable to open makefile**
The current directory does not contain a file named MAKEFILE, MAKEFILE.MAK, or does not contain the file you specified with **–f**.

**Unable to redirect input or output**
MAKE was unable to open the temporary files necessary to redirect input or output. If you are on a network, make sure you have rights to the current directory.

**Unexpected end of file**
The end of the makefile was reached without a temporary inline file having been closed.

**Unexpected end of file in conditional started on line *line number***
The source file ended before the compiler (or MAKE) encountered an **!endif**. The **!endif** was either missing or misspelled.

**Unknown preprocessor statement**
A ! character was encountered at the beginning of a line, and the statement name following was not **error**, **undef**, **if**, **elif**, **include**, **else**, or **endif**.

**Use of : and :: dependants for target *target***
You have tried to use the target in both single and multiple description blocks (using both the **:** and **::** operators). Examples:

```
filea: fileb
filea:: filec
```

# Help compiler messages

The Help Compiler displays messages when it encounters errors or warnings in building the Help resource file. Warnings indicate a problem during compilation that were not severe enough to prevent the Help file from being created. Help should be able to open the file but may encounter problems when displaying some

topics. A fatal error indicates a problem that prevents the compiler from creating a Help file. The compiler always reports fatal errors, regardless of the current warning level or reporting option.

While the Help Compiler processes the project file, it ignores lines that contain errors and attempts to continue. This means that errors ENCOUNTERED early in the file may result in many more errors being reported as the compiler continues.

When the Help Compiler processes topic files, it reports any errors it encounters and, if the errors are not fatal, compilation continues. A single error in a topic file may result in more than one error message being displayed by the compiler. For example, a typographical mistake in a topic's context string will cause an error to be reported every time the compiler encounters a reference to the correct topic identifier.

The error messages in this section are divided into those generated by the HC30 Help compiler and those generated by the HC31 Help Compiler.

## HC30 Help compiler messages

These errors are grouped according to project error messages and run-time (RTF) error messages.

Table A.3
Help message variables

| What you'll see in the manual | What you'll see on your screen |
|---|---|
| *contextname* | Context string alias |
| *context-string* | A context string |
| *filename* | A file name (with or without extension) |
| *fontname* | The name of a font |
| *optionname* | An option name |
| *sectionname* | A section heading |
| *tagname* | A build tag |
| *topicnumber* | A topic number |

### Project messages

Project error messages occur when the compiler encounters an error while processing the project file.

**P1003 Invalid path specified in Root option**
The path specified by the Root option cannot be found. The compiler uses the current working directory.

**P1005 Path and filename exceed limit of 79 characters**
The absolute path name, or the combined root and relative path name, exceed the DOS limit of 79 characters. The file is skipped.

**P1007 Root path exceeds maximum limit of 66 characters**
The specified root path name exceeds the DOS limit of 66 characters. The path name is ignored and the compiler uses the current working directory.

**P1009 [FILES] section missing**
The [Files] section is required. The compilation is aborted.

**P1011 Option *optionname* previously defined**
The specified option was defined previously. The compiler ignores the attempted redefinition.

**P1013 Project file extension cannot be .HLP**
You cannot specify that the compiler use a project file with the .HLP extension. Normally, project files are given the .HPJ extension.

**P1015 Unexpected end-of-file**
The compiler has unexpectedly come to the end of the project file. There might be an open comment in the project or included file.

**P1017 Parameter exceeds maximum length of 128 characters**
An option, context name or number, build tag, or other parameter on the specified line exceeds the limit of 128 characters. The line is ignored.

**P1021 Context number already used in [MAP] section**
The context number on the specified line in the project file was previously mapped to a different context string. The line is ignored.

**P1023 Include statements nested too deeply**
The **#include** statement on the specified line has exceeded the maximum of five include levels.

**P1025 Section heading *sectionname* unrecognized**
A section heading that is not supported by the compiler has been used. The line is skipped.

**P1027 Bracket missing from section heading sectionname**
The right bracket (]) is missing from the specified section heading. Insert the bracket and compile again.

**P1029 Section heading missing**
The section heading on the specified line is not complete. This error is also reported if the first entry in the project file is not a section heading. The compiler continues with the next line.

**P1030 Section *sectionname* previously defined**
A duplicate section has been found in the project file. The lines under the duplicated section heading are ignored and the compiler continues from the next valid section heading.

**P1031 Maximum number of build tags exceeded**
The maximum number of build tags that can be defined is 30. The excess tags are ignored.

**P1033 Duplicate build tag in [BUILDTAGS] section**
A build tag in the [BUILDTAGS] section has been repeated unnecessarily.

**P1035 Build tag length exceeds maximum**
The build tag on the specified line exceeds the maximum of 32 characters. The compiler skips this entry.

**P1037 Build tag *tagname* contains invalid characters**
Build tags can contain only alphanumeric characters or the underscore (_) character. The line is skipped.

**P1039 [BUILDTAGS] section missing**
The **BUILD** option declared a conditional build, but there is no [BuildTags] section in the project file. All topics are included in the build.

**P1043 Too many tags in Build expression**
The Build expression on the specified line has used more than the maximum of 20 build tags. The compiler ignores the line.

**P1045 [ALIAS] section found after [MAP] section**
When used, the [Alias] section must precede the [Map] section in the project file. The [Alias] section is skipped otherwise.

**P1047 Context string *contextname* already assigned an alias**
Because a context string can only have one alias, you cannot type a=b then a=c<_>. The specified context string has previously been aliased in the [Alias] section. The attempted reassignment on this line is ignored.

**P1049 Alias string aliasname already assigned**
You cannot do: a=b then b=c. An alias string cannot, in turn, be assigned another alias.

**P1051 Context string *contextname* cannot be used as alias string**
You cannot do: a=b then c=a. A context string that has been assigned an alias cannot be used later as an alias for another context string.

**P1053 Maximum number of font ranges exceeded**
The maximum number of font ranges that can be specified is five. The rest are ignored.

**P1055 Current font range overlaps previously defined range**
A font size range overlaps a previously defined mapping. Adjust either font range to remove any overlaps. The second mapping is ignored.

**P1056 Unrecognized font name in Forcefont option**
A font name not supported by the compiler has been encountered. The font name is ignored and the compiler uses the default Helvetica font.

**P1057 Font name too long**
Font names cannot exceed 20 characters. The font is ignored.

**P1059 Invalid multiple-key syntax**
The syntax used with a **MULTIKEY** option is unrecognized. See Chapter 6, "Windows Help compilers," for the proper syntax.

**P1061 Character already used**
The specified keyword-table identifier is already in use. Choose another character.

**P1063 Characters 'K' and 'k' cannot be used**
These characters are reserved for Help's normal keyword table. Choose another character.

**P1065 Maximum number of keyword tables exceeded**
The limit of five keyword tables has been exceeded. Reduce the number. The excess tables are ignored.

**P1067 Equal sign missing**
An option is missing its required equal sign on the specified line. Check the syntax for the option.

**P1069 Context string missing**
The line specified is missing a context string before an equal sign.

**P1071 Incomplete line in *sectionname* section**
The entry on the specified line is not complete. The line is skipped.

**P1073 Unrecognized option in [OPTIONS] section**
An option has been used that is not supported by the compiler. The line is skipped.

**P1075 Invalid build expression**
The syntax used in the build expression on the specified line contains one or more logical or syntax errors.

**P1077 Warning level must be 1, 2, or 3**
The **WARNING** reporting level can only be set to 1, 2, or 3. The compiler will default to full reporting (level 3).

**P1079 Invalid compression option**
The **COMPRESS** option can only be set to TRUE or FALSE. The compilation continues without compression.

**P1081 Invalid title string**
The **TITLE** option defines a string that is null or contains more than 32 characters. The title is truncated.

**P1083 Invalid context identification number**
The context number on the specified line is null or contains invalid characters.

**P1085 Unrecognized text**
The unrecognizable text that follows valid text in the specified line is ignored.

**P1086 Invalid font-range syntax**
The font-range definition on the specified line contains invalid syntax. The compiler ignores the line. Check the syntax for the **MAPFONTSIZE** option.

**P1089 Unrecognized sort ordering**
You have specified an ordering that is not supported by the compiler. Contact Borland Technical Support for clarification of the error.

## Help RTF messages

Run-time errors result from problems in trying to read or write RTF files.

**R2001 Unable to open bitmap file *filename***
The specified bitmap file is unreadable. This is a DOS file error.

**R2003 Unable to include bitmap file *filename***
The specified bitmap file could not be found or is unreadable. This is a DOS file error or an out-of-memory condition.

**R2005 Disk full**
The Help resource file could not be written to disk. Create more space on the destination drive.

**R2009 Cannot use reserved DOS device name for file *filename***
A file has been referred to as COM1, LPT2, PRN, etc. Rename the file.

**R2013 Output file *filename* already exists as a directory**
There is a subdirectory in the Help project root with the same name as the desired Help resource file. Move or rename the subdirectory.

**R2015 Output file *filename* already exists as read-only**
The specified file name cannot be overwritten by the Help resource file because the file has a read-only attribute. Rename the project file or change the file's attribute.

**R2017 Path for file *filename* exceeds limit of 79 characters**
The absolute path name, or the combined root and relative path name, to the specified file exceed the DOS limit of 79 characters. The file is ignored.

**R2019 Cannot open file *filename***
The specified file is unreadable. This is a DOS file error.

**R2021 Cannot find file *filename***
The specified file could not be found or is unreadable. This is a DOS file error or an out-of-memory condition.

**R2023 Not enough memory to build Help file**
To free up memory, unload any unneeded applications, device drivers, and memory-resident programs.

**R2025 File environment error**
The compiler has insufficient available file handles to continue. Increase the values for FILES= and BUFFERS= in your CONFIG.SYS file and reboot.

**R2027 Build tag *tagname* not defined in [BUILDTAGS] section of project file**
The specified build tag has been assigned to a topic, but not declared in the project file. The tag is ignored for the topic.

**R2033 Context string in Map section not defined in any topic**
The compiler could not find topics for are one or more context strings defined in the project file.

**R2035 Build expression missing from project file**
The topics have build tags, but there is no Build= expression in the project file. The compiler includes all topics in the build.

**R2037 File *filename* cannot be created, due to previous error(s)**
The Help resource file could not be created because the compiler has no topics remaining to be processed. Correct the errors that preceded this error and recompile.

**R2039 Unrecognized table formatting in topic *topicnumber* of file *filename***
The compiler ignores table formatting that is unsupported in Help. Reformat the entries as linear text if possible.

**R2041 Jump *context_string* unresolved in topic *topicnumber* of file *filename***
The specified topic contains a context string that identifies a nonexistent topic. Check spelling, and that the desired topic is included in the build.

**R2043 Hotspot text cannot spread over paragraphs**
A jump term spans two paragraphs. Remove the formatting from the paragraph mark.

**R2045 Maximum number of tab stops reached in topic *topicnumber* of file *filename***
The limit of 32 tab stops has been exceeded in the specified topic. The default stops are used after the 32nd tab.

**R2047 File *filename* not created**
There are no topics to compile, or the build expression is false for all topics. There is no Help resource file created.

**R2049 Context string text too long in topic *topicnumber* of file *filename***
Context string hidden text cannot exceed 64 characters. The string is ignored.

**R2051 File *filename* is not a valid RTF topic file**
The specified file is not an RTF file. Check that you have saved the topic as RTF from your word processor.

**R2053 Font *fontname* in file *filename* not in RTF font table**
A font not defined in the RTF header has been entered into the topic. The compiler uses the default system font.

**R2055 File *filename* is not a usable RTF topic file**
The specified file contains a valid RTF header, but the content is not RTF or is corrupted.

**R2057 Unrecognized graphic format in topic *topicnumber* of file *filename***

The compiler supports only Windows bitmaps. Check that metafiles or Macintosh formats have not been used. The graphic is ignored.

**R2059 Context string identifier already defined in topic *topicnumber* of file *filename***

There is more than one context-string identifier footnote for the specified topic. The compiler uses the identifier defined in the first # footnote.

**R2061 Context string *contextname* already used in file *filename***

The specified context string was previously assigned to another topic. The compiler ignores the latter string and the topic has no identifier.

**R2063 Invalid context-string identifier for topic *topicnumber* of file *filename***

The context-string footnote contains nonalphanumeric characters or is null. The topic is not assigned an identifier.

**R2065 Context string defined for index topic is unresolved**

The index topic defined in the project file could not be found. The compiler uses the first topic in the build as the index.

**R2067 Footnote text too long in topic *topicnumber* of file *filename***

Footnote text cannot exceed the limit of 1000 characters. The footnote is ignored.

**R2069 Build tag footnote not at beginning of topic *topicnumber* of file *filename***

The specified topic contains a build tag footnote that is not the first character in the topic. The topic is not assigned a build tag.

**R2071 Footnote text missing in topic *topicnumber* of file *filename***

The specified topic contains a footnote that has no characters.

**R2073 Keyword string is null in topic *topicnumber* of file *filename***

A keyword footnote exists for the specified topic, but contains no characters.

**R2075 Keyword string too long in topic *topicnumber* of file *filename***

The text in the keyword footnote in the specified topic exceeds the limit of 255 characters. The excess characters are ignored.

**R2077 Keyword(s) defined without title in topic *topicnumber* of file *filename***

Keywords have been defined for the specified topic, but the topic has no title assigned. Search Topics Found displays Untitled Topic<< for the topic.

**R2079 Browse sequence string is null in topic *topicnumber* of file *filename***

The browse-sequence footnote for the specified topic contains no sequence characters.

**R2081 Browse sequence string too long in topic *topicnumber* of file *filename***

The browse-sequence footnote for the specified topic exceeds the limit of 128 characters. The sequence is ignored.

**R2083 Missing sequence number in topic *topicnumber* of file *filename***

A browse-sequence number ends in a colon (:) for the specified topic. Remove the colon, or enter a "minor" sequence number.

**R2085 Sequence number already defined in topic *topicnumber* of file *filename***

There is already a browse-sequence footnote for the specified topic. The latter sequence is ignored.

**R2087 Build tag too long**

A build tag for the specified topic exceeds the maximum of 32 characters. The tag is ignored for the topic.

**R2089 Title string null in topic *topicnumber* of file *filename***

The title footnote for the specified topic contains no characters. The topic is not assigned a title.

**R2091 Title too long in topic *topicnumber* of file *filename***

The title for the specified topic exceeds the limit of 128 characters. The excess characters are ignored.

**R2093 Title titlename in topic *topicnumber* of file *filename* used previously**

The specified title has previously been assigned to another topic.

**R2095 Title defined more than once in topic *topicnumber* of file *filename***

There is more than one title footnote in the specified topic. The compiler uses the first title string.

**R2501 Using old key-phrase table**
Maximum compression can result only by deleting the .PH file before each recompilation of the Help topics.

**R2503 Out of memory during text compression**
The compiler encountered a memory limitation during compression. Compilation continues with the Help resource file not compressed. Unload any unneeded applications, device drivers, and memory-resident programs.

**R2505 File environment error during text compression**
The compiler has insufficient available file handles for compression. Compilation continues with the Help resource file not compressed. Increase the values for FILES= and BUFFERS= in your CONFIG.SYS file and reboot.

**R2507 DOS file error during text compression**
The compiler encountered a problem accessing a disk file during compression. Compilation continues with the Help resource file not compressed.

**R2509 Error during text compression**
One of the three compression errors—R2503, R2505, or R2507—has occurred. Compilation continues with the Help resource file not compressed.

**R2701 Internal error**
**R2703 Internal error**
**R2705 Internal error**
**R2707 Internal error**
**R2709 Internal error**
Contact Borland Technical Support for clarification of the error.

## HC31 Help compiler messages

HC31 Help compiler error message numbers have four digits; the first one or two of those digits identifies the message category. The message number prefixes and the categories they identify are defined as follows:

Table A.4
Help message categories

| Prefix | Error |
|---|---|
| 1 | Problems with files used to build the help file |
| 2 | Problems with the project file |
| 30–31 | Problems with build tags or build-tag expressions |
| 35–36 | Problems with Help macros |
| 40–41 | Problems with context strings |
| 42–45 | Problems with footnotes |

Table A.4: Help message categories (continued)

| | |
|---|---|
| 46–47 | Problems with topic file |
| 5 | Other problems |

This section lists the HC31 error messages in numerical order within each of these eight categories.

## File Errors

The following messages result from problems with files used to build a Help file.

**1019 Project file extension cannot be .HLP or .PH**

**1030 File name exceeds limit of 259 characters**
The combined length of the path and file name must not be more than the MS-DOS limit of 259 characters.

**1079 Out of file handles**
The compiler does not have enough available file handles to continue. If possible, increase the FILES setting in the CONFIG.SYS file.

**1100 Cannot open file *filename*: permission denied**
Requested files must have at least read privilege to be opened.

**1150 Cannot overwrite file *filename***
Files with read-only attribute cannot be overwritten.

**1190 Cannot use reserved DOS file name *filename***
Do not use reserved MS-DOS file names, such as COM1, LPT2, or PRN, when specifying topic or other data files.

**1230 File *filename* not found**
The specified file could not be found or is unreadable.

**1292 File *filename* is not a valid bitmap**
The specified bitmap file could not be found or is unreadable.

**1319 Disk full**

**1513 Bitmap name *filename* duplicated**

**1536 Not enough memory to compress bitmap *filename***
The specified bitmaps cannot be compressed due to insufficient memory.

## Project File Errors

The following messages result from errors in the Help project file (with the .HPJ file-name extension) used to build a Help file.

**2010 Include statements nested more than 5 deep**
The **#include** statement on the specified line has exceeded the maximum of five include levels.

**2030 Comment starting at line *linenumber* of file *filename* unclosed at end of file**
The compiler has unexpectedly come to the end of the project file. There may be an open comment in the project file or in an include file.

**2050 Invalid #include syntax**
The **#include** statement requires a file name.

**2091 Bracket missing from section heading *sectionname***

**2111 Section heading missing**
The section heading on the specified line is not complete. This error is also reported if the first entry in the project file is not a section heading.

**2131 Invalid OPTIONS syntax: 'option=value' expected**

**2141 Invalid ALIAS syntax: 'context=context' expected**

**2151 Incomplete line in *sectionname* section**

**2171 Unrecognized text**

**2191 Section heading *sectionname* unrecognized**

**2214 LIne in .HPJ file exceeds length limit of 2057 characters**

**2273 OPTIONS should precede FILES and BITMAPS for all options to take effect**
The OPTIONS section should be the first section in the project file. Also, if the ERRORLOG option is used, that option should be the first line in the OPTIONS section.

**2291 Section *sectionname* previously defined**
The compiler ignores the lines under the duplicated section and continues from the next valid section heading.

**2305 No valid files in FILES section**
The file section is empty or contains only invalid files.

**2322 Context string *context_name* cannot be used as alias string**
A context string that has been assigned an alias cannot be used later as an alias for another context string. That is, you cannot map a=b and then c=a in the ALIAS section. The compiler ignores the attempted reassignment.

**2331 Context number already used in MAP section**
The context number on the specified line in the project file was previously mapped to a different context string.

**2341 Invalid or missing context string**
The specified line is missing a context string before an equal sign.

**2362 Context string *context-name* already assigned an alias**
A context string can have only one alias. That is, you cannot map a=b and then a=c in the ALIAS section. The specified context string has already been assigned an alias in the ALIAS section. The compiler ignores the attempted reassignment.

**2372 Alias string *aliasname* already assigned**
You cannot alias an alias. That is, an alias string cannot, in turn, be assigned another alias. You cannot map **a=b** and then **b=c** in the ALIAS section. The compiler ignores the attempted reassignment.

**2391 Limit of 6 window definitions exceeded**
The maximum number of window definitions is one main-window definition and five secondary-window definitions.

**2401 Window maximization state must be 0 or 1**
The *sizing* parameter in a window definition must be either 0 or 1.

**2411 Invalid syntax in window color**
A window color in a window definition consists of three decimal numbers enclosed in parentheses and separated by commas.

**2421 Invalid window position**
The window position in a window definition consists of four decimal numbers enclosed in parentheses and separated by commas.

**2431 Missing quote in window caption**
The window caption in a window definition must be enclosed in quotation marks.

**2441 Window name *windowname* is too long**
The window name exceeds the maximum length of 8 characters.

**2451 Window position value out of range 0...1023**
One or more of the window-position coordinates exceed the maximum limit of 1023.

**2461 Window name missing**
A window definition in the project file is missing the window name.

**2471 Invalid syntax in WINDOWS section**

**2481 Secondary-window position required**
A window definition for a secondary window must specify the four window-position parameters.

**2491 Duplicate window name *windowname***
Window names must be unique.

**2501 Window caption *windowname* exceeds limit of 50 characters**

**2511 Unrecognized option *optionname* in OPTIONS section**

**2532 Option *optionname* previously defined**
The compiler ignores the attempted redefinition.

**2550 Invalid path *pathname* in *optionname* option**
The compiler cannot find the path specified by the **ROOT** or **BMROOT** option. The compiler uses the current working directory.

**2570 Path in *optionname* option exceeds *number of* characters**
The specified root path exceeds the maximum limit for MS-DOS. The compiler ignores the path and uses the current working directory.

**2591 Invalid MAPFONTSIZE option**
The font range syntax used is invalid. A font range consists of a low and high point size, separated by a hyphen (-).

**2612 Maximum of 5 font ranges exceeded**
The compiler ignores additional ranges.

**2632 Current font range overlaps previously defined range**
The compiler ignores the second mapping.

**2651 Font name exceeds limit of 20 characters**

**2672 Unrecognized font name *fontname* in FORCEFONT option.**
The compiler ignores the font name and uses the default Helvetica font.

**2691 Invalid MULTIKEY syntax**
The **MULTIKEY** option must specify a single capital letter other than the letter *K*.

**2711 Maximum of 5 keyword tables exceeded**
The compiler ignores the additional tables.

**2732 Character already used**
A character used for indicating the keyword table was previously used. The compiler ignores the line.

**2752 Characters 'K' and 'k' cannot be used**
These characters are reserved for Help's standard keyword table.

**2771 REPORT option must be 'ON' or 'OFF'**

**2811 OLDKEYPHRASE option must be 'ON' or 'OFF'**

**2832 COMPRESS option must be 'OFF', 'MEDIUM' or 'HIGH'**

**2842 OPTCDROM option must be 'TRUE' or 'FALSE'**

**2852 Invalid TITLE option**
The **TITLE** option defines a string that is empty or contains more than 32 characters.

**2872 Invalid LANGUAGE option**
You have specified an ordering that is not supported by the compiler. The compiler uses English sorting order.

**2893 Warning option must be 1, 2, or 3**
The compiler uses full reporting (level 3).

**2911 Invalid icon file *filename***
The compiler cannot find the icon file specified in the **ICON** option, or the file is not a valid icon file.

**2932 Copyright string exceeds limit of 50 characters**
The maximum length of the copyright string in the About box is limited to 50 characters.

**3011 Maximum of 32 built tags exceeded**
The compiler ignores the build tag.

**3031 Build tag length exceeds 32 characters**
Build tags can contain only alphanumeric characters or the underscore (_) character.

**3051 Build tag *tagname* contains invalid characters**
Build tags can contain only alphanumeric characters or the underscore (_) character.

**3076 [BUILDTAGS] section missing**
The **BUILD** option declared a conditional build, but there is no [BUILDTAGS] section in the project file. The compiler includes all topics in the build.

**3096 Build expression too complex**
The build expression has too many expressions ("~", "&") or is too deeply nested.

**3116 Invalid build expression**
The syntax used in the build expression on the specified line contains one or more logical or syntax errors.

**3133 Duplicate build tag in [BUILDTAGS] section**
The specified build tag has been assigned to a topic but not declared in the project file. The compiler ignores the tag for the topic.

**3178 Build expression missing from project file**
The topics have build tags, but there is no **build** expression in the project file. The compiler includes all topics in the build.

## Macro Errors

The following messages result from errors in the use of Help macros in footnotes, hot spots, and the [CONFIG] section of the Help project file.

**3511 Macro *macrostring* exceeds limit of 254 characters**

**3532 Undefined function in macro *macroname***
The specified macro is not on the list of macros supported by the compiler, nor is it specified in the *RegisterRoutine* macro. The compiler passes the macro to the Help file, however.

**3552 Undefined variable in macro *macroname***

**3571 Wrong number of parameters to function in macro *macroname***

**3591 Syntax error in macro *macroname***

**3611 Function parameter type mismatch in macro *macroname***

**3631 Bad macro prototype**
The prototype string passed to the *RegisterRoutine* macro is invalid.

**3652 Empty macro string**
The ! footnote or a hidden text starting with "!" does not contain a macro.

**3672 Macro *macroname* nested too deeply**
Macro strings that contain macro strings as parameters may not next more than three deep.

## Context-String Errors

The following messages are caused by problems with context-string footnotes or context strings specified in jumps or in Help project-file options.

**4011 Context string *contextname* already used**
The specified context string was previously assigned to another topic. The compiler ignores the latter string and the topic has no identifier.

**4031 Invalid context string *contextname***
The context string footnote contains nonalphanumeric characters or is empty. The compiler does not assign the topic an identifier.

**4056 Unresolved context string specified in CONTENTS option**
The Contents topic defined in the project file could not be found. The compiler uses the first topic in the build as the Contents topic.

**4072 Context string exceeds limit of 255 characters**
The compiler ignores the context string.

**4098 Context string(s) in [map] section not defined in any topic**
The compiler cannot find a context string listed in the [MAP] section in any of the topics in the build.

**4113 Unresolved jump or popup *contextname***
The specified topic contains a context string that identifies a nonexistent topic.

**4131 Hash conflict between *contextname* and *contextname.***
The hash algorithm has generated the same hash value for both of the listed context strings. Change one of the context strings and recompile.

**4151 Invalid secondary window with popup**
The window name for the secondary window is "main" or another disallowed member name.

**4171 Cannot use secondary window with popup**
The hidden text defining the pop-up identifier contains a secondary window name.

**4196 Jumps and lookups not verified**
Due to low memory conditions, the build continues without verifying the validity of jumps and popups. (The reference to "lookups" in the error message is incorrect.)

**4211 Footnote text exceeds limit of 1023 characters**
Footnote text cannot exceed the limit of 1,023 characters. The compiler ignores the footnote.

**4231 Footnote text missing**
The specified topic contains a footnote that has no characters.

**4251 Browse sequence not in first paragraph**
The browse sequence footnote is not in the first paragraph of the topic. The compiler ignores the browse sequence.

**4272 Empty browse sequence string**
The browse-sequence footnote is not in the first paragraph of the topic. The compiler ignores the browse sequence.

**4272 Empty browse sequence string**
The browse sequence footnote for the specified topic contains no sequence characters.

**4292 Missing sequence number**
A browse sequence footnote ends in a colon (:) for the specified topic. Remove the colon or enter a "minor" sequence number and then recompile.

**4312 Browse sequence already found**
A browse sequence footnote already exists for the specified topic. The compiler ignores the latter sequence.

**4352 Empty title string**
The title footnote for the specified topic contains no characters. The compiler does not assign the topic a title.

**4372 Title defined more than once**
There is more than one title footnote in the specified topic. The compiler uses the first title string.

**4393 Title exceeds limit of 128 characters**
The compiler ignores the additional characters.

**4412 Keyword string exceeds limit of 255 characters**

**4433 Empty keyword string**
There are no characters in the keyword footnote.

**4452 Keyword(s) defined without title**
The topic has a keyword assigned to it, but no title.

**4471 Build tag footnote not at beginning of topic**
The build-tag footnote marker, if used, must be the first character in the topic.

**4492 Build tag exceeds limit of 32 characters**
The compiler ignores the tag for the topic.

**4551 Entry macro not in first paragraph**
The ! footnote (for executing a macro) is not in the first paragraph of the topic. The compiler ignores the macro.

### Topic File Errors

The following messages result from problems in rich-text (RTF) formatting in one or more topic files.

**4616 File *filename* is not a valid RTF topic file**

**4639 Error in file *filename* at byte offset 0x*offset***
The specified file contains unrecognized RTF at that byte offset.

**4649 File *filename* contains more than 23767 topics**

**4652 Table formatting too complex**
The compiler encountered a table with borders, shading, or right justification.

**4662 Side by side paragraph formatting not supported**
The side-by-side paragraph formatting is not supported in Microsoft Windows Help 3.1.

**4671 Table contains more than 32 columns**

**4680 Font *fontname* in file *filename* not in RTF font table**
The compiler uses the default system font.

**4692 Unrecognized graphic format**
The compiler supports only Windows bitmaps. Windows metafiles, segmented graphics, and multi-resolution graphics. The compiler ignores the graphic.

**4733 Hidden page break**
A page break is a part of the hidden text. A page break formatted as hidden text will not separate two topics.

**4753 Hidden paragraph**
A paragraph marker is part of the hidden text. The compiler ignores the paragraph marker.

**4763 Hidden carriage return**
A carriage return is part of the hidden text. The compiler ignores the carriage return.

**4774 Paragraph exceeds limit of 64K**
A single paragraph has more than 64K of text of 64K of graphics. (This limit does not include graphics stored separately from the data, using the **bmc, bml,** or **bmr** statements.)

**4792 Non-scrolling region defined after scrolling region**
A **keepn** statement precedes a paragraph that is not the first paragraph is treated as regular text and is part of the regular topic text.

**4813 Non-scrolling region crosses page boundary**
A **pard** statement must appear before the **page** statement in a topic containing a **keepn** statement.

## Miscellaneous Errors

The following messages are caused by conditions such as MS-DOS file errors or out-of-memory conditions.

**5035 File *filename* not created**
There are no topics to compile or the build expression is false for all topics. The compiler does not create a Help file.

**5059 Not enough memory to build help file**
To free memory, unload any unneeded applications, device drivers, and memory-resident programs.

**5075 Help Compiler corrupted. Please reinstall HC.EXE.**
Virus-checking code has detected a corruption in the compiler. Reinstall the compiler.

**5098 Using old key-phrase table**
Maximum compression can result only by deleting the .PH file before each recompilation of the Help topics or by setting the **OLDKEYPHRASE** option to "0".

**5115 Write failed**
Write-to-disk operation failed. Contact Microsoft Product Support Services.

**5139 Aborted by user**
Compilation terminated when the user pressed *Ctrl+C*.

# I N D E X

## D



## E



## F



## G



## H



## I

## K



## L



## M

## N



## O



## P

## R



## S



## T



## U

## V



## W



## X



## Z

7.0

# BORLAND® PASCAL WITH OBJECTS

BORLAND

Corporate Headquarters: 1800 Green Hills Road, P.O. Box 660001, Scotts Valley, CA 95067-0001, (408) 438-8400. Offices in: Australia, Belgium, Canada, Denmark, France, Germany, Hong Kong, Italy, Japan, Korea, Malaysia, Netherlands, New Zealand, Singapore, Spain, Sweden, Taiwan, and United Kingdom ▪ Part #11MN-BPL05-70 ▪ BOR 4688