TURBO PROFILER™

1.0

USER'S GUIDE

BORLAND

Turbo Profiler®

Version 1.0

User's Guide

BORLAND INTERNATIONAL, INC. 1800 GREEN HILLS ROAD
P.O. BOX 660001, SCOTTS VALLEY, CA 95066-0001

Copyright © 1990 by Borland International. All rights reserved. All Borland products are trademarks or registered trademarks of Borland International, Inc. Other brand and product names are trademarks or registered trademarks of their respective holders.

PRINTED IN THE USA.
R1 10 9 8

C O N T E N T S

Introduction 1
The difference between optimizing and profiling . . . 2
Hardware and software requirements . . . 3
Installing Turbo Profiler . . . 3
The README file . . . 3
What's in this manual . . . 4
A note on terminology . . . 5
How to contact Borland . . . 5

Chapter 1 A sample profiling session 7
Profiling a program (PRIME0) . . . 9
Setting up the profile options . . . 10
Collecting data . . . 11
Displaying statistics . . . 12
Printing modules and statistics . . . 14
Time and counts profile listing . . . 14
Profile statistics report . . . 16
Saving and restoring statistics . . . 17
Analyzing the statistics . . . 17
Viewing both source code and statistics . . . 18
Saving the window configuration . . . 19
Measuring an area's efficiency . . . 20
A modularized primes test (PRIME1) . . . 21
Modifying the program and reprofiling . . 22
Loading another program (PRIME2) . . 23
Reducing calls to a routine (PRIME3) . . 24
Still more efficiency (PRIME4) . . . 24
Reducing I/O time (PRIME5) . . . 26
Eliminating CR/LF pairs (PRIME6) . . . 26
Where to now? . . . 27

Chapter 2 Inside the profiler 29
Phantom tollbooths . . . 30
Determining the overhead of routine calls . . . 31
Who pays for loops? . . . 32
Logging callers . . . 35
Sampling vs. counting . . . 37
Profiler memory use . . . 38

Chapter 3 Profiling strategies 39
Preparing to profile . . . 40
Adjust your program . . . 41
Compile your program . . . 42
Set profile areas . . . 42
What level of detail do you need? . . . 44
What type of data do you need? . . . 44
When should data collection start? . . 45
How do you want time data grouped? . . . 46
Which data do you want to look at? . 46
Profiling your program . . . 47
What are you trying to find out? . . . 47
Testing algorithms . . . 48
Verifying and testing programs . . . 48
Timing execution and monitoring performance . . . 49
Studying unfamiliar code . . . 50
Which analysis mode do you use? . . . 50
Active analysis . . . 51
Passive analysis . . . 51
Some things to watch out for . . . 51
Profiling object-oriented programs . . . 52
How to speed up profiling . . . 52
How to improve statistical accuracy . . . 53
Insufficient data . . . 53
Resonance . . . 53
Some tips for profiling overlays . . . 54
Interpreting and applying the profile results . . . 55
How to analyze profile data . . . 55

Execution Profile window 55
Callers window 56
Overlays window 56
Interrupts window 56
Files window 56
How to filter collected data 56
Revise your program 57
Modify data structures 58
Store precomputed results 58
Cache frequently accessed data 59
Evaluate data as needed 59
Optimize existing code 59
Loops 59
Routines 60
Expressions 60
Wrapping it up 61

Chapter 4 The Turbo Profiler environment 63

Part 1: The environment components 63
The menu bar and menus 63
Choosing menu commands from the keyboard 64
Choosing menu commands with the mouse 64
Shortcuts 65
Turbo Profiler windows 65
Window management 67
The status line 68
Dialog boxes 68
Check boxes and radio buttons 69
Input boxes and lists 70
Part 2: The menu reference 71
≡ menu (System) 71
Repaint Desktop 73
Restore Standard 73
About 73
File menu 73
Open 73
Using the File Name input box ... 74
Using the Files list box 75
Change Dir 75
New Directory dialog box components 76
Get Info 76
DOS Shell 77
Quit 78
View menu 78
Module 79
Line 80
Search 80
Next 81
Goto 81
Add Areas 81
Remove Areas 82
Operation 82
Callers 83
Module 84
File 85
Edit 86
Execution Profile 86
Display 88
Filter 89
Module 90
Remove 91
Callers 91
Inspect (in left pane) 94
Inspect (in right pane) 94
Sort (in right pane) 94
Overlays 95
Display 95
Inspect 96
Interrupts 96
Collection (in top pane) 97
Subroutines (in top pane) 97
Add (in top pane) 97
Pick (in top pane) 98
Remove (in top pane) 98
Delete All (in top pane) 98
Display (in bottom pane) 98
Files 98
Collection (in top pane) 100
Detail (in top pane) 100
When Full (in top pane) 100
Display (in bottom pane) 100
Areas 101
Add Areas 102
Remove Areas 102

Inspect 102
Options 103
Sort 104
Routines 104
Local Module (in right pane) 105
Areas (in both panes) 106
Callers (in both panes) 106
Module (in both panes) 106
Profile (in both panes) 106
Disassembly (CPU) 106
Goto 108
Origin 108
Follow 108
Previous 109
View Source 109
Mixed 109
Run menu 110
Run 110
Program Reset 110
Arguments 110
Statistics menu 111
Callers 112
Files 112
Interrupts 112
Overlays 113
Profiling Options 113
Accumulation 114
When you would disable accumulation 115
Delete All 117
Save 117
Saving Files 118
Restore 118
Print menu 119
Statistics 119
Module 119
Options 120
Options menu 121
Macros 121
Create (Alt=) 123
Stop Recording (Alt-) 123
Remove 123
Delete 123
Display Options 123
Display Swapping 124
Screen Lines 124
Tab Size 124
Width of Names 124
Path for Source 125
Save Options 125
Restore 126
Window menu 127
Zoom 127
Next 127
Next Pane 127
Size/Move 127
Iconize/Restore 127
Close 128
Undo Close 128
User Screen 128
The open window list 129
Help menu 129
Index 129
Previous Topic 129
Help on Help 130
Appendix A Turbo Profiler's command-line options 131
The command-line options 132
Configuration file (-c) 133
Display update (-d) 133
Help (-h and –?) 133
Process ID switching (-i) 134
Modify heap size (-m) 134
Mouse support (-p) 134
Remote profiling (-r) 134
Source code and symbols (-s) 135
Video hardware (-v) 136
Overlay area size (-y) 136
Appendix B Customizing Turbo Profiler 139
Running TFINST 140
Setting the screen colors 140
Customizing screen colors 140
Windows 140
Dialog boxes and menus 141
Screen 142

The default colors 142
Setting Turbo Profiler display parameters 142
Display Swapping 142
Screen Lines 143
Fast Screen Update 143
Permit 43/50 Lines 143
Full Graphics Saving 143
Tab Size 144
User Screen Updating 144
Turbo Profiler options 145
The Directories dialog box 145
The User Input and Prompting dialog box 145
History List Length 145
Beep on Error 145
Mouse Enabled 146
Control Key Shortcuts 146
The Miscellaneous Options dialog box 146
Printer Output 146
Use Expanded Memory 146
NMI Intercept 146
Ignore Case of Symbol 147
DOS Shell Swap Size (Kb) 147
Remote Analyzing 147
Remote Link Port 147
Link Speed 147
Setting the mode for display 147
Default 147
Color 147
Black and White 148
Monochrome 148
LCD 148
Command-line options and TFINST equivalents 148
When you're through... 149
Saving changes 149
Save Configuration File 149
Modify TPROF.EXE 149
Exiting TFINST 150

Appendix C Remote profiling 151

Remote hardware requirements 151
Installing TFREMOTE 152
Starting the remote link 153
Starting Turbo Profiler on the remote link 153
Loading the program to the remote system 154
TFREMOTE command-line options .. 154
Getting it all to work 155
TFREMOTE messages 156

Appendix D Virtual profiling on the 80386 processor 159

Equipment required for virtual profiling 159
Installing the virtual profiler device driver 160
Starting the virtual profiler 160
Differences between normal and virtual profiling 162
TF386 error messages 163
TDH386.SYS error messages 164

Appendix E Prompts and error messages 165

Turbo Profiler prompts 165
Turbo Profiler error messages 169

Index 179

T A B L E S

3.1: Ways of using a profiler 48
3.2: Local menu commands for filtering collected statistics 56
4.1: Manipulating windows 67
4.2: Summary of Turbo Profiler windows .78
A.1: Turbo Profiler command-line options 132
B.1: Command-line options and TFINST equivalents 148
C.1: TFREMOTE command-line options .154

F I G U R E S

1.1: Turbo Profiler with PRIME0 loaded .. 10
1.2: Program statistics, PRIME0 11
1.3: The Display Options dialog box 12
1.4: Counts display in Execution Profile window 13
1.5: Time and Counts in the Execution Profile window 14
1.6: Time and count statistics, PRIME1 ... 22
1.7: Time and count statistics, PRIME4 ... 25
1.8: PRIME6's execution times and counts 27
2.1: Execution time and count compartments for PTOLL/PTOLLPAS 30
2.2: Memory map for Turbo Profiler 38
3.1: The Callers window 50
4.1: A typical window 66
4.2: A typical dialog box 69
4.3: The File Name history list 71
4.4: Turbo Profiler's menu bar and global menus 72
4.5: The Program Load dialog box 74
4.6: The New Directory dialog box 76
4.7: The Get Info text box 76
4.8: The Module window 79
4.9: The Area Options dialog box 83
4.10: The Stack Trace dialog box 84
4.11: The Pick a Module dialog box 85
4.12: The File dialog box 86
4.13: The Execution Profile window 87
4.14: The Display Options dialog box 88
4.15: The Callers window, showing calls in CALLTEST 91
4.16: The Callers window local menus ... 93
4.17: The Pick a Caller dialog box 94
4.18: The Overlays window 95
4.19: The Interrupts window 96
4.20: The Interrupt window local menus . 97
4.21: The Files window 99
4.22: The Files window local menus 99
4.23: The Display Options dialog box ... 101
4.24: The Areas window 101
4.25: The Area Options dialog box 103
4.26: Propagation of time 104
4.27: The Routines window 105
4.28: The Routines window local menus .105
4.29: The Pick a Module dialog box 106
4.30: The Disassembly (CPU) window .. 107
4.31: The Profiling Options dialog box .. 113
4.32: The Save dialog box 117
4.33: The Restore dialog box 118
4.34: The Pick a Module dialog box 119
4.35: Annotated source listing for PRIME0 120
4.36: The Printing Options dialog box .. 120
4.37: The Display Options dialog box ... 123
4.38: The Save Configuration dialog box .125
4.39: The Restore dialog box 126
4.40: Windows and window icons 128
A.1: Turbo Profiler DOS-level help 134
B.1: Customizing colors for windows ... 141
B.2: The Display Options dialog box 142
B.3: The User Input and Prompting dialog box 145
B.4: The Miscellaneous Options dialog box 146

INTRODUCTION

Borland's Turbo Profiler is the missing link in your software development cycle. Once you have your code doing what you want, Turbo Profiler helps you do it faster and more efficiently.

So what is a profiler? Profilers (also known as performance analyzers) are software tools that measure a program's performance by finding its bottlenecks:

- where your program spends its time
- how many times a line executes
- how many times a routine is called, and by which routines
- what files your program accesses most and for how long

Profilers also monitor critical computer resources:

- processor time
- disk access
- keyboard input
- printer output
- interrupt activity

By monitoring vital activities and providing detailed statistical reports on every part of your program's performance, Borland's Turbo Profiler enables you to fine-tune your programs. By opening up the inside of your program and exposing its most intricate operations—from execution times to statement counts, from interrupt calls to file access activities—Turbo Profiler helps you polish your code and speed up your programs.

Turbo Profiler surpasses other profilers on the market both in power and ease of use:

- Interactive profiling quickly reveals inefficient code in a program.
- Profiles any size program that runs under DOS.

- Handles programs written in Turbo Pascal, Turbo C++, Turbo C, and Turbo Assembler, as well as programs compiled with Microsoft C and MASM.
- Has an easy-to-use interface with multiple overlapping windows, mouse support, and context-sensitive help.
- Reports execution time and execution count for routines and individual lines of a program.
- Tracks complete call path history for all routines. Analyzes frequency of calls with complete call stack tracing.
- Monitors DOS file activities from the Files window by file handle and time of open, close, read, or write. Event list logs number of bytes read or written.
- Supports selective interrupt monitoring. Monitors all video, keyboard, disk, and mouse interrupts, and custom interrupts. Provides a complete event list or frequency monitoring. List of common DOS calls by symbolic name lets you quickly identify interrupts.
- Supports complete tracking of Turbo Pascal and Turbo C overlays.
- 386 virtual machine profiling uses no RAM—leaves all of your RAM available for your program.
- Allows remote profiling.
- Supports programs in C and assembler compatible with Codeview .EXE format files.
- Profiles any program from any compiler if it's accompanied by standard Microsoft format .MAP files.

By picking up where code optimizers leave off, Turbo Profiler directs you immediately to slow code, pointing out where to open up bottlenecks and when to rework algorithms.

The difference between optimizing and profiling

An optimizer makes your program run a little faster by replacing time-consuming instructions with less expensive ones. But optimizing can't fix inefficient code.

The profiler helps you to detect the part of your code that is least efficient and helps point to algorithms which can be modified or rewritten. Studies show that the largest performance improvements in programs come from changing algorithms and data

structures, rather than from optimizing small segments of compiled code. Trying to find bottlenecks without a profiler is like trying to find bugs without a debugger; Turbo Profiler reduces both time and effort.

Hardware and software requirements

Turbo Profiler runs on the IBM PC family of computers, including XT, AT, the PS/2 series, and all true IBM compatibles. DOS 2.0 or higher is required and at least 384K of RAM. Turbo Profiler will run on any 80-column monitor. We recommend a hard disk, although it is possible to profile on a computer with two floppy disk drives.

Turbo Profiler does not require an 80x87 math coprocessor.

Installing Turbo Profiler

To install Turbo Profiler on your system, run INSTALL.EXE, the easy-to-use installation program on your distribution disks. This program automatically copies files from the distribution disks to your hard disk. Just insert the Install disk in your A drive, type `A: INSTALL`, and press *Enter*. Then follow the instructions on the screen.

The distribution disks are formatted for double-sided, double-density disk drives and can be read by IBM PCs and close compatibles. For a list of the files on your distribution disks, see the README file on the Installation disk.

The README file

Take a look at the README file on the Install disk before you do anything else with Turbo Profiler. This file contains last-minute information that might not be in the manual. It also lists every file on the distribution disks, with a brief description of what each one contains.

To access the README file, insert the Installation disk in drive A, switch to drive A by typing `A:` and pressing *Enter*, then type `README` and press *Enter* again. Once you are in README, use the ↑ and ↓ keys to scroll through the file. Press *Esc* to exit.

What's in this manual

Introduction (this section) tells you what profilers are in general, summarizes Turbo Profiler's features, and gets you ready to run Turbo Profiler on your system.

Chapter 1, A sample profiling session, is a tutorial that takes you through a typical (albeit simple) profiling session. This chapter starts with a "let's see what's going on" profile, then takes you through interpreting the profile data collected, modifying and refining the program based on insight gained from the profile, and running additional profiles to gauge the effect of each successive modification.

Chapter 2, Inside the profiler, uses analogy to explain how the profiler gathers execution-time and execution-count data while your program runs.

Chapter 3, Profiling strategies, provides general guidelines, along with some tips, for conducting a fruitful profiling session.

Chapter 4, The Turbo Profiler environment, explains in detail each menu item and dialog box option in the Turbo Profiler environment.

Appendix A, Turbo Profiler's command-line options, lists each Turbo Profiler command-line option and explains what the option accomplishes.

Appendix B, Customizing Turbo Profiler, explains how to use TFINST to change the configuration defaults of TPROF.

Appendix C, Remote profiling, describes how to profile with two systems; you run your program on one and Turbo Profiler on the other.

Appendix D, Virtual profiling on the 80386 processor, explains how to run the profiler in 80386 extended memory, leaving the full 640K of real memory for your program.

Appendix E, Prompts and error messages, lists all prompts and error messages that can occur, with suggestions on how to respond to them.

A note on terminology

For convenience and brevity, we use some terms in this manual in slightly more generic ways than usual. These terms are *module, routine,* and *argument.*

module

A *module* in this manual refers to what is usually called a module in C and in assembler, but also refers to what is called a *unit* in Pascal.

routine

Similarly, a *routine* in this manual refers to both a C function and to what is known in Pascal as a subprogram (or routine), which encompasses *functions, procedures* and *object methods.* In C, a function can return a value (like a Pascal function) or not (like a Pascal procedure). (When a C function doesn't return a value, it's called a *void function.*) We use *routine* in a generic way to refer to both C functions and Pascal functions and procedures.

argument

Finally, the term *argument* is used interchangeably with *parameter* in this manual. This applies to references to command-line arguments used to invoke a program from DOS, as well as arguments passed to routines.

How to contact Borland

If, after reading this manual and using Turbo Profiler, you would like to contact Borland with comments, questions, or suggestions, we suggest the following procedures:

- The best way is to log on to Borland's forum on CompuServe: Type `GO BPROGB` at the main CompuServe menu and follow the menus to Turbo Profiler. Leave your questions or comments there for the support staff to process.
- If you prefer, write a letter detailing your problem and send it to

  Technical Support Department—Turbo Profiler
  Borland International
  P.O. Box 660001
  1800 Green Hills Drive
  Scotts Valley, CA 95066-0001
- You can also telephone our Technical Support department between 6 a.m. and 5 p.m. at (408) 438-5300. To help us handle

your problem as quickly as possible, have these items handy before you call:

- product name (Turbo Profiler) and version number
- product serial number
- computer make and model number
- operating system and version number
- the contents of your CONFIG.SYS and AUTOEXEC.BAT files

If you're not familiar with Borland's No-Nonsense License statement, now's the time to read the agreement included with this package and mail in your completed product registration card.

C H A P T E R

1

A sample profiling session

Profiling is one of the least-understood yet most useful and vital areas of good software development. Surveys indicate that only a small fraction of professional programmers actually use profilers to improve their code. Other studies show that, most of the time, even the best programmers guess wrong about where the bottlenecks are in their programs.

What is the advantage to using this widely overlooked tool? For one, profiling your program can increase its overall performance. Second, profiling can augment your ability to produce efficient code. The bottom line is that profiling, like debugging, can be a cog in the wheel of the program development cycle.

We've based the examples in this chapter on Jon Bentley's "Programming Pearls" column (July 1987) in Communications of the ACM.

All the tutorial examples were run on a 286 machine with a Hercules video adapter.

In this chapter we show you an example of profiling put to good use, and how—in the long run—profiling can save you hours of hunting for that expensive line of code. You use Turbo Profiler to

- see where your program spends its time
- create an annotated source listing and a profile statistics report
- save profile statistics, then start up again with saved statistics
- analyze profile statistics and source code in side-by-side windows

The examples in this chapter are based on finding and printing all prime numbers between 1 and 1,000. Recall that a number is prime if it is divisible by only the integer 1 and itself; it must also be odd, since any even number is divisible by 2 and therefore is not prime. You can tell whether a particular number is prime by

checking to see if it is divisible by other, smaller primes, or by any integer larger than the first two primes, 2 and 3.

The object of profiling the example programs is to speed up the process of finding and printing the prime numbers. As you work through the examples, you'll learn how to use Turbo Profiler to test the efficiency of each example's structure.

The first program you'll look at is PRIME0. Once you've profiled it and seen where to modify the code, all you need to do is load and profile PRIME1. With the exception of PRIME1, each of the programs covered in this chapter (PRIME2, PRIME3, PRIME4, PRIME5, and PRIME6) is a variation on its predecessor.

Pascal users

The PRIME*n*.* programs are Turbo C programs. For Pascal programmers, we've also supplied Turbo Pascal versions of them on disk (PRIME*n*PA.*) that you can run as you go through this chapter. To each discussion of the C program's profile results, we've added comments about what you'll see if you're running the Pascal programs instead.

Make sure the files for all the example programs (PRIME*n*.C and PRIME*n*.EXE or PRIME*n*PA.PAS and PRIME*n*PA.EXE) are in your current directory.

For each of the examples we've provided both the source code and the executable code; Turbo Profiler requires both to analyze a program. Each of the examples was compiled with full symbolic information, since the profiler also requires this information.

To ensure that your programs contain full symbolic information, compile them with the appropriate compiler options turned on:

- **Turbo C++:** If you are compiling in the IDE, turn on **O**ptions | **F**ull Menus, then open the Debugger dialog box (choose **O**ptions | De**b**ugger), and set Source Debugging to Standalone. If you are compiling from the command line, use the **–v** command-line option.
- **Turbo C:** If you are compiling in the IDE, specify **S**tandalone in the **D**ebug | **S**ource Debugging option before you compile your modules. If you are compiling from the command line, specify the **–v** command-line option.
- **Turbo Pascal:** If you are compiling in the IDE, set the **O**ptions | **D**ebug Information and **D**ebug | **S**tand-Alone Debugging options to *On*. If you are compiling from the command line, use the **/V** command-line option.

- **Turbo Assembler:** Use the **/zi** command-line option, then link the program with TLINK, using the **/v** option.

80x87 users

While TPROF works with numeric coprocessors, if your system has an 80x87 chip, you have to fool it temporarily into thinking the 80x87 is unavailable so you'll get results similar to the ones shown in this tutorial. (Otherwise, the statistics here won't even approach what you'll see on your screen.) Just set an environment variable at the DOS prompt with the command `SET 87=N` before running the profiler. Of course, the statistics you get might still vary from what appears in the figures in this adapter; this occurs because of differences in individual systems (CPU speed, etc.).

Profiling a program (PRIME0)

You profile and improve a program in four steps:

1. Set up the program before profiling it.
2. Collect data while the program runs.
3. Analyze the collected data.
4. Modify the program and recompile it.

After modifying your program, repeat Steps 1 through 3 to see if the modifications have improved your program's performance.

PRIME0 uses Euclid's method of testing for prime numbers, a straightforward integer test for a remainder after division. As each prime number is found, it is stored in the array *primes*, and each successive number is tested for "prime-ness" by being divided by each of the numbers already stored in *primes*.

Leaving Turbo Profiler at any time is a simple, one-step procedure: just choose File | Quit or press Alt-X.

Load PRIME0 into Turbo Profiler by typing

```
TPROF PRIME0
```

and pressing *Enter*.

Pascal users

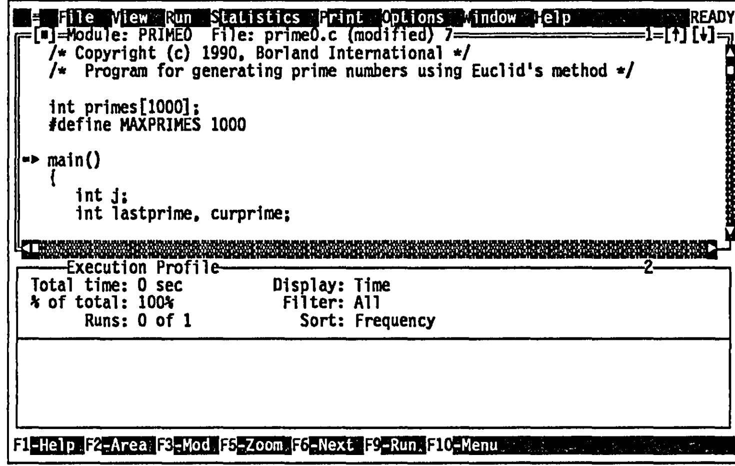
If you want to profile the Turbo Pascal version of PRIME0.C, make sure PRIME0PA.PAS and PRIME0PA.EXE are in your current directory, and enter

```
TPROF PRIME0PA
```

The profiler comes up with two windows open: the Module window (which displays PRIME0's source code) and the

Execution Profile window (which will display profile statistics after you run PRIMEO).

Figure 1.1
Turbo Profiler with PRIMEO loaded



For a more detailed description of the profiler's environment, see Chapter 4.

The Module and Execution Profile windows are concerned with Steps 1 and 3 in the profiling process. You use the Module window to determine what parts of the program to profile. Once you run a program, the Execution Profile window displays the information you need to analyze your program's behavior.

Setting up the profile options

Before you begin to profile your program, you must specify the areas you want to profile. An *area* is a location in your program where you want to collect statistics: An area can be a single line, a construct such as a loop, or an entire routine. For your first profile, you want more information than Turbo Profiler's default area settings provide.

To analyze a small number of short routines (like **prime** and **main** in this program), you have to know how often each line executes and how much time each line takes. To get this information, you must mark every line in the program as an area.

1. Press *Alt-F10* to open the Module window local menu.
2. Choose **A**dd Areas from the local menu. This menu lists area boundaries for you to choose from.

3. Choose **Every Line in Module**. This sets area markers for all lines in the module, then returns the cursor to the Module window.

Notice that all executable lines inside the module are now tagged with a marker symbol (=▸).

Collecting data

Now you're ready for the second step in the profiling process. Press *F9* to run PRIME0 under Turbo Profiler. The program prints the prime numbers between 1 and 1,000 on the User screen. When the program finishes, look at the information in the Execution Profile window. These are your *program statistics*.

Zoom the Execution Profile window: Press *F5* or choose **Zoom** from the Window menu. The Execution Profile window should now look like this:

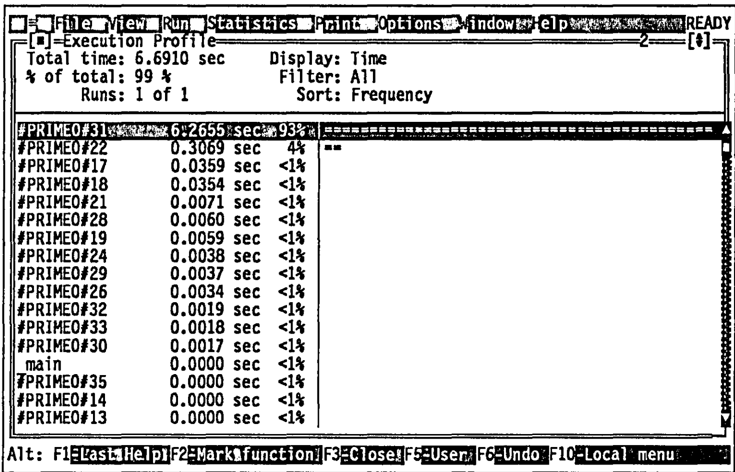


Figure 1.2
Program statistics, PRIME0

The upper pane of the Execution Profile window displays the program's total execution time, along with information about the data in the lower pane. The lower pane has four entries on each line:

- an area name
- the number of seconds spent in that area
- the percentage of total execution time spent in that area
- a *magnitude bar* proportional to the percentage

This line

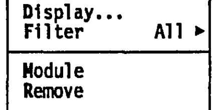
```
#PRIMEO#31   6.2655 sec  93%  |====================================
```

tells you that the thirty-first line of code in module PRIME0 executed for about 6.3 seconds—which was 93% of the total execution time for all marked areas. The magnitude bar automatically shows Line 31's time full-scale (because Line 30 is the most time-consuming of the marked areas).

Pascal users In PRIME0PA, the corresponding line of code is line 42.

Displaying statistics

You can also display this program's collected data as *execution counts.*



1. Press *Alt-F10* to bring up the local menu for the Execution Profile window.
2. Choose **D**isplay on the local menu.

   The Display Options dialog box lists five possible ways to display data in the Execution Profile window.

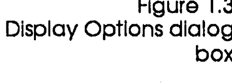
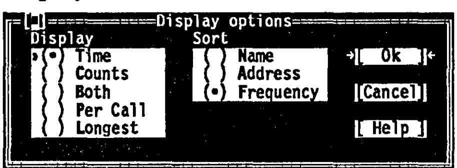


Figure 1.3 The Display Options dialog box



   - Time shows the total time spent in each marked area. (This is the default.)
   - Counts displays the number of times program control entered each area.
   - Both shows time and counts data on the same screen.
   - Per Call displays the average amount of time per call.
   - Longest shows the longest time spent in each area.

3. Choose Counts under Display in this dialog box. (Click Counts with the mouse, or use the arrow keys to move to it and press *Enter*, or press *C*, the hot key for this option.)
4. Choose OK (or press *Enter*).

The Execution Profile window now displays PRIME0's statistics as execution counts instead of execution times, as shown in this figure:

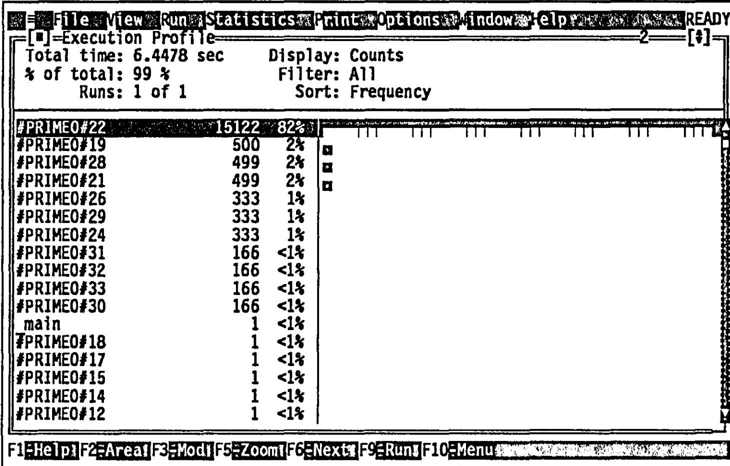


Figure 1.4
Counts display in Execution Profile window

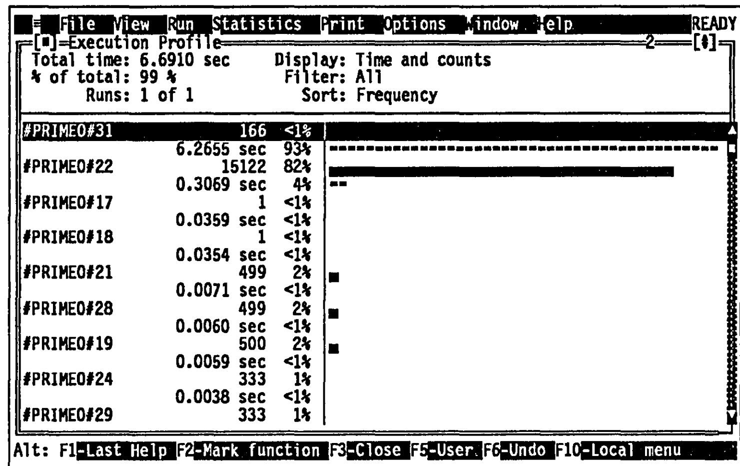
This display of PRIME0's statistics shows that line 22 is the most often called line in PRIME0. (It's line 31 in PRIME0PA.)

You can also see counts and times together. Bring up the Display Options dialog box again (either press *Alt-F10* and choose Display, or press *Ctrl-D*).

Choose Both under Display, then choose OK or press *Enter*. (To choose Both, either click it, or press ↓ to get to it, then press *Enter*, or press *B*, the hot key for this option.)

Now the Execution Profile window looks like this:

Figure 1.5
Time and Counts in the Execution Profile window



When the Execution Profile window displays time and counts together, the first entry for each area is execution counts, and the second is execution time. Figure 1.5 shows that the area PRIME0#22 (line 22 in PRIME0) was called 15,122 times (execution counts) and took up 0.31 seconds of total execution time.

Printing modules and statistics

In this section, you print two things:

1. a profile source listing of the code that's in the Module window, with time and counts data attached to each marked area
2. the profile statistics displayed in the Execution Profile window

Time and counts profile listing

Before you print a time-and-counts profile listing to a file, you must set the appropriate printing options.

1. Choose **Print | Options**.
2. In the Printing Options dialog box, choose the File radio button (press *Tab* until the radio buttons become active, then press ↓ to turn the setting to File).

3. Tab to the Destination File input box and type

   ```
   PRIMEOSC.LST
   ```

4. Choose ASCII to use the standard ASCII character set (rather than the IBM extended character set).
5. Choose OK (or press *Enter*).

The cursor returns to the active Execution Profile window.

Now, to print the listing file, choose **P**rint | **M**odule. In the Pick a Module dialog box, press ↓ to highlight the module name PRIME0, then press *Enter* (or choose OK).

You can shell out to DOS to see the file PRIME0SC.LST, which prints to the current directory. Choose **F**ile | **D**OS shell and, at the DOS prompt, type

```
TYPE PRIMEOSC.LST
```

This is what you see if you're profiling the Turbo C program, PRIME0:

The times in your file probably vary from these somewhat, because each computer is a little different.

```
Turbo Profiler  Version 1.0  Tue Feb 27 15:16:47 1990

Program: D:\TPROF\PRIME0.EXE  File prime0.c

Time  Counts
              /* Copyright (c) 1990, Borland International */
              /*  Program for generating prime numbers using Euclid's
method */

              int primes[1000];
              #define MAXPRIMES 1000

0.0000 1      main()
              {
                 int j;
                 int lastprime, curprime;

0.0000 1         primes[0] = 2;
0.0000 1         primes[1] = 3;
0.0000 1         lastprime = 1;
0.0000 1         curprime  = 3;

0.0359 1         printf("prime %d = %d\n", 0, primes[0]);
0.0354 1         printf("prime %d = %d\n", 1, primes[1]);
0.0059 500       while(curprime < MAXPRIMES)
                 {
0.0071 499          for(j = 0; j <= lastprime; j++)
0.3069 15122           if((curprime % primes[j]) == 0)
```

```
                          {
0.0038 333                   curprime += 2;
0.0034 333                   break;
                          }
0.0060 499              if(j <= lastprime)
0.0037 333                 continue;
0.0017 166              lastprime++;
6.2655 166              printf("prime %d = %d\n", lastprime, curprime);
0.0019 166              primes[lastprime] = curprime;
0.0018 166              curprime += 2;
                    }
0.0000 1        }
```

This profile source listing is useful because it's a permanent record that shows, for each area in your program, the execution time and execution counts.

Now type EXIT at the DOS prompt and press *Enter* to return to Turbo Profiler.

Profile statistics report

You can also print a replica of the open Execution Profile window's contents to your printer or to a disk file.

1. Choose **Print | Options** again.
2. Choose the Printer radio button.
3. Choose Graphics to include IBM semi-graphic characters in the printed report. (If your printer does not support IBM high ASCII characters, like ╓ and ╨, skip this step and proceed to Step 4.)
4. Press *Enter* (or choose OK).
5. Choose **Print | Statistics.**

The resulting printout, like the profile source listing, is a permanent record of your progress as you go through the steps of profiling, modifying, recompiling, and reprofiling in your quest for the sleekest and most efficient code possible (and practical) for your program.

Saving and restoring statistics

Before you go on, here's how to save PRIME0's profile statistics to a file, so you can quit Turbo Profiler at any time without losing the data. We also show you how to restore those statistics the next time you start the profiler.

Choose **S**tatistics | **S**ave to save your program's profile statistics to a .TFS (Turbo Profiler Statistics) file. Because PRIME0 is in the Module window, the File Name input box lists PRIME0.TFS as the default. Choose OK to create this file.

All the statistical data from the current profile run of PRIME0 is now saved in the file PRIME0.TFS in the current directory, so you can quit the profiler at any time without losing any of that information. Open PRIME0 in the profiler, and choose **S**tatistics | **R**estore. As before, the File Name input box lists PRIME0.TFS as the default. Press *Enter* to go to the Files list box, highlight PRIME0.TFS, and choose OK to recover the data from this file.

Analyzing the statistics

In this section you learn how to analyze the statistics in the Execution Profile window, so you can use what they reveal to streamline your program.

First, though, take another look at the time and count statistics in the Execution Profile window. Unzoom the Execution Profile window (choose **Z**oom from the Window menu or press *F5*) and look at the statistics for lines 22 and 31 (the **if** and **printf** statements).

Pascal users

In PRIME0PA it's line 31 (**if**) and line 42 (*Writeln*).

*We cover modifications to the **printf** statement in program PRIME5 and—for you Pascal users—introduction of the **uses** CRT statement in PRIME5PA.*

A time and count profile like this tells a lot about a program. For instance, you can see that line 22 in PRIME0 executes far more frequently than any other statement. It makes sense that line 22 executes 15,122 times, since it tests every number between 4 and 1,000 against every number in the array *primes*, until there is even division or the array is exhausted. That means a lot of numbers to be tested. You can also see that line 31, the **printf** statement, accounts for most of the program total execution time.

Viewing both source code and statistics

The data in the Execution Profile window shows that the test in line 22 is doing more work than it should. But you can't really get the entire picture until you look at execution time and count data and source code together.

What you need to do is compare time and count data in the Execution Profile window and the corresponding source code in the Module window.

Here's one way to display source code and profile statistics simultaneously:

1. Resize and move the Execution Profile window so it occupies the right half of your screen: Choose **Window | Size/Move**, or press *Ctrl-F5*.
2. Follow the directions on the status line to

   a. Resize the window to full-screen height and half-screen width.
   b. Move the resized window to the right.

   When you've done steps *a* and *b*, press *Enter*.
3. Activate the Module window by pressing *F6*, then resize and move it so it occupies the left half of the screen.
4. Go back to the Execution Profile window (press *F6* again).

To resize a window with the mouse, drag the Resize box in the lower right corner; to move the window, drag the title bar or any double-line left or top border character (|| or =).

There is an automatic link between the Execution Profile window and the Module window, so that when you move through the source code, the execution profile display tracks the cursor's current line position. To see this tracking feature in action,

1. Activate the Execution Profile window (press *F6*), and move the highlight bar to the first line (statistics for line 31 of PRIME0, line 42 of PRIME0PA).
2. Open the local menu (press *Alt-F10*) and choose **Module** (or just press *Ctrl-M*).

   The profiler positions the cursor on line 31 in the Module window.

3. Use the arrow keys to move through the source code to line 22 (line 31 in PRIME0PA).

   This line is the second largest time consumer in PRIME0. The top two statistics lines in the Execution Profile window now display the profile data for this **If** statement.
4. Move the cursor in the Module window to line 21 (line 29 of PRIME0PA) and note how the display in the Execution Profile window tracks with it. The top lines in the Execution Profile window are now the profile statistics for line 21.
5. Move the cursor to line 30 (line 42 of PRIME0PA) and note the display in the Execution Profile window.

Having the two windows synchronized this way makes it easy to find the greatest resource hogs in your program. Once you get a better feel for interpreting the data onscreen, you won't need to rely as much on profile listings like the one on page 16.

Saving the window configuration

This is a good time to save your customized version of Turbo Profiler. If you don't save your customized window arrangement, the windows will revert to their default size and placement the next time you load a program into the profiler.

1. Choose **O**ptions | **S**ave Options. This brings up the Save Configuration dialog box.
2. By default, the Options check box is already checked. This records settings (such as the Execution Profile window's display options) in the configuration file.
3. In the Save Configuration dialog box, tab to Layout and press *Spacebar*. This causes your side-by-side window layout to be saved in the configuration file.
4. By default, the configuration file to be saved is TFCONFIG.TF, listed in the Save To input box. Choose OK, or press *Enter*, to save your options to this file in the current directory.

Wherever you start up Turbo Profiler, it looks for TFCONFIG.TF, the default configuration file. When the profiler finds that file, the options and layout you've set will come up automatically.

Measuring an area's efficiency

The ratio of execution time to execution counts is a good measure of a line's or routine's overall efficiency. To see this ratio for the areas in PRIME0, change the display option in the Execution Profile window. Here's how:

1. From the Execution Profile window's local menu (press *Alt-F10*, choose **D**isplay.
2. Under Display in the dialog box, choose Per Call.
3. Choose OK (or press *Enter*).

Now you can see that line 22 is much more efficient than line 31 (in PRIME0PA, lines 30 and 41). It uses up a lot of execution time because it executes so many times, but each individual call averages much less than a millisecond. Line 31, on the other hand, averages nearly 38 milliseconds per call (in PRIME0PA, line 42 averages 28 milliseconds).

The output from the profiler points the way to improving the execution time of PRIME0 and making it structurally simple. The task of improving the program can be divided into two strategies:

1. Reduce the amount of time spent in input/output.
2. Rewrite the looping structure to be more streamlined and efficient.

The input/output problem can be partially resolved by reducing the **printf** statement from its present form

```
printf("prime #%d = %d\n", lastprime, curprime)
```

to simply

```
printf("%d\n", curprime).
```

Pascal users You can change the *Writeln* statement to

```
Writeln(CurPrime);
```

Just this simple modification results in a considerable savings in the execution time. However, you can't reduce the number of times you call the output statement; for the given problem, there will always be 168 primes to print out. And apart from this minor improvement, there is not a great deal you can do to speed up the execution of PRIME0. Its algorithm, which requires saving all the

previous results in an array and then using them to divide, is thorough but virtually impossible to streamline. (It is also not very memory-efficient, because the array requires an allocation of memory equal to the number of primes being tested. Eventually this imposes a limit on the number of primes that could be tested without running out of memory.)

Fortunately, there is a better way to test for prime numbers: You can change the algorithm itself. That's what happens in the next example program, PRIME1.

A modularized primes test (PRIME1)

Pascal users: Load PRIME1PA.

You're finished with PRIME0 now, so load PRIME1 (Pascal users: PRIME1PA), the next version of the prime number program, into the Module window and look at the code.

1. Choose **File | Open**.
2. By default, the File Name input box is activated and contains the file-name mask *.EXE. Press *Enter.*
3. In the Files list box, use the ↑ and ↓ keys to highlight PRIME1.EXE (PRIME1PA.EXE).
4. Press *Enter.* Turbo Profiler loads PRIME1 (PRIME1PA) into the Module window.
5. Zoom the Module window (press *F5*). Note the added **prime** (*Prime*) routine on line 4.

You can see right away that two major changes have occurred:

- The array *primes* is gone. This program does not test by dividing each number by all smaller primes; it simply uses a loop to divide by all the odd numbers up to but not including the suspected prime. Initially this algorithm results in more iterations, but we will see that it eventually can be refined into a more streamlined and readable program.
- The prime number test itself has been placed in a separate routine that is called from the main program.

Set your areas (choose **Add Areas | Every Line in Module** from the Module window local menu), press *Enter,* then run (*F9*) PRIME1 in Turbo Profiler and look at the statistics. Then choose **Display** from the Execution Profile window local menu to open the Display

Options dialog box and turn on the Both radio button. Press *Enter*, then zoom the Execution Profile window (*F5*).

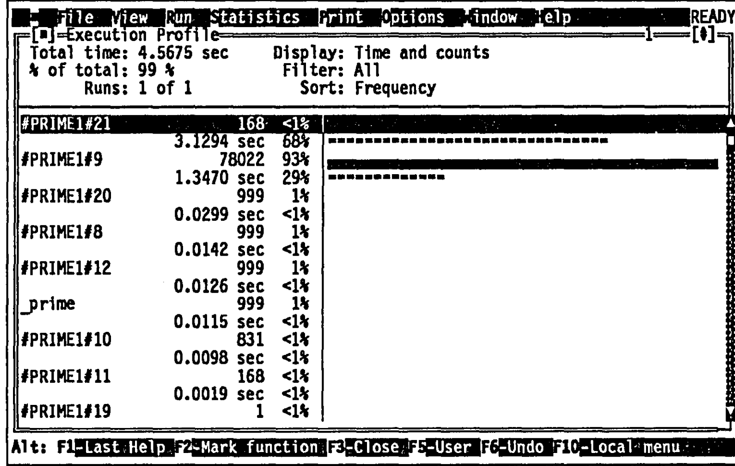


Figure 1.6
Time and count statistics, PRIME1

You can see that the execution time has improved somewhat (this is due in part to the fact that PRIME1 prints out less information than PRIME0). The main bottleneck is still the **printf** statement (now line 21). (In PRIME1PA it's the *Writeln* statement, line 24.)

Notice in particular that the test for prime numbers (line 9 in PRIME0, line 12 in PRIME0PA) now executes 78,022 times instead of 15,122. This looks impressive at first, but notice that it only increases execution time for this line by about 1 second; we have already seen that this statement is very time-efficient.

One obvious way to improve efficiency, now that we have isolated the test loop in a separate routine, is to cut down on the number of calls to the routine. There are ways of limiting the number of integers that have to be passed to the routine for testing; the more you can eliminate at the main program level, the fewer calls you have to make and the faster your program executes. That is the strategy we employ in the next several sample programs.

Modifying the program and reprofiling

Bentley points out that instead of testing for all factors between 1 and *n* in the modulus statement, you can set the upper limit of the

test to the square root of the number you're testing. That's what we've done in program PRIME2 (PRIME2PA).

Loading another program (PRIME2)

Go ahead and load PRIME2, the next version of the sample program, into the Module window. In program PRIME2, we've added a **root** (*Root*) routine that calls a square root library routine and returns an integer result.

Pascal users Load PRIME2PA into the Module window.

You need to set areas for all lines in the module, so bring up the local menu and choose **A**dd Areas | **E**very Line in Module, then press *Enter.*

Press *F9* to start the profile. Once again, you'll see the primes between 1 and 1,000 print to the User screen.

When the program finishes running, open the Display Options dialog box (choose **D**isplay from the Execution Profile local menu) and set Display to *Both*. Press OK. Despite decreasing the number of calls to line 15 (from 78,022 to 5,288) and reducing the time spent in the same statement, there's still a substantial increase in overall execution time.

The problem with PRIME2 (and PRIME2PA) is the expense of the new root routine. Line 7 inside the routine executes 5,456 times and consumes almost 5 seconds. At approximately 1 millisecond per call, you can't afford too many passes through this routine. (In PRIME2PA it's line 9.)

When the Execution Profile window shows both time and count information, certain patterns are worth looking for. In inefficient routines, the second line (time data) is much longer than the first line (count data), which means the ratio of time to counts is high. This is the case for line 27, the **printf** statement (in the Pascal program, it's line 28).

When the routine's time:count ratio is high, the best thing to do is substitute another routine.

However, the **return** statement in the **root** routine (line 7), presents a different problem. It accounts for the largest number of calls and the largest amount of time. Two other lines (line 5 and line 8) have 5,456 calls, but the histogram bar for each of these cases shows small execution times. This is good: It means the

statements are fast. So the biggest problem right now is the number of calls made to the **root** routine.

Reducing calls to a routine (PRIME3)

The problem now is to reduce the number of calls to the **root** routine. Load PRIME3 into the Module window, then zoom the Module window and take a look at the source code.

Pascal users Load PRIME3PA into the Module window.

In PRIME3, the only routine modified is **prime**. We've added a new integer variable, *limit*, and set *limit* equal to **root**(*n*) before entering the **for** loop. The test in the **for** loop is based on *limit*.

Pascal users In PRIME3PA, we've added the integer variable *Limit* and set it equal to the root of *n* before entering the *for* loop. The test in the **for** loop is based on *Limit*.

In the Module window local menu, set areas to Every Line in Module. When you profile the program this time (choose Run | Run or press *F9*), the program runs quite a bit faster. PRIME3 shows an almost 50% decrease in total execution time.

The **printf** routine is now the major resource consumer, eating up over half the execution time. By reducing the number of calls to the square root routine in **root** (from 5,456 to 999), we've decreased computational time substantially.

Still more efficiency (PRIME4)

There are still more ways to increase the efficiency of the **prime** routine. Load PRIME4 into the Module window now, then examine lines 8 through 17 of the source code.

Pascal users Load PRIME4PA and examine lines 11 through 32.

```
/****** PRIME4.C ******/

if (n % 2 == 0)

    return (n==2);


if (n % 3 == 0)

    return (n==3);
```

```
{****** PRIME4PA.PAS ******}

if (N MOD 2 = 0) then
begin
  Prime := N = 2;
  exit;
end;
if (n MOD 3 = 0) then
begin
  Prime := N = 3;
  exit;
```

```
    if (n % 5 == 0)                          end;
                                             if (N mod 5 = 0) then
                                             begin
        return (n==5);                         Prime := N = 5;
                                               exit;
                                             end;
    for (i=7; i*i <= n; i+=2)                for I := 7 to N-1 do
        if (n % i == 0)                        if (N mod I = 0) then
                                               begin
            return 0;                            Prime := False;
                                                 exit;
                                               end;
    return 1;                                Prime := True;
```

There are a number of improvements here.

- The three **if** statements in the **prime** routine weed out factors that are multiples of 2, 3, and 5, respectively. If you can't throw out a number *n* based on one of these tests, you must test the remaining numbers, up to the root of *n*. You can start at the value 7—the **if** statements have eliminated all possibilities below this number.
- The **for** loop now increments by two on each iteration, because there's no point in testing even numbers.
- The test *i* * *i* <= *n* has replaced the more expensive test involving the **root** routine.

The net result is that we've shaved more than one second off the execution time. The count data in Figure 1.7 shows that **printf** now consumes 96% of run time.

Figure 1.7
Time and count statistics, PRIME4

```
≡ File View Run Statistics Print Options Window Help                READY
[■]=Execution Profile=============================================1===[↕]
 Total time: 3.2654 sec      Display: Time and counts
 % of total: 99 %             Filter: All
       Runs: 1 of 1             Sort: Frequency

#PRIME4#27                 168   2%
                    3.1261 sec  96%
#PRIME4#26                 999  12%
                    0.0299 sec  <1%
#PRIME4#15                1530  19%
                    0.0278 sec  <1%
#PRIME4#8                  999  12%
                    0.0145 sec  <1%
#PRIME4#18                 999  12%
                    0.0138 sec  <1%
_prime                     999  12%
                    0.0124 sec  <1%
#PRIME4#10                 499   6%
                    0.0075 sec  <1%
#PRIME4#9                  500   6%
                    0.0066 sec  <1%
#PRIME4#12                 332   4%

Alt: F1-Last Help F2-Mark function F3-Close F5-User F6-Undo F10-Local menu
```

Reducing I/O time (PRIME5)

This change in the amount of time consumed by **printf** shows that the program is now I/O bound, rather than computationally bound. Perhaps that's acceptable. But just for fun let's try to squeeze a little more blood out of the I/O turnip.

Pascal users load PRIME5PA

Load PRIME5 into the Module window and look at line 28 (line 3 in the Pascal version).

Turbo C has a fast version of **printf** called **cprintf**, which is PRIME5's only statement change from program PRIME4. **cprintf** handles newlines differently than **printf** does; in **cprintf**, an explicit carriage-return/linefeed pair, *\r\n*, replaces the single newline character of **printf**.

Pascal users

Turbo Pascal also has a fast version of *Writeln* in the *Crt* unit. We tell PRIME5PA to use this fast version by including the **uses** *Crt* statement at the beginning of the program. This is the only change to PRIME5PA.

Call up the Module window's local menu and set areas for every line in PRIME5's source module. Run PRIME5, then examine the profile count data for line 28 in PRIME5 (or line 3 in PRIME5PA).

The faster print routine saves almost a second in 168 calls.

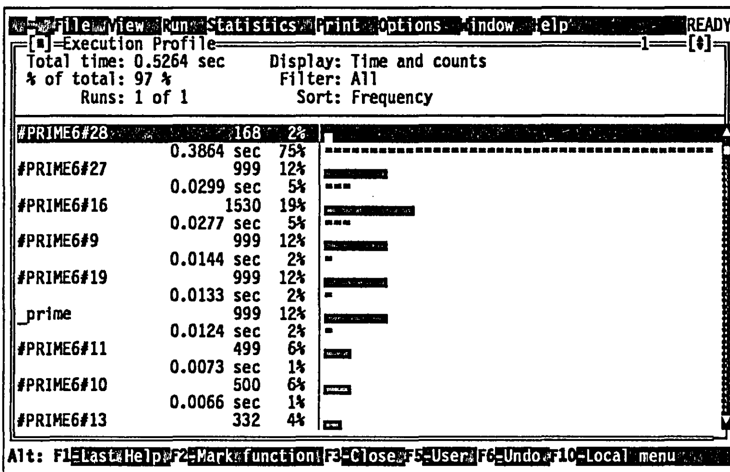
Eliminating CR/LF pairs (PRIME6)

Here's one last change. Instead of printing a carriage-return/line-feed pair after each prime number, try printing just a space. This is the only change made in program PRIME6.

Load PRIME6 (Pascal users load PRIME6PA), set areas for every line, then run it.

Surprise! Eliminating the carriage return/linefeed pair cuts execution time by a factor of almost 7. Apparently, printing new lines is expensive. The distribution of profiles is fairly even for execution times and counts (Figure 1.8). We'd be hard pressed to squeeze more out of this program without substantially changing the algorithm.

Figure 1.8
PRIME6's execution times and counts



Where to now?

We've taken you through the basics of profiling in this tutorial. By now, you should be familiar with using Turbo Profiler: loading and profiling programs, printing the contents of various windows, saving and restoring profile statistics, and rearranging the windows so you can analyze the statistics.

Go ahead and quit the profiler now (choose **File | Quit**, or press *Alt-X*).

For more information about Turbo Profiler's environment, as well as details about parts of the profiler not mentioned here, refer to Chapter 4, the complete Turbo Profiler environment reference.

If you want more challenges than we've given in this tutorial, try these:

- Profile for primes less than
  - 2,500
  - 5,000
  - 7,500
  - 10,000
- Set the profile mode (choose **Statistics | Profiling Options** to bring up the Profiling Options dialog box) to Passive analysis. What does this do to profiler overhead? What kinds of

information do you lose in passive analysis? (See Chapter 3 for information on passive profiling.)

- Find out what kind of performance improvement you get by implementing the Sieve of Eratosthenes to compute primes up to 10,000.
- Compare the cost of printing new lines with calls to position the cursor.

There are a number of articles on the subject of profiling, but not many books. Jon Bentley's book, *Writing Efficient Programs,* provides a summary of rules for designing efficient code, suggests a comprehensive methodology for profiling, and contains an extensive bibliography.

C H A P T E R

2

Inside the profiler

If you want to use Turbo Profiler to your best advantage, you need to understand its inner workings. Knowing what the profiler does when it encounters an area marker or what happens each time the profiler interrupts program execution allows you to fine tune your techniques for specifying the type of information to collect and for interpreting the resulting reports.

Consider the source code in PTOLL and PTOLLPAS:

```
/* ****** PTOLL.C ***** */

#include <stdio.h>
#include <dos.h>


void main()
{
    printf("Entering main\n");
    route66();
    printf("Back in main\n");
    delay(1000);
    highway80();
    printf("Back in main\n");
    delay(1000);
    printf("Leaving main\n\n");
}

route66()
{
    printf("Entering Route 66\n");
    delay(2000);
```

```
{****** PTOLLPAS.PAS ******}

Uses Crt;

procedure Route66;
begin
  Writeln( 'Entering Route 66' );
  Delay(2000);
  Writeln( 'Leaving  Route 66' );
end;


procedure Highway80;
begin
  Writeln( 'Entering Highway 80' );
  Delay(2000);
  Writeln( 'Leaving  Highway 80' );
end;


begin
```

```
    printf("Leaving  Route 66\n");
}

highway80()
{
    printf("Entering Highway 80\n");
    delay(2000);
    printf("Leaving Highway 80\n");
}
```

```
  Writeln( 'Entering main' );
  Route66;
  Writeln( 'back in main' );
  Delay(1000);
  Highway80;
  Writeln( 'back in main' );
  Delay(100);
  Writeln( 'Leaving main' );
end.
```

Setting the areas to All Routines in Module effectively sets up four time-collection compartments and four count-collection compartments.

Figure 2.1
Execution time and count compartments for PTOLL/PTOLLPAS

	Execution Time	Execution Counts
Total	☐	☐
main()	☐	☐
route66()	☐	☐
highway80()	☐	☐

Phantom tollbooths

In this section, you follow program execution and see what happens as it passes each area marker. Think of this process as going through a series of toll booths. When you pass a toll booth, you're on a section of road associated with that toll booth until you come to another toll booth.

You're in free space before you pass the first toll booth and after you leave the last toll booth. Each toll booth knows how long you spend on its road; it also keeps track of how many times you pass by. The only weird thing about this highway is that you can only go one direction down the road: Loops and jumps are like airlifts that take you back to some previous position on the road. As you move down the highway in program PTOLL, think of each area as a stretch of highway with an imaginary toll booth at each end.

Here's how time and count collection works for a typical C or Pascal program.

Before you enter the main program block, C startup code is executed. This is no man's land. Any timer ticks encountered here

are thrown away, unless you have explicitly set an area in the startup code.

As soon as you pass the area marker (toll booth) at **main**, the count associated with **main** increments by 1. Any timer tick that occurs between the time you enter **main** and the time when **route66** is called goes into **main**'s timer compartment.

Next, **main** calls **route66** and you enter a new stretch of highway. The moment execution passes through the area marker (toll booth) at **route66**, several things happen:

- The current area is set to **route66**.
- The compartment for the caller (**main**, in this case) goes on a stack.
- The count-collection compartment associated with **route66** increments by 1.

Any timer tick that occurs between now and the time you return from **route66** automatically increments **route66**'s time-collection compartment. The global program time-collector also continues to increment with each timer tick.

As soon as execution passes through a return point for **route66**, the profiler pops the caller's compartment from a stack. The caller's count compartment is *not* incremented on a return. However, any timer ticks that occur between now and the call to **highway80** are added to the time-collection compartment for **main** as well as to the program's global compartment. To verify this, try turning off **route66**'s area marker and comparing the result with a profile for which that area marker was set. You should see essentially the same total execution time. However, **main**'s execution time should increase by the amount of time it formerly took to execute **route66**.

Determining the overhead of routine calls

You might want to measure the time consumed by *calling* a routine (for example, **route66**) and ignore the time spent *inside* the routine. The easiest way to get there from here is to disable collection of information at the entry point for **route66**, and then to reenable collection upon return from **route66**.(You can also get this kind of information using *passive analysis*, which we discuss in Chapter 3). For now, position the cursor on the first line of **route66**, then choose **O**peration from the Module window local

menu to open the Area Options dialog box. Set Operation to Disable. Press *Enter.*

When you disable collection on entry to **route66**, returning from it doesn't automatically reenable collection. You must set an area marker at the closing brace for **route66**, and set the area operation for that area marker to *Enable* again (in the Area Operations dialog box).

Who pays for loops?

The toll booth analogy helps explain why passing through an area marker and jumping back to an address that precedes that marker (using a loop or a goto statement) doesn't change the current area. Even though you are lexically outside the scope of the marker, you haven't passed through any new markers. Any timer ticks that occur will still be associated with the most recently tripped marker.

```
 /* ****** PLOST.C ****** */

 #include <stdio.h>
 #include <dos.h>
 lost_in_town();

=>void main()
 {
    printf("Entering main\n");
    lost_in_town();
    delay(1000);
    printf("Leaving main\n\n");
    delay(1000);
 }

=>lost_in_town()
 {
    int i;
    printf("Looking for highway...\n");
    delay(100);
    for (i=0; i<10; i++)
    {
       printf("Ask for directions\n");
=>     printf("Wrong turn\n\n");
       delay(1000);
    }
    printf("On the road again\n");
 }
```

```
   {****** PLOSTPAS.PAS ******}

   uses Crt;

=> procedure Lost_in_town;
   var
     I : Integer;
   begin
     Writeln( 'Looking for highway
     Delay(100);
     for I := 0 to 9 do
     begin
       Writeln( 'Ask for direction
=>     Writeln( 'Wrong turn' );
       Writeln;
       Delay(1000);
     end;
     Writeln( 'on the road again'
   end;

=> begin
     Writeln( 'Entering Main' );
     Lost_in_town;
     Delay(1000);
     Writeln( 'Leaving main' );
     Writeln;
     Delay(1000);
   end.
```

In program *plost*, we've complicated the routine **lost_in_town** by using a compound statement inside a loop. Assume that three markers have been set: one for **main**, one for **lost_in_town**, and a line marker for the statement that prints `Wrong turn`.

Things get tricky when you get into **lost_in_town**. When you first enter the routine, **lost_in_town** becomes the current area. The time associated with printing `Looking for highway` is associated with this marker.

Time for executing the loop statement is still associated with the routine marker, and the first time you "Ask for directions," the time is associated with the routine marker. However, once you trip the line marker for "Wrong turn," the remainder of the time spent in the routine is associated with that line marker.

Just because you pass into an area that was previously associated with another marker doesn't mean the current area changes. The current area only changes when you trip an area marker. This can produce unexpected results.

For instance, if you set the three markers for program *plost* as we've already described (one each for **the main program block, lost_in_town**, and the `Wrong Turn` statement), approximately 84% of program time will be associated with printing "Wrong turn," while only 1% of execution time will be associated with **lost_in_town**. This is because nine out of ten calls to "Ask for directions," plus all calls to the subsequent **delay** statement, are associated with the `Wrong Turn` marker.

If you toggle off the area marker for "Wrong turn," 84% of the remaining execution time will be logged to the routine **lost_in_town**.

Consider the following code:

```
main
{
   while(!kbhit() )
   {
      func1();
      statement1;
      statement2;
      func2();
   }
}
```

```
func1()
{
}

func2()
{
}
```

Assume that areas are set for all routines in the module. The routines **main**, **func1**, and **func2** each mark the beginning of an area, and the interrupt timer is ticking away at 100 times a second. (Pretend that 1/100 second is a long time, so you can see what's going on.)

You enter **main**, which trips **main**'s area marker. When this happens, Turbo Profiler internally encounters a breakpoint. This encounter sets a variable indicating that, until you trip another breakpoint, **main** is the current area. This encounter also increments a variable associated with execution counts for **main** by 1.

The scope of these areas is dynamic rather than lexical. That is, **main** is the current area until **func1** is called. As soon as you enter **func1**, you're in a new area until you encounter another function call or until you return from **func1**. This means that the profiler puts the caller (**main**, in this case) on a stack.

When you exit from **func1**, you trip a return marker that the profiler set up when it entered **func1**. The routine **main** becomes the current area again. Any timer ticks that occur while the program is executing *statement1* or *statement2* will update the timer for the area associated with **main**.

Two things are going on here:

1. Every time you encounter an area, the profiler calls an internal routine that adjusts variables and updates a routine call stack. Two variables are associated with each area: execution counts and execution time. Each time you enter an area, the execution count associated with that area increments.
2. Every time a timer tick occurs, the profiler calls another internal routine that checks to see what area is current, then increments the timer variable associated with that area by the appropriate amount of time.

When the program terminates, Turbo Profiler converts the *counts* variable for each area to an actual time (based on the total number of timer ticks that occurred for the entire program).

When you disable collection on entry to routine **func1**, returning from **func1** does not automatically reenable collection. You must explicitly reenable collection at **func1**'s return point.

What do you do about multiple **return** statements? The answer is related to the implicit return points at the end of routines.

Something to keep in mind about return points.

Even though you might have several explicit return points in your program, Turbo C actually turns all returns into jumps to a single exit point at the end of the routine. The line that receives the area marking for a return statement is the line associated with the closing brace for the routine. This is the actual assembly language return statement to which all other return statements in the routine are vectored.

To disable collection for a routine, set an area marker with a disable operation at the first line of the function and an enable on the line after the call to the function.

If you want to throw out the time spent in **func1** but continue collection upon return from **func1**, you must set an area marker at **func1**'s return statement. If no explicit return statement exists, mark the closing curly brace associated with the end of the routine.

Logging callers

An active routine is a routine currently on the profiler's routine call stack. In active analysis, (that is, with the profiler collecting call histories and other data not related to times) Turbo Profiler maintains its own routine call stack. This stack is similar to the stack found in any DOS program. However, the profiler's stack is separate from the user's program stack and is used strictly to retain information about routine calls for which a return statement has not yet been executed.

In order to maintain an active routine stack, Turbo Profiler recognizes two types of area markers:

- Routine-entry area markers (routine markers)
- Normal area markers (label markers)

When the profiler encounters a routine-entry area marker, it pushes the currently active routine (the *last* encountered routine-entry marker) onto its active routine stack. The newly encountered routine marker then becomes the active routine marker.

Now, if a normal area marker trips, this encounter will have no effect on the current routine or on the active routine stack. When a normal area marker trips, it simply becomes the active *area*, which means that the profiler forgets the previously active area. The currently active *routine*, however, remains on the stack until the profiler encounters a return statement.

When a return is issued within an active routine, the area marker associated with that routine becomes inactive. The routine on top of the profiler's active routine stack pops off the stack and becomes the active routine, until a return statement executes within that routine, or until another routine-entry area marker is tripped.

Thus the profiler can maintain a complete call history for every marked routine. If you have enabled **S**tatistics | **C**allers for all marked routines, then each time a routine-entry area marker is tripped, the profiler saves the entire profiler call stack in a buffer linked directly to the routine-entry marker.

If that call stack is identical to a call stack that was saved for a prior entry to this routine, the profiler increments a counter, rather than saving the call stack again. If, however, the call stack is different, the profiler allocates a new buffer and logs the profiler call stack to that new buffer. This makes it possible to maintain a record of every call path to a routine and the number of times each call path is traversed.

The profiler's active routine stack is related to two menu settings:

- **S**tatistics | **C**allers (set to either Enabled or Disabled)
- the **C**allers option for each marked area in the Areas window

You gain finer control over logging call paths by using the local menus of the Module and the Areas windows. You can set the **C**allers option for each of the marked areas separately. Both the **C**allers command on the Module local menu and the **O**ptions command on the Areas local menu lead to a dialog box where you can specify Callers as All Callers, Immediate Caller, or None.

- All Callers means log the entire routine call stack each time the entry point is tripped.
- Immediate Caller means log only the top entry on the routine call stack when the entry point is tripped.
- None means don't log any routine stack information when this routine-entry marker is tripped.

By default, when you first profile a program, the Callers option for all routine-entry area marks is set to None.

Enabling **S**tatistics | **C**allers from the main menu is the same as setting the Callers option to All Callers for each area marker listed in the Areas window. However, once you've hand-set any of the Callers options in the Areas window, setting **S**tatistics | **C**allers to Enable won't change the value of the Callers options for any of the areas.

Disabling the **S**tatistics | **C**allers option at this point tells the profiler not to log any stack information, but doesn't change the Caller settings in the Areas window. Neither does setting **S**tatistics | **C**aller.)

Sampling vs. counting

This happens only in passive mode.

The profiler does not actually measure time: It comes up with a very accurate estimate of time based on information from timer tick counts. This is a form of *statistical sampling*. By taking regular periodic samples of the current area, and by keeping a count for each area (which increments each time that area is active when the timer interrupts), the profiler can estimate the time spent in a given area.

The profiler knows the total time taken to run the program. It also knows the total number of times the timer interrupted the program. The time spent in a given area can be calculated as

$$time_{area} = timer_{total} * counts_{area} / counts_{total}$$

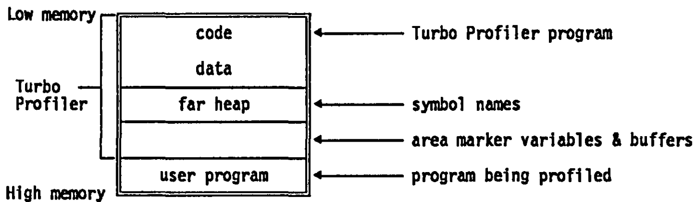
This is *not* the true time spent in an area. If your program iterates over some routine at a frequency that is a multiple of the timer frequency (for example, a routine that generates a steady sound tone), the execution of a particular line (or area) might exactly coincide with most of the timer interrupts. This resonance could occur even though that line is not where the program is spending most of its time. This is rare, but possible.

If you suspect this sort of frequency collision, change the **Statistics | Profiling Options | Clock** *speed value* and compare the resulting profile to the previous one.

Profiler memory use

The following figure maps memory usage when Turbo Profiler is running a program.

Figure 2.2 Memory map for Turbo Profiler



The profiler allocates memory for area information on the far heap. If you add areas while the program is running, the far heap will expand into the user program area to make room for new area variables and buffers. This is why, if you modify areas during a run, you should always reset the program with **Run | Program Reset**. If you don't, the results of a profile might be unpredictable; you could hang your computer.

C H A P T E R

3

Profiling strategies

Improving your program's performance through profiling is not a simple linear process; you don't just profile the program, modify the source code, and call it a day. Profiling for improved performance with Turbo Profiler is dynamic and interactive. You collect statistics, analyze the results in a variety of windows, perhaps change the profiling parameters so you'll get different statistics, profile again, analyze again, modify the source code and recompile, profile again, analyze again, and so on.

If you're not sure at first where the bottlenecks in your program are, go ahead and profile using Turbo Profiler's default settings. When you look at the results in the Execution Profile window, you get an idea of which routines in your program consume the most overall time. By looking at time and count data together, you find out which parts of the program are most expensive in terms of time *per call*. Armed with that knowledge, you can start zeroing in on your program's problem areas.

Turbo Profiler provides several different report windows for analyzing the collected data; you can also print report window contents to paper or disk for a running account of performance improvements. In the report windows, you can look at your program's execution times and counts, file-access activity, DOS interrupts, and overlay activity, along with call histories for routines.

For extensive coverage of profiling in general, there are many articles and books you can refer to.

What do you do with all this power and flexibility? How do you use Turbo Profiler for efficient and effective profiling? And what are the tricks of the profiling trade? Obviously, we can't answer all of these questions in this chapter. We do, however, provide some general guidelines, techniques, and strategies to get you moving.

The first time you run Turbo Profiler on a program, it

- automatically scans through your .EXE file to find the main program module
- sets area markers for the program
- determines what source module contains the main part of your program
- loads that main module into the Module window
- positions the cursor at the main module's starting point

The main module is the one that contains the first source line to be executed in your program. Area markers are "trip points" that mark the locations where you want to gather statistics; the number of markers set depends on the number of symbols found in your program's debug information.

Whenever you exit Turbo Profiler, *it saves information* about the areas you set up for the currently loaded program in an *area file* named *filename*.TFA, where *filename* is the name of your program. Each time you load a program to profile, Turbo Profiler looks for a corresponding .TFA file. If it finds one, it automatically uses the area settings in that file.

It's a good idea to save the results of a profile that takes a long time to run, in case you want to come back and study the results later.

You can also save the results of a profile to a .TFS file with the Statistics | Save command. By default, the file name assigned to a statistics file is *filename*.TFS. You can use the default or change the name (in case you want to save more than one set of statistics for a single program).

Preparing to profile

The examples in Chapter 1 are small and simple; we designed them to show the general process of profiling. The problem in that chapter was to optimize the routine **prime**, rather than to identify specific program bottlenecks.

However, you actually need a profiler more when you're writing very large programs, rather than small ones, because you must identify which program fragments are bottlenecks *before* you can figure out how to optimize any given fragment. In many ways, it's easier to find the bottlenecks than it is to figure out what to do about them.

Before profiling your program, adjust your source code so the profile statistics gathered are useful and sufficient. Once the source code is modified (or if it doesn't need to be), compile the program with debug information turned on. Then set the markers that tell the profiler where to collect statistics and what kind of statistics to collect.

Adjust your program

The first thing to do is set up your program in a way that lets you find out what you need to know from the profile. For example, if you're writing an interactive program that gets a lot of input from the keyboard, you don't need to find out that most of your time is spent waiting for the user to press a key.

Here are some basic techniques for finding bottlenecks in large programs:

Selecting input data is important.

- Select data sets large enough to give you a useful profile.

  A string search program on a three-line file won't tell you very much. Likewise, searching for a short string found in nearly every line of a 10,000-line file will give a different kind of profile than searching for a long string found only once in 10,000 lines.
- If you know your program runs quickly, set the profiler to collect statistics over several runs. (Modify the Run Count setting in the Profiling Options dialog box.)
- Modify the program to work independent of keyboard input, or remove any areas that have code doing keyboard input.

  Read data numbers from a file or use a random number generator to stuff numbers in an array. The main idea is to select data that's typical of the real-world data the module operates on.
- Isolate the modules of the program you know need improvement.

Compile your program

After you've adjusted your program so that the profiling session won't become a wild goose chase, compile it again with debug information turned on.

Files that you've compiled for debugging with Turbo Debugger can be handled by Turbo Profiler without recompilation.

To use Turbo Profiler with Borland products, you must have Turbo Pascal 5.0 or later, Turbo C 2.0, Turbo C++, or Turbo Assembler 1.0 or later. You must compile your source code with full symbolic debugging information turned on.

- **Turbo Pascal**: Standalone Debugging and Debug Information must be set to *On*.
- **Turbo C++**: The Standalone radio button must be selected.
- **Turbo C**: **S**tandalone must be specified in **D**ebug | **S**ource Debugging.
- **Turbo Assembler**: Source code must be assembled with the **/zi** command-line option and linked with TLINK, using the **/v** option.

You can also run Turbo Profiler programs compiled with a Microsoft C compiler or assembled with MASM, if you convert them with TDCONVRT or TDMAP. (See documentation for Turbo Debugger utilities included in the MANUAL.DOC file on disk.)

To run Turbo Profiler, you need both the .EXE file and the original source files. Turbo Profiler searches for the source files in these directories, in this order:

1. in the directory in which they were found at compile time (this information is included in the executable file.)
2. in the directory specified with the **O**ptions | **P**ath for Source command
3. in the current directory
4. in the directory containing the .EXE program being profiled

Set profile areas

Once you've adjusted your program so you can concentrate on the troublesome areas and compiled it with debug information turned on, you're ready to run it through the profiler and collect statistics for individual areas. You can start out by profiling your

whole program in general, then focus in more and more detail as you find the trouble spots. Start by accepting the default area settings—Turbo Profiler sets default areas based on the density of the symbols it finds appended to the executable file.

An *area*, remember, is a location in your program where you want to collect statistics: An area can be a single line, a construct such as a loop, or an entire routine. An *area marker* sets an internal breakpoint. Whenever the profiler encounters one of these breakpoints, it executes a certain set of code—depending on the options that you've set for the area in question. This profiling code could be a bookkeeping routine or a simple command to stop program execution.

These are the actions the profiler can perform when execution enters an area:

Operation	What it does
Normal	Activates the default counting behavior (collects execution time and counts for all marked areas).
Enable	Turns on the collection of statistics (if they've been previously disabled).
Disable	Turns off the collection of statistics, but lets your program keep running. When your program enters an area where the action is set to *Enable,* the profiler resumes data collection.
Stop	Stops the program, and returns control to the Turbo Profiler environment. At that point, you can examine the collected statistics, then resume execution.

By default, Turbo Profiler counts the number of times execution enters an area and how long it stays there. You can change what the profiler does when an area executes by setting the Operation option in the Area Options dialog box—accessed through the Module or Areas window local menus.

When you're setting areas in your program before running a profile, you should consider these questions:

- How many areas should statistics be collected for?
- Which parts of the program should be profiled?
- What should happen at each marked area?

What level of detail do you need?

You must first decide how much information you want. Keep in mind how large your program is and how long it takes to run.

- For a small program, you probably want statistics for every executable line—the maximum level of detail.
- For large programs, you need less detail; just profiling the amount of time spent in each routine is probably enough.

"Large" is a bit vague: You need to take into account the number of modules of source code, the number of routines, and the number of lines.

If your source consists of 10,000 lines in ten modules, you should probably analyze only one module at a time in active analysis. (Your program is factored into discrete functional modules, right?)

On the other hand, if your program is less than 100 lines and you need detailed analysis, you probably want to collect statistics for all lines.

If your program runs in less than five seconds, you'll get more accurate profile results if you set up multiple runs with averaged results. (Set the number of runs with the **S**tatistics | **P**rofiling command.) If the program takes an hour to run (not counting profiler overhead), be careful not to set so many areas that you slow down execution to an unacceptable crawl.

You divide your program into a number of areas by selecting **A**dd Areas from the Module window's local menu, then run your program to accumulate statistics for each area.

If you don't tell Turbo Profiler how to divide your program, it uses a default scheme to intelligently select appropriate areas in your program. Based on information it finds in a program's symbol tables, Turbo Profiler selects one of several default options for setting areas in a program.

- If there are few symbols in the table, and there is a single module, Turbo Profiler selects **E**very Line in Module as the default area setting.
- If there are many symbols and several modules, Turbo Profiler selects **A**ll Routines as the default area setting.

Suggestion If your program is very large, profile it first in passive mode to get the big picture, then select areas for more detailed analysis.

What type of data do you need?

For each area in your program, Turbo Profiler accumulates the following default information:

- the number of calls to the area
- how much time was spent in the area (active mode)
- how many clock ticks occurred while the area executed (passive mode)

You can also collect more extensive information during the profiling session.

- By enabling **S**tatistics | **C**allers and setting Call Stack options in the Area Options dialog box, you can track which routines call a marked routine—how often and through what pathway.
- With the **S**tatistics | **F**iles option enabled, you can monitor your program's file-access activity.
- The **S**tatistics | **I**nterrupts option, when it is enabled, records your program's interrupts.
- You can monitor your program's overlay file activity by enabling the **S**tatistics | **O**verlays option.

Once you've enabled the appropriate **S**tatistics menu options, you can open the corresponding profile report windows (through the **V**iew menu), then call up each window's local menu to specify details about how you want the data collected.

Remember, to get the Turbo Profiler reports you want, you need to set options *before* you run the program.

When should data collection start?

Often, you only want to collect timing information when a certain portion of a program is running. To do this, start the program executing without collecting any information; set the **S**tatistics | **A**ccumulation option to *Disabled*. You can determine the Accumulation option's setting at any time by bringing up the **F**ile | **G**et Info box and checking the status of Collection.

With **A**ccumulation disabled, you must set an area marker to *Enable* for the area where you want data collection to start, then set another marker to *Disable* for the area where you want data collection to stop. The actual number of start and stop points you set is determined by the amount of available memory; generally, you can set as many as you need.

How do you want time data grouped?

The profiler can keep each routine's execution-time statistics separate from others, or it can combine routines' times with those of the routines calling them.

By default, as soon as an active routine calls a routine that has an area marker, the profiler puts the calling routine on the call stack and makes it inactive. The profiler associates any timer counts made while program control is in the routine with that routine only, not with the caller.

However, if you specify that the caller should use a combined clock (rather than a separate clock), the profiler associates timer ticks that occur while control is in the routine with both the routine and the caller.

- If routine **A** makes no routine calls to other routines, it (routine **A**) won't appear as an area in the Execution Profile window. Instead, the routine that called **A** appears with a time equivalent to its own execution time *plus* the time of routine **A**.
- If routine **A** does call other routines, it (routine **A**) appears as an entry in the Execution Profile window. The time associated with routine **A** is the time required to execute **A**'s routines, but not **A**'s *self time* (the time spent in its own execution).

Turbo Profiler's default analysis mode uses a separate timer for each marked routine. So normally, the time spent in a routine is measured exclusive of calls to routines. If you want a routine's time data to include time spent in routines, choose Combined under Timing in the Areas window's (Options) dialog box.

Which data do you want to look at?

It's important to know how to control the amount of information Turbo Profiler collects and subsequently displays, particularly if you want detailed information about just part of a large program. Turbo Profiler provides two ways to control how much information you view about your program:

- Before you profile, you can limit the collection to specific areas and types of data by setting options and parameters.
- After the profile, you can filter the collected statistics (without erasing any) and display only the data you're currently interested in.

In the Module, Areas, and Interrupt windows, you can specify which parts of your program you want Turbo Profiler to collect

information about, and how much information to collect. You can choose to make data collection as coarse as all routines in a module or as fine as a single statement. You can choose to collect time-related data only (*by setting the analysis mode to Passive*), or you can choose to collect the full gamut of data, including complete call-stack histories, all file-access and overlay activities, and all DOS interrupt calls. You can slow down or speed up the profiler's timer, thus decreasing or increasing the resolution of data collected (passive mode only).

There's a basic tradeoff in how much data you choose to collect: The more information Turbo Profiler collects, the slower your program runs and the more memory it needs to store the collected statistics.

Once you've collected the data, you can use commands in the profile report windows to temporarily exclude the data you don't want to look at from the displayed statistics. (See page 56 for more information about filtering displayed statistics.)

Profiling your program

You might not know if a profile is worth saving until you look at several Execution Profile windows.

Once you've selected the areas, run the profile. You can save the resulting profile with the **S**tatistics | **S**ave... command. This command saves the statistics to a .TFS (Turbo Profiler Statistics) file. If you plan to save several different profile results, use some file-naming convention that uniquely identifies each of the runs (for example, RUN1.TFS, RUN2.TFS, and so on). This simplifies your task of comparing them later.

After you save the .TFS file, you can study the profile's results in the profile report windows, sorting and filtering the displayed data as you explore their meanings. You won't lose any area markers or statistical reports, because all this information can be reproduced (simply restore the profile from the .TFS file). In general, if a profile took a long time to create, save it unless you're absolutely sure you won't need it.

What are you trying to find out?

Normally, programmers use a profiler to get answers to one or more of these questions:

- How efficient is this algorithm? (*Algorithm testing*)

- Is this program doing what I think it is? Is all of it running? (*Verification and testing*)
- How long does each routine run? How much time does the program spend using various resources? (*Execution timing and resource monitoring*)
- What's the structure of this code? (*Code structure*)

The following table relates why you're profiling to the type of information you're likely to gather.

Table 3.1
Ways of using a profiler

Purpose of profile	Type of information gathered
Algorithm testing	Line-count information Dynamic call history
Program testing and verification	Execution-count at the routine level (possibly at line level) Dynamic call history
Execution timing and resource monitoring	Execution time Execution counts Interrupt activity File-access activity Overlay activity
Program structure analysis	Dynamic call history File-access activity Execution profile (time and counts) Interrupt activity Overlay activity

Testing algorithms

If you're analyzing an algorithm, you'll probably concentrate on a small number of routines, so line count information matters more than execution times do. You need to do the following:

1. Isolate the algorithm and its supporting routines by marking them as an area.
2. Make sure you've set area markers for all lines in all routines that implement the algorithm in question.

The examples in Chapter 1 demonstrate algorithm analysis, especially as it relates to execution time statistics.

Verifying and testing programs

In program verification and testing, line-count information is more pertinent than execution times. But since the verification and testing process looks at the program as a whole, you want to see how everything works together in an integrated system.

Profiling a program while you run it through standard tests can point out areas of the program that execute very little or not at all. For example, by studying *call paths* in the Callers window and printing out a source-code listing (annotated with execution counts) from the Module window, you can verify that every statement in your source code has actually executed.

Because you deal with large pieces of code when you test and verify programs, you don't need as much detail as you do for algorithm analysis. However, it's still useful to know how many times a routine has been called. And, if you want to organize the test down to groups of routines that constitute some hierarchy, the execution-count information can help prove that every path in a switch statement or a conditional branch has executed at least once.

Timing execution and monitoring performance

For timing a large program to see where it's slow, you rarely need information at the line-count level. In execution timing, you need to know two things:

1. how much time is spent in individual routines
2. what times propagate from low-level routines to higher-level routines

Before timing a program's execution, you need to set areas for all routines with source code. In very large programs, limit your selection of area markers to a single module.

Once you've set the area markers in a single module, profiling becomes a matter of successive grouping and refinement. These are techniques you use to refine the profiling process:

- Use filters to temporarily mask out unwanted information (with the Execution Profile window's local Filter command).
- Unmark routines whose statistics you don't want (with the local Remove command in the Module, Execution Profile, and Areas windows).
- Combine the timer counts for specified routines (with the Timer option, which you set from either the Statistics | Profiling Options... command or the Areas window's local Options... command).

If you're not completely familiar with the program you're profiling, you can use execution timing and performance monitoring in conjunction with studying the unfamiliar code.

Studying unfamiliar code

One of the best ways of studying code you don't know is to analyze the dynamic call history that Turbo Profiler generates in the Callers window. This history shows the program's structural hierarchy. Although you can see only one routine's *call paths* at a time in the Callers window, you can print all recorded call paths by choosing **P**rint | **S**tatistics with the Callers window open.

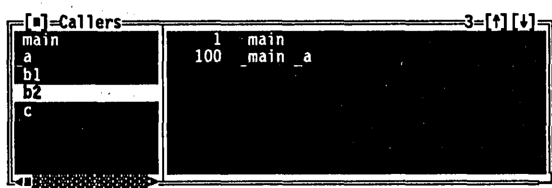


Figure 3.1
The Callers window

By noting a program's called routines, their callers, and the number of times the program traverses each call path, you can see which routines are most important. You can also predict which higher-level routines will be affected by changes you make to lower-level routines.

By looking at execution times and counts, you can get a sense of the program's important routines. File and overlay monitoring reveal any temporary files opened and closed during program execution as well as any overlays swapped into memory. This information is harder to find through lexical program analysis.

The profiler's link between the Execution Profile, Module, and Areas windows enables you to move back and forth quickly to specified symbols, thus revealing the connections between functionally related but physically separated pieces of source code.

Which analysis mode do you use?

One important consideration when you're profiling is whether to use active or passive analysis. You set the mode under Profile Mode in the Profiling Options dialog box (choose **S**tatistics | **P**rofiling Options). Turbo Profiler's default mode is active analysis; it collects execution times and execution counts automatically, as well as any other data (such as call histories or DOS interrupts) that you've enabled in the Statistics menu. If your program runs very slowly and you can do without execution counts and call histories, use passive analysis; in passive mode, the profiler collects only time-related statistics for areas (such as execution times,

interrupt calls, and file activity), and your program runs much faster.

Active analysis

See the section "How to speed up profiling" below for other ways to make your profiling go faster.

When you profile in active mode, it matters how frequently program execution trips area markers. For instance, you can mark every line in a program except a loop statement, but if the program spends 95% of its time inside that loop, the number of areas set won't slow the program much.

The profiler slows down program execution if it must perform a lot of bookkeeping every time it executes a source statement. If that happens, you can always switch to passive analysis, which turns off all automatic calls to expensive bookkeeping code the profiler normally makes each time program execution trips an area marker.

Passive analysis

In passive analysis, Turbo Profiler interrupts your program's execution at regular intervals to sample the value of the program counter, CS:IP. If the sampled value points to an address inside an area that you're monitoring (a marked area), the profiler increments the ticks in that area's timer compartment. If the value in the CS:IP does not point to an address inside a marked area (for example, it points to an address within a DOS interrupt or BIOS call), the profiler throws out that timer tick.

It's hard to interpret the results of passive analysis unless your program runs a long time, or unless you accumulate timing statistics over many runs. Some areas of your code might never show up even though they execute, because they're never being executed at the time the profiler interrupts the program's execution.

Passive analysis doesn't add noticeable overhead to program run time, but it does sacrifice some detail in the resulting reports.

When you set passive analysis, there is no noticeable slowdown in program execution. However, you might not be able to get all the information you require. You can't get count information or callers information, but you can monitor interrupt calls and file activity.

When you're profiling in passive mode, you'll get greater statistical accuracy by running your program several times (set Run Count in the Profiling Options dialog box to a value greater than 1).

Some things to watch out for

Some of the data you collect under passive analysis might be misleading if you don't take these points into consideration when you analyze the results:

- If your program does disk I/O, the profiler attributes file-access time to the calling routine under active mode, but not under passive mode.
- If your program calls an interrupt that's not marked as an area, the profiler attributes the interrupt's time to the calling routine in active mode, but tosses out the interrupt's time in passive mode.

Profiling object-oriented programs

In general, profiling object-oriented programs is not much different from conventional profiling. You can treat them just like ordinary programs and consider each method to be just like a call to a routine.

How to speed up profiling

Each time your program enters a routine that you have defined as a data-collection area, Turbo Profiler must perform certain processing ("bookkeeping" code). The execution speed of a program under the control of Turbo Profiler depends on how frequently area markers are tripped and on the kind of information being collected for the most frequently tripped areas. The greater the level of information being collected (particularly call stack history), the longer it takes to execute bookkeeping code associated with an area.

Even if your program runs slower, Turbo Profiler still keeps track of timing information properly.

Sometimes your program speed might be unacceptably slow under Turbo Profiler. That might be because your program is frequently calling a deeply nested routine with call-stack tracing set to *All Callers* for *All Areas*. If you've defined this deeply-nested routine as an area, Turbo Profiler will spend a lot of time keeping track of the calls to it.

To determine if your program is frequently calling a low-level routine, switch to the Execution Profile window and display the areas by execution counts. (Set the local menu **D**isplay option to *Counts*.) This displays an execution-count histogram sorted by the number of times each area is executed.

If the program calls one or more routines much more frequently than the rest, you can exclude them from the list of displayed areas with the Execution Profile window's local **F**ilter | **C**urrent command. You can also unmark areas with the local **R**emove command in the Module, Areas, and Execution Profile windows.

How to improve statistical accuracy

If you don't collect enough data (because your program runs too fast for the profiler to gather a statistically significant number of data points) or if you collect a skewed data set (because of resonance; the profiler's timer-tick frequency coincided with the execution frequency of some part of your program), you won't be able to make informed decisions about the changes needed in your source code. Here's what to do if either of these problems should occur.

Insufficient data

To improve the accuracy of timing statistics and to get a statistically significant average, run your program more than once, using the Run Count option of the Profiling Options dialog box. When your program terminates and you run it again, the profiler adds the times for the new run to times accumulated for previous runs. This continues until you've run your program the number of times specified in the Run Count option.

Resonance

If resonance is causing the profiler to return inaccurate data, use the Clock Speed setting in the Profiling Options dialog box to set the profiler's clock tick speed anywhere between 18 and 1,000 ticks per second. Choose a speed that is not an integral multiple or fraction of the speed that is causing the resonance. For example, if your program exhibits resonance at 100 ticks per second, try 70 or 130 ticks per second. (If you suspect that resonance is causing biased statistics, try different clock speeds that are not integral multiples, and compare the collected statistics. If resonance is the problem, the various sets of statistics will vary considerably.)

The faster the clock speed, the more accurately Turbo Profiler can determine where your program spends its time.

Changing the clock speed can only be done in passive mode; active mode doesn't use clock ticks.

Will setting the clock speed to 1000 ticks per second produce incredibly accurate timing information? Not necessarily. The faster you set the clock speed, the slower your program will run (because Turbo Profiler must perform certain lookup operations each time a clock tick occurs). So if you want greater accuracy than the default 100 ticks per second, increase the clock speed until you reach an acceptable compromise between accuracy and execution speed.

Some tips for profiling overlays

Overlays allow large programs to run in limited memory by storing portions of the code on disk, and loading that code only as needed. If you use overlays, the program's modules share the same memory—thereby reducing total RAM requirements.

Unfortunately, swapping code in and out of memory can lead to slow program execution because it wastes time accessing disk drives. Because even a fast disk drive is still the slowest storage device in most PCs, improper overlay management can dramatically reduce performance. To make a difficult situation worse, the overlay manager code in the compiled program is normally hidden. Turbo Profiler brings overlay management code out in the open so you can adjust your program's overlay behavior.

To fine-tune overlay performance, you need to choose the right overlay buffer size, select algorithms for managing overlays in the buffer, and set other parameters that can help keep the most frequently-used overlay modules in memory for longer periods of time. You can reduce "thrashing", which results from too many disk accesses as the program reads overlay files, by keeping frequently used overlays in memory longer.

Statistics displayed in the profiler's Overlay window include

- the number of times your program loads each overlay from disk
- the time-ordered event sequence in which your program loads overlays

The load-count and execution-time information is useful for determining which overlays should stay in RAM longer. By comparing this data with a profile of non-overlay routines, you can decide which modules should be overlays and which shouldn't.

With the overlay event history, you can choose optimal algorithms for overlay buffer management. By examining a list of overlays and seeing when and how often each was loaded, you can decide which main program modules might work better as overlays, and which overlays might benefit from being made part of your main program.

Interpreting and applying the profile results

OK, so you've decided what profile statistics you want to collect, adjusted your program accordingly, and run it enough times to gather a statistically significant (if not downright daunting) set of data. Now what?

Now comes the fun part. First you analyze the data to figure out what the profiler is telling you, then you apply that new-found knowledge to your source code to make your program faster and more efficient than ever.

How to analyze profile data

The Turbo Profiler windows you'll use to study the collected statistics fall into two categories: *program source* windows and *profile report* windows.

Turbo Profiler's program source windows are the Module, Areas, Routines, and Disassembly (CPU) windows. Before running the profile, you mainly use source windows to set areas and to specify profiling actions at the marked areas. After you examine the profile statistics (in one or more report windows), you use source windows again to analyze your program's source code.

Turbo Profiler's report windows are the Execution Profile, Callers, Overlays, Interrupts, and Files windows. You use report windows to display profile statistics gathered from your running program, so you can evaluate the collected data and determine where changes in the source code might improve your program's performance.

Execution Profile window

This window will be your primary focus for improving the performance of your program. In general you will want to examine those lines of source code which account for most of the program execution time. Next, look for lines (or routines) which have a high ratio of execution time to execution count. And finally, it is always good form to check on the routines which account for the most "per call" execution time.

Callers window

Once you have isolated a routine that you wish to improve, use the Callers window to locate all of the areas in your program which call the selected routine. The Callers window displays the number of times the routine was called, and the source of those calls (the caller).

Overlays window

The Overlays window will allow you to detect excessive overlay calls which will then become candidates for placement into a non-overlaid module (unit).

Interrupts window

The Interrupts window will reveal all of the (selected) interrupts made by your program. This revelation may prompt you to combine video output for some lines of code. Or for file-intensive programs, suggest that disk I/O be buffered.

Files window

The Files window quickly discloses the number of reads and writes performed to the files manipulated by your program. In I/O intensive applications this window will point out very quickly which files deserve your attention.

How to filter collected data

The Execution profile report window provides local menu commands for temporarily or permanently filtering data out of the current display. Here's a table summarizing the profiler's filtering options:

Table 3.2
Local menu commands for filtering collected statistics

Window	Local menu command	What it does for you
Execution Profile	Filter	Temporarily removes the current area's statistics, or shows only the current module's statistics, or restores all collected statistics to the window. (You choose **C**urrent, **M**odule, or **A**ll from the Filter menu.)
	Remove	Permanently erases the current area's statistics from the collected data. ***Use with caution!***
Files	**C**ollection (top pane)	Disabled, disables file statistics collection.

Table 3.2: Local menu commands for filtering collected statistics (continued)

	Detail (top pane)	Disabled, displays only file *open* and *close* activities. Enabled, also displays file *read* and *write* activities.
	Display...	Displays each event either as a bar graph element, or as text showing the exact time and duration of the event.
Interrupts	**R**emove (top pane)	Removes the currently selected interrupt from the top pane.
	Display (bottom pane)	Displays an interrupt's statistics as either (1) summary histograms of time, calls, or both, or (2) a detailed sequence of events.
Overlays	**D**isplay	Displays each overlay's profile statistics as either (1) Count, a summary of memory consumed and times loaded, or (2) History, a detailed sequence of events, with a line of data for every time the overlay loaded.

When you choose **R**emove from the Execution Profile's local menu to permanently filter out an area's statistics, the profiler

- adjusts the report by discounting time spent in that area
- adjusts the percentages of remaining areas by calculating them as percentages of the revised total time

  (*revised total time = total profile time — time for the removed area*)
- unmarks that area in the Module window
- removes the area from the areas list in the Areas window

Revise your program

Here is a general plan of attack for finding routines where simple changes in control constructs can improve your program's performance.

1. Look for large routines with a disproportionate share of execution time, or for routines with a large number of calls. Working from the highest level of your program, follow flow of control through successive levels of calls, looking for places to

optimize by reducing or eliminating excessive calls and operations.

2. Look for statements and routines that have a high ratio of time to count. From the Execution Profile window's local menu, set **D**isplay to **B**oth or **P**er Call. Then look for those areas that show a long time magnitude bar and a short count magnitude bar. Statements and routines of this sort usually represent an inefficient segment of code. Recode them to produce the same result in a more efficient way.
3. As a last resort, you can optimize the program's innermost loops; here are some techniques:
   - unroll loops
   - cache temporary results calculated on each iteration
   - put calculations for which results don't change outside loops.
   - hand-code assembly language

Usually you'll see less improvement with inner-loop optimization than you'll see if you modify control constructs, algorithms, or data structures.

Besides these three general procedures, here are some specific things you can do to improve your program's performance:

- Modify data structures and algorithms
- Store precomputed results
- Cache frequently accessed data
- Evaluate data only as needed
- Optimize loops, procedures, and expressions

Modify data structures

Use more sophisticated data structures or algorithms. A QuickSort routine will generally operate faster than a bubble sort for a random distribution of key values. Consult a book on data structures and algorithms for other examples.

Switch from real numbers to integers for fast calculations, such as window and string management for screen I/O and graphics routines. Use long integers for data manipulation or any other value that does not require floating point precision.

Instead of sorting an array of lines of text, add an array of pointers into the text array. All text access occurs via the pointers. To sort or insert a new line of text, you only need to reorder the pointers, rather than entire lines of text.

Store precomputed results

Build a precomputed sine table, then look up sine as a function of degrees based on an integer index.

Cache frequently accessed data

C buffers low-level character input from files. The **getc** routine reads a whole sector of bytes from the disk into a buffer, but returns only the first character read. The next call to **getc** returns the next character in the buffer, and so on until the buffer is empty, in which case **getc** reads another sector in from disk. The *Read* routine does exactly the same thing in Pascal.

Turbo Pascal has the SetTextBuff routine, which can also help to reduce disk accesses. By use of this routine to allocate a large text buffer on the heap, you can reduce disk file access for text.

In an interactive editor or file-dump utility, you can keep a number of buffers that are updated while the program waits for user input. You might have two buffers that always contain screenfuls of information read from the beginning and the end of the file. Another two buffers can keep the previous and next screenful of bytes in the disk file relative to the position currently onscreen. This way, for those file-navigation commands the user is most likely to select, your interactive program can update the screen without disk access.

Evaluate data as needed

Structure the order of conditional tests and switches so that those most likely to yield true results are evaluated first.

For a large table of lookup information, evaluate entries only as you need them, and use a supplemental array to track entries that have already been computed.

You might only need to calculate the length of a line when you need to reformat output—not each time a new line is read from a file.

Optimize existing code

Loops, procedures, and expressions all offer potential for improvement.

Loops

- Whenever possible, move calculations outside of loops. Repeatedly calculating the same value inside a loop is both time-consuming and unnecessary.

- Store the results of expensive calculations (use **S**tatistics | **S**ave Option).

  For example, an insertion sort routine doesn't need to swap every pair of numbers as it works up an array. If you save the value of the starting element, the inner loop only needs to move the successive element down as long as that element is less than the starting one. When this test fails, you insert the stored value at the current position. This process replaces the expensive swap operation for each element called for in the traditional insertion sort algorithm.
- If two loops perform similar operations over the same set of data, combine them into a single loop.
- Reduce two or more conditional tests in a loop to a single test, if possible.

  For example, add an extra element to an array and initialize it to some sentinel value that will cause the loop test to fail. (This is how C handles text strings.)
- Unroll loops.

  For example, replace this

  ```
  for (x = 0; x < 4; x++)
     y += items[x];
  ```

  with this

  ```
  y += items[0];
  y += items[1];
  y += items[2];
  y += items[3];
  ```

Routines

- Rewrite frequently called routines as inline routines, or replace their definitions with inline macros.
- Use coroutines for multipass algorithms that operate on large data files. (See the **setjmp** and **longjmp** routines in C.) (In Pascal, investigate *procedural types* that allow you to use procedures and functions much like variables to execute coroutines.)
- Recode recursive routines to use an explicitly managed data stack.

Expressions

- Use compile-time initialization.

- Combine returned results in a single call.

  For example, write routines that return *sine/cosine*, *quotient* and *remainder*, or *x-y* screen coordinates as a pair.
- Replace indexed array access with pointer indirection.

Wrapping it up

In this chapter, we've covered most of the things you need to consider before, during, and after a profiling session. We've explained how to prepare your program, and yourself, for the profile; we've given you some hints and caveats about the process of profiling; and we've given you some ideas about how to apply the results after you've run the profile. In the next chapter, we describe each menu item and dialog box option in the Turbo Profiler environment.

C H A P T E R

4

The Turbo Profiler environment

Turbo Profiler makes it as easy and efficient as possible for you to profile your programs. When you start Turbo Profiler, everything you need is literally at your fingertips. That's what an *environment* is all about.

The Turbo Profiler environment also boasts these extras to make program profiling smooth:

- multiple, movable, resizable windows
- mouse support for any mouse compatible with the Microsoft mouse version 6.1
- dialog boxes to replace multilevel menus

Part 1: The environment components

There are three visible components to the integrated environment: the menu bar at the top, the window area in the middle, and the status line at the bottom. Many menu items also offer *dialog boxes*. Before we discuss each menu item in the environment, we'll describe these more generic components.

The menu bar and menus

Turbo Profiler has global and local menus. Global menus are ones you access via the menu bar, and local menus are ones you access from within a window.

The menu bar is your primary access to all the global menu commands. In addition, it displays a program activity indicator on the right side that tells, for example, whether the profiler is READY for you to do something, RUNNING your program, or WAITing while it processes a processor-intensive task. The only time the menu bar is not visible is when you're viewing your program's output in the user screen.

Choosing menu commands from the keyboard

Here is how to execute global menu commands using just the keyboard:

1. Press *F10*. This makes the menu bar *active*, which means the next thing you type pertains to it, and not to any other component of the environment.

   You see a highlighted menu title when the menu bar is active. The menu title that's highlighted is the currently *selected* menu.
2. Use the arrow keys to select the menu you want to display. Then press *Enter*.

   To cancel an action, press Esc.

   As a shortcut for this step, just press the initial letter of the menu title. (For example, press *F* to display the Files menu.)

   If an ellipsis (...), follows a menu command, choosing the command displays a dialog box. If an arrow (▶) follows the command, the command leads to another menu.
3. If the command opens another menu, use the arrow keys again to select the command you want. Then press *Enter*.

   Again, as a shortcut, you can just press the highlighted letter of a command to choose it, once the menu is displayed.

   At this point, Turbo Profiler either carries out the command, displays a dialog box, or displays another menu.

In addition to the global menus that you access through the menu bar, each of Turbo Profiler's windows has its own unique *local menu* (or menus). When you're in a window, press *Alt-F10* to bring up the local menu.

When the local menu pops up, use the arrow keys to select the command you want and press *Enter*, or press the highlighted letter. Once you choose a local menu command, Turbo Profiler either carries it out, displays a dialog box, or displays another menu. To activate a local menu item directly (without bringing up the menu), press the *Alt-(letter)* hot key, where *letter* is the menu item's highlighted letter.

Choosing menu commands with the mouse

To choose commands from global menus with the mouse, click the desired title on the menu bar to display the menu, then click the desired menu command. You can also drag straight from the menu title down to the menu command. Release the mouse button on the command you want. (If you change your mind, just drag off the menu; no command will be chosen.)

To choose the active window's local menu commands, click the mouse's right button to pop up the local menu, then click the desired menu command.

Shortcuts

Turbo Profiler offers many quick ways to choose menu commands. For example, with a mouse you can combine the two-step process into one: Drag from the menu title down to the menu commands, then release the mouse button when the command you want is selected.

From the keyboard, you can use keyboard shortcuts (or *hot keys*) to access the menu bar and choose commands. Here's a list of the shortcuts available:

Press this shortcut...	To accomplish this...
Ctrl and the highlighted letter of the local menu command	Carry out the local menu command
Alt plus the highlighted letter of the menu command	Display a menu from the menu bar
The highlighted letter of the dialog box component	Execute that menu command or select that dialog box component
The hot key combination listed next to a menu command.	Carry out the menu command

Turbo Profiler windows

Most of what you see and do in the Turbo Profiler environment happens in a *window*. A window is an area of the screen that you can move, resize, zoom, layer, close, and open.

You can have many windows open in Turbo Profiler (memory allowing), but only one window can be *active* at any time. The

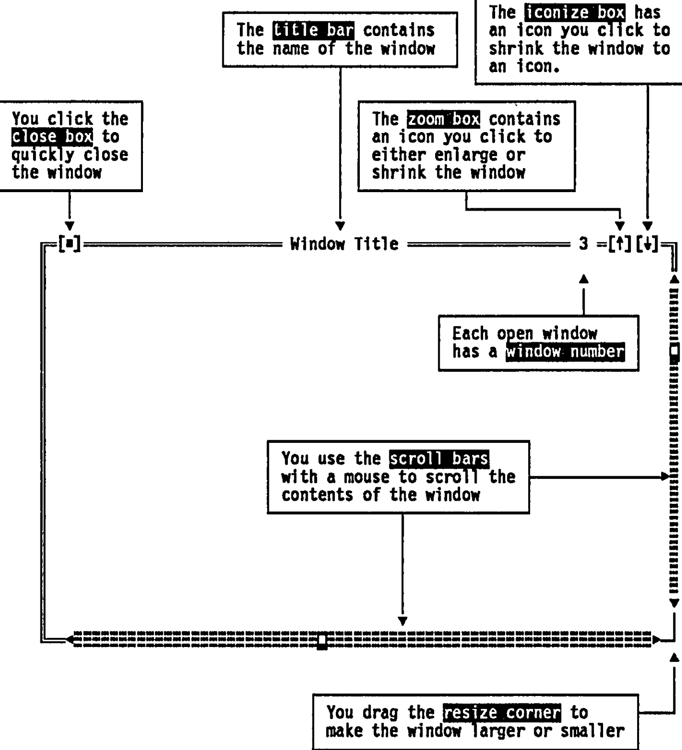
active window is the one that you're currently working in. Any command you choose or text you type applies only to the active window.

Turbo Profiler makes it easy to spot the active window by placing a double-lined border around it. The active window always has a *close box*. If your windows are overlapping, the active window is also the one on top of all the others (the front most one).

There are several types of windows, but most of them have these seven things in common: a title bar, a close box, two scroll bars, a resize corner, a zoom box, an iconize box, and a window number (1 to 9).

This is what a typical Turbo Profiler window looks like:

Figure 4.1
A typical window



Window management

Some windows are divided into two or more panes for displaying different kinds of information. Individual panes often have their own local menu.

The following table provides a quick rundown of how to handle windows in Turbo Profiler. You can perform these actions with a mouse or the keyboard.

Table 4.1: Manipulating windows

To accomplish this...	Use one of these methods...
Open a window	Choose View to open a profiler window that's not already open.
Close a window	Choose Close from the Window menu or press *Alt-F3*, or Click the window's close box.
Activate a window	Click anywhere in the window, or Press *Alt* plus the window number (1 to 9, in the upper right border of the window), or Choose Window and select the window from the list at the bottom of the menu, or Choose Next from the Window menu (or press *F6*) to make the next window active (next in the order you first opened them).
View the window's contents	Use the cursor keys to scroll the window up and down or left and right, or Use the mouse to operate the scroll bars: ■ Click the direction arrows at the ends of the bar to move one line or one character in the indicated direction. ■ Click the gray area in the middle of the bar to move one window size in the indicated direction. ■ Drag the scroll box to move as much as you want in the direction you want.
Move the active window	Drag its title bar, or any border character (=)s that is not a scroll bar. Choose Size/Move from the Window menu (or press *Ctrl-F5*), use the arrow keys to place the window where you want it, then press *Enter*.
Resize the active window	Drag the resize corner. Choose Size/Move from the Window menu (or press *Ctrl-F5*), press *Shift*-(arrow key) to change the size of the window, then press *Enter*, or Drag the right or bottom border to resize the window in that direction only.
Zoom the active window	Click the zoom box, or Double-click the window's title bar, or

Table 4.1: Manipulating windows (continued)

	Choose **Zoom** from the **Window** menu, or press *F5*.
Iconize the active window	Click the iconize box, or
	Choose **Iconize/Restore** from the **Window** menu.
	When a window is fully zoomed, it has only an iconize box.([⇕]) When it is iconized, it has only a zoom box ([↑]).
Move from pane to pane	Press *Tab* or *Shift-Tab*, or
	Choose **Window** | **Next Pane**.

The status line

The status line at the bottom of the Turbo Profiler screen provides the following information:

- It reminds you of basic keystrokes and shortcuts applicable at that moment in the active window. (You will see that the status bar changes if you hold down *Alt* or *Ctrl*.)
- It provides on-screen shortcuts you can click to carry out the action (instead of choosing the command from the menu or pressing the hot key on the keyboard).
- It offers one-line information on any selected menu command or dialog box item.

The only time the status line is unavailable is when a dialog box or menu is open. You must close the dialog box or menu before doing anything else.

The status line changes as you switch windows or activities. You can click any of the shortcuts to carry out the command.

When you've selected a menu command, the status line changes to display a one-line summary of the routine of the selected item. For example, if the **Options** menu title is selected (highlighted), the status line displays the currently selected item in the Options menu.

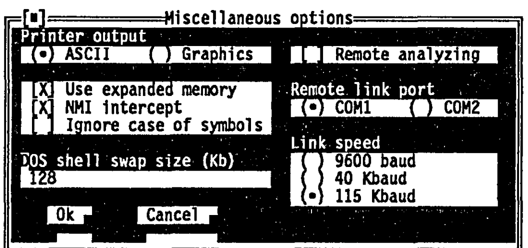
Dialog boxes

If a menu command has an ellipsis after it (...), the command opens a *dialog box*. A dialog box is a convenient way to view and set multiple options.

When you're making settings in dialog boxes, you work with five basic types of controls: radio buttons, check boxes, action buttons, text boxes, and list boxes. Here's a typical dialog box that illustrates some of these items:

Figure 4.2
A typical dialog box

If you have a color monitor, Turbo Profiler will use different colors for various elements of the dialog box.



This dialog box has three standard buttons: OK, Cancel, and Help. If you choose OK, the choices in the dialog box are recorded in Turbo Profiler; if you choose Cancel, nothing changes and no action is made, but the dialog box is put away. Choose Help to open a Help window about this dialog box. *Esc* is always a keyboard shortcut for Cancel (even if no Cancel button appears).

If you're using a mouse, just click the button on the item you want. If you're using the keyboard, press *Tab* or *Shift-Tab* to move from section to section; each section highlights when it becomes the active one.

You can select another button with Tab; press Enter to choose that button.

Note that the OK button has a special look. It has a special color (in monochrome systems, arrows point to it →like this←). This indicates that OK is the *default button*, which means you need only press *Enter* to choose that button. Be aware that tabbing to a button makes that button the default.

To choose a button with a mouse, click it. From the keyboard, you choose a button by pressing *Tab* until the button is highlighted, and then pressing *Enter*. (Once you've tabbed past the buttons, pressing *Enter* chooses only the preset default button.) You can also press the highlighted letter associated with the button (*K* for OK).

Check boxes and radio buttons

The dialog box also has *check boxes*. When you select a check box, an X appears in it to show that it's on; an empty box indicates it's off. To check a check box (set it to on), click it or its text, press *Tab* until the check box is highlighted and then press *Spacebar*, or press *Alt* and the highlighted letter. You can have any number of the check boxes checked at any time.

If several check boxes apply to a topic, they appear as a group.

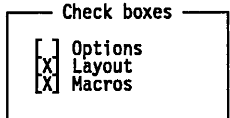
On monochrome monitors, Turbo Profiler indicates the active check box by placing a chevron symbol (») next to it. When you press *Tab*, the chevron moves to the next check box.

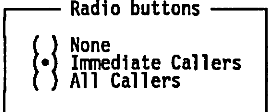
Radio buttons are so called because they act just like the group of buttons on a car radio. There is always one—and only one—button pushed in at a time.

The dialog box also has *radio buttons*. Radio buttons differ from check boxes in that they present mutually exclusive choices. For this reason, radio buttons always come in groups, and only one radio button can be on in any one group at any one time.

To choose a radio button, click it or its text. From the keyboard, press *Tab* until the group is highlighted, then use the arrow keys to choose a particular radio button. Press *Tab* (or *Shift-Tab*) again to leave the group with the new radio button chosen.

Here's what check boxes and radio buttons look like on and off:





Input boxes and lists

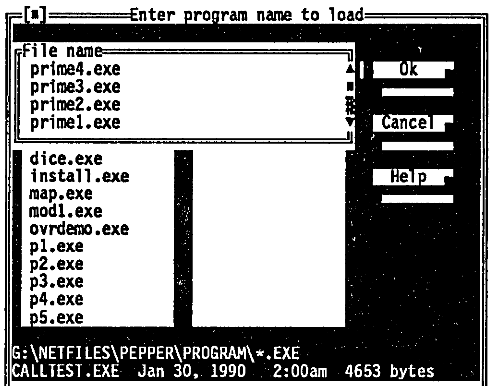
Dialog boxes can also contain input boxes. These boxes allow you to type in text. All the regular text-editing keys work in the input box (for example, arrow keys, *Home*, and *End*). If you continue to type once you reach the end of the box, the contents automatically scroll. If there's more text than what shows in the box, arrowheads appear at the end (◄ and ►). You can click the arrowheads to scroll or drag the text.

If an input box has a ↓ icon to its right, there is a *history list* associated with that input box. This history list lists the text you typed into this box the last few times you used this dialog. The Search box, for example, has a such a history list, which keeps track of the text you searched for previously.

If you want to reenter text that you already entered, press ↓ or click the ↓ icon. Y ou can edit an entry in the history list directly. Press *Esc* to remove the history list without making a selection.

Here is what a history list for the File Name input box might look like if you had already used it four times:

Figure 4.3
The File Name history list



Many dialog boxes also have a *list box*. You use a list box to scroll through long lists without leaving the dialog box. Turbo Profiler typically uses list boxes to display file names in dialog boxes.

To display a list box, you click it, or press *Tab* until it's highlighted and then press *Enter*. To move through the list once the list box is displayed, you can use the scroll box or press ↑ or ↓ from the keyboard.

Part 2: The menu reference

This section gives you an item-by-item description of each menu command and dialog box option in the Turbo Profiler environment. The figure on page 72 is a "road map" to the profiler's global menus (the menus that are called from the menu bar).

≡ menu (System)

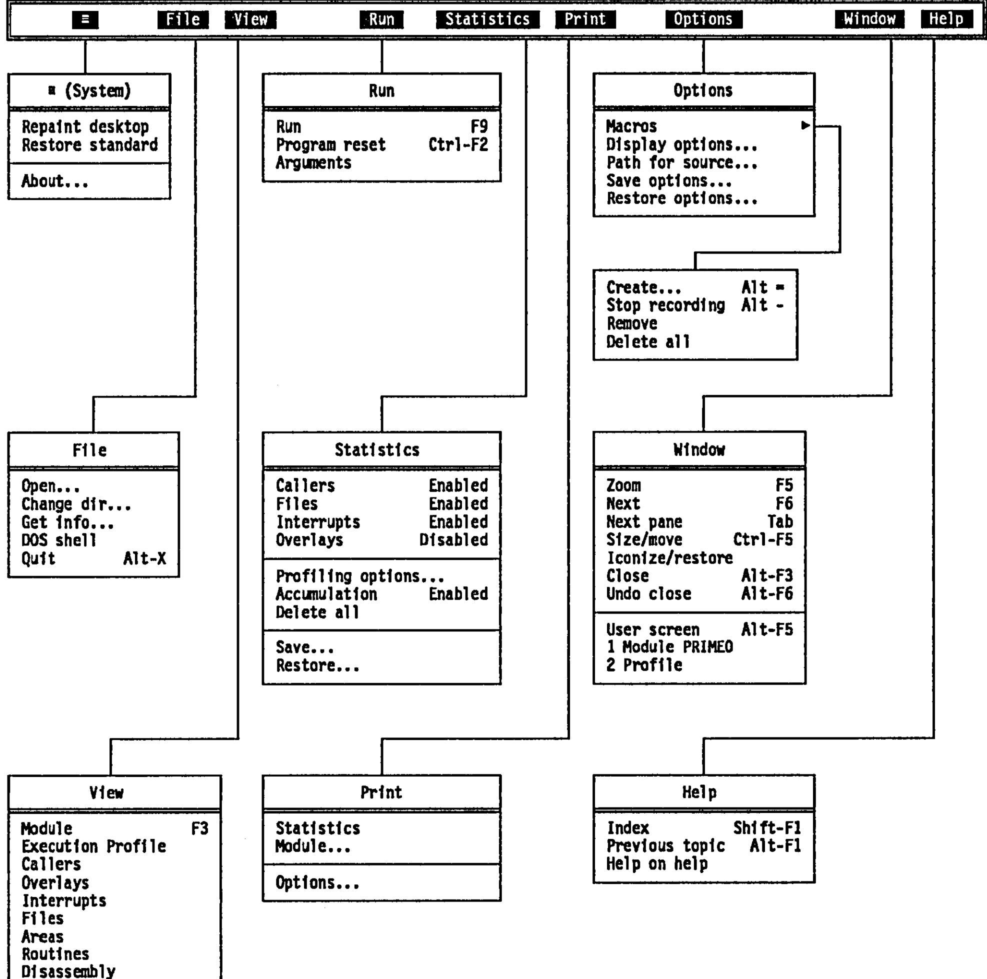
The ≡ menu (called the *System menu*) appears on the far left of the menu bar. To activate the ≡ menu, either (1) press *Alt Spacebar*, or (2) press *F10*, then use → or ← to go to the ≡ symbol and press *Enter*.

```
Repaint desktop
Restore standard
----------------
About...
```

With the commands in the ≡ menu, you can

- repaint the screen
- restore your original window configuration
- activate a Turbo Profiler information box

Figure 4.4: Turbo Profiler's menu bar and global menus



Repaint Desktop

Choose **R**epaint Desktop when you want Turbo Profiler to redraw the screen. You might need to do this, for example, if a memory-resident program has left stray characters on the screen, or possibly if you have display swapping turned off.

Restore Standard

When you start up Turbo Profiler, it sets the environment windows' size, status (open or closed), and placement according to information stored in the configuration file, TFCONFIG.TF. Once Turbo Profiler is onscreen, you can move and resize the windows, close some and open others, and generally make a real mess of your screen. The Restore Standard command provides a quick way to rectify such a situation.

When you choose Re**s**tore Standard, Turbo Profiler puts all the windows back the way they were when you first started up the profiler.

About

When you choose **A**bout from the ≡ menu, the About box pops up. This box lists the Turbo Profiler version number and other interesting facts. Press *Enter* or choose OK to close the box.

File menu

```
Open...
Change Dir...
Get Info...
DOS Shell
Quit     Alt-X
```

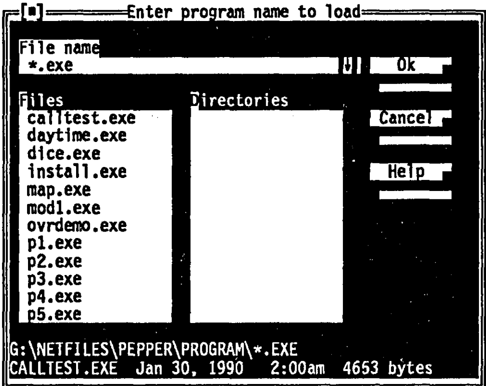
The **F**ile menu contains commands for

- opening and loading a program to be profiled
- changing the current directory
- obtaining information about your program and system memory allocation
- shelling out to the operating system
- quitting the profiler

Open

The **File** | **O**pen command opens the Program Load dialog box, shown here:

Figure 4.5
The Program Load dialog box



With this dialog box, you can do any of the following:

- load an explicit file into the Module window
- use wildcards to filter the file list to match your specifications
- choose a file from a history list of previously-entered file names
- view the contents of different directories or drives

There are three ways to load a file from this dialog box:

1. Type in the file name, then choose OK (or press *Enter*).
2. Press *Enter* or *Tab* to activate the files list box. Select (highlight) the file name you want, then choose OK or press *Enter*.
3. Double-click the file name.

You'll get an error message if you attempt to load a non-existent file or a file that isn't an .EXE file or that doesn't have debug information.

Choose **C**ancel to leave the Program Load dialog box without loading a file.

Using the File Name input box

When the File Name input box is active (the cursor is blinking in the box), you can do any of the following:

- **Load an explicit file:** Type in a full *executable* file name (including disk drive and relative or absolute path, if you want; you don't have to type the extension). Then choose OK (or

press *Enter*) to load that executable file's main source file into the Module window.

- **Filter the file list:** Type in a file name (including disk drive and relative or absolute path) with DOS wildcards (? and *). Then move to the Files list box of matching file names to choose the file you want, or to the Directory list box to change to a different directory.

A history list shows the last eight file names you've entered.

- **Choose from a history list:** Press ↓ to make a history list drop down below the Name input box. To choose a file from the history list, double-click the file name, or select it with an arrow key and press *Enter*.

Using the Files list box

By default, the Files list box displays all file names in the current directory that match the specifications in the Name input box as well as the names of directories you can move to from the current directory. If the Name input box specification includes a drive or path name, the list box displays all matching file names in the specified drive and directory.

To load a file from the Files list box,

1. Click the list box (or press *Tab* until the list box name is highlighted).
2. When the name is highlighted, either press ↓ or ↑ to select a file name (then press *Enter*), or double-click the file name.

You can scroll the list box, if necessary, to see all the file names.

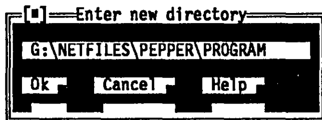
If the file you want is in another directory, tab to the Directories list box and select the directory you want to move into. (To access the parent directory of the one you are currently in, type `..*.exe` and press *Enter*.

If you need to load your program with some command-line arguments, refer to the description of the **R**un | **A**rguments input box on page 110.

Change Dir

The **F**ile | **C**hange Dir command brings up the New Directory dialog box.

Figure 4.6
The New Directory dialog box



From this dialog box, you can log to a different current directory. (The current directory is where Turbo Profiler saves and looks for files.)

New Directory dialog box components

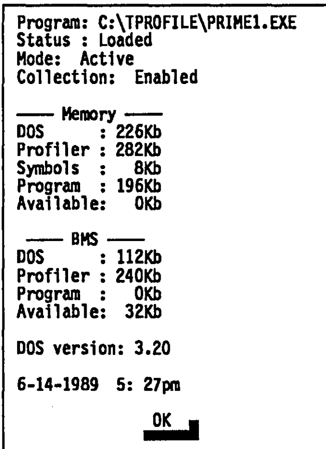
You'll get an error message if the new directory can't be found.

The New Directory dialog box contains an input box in which you type the path to the directory you want to access. When you have done so, choose OK to change directories, or Cancel to remain in your present one.

Get Info

The **File | Get** Info command displays a text box with information about the program being profiled and your system's current memory configuration.

Figure 4.7
The Get Info text box



The information in the Get Info box is for display only; you can't change any settings from this box. Here's what the categories in this information box represent:

- Program is the program being profiled; you determine which file to profile with the **File | Open** command.
- Status is the reason why Turbo Profile gained control: it can be any one of these loaded messages:

```
No program loaded
Control-Break
Terminated, exit code XX
Stopped by area
NMI Interrupt
Exception XX
Divide by zero
```

- Mode is the profiling mode (active or passive); you specify the profiling mode with the Profile Mode radio button in the Profiling Options dialog box (accessed by choosing **Statistics | Profiling Options**).
- Collection tells whether automatic data collection is enabled or disabled; you specify the data-collection setting with the **Statistics | Accumulation** command.
- Memory tells
  - DOS: Memory occupied by DOS and/or various device drivers.
  - Profiler: Total memory used by the profiler.
  - Symbols: Memory allocated for the program's symbol table.
  - Program: Memory allocated to the current program being profiled.
  - Available: Amount of remaining available memory.
- EMS shows use of expanded memory by DOS, Turbo Profiler, the program's symbol table, the program being profiled, and available memory, like the base memory display. EMS appears only if expanded memory is present.
- DOS version shows the current DOS version on your system.
- Current date and time shows today's date and the time of day.

After reviewing the information in the Get Info box, click OK or press *Enter* to return to the current window.

DOS Shell

The **File | DOS** Shell command steps you out of Turbo Profiler and back to the DOS prompt, so you can enter a DOS command or program.

To return to Turbo Profiler, type `EXIT` at the DOS prompt.

In remote profiling mode, the DOS command line appears on the Turbo Profiler screen rather than on the User screen; this allows you to switch to DOS without disturbing your program's output. Because your program's output is always available on one

monitor in the system, **Window | User Screen** and *Alt-F5* are disabled. (See Appendix C for details about remote profiling.)

Quit

[Alt][X]

The **File | Quit** command exits Turbo Profiler, removes it from memory, and returns to the DOS command line. Each time you exit Turbo Profiler, it remembers the areas you set up for the current program.

If you have any profile data or setup parameters that you want to keep (such as the profile statistics, profiling and display options, and screen layout options), save them with **Statistics | Save** and **Options | Save** before exiting. Otherwise, you will lose the options you've set.

View menu

The View menu lets you open several kinds of windows in which you can examine information about your program's performance.

```
Module...          F3
Execution Profile
Callers
Overlays
Interrupts
Files
Areas
Routines
Disassembly
```

Table 4.2
Summary of Turbo Profiler windows

Window name	**What this window displays**
Module	Source code for the program being profiled
Execution Profile	Statistical information about a program after the program has run
Callers	Information about how often a routine is called and which routines call it
Overlays	Information about overlays for Turbo Pascal, Turbo C, and Turbo Assembler
Interrupts	Information about interrupt calls made by the program
Files	Information about file activity
Areas	Detailed information about data-collection activities at the places marked in your source code
Routines	All routines that can be used as profile area markers
Disassembly (CPU)	The current profile area in the Module window, as disassembled source code

Module

When you choose **Module**, a dialog box appears in which you type the name of the module you want to open. Press OK to display this module in the Module window. The Module window displays source code for the program being profiled. In the Module window, you can examine code and set areas to be profiled. Special hot keys and window links connect the code in this window to data and statistics in the other windows.

Figure 4.8
The Module window

```
=[■]=Module: PRIME0  File: prime0.c (modified) 7===============1===[↕]=
   /* Copyright (c) 1990, Borland International */
   /*  Program for generating prime numbers using Euclid's method */

   int primes[1000];
   #define MAXPRIMES 1000

=► main()
   {
      int j;
      int lastprime, curprime;

=►    primes[0] = 2;
=►    primes[1] = 3;
=►    lastprime = 1;
=►    curprime  = 3;

=►    printf("prime %d = %d\n", 0, primes[0]);
=►    printf("prime %d = %d\n", 1, primes[1]);
=►    while(curprime < MAXPRIMES)
      {
=►       for(j = 0; j <= lastprime; j++)
```

When you run the profiler, you'll need *both* the .EXE file and the original source file available. Turbo Profiler looks for your program's source code in these places, in this order:

1. In the directory where the program was originally compiled
2. In the directories (if any) you've listed under **O**ptions | **P**ath for Source
3. In the current directory
4. In the directory that contains the program you're profiling

(The name of the directory where the program was originally compiled is contained in .EXE and .OBJ files *if* you compiled your program with symbolic debugging information turned on.)

```
Line...
Search...
Next
Goto...
---
Add areas    ▶
Remove areas ▶
Operation...
Callers...
---
Module...
File...
Edit
```

Press *Alt-F10* or click the right mouse button to bring up the Module window's local menu. With the local menu commands, you can perform these actions:

- move the cursor to a specific line or code label
- search for text in the source code
- add and remove profile areas
- set the profiling action that will occur for a given area
- specify the level of call-path recording for a given routine
- load another module or another source file of the current module into the Module window
- invoke the editor of your choice

To activate a local menu item directly (without bringing up the menu), press the *Ctrl-(letter)* hot key, where *letter* is the menu item's highlighted letter.

Ctrl L

Line

To move swiftly to a particular line of code in the Module window, choose **L**ine from the local menu. The dialog box that pops up requests the line number you seek; type in the new line number to go to, then choose OK (or press *Enter*). If you enter a line number after the last line in the file, you will be positioned at the last line in the file.

Ctrl S

Search

To search for a character string in the current module, choose **S**earch from the Module window's local menu. The prompt box that pops up requests the string to search for; type in the string, then choose OK (or press *Enter*).

If the cursor is positioned over text that looks like a variable name, the prompt box comes up initialized to that name. If you mark a block in the file, the profiler uses that block to initialize the search prompt. This saves you from extraneous typing if the text you want to search for is a string already in the Module window.

You can use the standard DOS wildcards (? and *): The ? indicates a match on any single character, and * matches 0 or more characters.

The search does not wrap around from the end of the file to the beginning. To search the entire file, start at the first line.

Ctrl N

Next

Once you've defined a search string with the Module window's local **S**earch command, you can search for successive occurrences of that string with the **N**ext command. Choose **N**ext from the local menu, or press the shortcut, *Ctrl-N*. You can only use **N**ext after issuing a **S**earch command.

Ctrl G

Goto

Use the hex format of whichever language your program's in.

To position the Module window's cursor on a particular routine or other code label in your program's source code, choose **G**oto. The prompt box that pops up requests the address you want to examine. Type in a line number, a routine name, or a hex address, then choose OK (or press *Enter*).

For information on address syntax, see the chapter "Expressions" in the *Turbo Debugger User's Guide.*

Ctrl A

Add Areas

```
All routines
Modules with source
Routines in module
Every line in module
Lines in routine
Current routine   Alt-F2
This line             F2
```

Add Areas on the Module window's local menu leads to the menu shown here.

- **A**ll Areas adds area markers for all routines in the program being profiled, including routines for which source code is unavailable (such as library routines linked in as object modules from library files).
- **M**odules with Source adds area markers for all routines in modules whose source code is available.
- **R**outines in Module adds area markers for all routines in the current module (the one in the Module window).
- **E**very Line in Module adds area markers for all lines in the current module.
- **L**ines in Routine adds area markers for all lines in the current routine (whichever routine the cursor is on in the Module window).
- **C**urrent Routine adds an area marker for whichever routine the cursor is on in the Module window.

- This Line adds an area marker for the line the cursor is on in the Module window.

Ctrl R

Remove Areas

```
All areas
Modules with source
Routines in module
Every line in module
Lines in routine
Current routine   Alt-F2
This line             F2
```

Remove Areas on the Module window's local menu leads to the menu shown here.

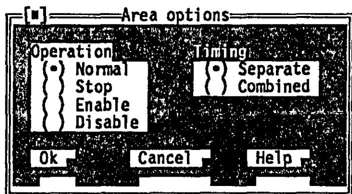
- All Areas removes area markers for all routines in the program being profiled, including routines for which source code is unavailable (such as library routines linked in as object modules from library files).
- Modules with Source removes area markers for all routines in modules whose source code is available.
- Routines in Module removes area markers for all routines in the current module (the one in the Module window).
- Every Line in Module removes area markers for all lines in the current module.
- Lines in Routine removes area markers for all lines in the current routine (whichever routine the cursor is on in the Module window).
- Current Routine removes the area marker for whichever routine the cursor is on in the Module window.
- This Line removes the area marker for the line the cursor is on in the Module window.

Ctrl O

Operation

The Operation command opens the Area Options dialog box, which contains settings for the current area (the one where the cursor is in the Module window).

Figure 4.9
The Area Options dialog box



You can specify two options from this dialog box: Operation and Timing. When you mark an area, a marker symbol signifying the chosen operation appears to the left of that area.

- **Operation** specifies what profiling action will occur for the current area.

  Normal collects profile statistics for this area as specified in the Statistics menu (callers, file activity, interrupts, overlays, etc.) and Area Operations dialog box, which you reach through the local menus of the Module and Areas windows.

  Stop stops program execution at this marker.

  Enable turns on the collection of statistics at this point in the program.

  Disable temporarily turns off the collection of statistics at this point in the program.

- Timing specifies whether the profiler will add the current area's execution time to a higher-level area or keep it separate.

  Separate adds any timer ticks occurring while program control is in the marked area to that area's timer-tick compartment only, not to the caller's compartment.

  Combined adds timer ticks that occur while control is in the marked area to the area's timer-tick compartment *and* to the caller's compartment.

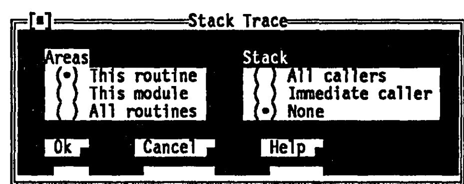
The Areas window displays operation and timer information for all marked areas.

Ctrl C

Callers

The **C**allers command on the local menu leads to the Stack Trace dialog box.

Figure 4.10
The Stack Trace dialog box



You can specify two options from this dialog box, Areas and Stack.

- Areas specifies which areas you want call paths recorded for.

  This Routine records call-path information for the current routine (the one the cursor is on in the Module window).

  This Module records call-path information for all marked routines in the current module (the one in the Module window).

  All Routines records call-path information for all routines in all modules in the program.

- Stack specifies how extensive ("deep") the recorded call stack should be.

  All Callers records all available call stack information for the routine(s) you've specified with the Areas option.

  Immediate Callers records only "parent" information for the routine(s) you've specified with the Areas option.

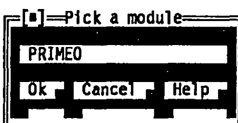
  None turns off call stack information for the routine(s) you've specified with the Areas option.

Alt M

Module

The Module command on the local menu leads to the Pick a Module dialog box that lists all your program's modules for which source is available.

Figure 4.11
The Pick a Module dialog box



Most modules only have a single source code file; other files included in a module (such as C header files) usually define only constants and data structures. Use this command to open a different module in the Module window.

This option displays only modules for which source code exists, if they are associated with the program being profiled. It allows you to move rapidly from one module to another without having to search your source directory explicitly.

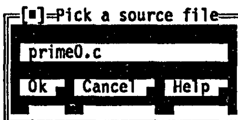
The **M**odule command searches for the source code in the following places, in the order listed:

1. in the directory where the program was originally compiled
2. in the directories (if any) you've listed under **O**ptions | **P**ath for Source
3. in the directory that contains the program you're profiling
4. in the current directory

File

The **F**ile command on the local menu leads to a dialog box that lists all the source files used to compile the current module. Use this command if your module has source code in more than one file and the file you want is not displayed in the module window.

Figure 4.12
The File dialog box



The **F**ile command searches for the source code in the following places, in the order listed:

1. in the directory where the program was originally compiled
2. in the directories (if any) you've listed under **O**ptions | **P**ath for Source
3. in the directory that contains the program you're profiling
4. in the current directory

[Ctrl][E]

Edit

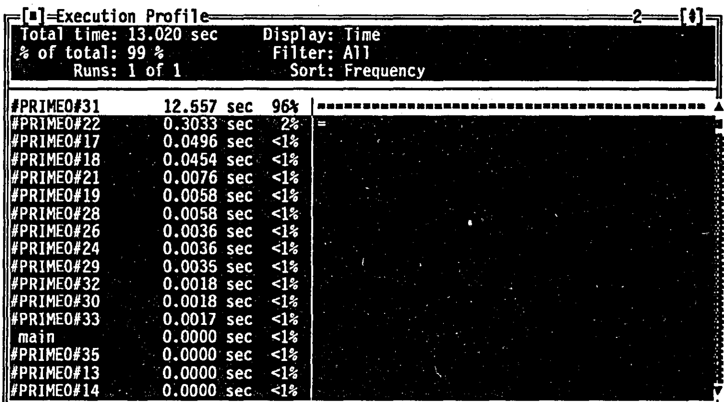
Although Turbo Profiler does not have a built-in editor, you can specify your own favorite editor as an option when you customize the profiler with the Turbo Profiler installation program, TFINST. See Appendix B for information about TFINST.

Once you've installed an editor via TFINST, whenever you choose **E**dit from the Module window's local menu, Turbo Profiler automatically shells out to DOS and invokes your editor. To return to the profiler from your editor, simply quit the editor.

Execution Profile

The Execution Profile window is where Turbo Profiler displays your program's profile statistics (after you've set areas and run the program under control of the profiler).

Figure 4.13
The Execution Profile window

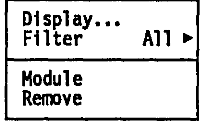


The Execution Profile window consists of one pane, divided into two display areas (top and bottom). The top display area lists

- Total Time: your program's total execution time
- % of Total: how much of that total (a percentage) is represented by the statistics currently displayed in the bottom area of the window
- Runs: the current profile run (if you're collecting and averaging statistics from more than one run)
- the options you've chosen from the local menu (display format, filter status, sort order)

The bottom display area lists one or two lines of profile data for each area you've marked. The information shown in this display area can include each area's name or line number, the execution counts for each marked area, the time spent in each marked area, the average time per pass for each marked area, and the most time spent in a marked area on a single pass.

If you have a Module window and an Execution Profile window onscreen at the same time, the Execution Profile window is positioned automatically to show the statistics for the area the cursor is on in the Module window.



To specify how the Execution Profile window displays your program's statistics, activate the local menu (press *Alt-F10*). Through this local menu, you can

- choose any one of five different ways to display profile statistics in the Execution Profile window
- sort the displayed statistics
- temporarily remove one or more areas' statistics from the display
- examine the source code for an area
- delete an area's statistics from memory

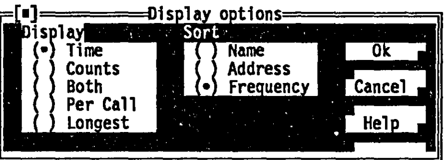
To activate a local menu command directly (without bringing up the menu), press the *Ctrl-(letter)* hot key, where *letter* is the menu item's highlighted letter.

Ctrl D

Display

When you choose **Display** from the Execution Profile window's local menu, this Display Options dialog box comes up.

Figure 4.14
The Display Options dialog box



You can specify two options from this dialog box, Display and Sort.

- Display specifies what form the data will be displayed in.
  - Time displays the profile statistics for each area as the time (in milliseconds) program control was in that area.
  - Counts displays profile statistics for each area as pass counts: how many times program control entered that area.
  - Both displays the statistics for each area as both time (the top line) and counts. This provides a graphic measure of a routine's efficiency.
  - Per Call displays each area's statistics as the *Time : Counts* ratio. This provides the average time spent in each call to the routine.

- Longest displays, for each area, the longest single time program control was in that area.

- Sort specifies what order the data will be sorted in.
  - Name sorts the profile statistics by area name, in alphanumeric order.
  - Address sorts profile statistics by memory location, starting with the lowest address.
  - Frequency sorts the statistics numerically, with the highest frequency at the top.

The top display area of the Execution Profile window lists the current display and sort options.

Filter

All
Module...
Current

The **F**ilter command on the local menu leads to the three-item menu shown here.

- **A**ll restores all collected statistics for the current program to the Execution Profile window.

  After you've filtered out certain statistics from the Execution Profile window (with **F**ilter | **M**odule or **F**ilter | **C**urrent), choose **F**ilter | **A**ll to restore all profile statistics to the window.

- **M**odule filters out all but one module's statistics.

  This command leads to the Pick a Module dialog box, which lists all modules for the current program. Use the ↑ and ↓ arrow keys to highlight one module in the list, then press *Enter*. Only the areas in the chosen module show up in the Execution Profile window.

- **C**urrent temporarily removes the current routine's statistics from the Execution Profile window.

  Choose **F**ilter | **C**urrent if you want to throw out one routine's statistics and see what happens to the remaining percentages. The **C**urrent command is a temporary filter that hides report information from sight without deleting any information; it does the following:

  1. Removes the current area's statistics from the Execution Profile window.
  2. Calculates original total execution time minus the time of the removed area.

3. Recalculates the remaining areas' percentages as fractions of the newly calculated total execution time.

When you filter one or more areas' statistics from the Execution Profile window, the profiler calculates a new total execution time based on the statistics displayed in the window, but the Total Time value shown in the top of the window does not change.

When you use **Filter | Current**, the original total execution time for the entire program remains displayed in the Execution Profile window's top display area.

*Don't confuse **Filter | Current** with the **Remove** command on the Execution Profile window's local menu.*

Filter | Current is a temporary filter that hides report information from sight; **Remove** actually affects area marker settings by removing them in both the Module and Areas windows.

[Ctrl][M]

Module

The **Module** command on the local menu takes you to the line of source code in the Module window for which the statistics are highlighted in the Execution Profile Window.

For instance, suppose you highlight the statistics for routine **fred** in the Execution Profile window, then choose **Module** from the local menu to activate the link. Turbo Profiler activates the Module window and places the cursor on the first line of **fred** in the source code. After that, you move the cursor to line 25 in the Module window (line 25 has an area marker). Automatically, the Execution Profile window's contents scroll so that the statistics for line 25 show at the top of the statistics display area.

The link is one-directional: If you go back to the Execution Profile window (after going to the Module window) and move the highlight bar, the source code in the Module window does *not* scroll or track the highlight bar's position. (If it did, you could get very frustrated.)

When you choose the **Module** command from the Execution Profile window's local menu, if source code for the highlighted line is unavailable, the link goes to the corresponding line of code in the Disassembly (CPU) window. This happens, for example, if you've marked areas for **All** Routines and the highlighted line is a library routine. (See page 106 for details about the Disassembly window.)

Remove

The **R**emove command removes area marker settings for the highlighted line from the Module and Areas windows.

***Attention!** The **R**emove command **erases** statistical data. Use it with discretion.*

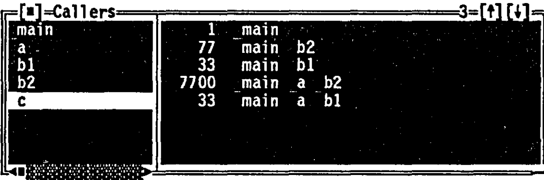
Once you remove the line's area markers with the **R**emove command, the statistics you had gathered for that line are erased and no more statistics are gathered for that line of code. To undo a **R**emove action, you must

1. Activate the Module window and bring up its local menu.
2. Place the cursor on the line whose marker you removed.
3. Choose **A**dd Areas | **T**his line.
4. Run the program again (collecting a new set of statistics).

Callers

The Callers window is where Turbo Profiler displays the call paths for each marked routine in your program. You must set the **S**tatistics | **C**allers menu item to *Enabled* before the profiler will record any call-path information.

Figure 4.15
The Callers window, showing calls in CALLTEST



An underscore precedes the identifier names in this Callers window because Turbo C adds the underscore to all symbol names that appear in .OBJ files and in symbolic debugging information.

The left pane in the Callers window lists each marked routine by name. When you highlight a routine name in the left pane, the right pane displays each unique call path for that routine. If a call path is wider than the right pane, you can zoom the window or switch to the right pane and scroll left and right through the path.

Although the Callers window *displays* the call-path information, you *specify* what type of call-path recording you want through either the Module window or the Areas window.

In the Module window, you can set callers options for whole groups of routines.

1. With the cursor on a marked routine in the Module window, press *Alt-F10* to bring up the local menu.
2. Choose **C**allers to see the Stack Trace dialog box.

3. Set the Areas option. You can choose to record call paths for the current routine, all routines in the current module, or all routines in the program (including library routines).
4. Set the Stack option. You can choose to record all callers for the chosen routine(s), immediate callers (the routines' parents, only), or no callers at all.
5. Press *Enter* or choose OK to go back to the Module window.

In the Areas window, you can set callers options for individual marked routines. (See page 101 for more information about the Areas window.)

1. In the Areas window, place the highlight bar on the routine you want to set call-path options for, then press *Alt-F10* to bring up the local menu.
2. Choose **O**ptions to see the Area Options dialog box.
3. Set the Areas option. You can choose to record call paths for the current routine, all routines in the current module, or all routines in the program (including library routines).
4. Set the Callers option. You can choose to record all callers for the chosen routine(s), immediate callers (the routines' parents, only), or no callers at all.
5. Press *Enter* or choose OK to go back to the Areas window.

Figure 4.15 shows routine **c** highlighted in the left pane of the Callers window, after a profile run of this program, CALLTEST:

```
/* Program CALLTEST */
/* Copyright (c) 1990, Borland International */
#include <stdio.h>

main()
{
   c();
   b2();
   b1();
   a();
}

a()
{
   int i;

   for (i = 0; i < 100; i++)
      b2();
   b1();
```

```
}
b1()
{
   int i;

   for (i = 0; i < 33; i++)
      c();
}
b2()
{
   int i;

   for (i = 0; i < 77; i++)
      c();
}
c()
{
   int i;

  for (i = 0; i < 3; i++)
    ;
}
```

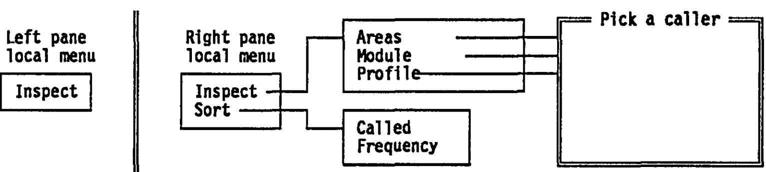
The Callers window's right pane lists each unique call path for routine **c**:

- 1 call from **main** to **c**
- 7,700 calls from **main** to **a** to **b2** to **c**
- 33 calls from **main** to **a** to **b1** to **c**
- 33 calls from **main** to **b1** to **c**
- 77 calls from **main** to **b2** to **c**

You'll find the Callers window useful when you must make decisions about restructuring code, especially when it's possible to reach a routine through several different call paths.

Both panes of the Callers window have local menus. In the Callers window's right pane, both local menu items bring up subsequent menus, as shown here:

Figure 4.16
The Callers window local menus



To activate the local menu from the current pane, press *Alt-F10.* To alternate between the window's panes, press *Tab.* To activate a local menu item directly (without bringing up the menu), press the *Ctrl-(letter)* hot key, where *letter* is the menu item's highlighted letter.

Ctrl I

Inspect (in left pane)

When the highlight bar is on a routine name in the left pane, choose Inspect (or press its shortcut, *Ctrl-I*) to view the source code for that routine in the Module window.

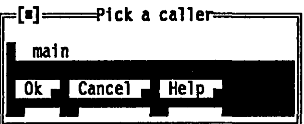
Ctrl I

Inspect (in right pane)

When the highlight bar is on a call path in the right pane of the Callers window, you can "inspect" (view information about) elements in that call path in one of three other windows.

1. Choose Inspect to bring up a list of those other windows.
2. Choose the window you're interested in (**A**reas, **M**odule, or **P**rofile) from the list. This brings up the Pick a Caller dialog box, which lists all callers on the current call path.

Figure 4.17
The Pick a Caller dialog box



3. In the dialog box, highlight the caller in question (use the arrow keys or a mouse click), then choose OK or press *Enter.* If the window you choose to inspect in isn't already open, the profiler opens it automatically, then goes to the caller's location in that window.

Ctrl S

Sort (in right pane)

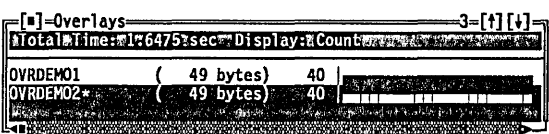
With the local **S**ort command in the Callers window's right pane, you can sort the list of call paths in two ways:

- **C**alled sorts the call paths in the same order that program control traversed them at run time.
- **F**requency sorts the call paths by how often program control traversed each path, with the most-used path at the top of the list.

Overlays

The Overlays window is where Turbo Profiler displays information about overlay activity for Turbo Pascal, Turbo C, and Turbo Assembler programs. You must set the **S**tatistics | **O**verlays menu item to *Enabled* before the profiler will record any overlay information.

Figure 4.18
The Overlays window



The information listed in this window can include

- how many times your program loads each overlay into memory
- how long it takes to load each overlay
- the sequence in which your program loads the overlays
- the size of the overlap

Like the Execution Profile window, the Overlays window is divided into two display areas, top and bottom. The top display area lists total execution time for your program and the current display option for overlay statistics. The bottom display area lists the overlay statistics as either a histogram or a list of events.

For a live demonstration of how the Overlays window works, load the Turbo Pascal demonstration program OVRDEMO into the profiler. Then set area markers for every line in the module OVRDEMO, enable **S**tatistics | **O**verlays, and run the program. (You'll need the files OVRDEMO.PAS, OVRDEMO1.PAS, OVRDEMO2.PAS, and OVRDEMO.EXE to profile this program.)

```
Display     Count
Inspect
```

The Overlays window's local menu provides two commands, shown here. To activate the local menu from the Overlays window, press *Ctrl-F10*. To activate a local menu item directly (without bringing up the menu), press the *Ctrl-(letter)* hot key, where *letter* is the menu item's highlighted letter.

Display

Ctrl D

Display specifies how the data will appear; you toggle between Count and History by pressing *Enter*.

Count produces a histogram that shows, for each overlay, how much memory that overlay consumes and how many times your program loaded the overlay into memory.

History lists your program's overlay activity as a sequence of events; each line names the overlay and specifies when, in the course of program events, that overlay was loaded.

Ctrl I

Inspect

Inspect goes automatically to the Module window (opening it, if necessary) and places the cursor on the source code for the highlighted overlay.

Interrupts

The Interrupts window is where Turbo Profiler displays information about the video, disk, keyboard, DOS, and mouse interrupt events in your program. The **Statistics | Interrupts** menu item must be *Enabled* before the profiler will record any interrupt-call information.

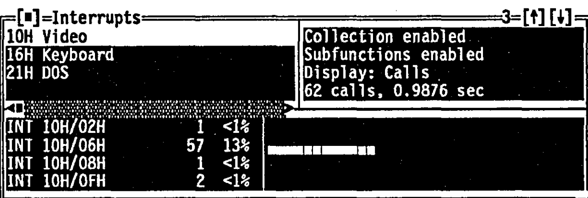


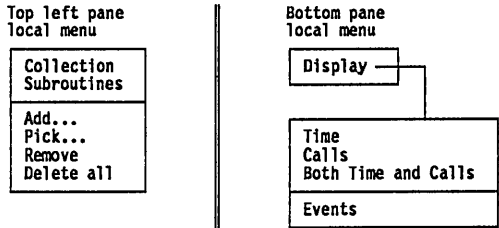
Figure 4.19
The Interrupts window

The Interrupts window is divided into three panes: top left, top right, and bottom.

- The top left pane displays the specific interrupts called by your program (by INT number and name).
- The top right pane lists information about the display mode and the current interrupt (the one highlighted in the top left pane), number of calls, and execution time. You cannot tab to the top right pane; it only displays information.
- In the bottom pane, you see a profile of data for each interrupt, shown as a histogram or as start time and duration.

Both active panes of the Interrupts window have local menus.

Figure 4.20
The Interrupt window local menus



To activate the local menu from the current pane, press *Alt-F10*. To alternate between the window's panes, press *Tab*. To activate a local menu item directly (without bringing up the menu), press the *Ctrl-(letter)* hot key, where *letter* is the menu item's highlighted letter.

Each entry in the bottom pane of the Interrupt window can list

- the interrupt by name or INT number (or both)
- the number of calls to that interrupt (as an absolute number and as a percentage)
- the total amount of execution time spent in that interrupt (as an absolute number and as a percentage)

Ctrl C

Collection (in top pane)

The **C**ollection command enables or disables collection of statistics for the current interrupt (the one highlighted in the left display area of the top pane).

Ctrl S

Subroutines (in top pane)

The **S**ubroutines command enables or disables collection of statistics for subroutines of the current interrupt (particularly useful for DOS INT 21H calls). Subroutine numbers are determined from the value in the AH register when the interrupt is called.

Ctrl A

Add (in top pane)

The **A**dd command adds an interrupt, by number, to the list in the pane's left display area. (You type in the INT number in decimal notation: 33, not 21H.)

Ctrl P **Pick (in top pane)**

The **P**ick command displays a predetermined list of interrupts, so you can pick one to add to the list in the left display area.

Ctrl R **Remove (in top pane)**

The **R**emove command removes the current highlighted interrupt from the list in the pane's left display area.

Ctrl D **Delete All (in top pane)**

The **D**elete All command removes all the listed interrupts in the pane's left display area.

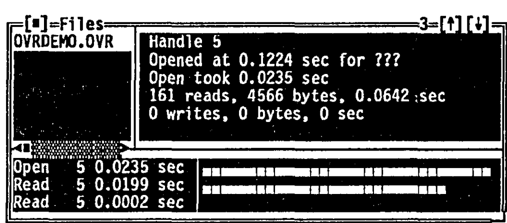
Ctrl D **Display (in bottom pane)**

The Interrupt window's bottom pane has a one-item local menu; its command, **D**isplay, leads to a subsequent menu. From this second menu, you can choose to display interrupt statistics in one of four different formats:

Time	Displays the amount of time spent in each interrupt and its subfunctions.
Calls	Displays the number of times each interrupt and its subfunctions were called.
Both Time and Calls	Displays both the amount of time and the number of times that each interrupt and its subfunctions were called.
Events	Displays a time ordered list of interrupt calls.

Files The Files window is where Turbo Profiler displays information about file activity that occurred during your program's run. **S**tatistics | **F**iles must be set to *Enabled* (the default), so the profiler will record any file-activity information, such as read, write, open, or close.

Figure 4.21
The Files window



The Files window is divided into three panes: top left, top right, and bottom.

The top left pane lists files by name, including STDIN and STDOUT. As you move the highlight bar over the file name you're interested in, the top right pane shows, for that file,

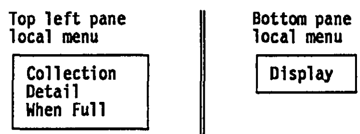
- the handle number
- the time when the file was opened
- how long the file was open
- the number of reads and writes from and to the file
- the total number of bytes read and written
- the time for all reads from and writes to the file
- the time required to close the file

The top right pane only displays information. You can't tab to it, and it does not have a local menu.

The lower pane displays file activity statistics (reads, writes, opens, and closes) as individual entries, rather than as statistical totals associated with a single file-name entry. Each entry provides information about a given file activity.

Both active panes of the Files window have local menus.

Figure 4.22
The Files window local menus



To alternate between the window's panes, press *Tab*. To activate the current pane's local menu, press *Ctrl-F10*. To activate a local menu item directly (without bringing up the menu), press the *Ctrl-(letter)* hot key, where *letter* is the menu item's highlighted letter.

Ctrl C **Collection (in top pane)**

The **C**ollection command enables or disables the collection of file activity statistics for the current file (the one highlighted in the left display area of the top pane).

Each entry in the bottom pane of the Files window provides information about a given file activity.

Ctrl D **Detail (in top pane)**

The **D**etail command enables or disables a detailed listing of file-activity statistics. A detailed listing logs each file read and write activity separately, with the time from the program start the activity occurred and the number of bytes transferred. When **D**etail is disabled, only file open and close activities are logged individually; reads and writes are summarized.

Ctrl W **When Full (in top pane)**

The **W**hen Full command specifies what happens when the memory set aside for file-activity statistics fills up.

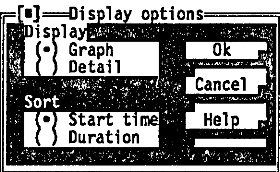
Wrap means that the newest file-activity statistics will overwrite the oldest ones when the memory area fills up.

Stop means that file-activity statistics gathering will stop when the memory area fills up.

Ctrl D **Display (in bottom pane)**

In the Files window's bottom pane, you can choose one menu item, **D**isplay, which leads to the Display Options dialog box, shown here.

Figure 4.23
The Display Options dialog box



You can specify two options from this dialog box: Display and Sort.

- Display specifies how you want file-activity statistics to appear in the bottom pane.

  Graph displays each activity's *total* time as a bar graph.

  Detail displays each activity's exact time in seconds.

- Sort specifies the order in which Turbo Profiler sorts the displayed statistics.

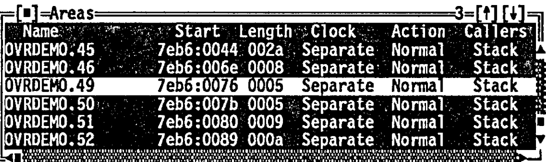
  Start Time sorts the files' statistics by sequential order of occurrence.

  Duration sorts the files' statistics by how long the open, read, write, or close operation took.

Areas

The Areas window is where Turbo Profiler displays detailed information about your program's marked profile areas. Use it to verify that the point and shoot Add/Remove Areas commands in the Module window local menu have set or cleared the desired area, and to adjust the behavior of individual areas.

Figure 4.24
The Areas window



By default, the Areas window lists each area in alphabetical order. For typical programs, these areas are designated by the names of the routines to which they correspond. However, if you mark each line in a routine, the area name is (generically)

```
ModName.FileName.NN
```

The file name appears only if the module is made up of more than one file.

where *ModName* is the module name, *FileName* is the file name, and *NN* is the line number. If you mark a line associated with a label (for example, a routine name), the profiler uses the label as the area name.

The Areas window shows the following information associated with each marked area:

- Start: starting address in hexadecimal
- Length: length in bytes, as a hexadecimal number
- Clock: whether the area uses a separate or combined clock in timing descendent areas
- Action: the area operation (what Turbo Profiler should do when it enters or leaves that area)
- Callers: whether the profiler tracks the area's immediate caller only, all callers, or no callers

```
Add areas      ►
Remove areas   ►
Inspect
----------------
Options...
Sort      Name ►
```

The Areas window is more than a source window for static display of information. With the local menu, you can

- add or remove areas
- inspect areas
- change options for individual areas
- sort the displayed information

To activate the window's local menu, press *Alt-F10*. To activate a local menu item directly (without bringing up the menu), press the *Ctrl-(letter)* hot key, where *letter* is the menu item's highlighted letter.

Ctrl A

Add Areas

Choose **A**dd Areas to specify all routines in your module as a specific routine or module that you want marked as an area. This command resembles the Module window's local **A**dd Areas command.

Ctrl R

Remove Areas

The **R**emove command removes all information displayed for an area and removes that area's markers.

Ctrl I

Inspect

When you choose **I**nspect, the profiler switches to the Module window and places the cursor on the first line of source code corresponding to the current area (the one highlighted in the Areas window).

Options

When you choose **Options** from the Areas window's local menu, the Area Options dialog box comes up.

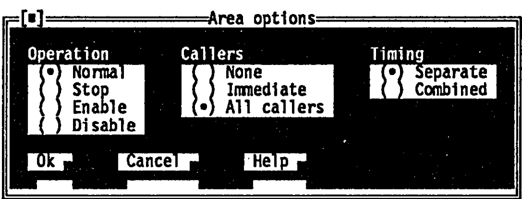


Figure 4.25
The Area Options dialog box

You can specify three options from this dialog box: Operation, Callers, and Timing.

- Operation specifies what profiling action will occur for the current area.

  Normal collects profile statistics for this area.

  Stop stops program execution at this marker.

  Enable turns on the collection of statistics at this area marker.

  Disable turns off the collection of statistics at this marker. Data collection resumes once program control passes an enabled marker.

- Callers specifies which level of callers the profiler will track.

  All Callers records all available call-path information for the current routine.

  Immediate Callers records only "parent" information for the current routine.

  None turns off call-path information for the current routine.

- Timing specifies whether the profiler will add the current area's execution time to a higher-level area or keep it separate.

  Separate adds any timer ticks occurring while program control is in this marked area to the area's timer-tick compartment only, not to the caller's compartment.

  Combined adds timer ticks that occur while control is in this marked area to the area's timer-tick compartment *and* to the caller's compartment. You can specify combined time for an area only if that area's Callers setting is *Immediate* or *All.*

The following figure illustrates how time propagates from called routines to the calling routines when the caller's clock is set to Combined.

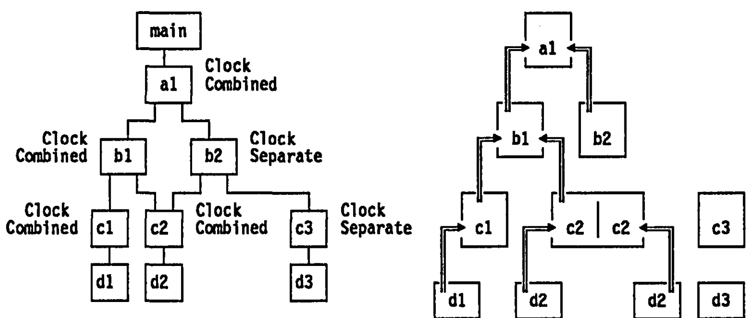


Figure 4.26
Propagation of time

In this figure, **b1**'s clock is combined and **b1** calls **c1**, so **c1**'s time is combined with **b1**'s time. Part of **c1**'s time is actually **d1**'s time, because **c1**'s clock is combined.

Both **b1** and **b2** call routine **c2**.

- Routine **c2**'s time from when **b1** calls it—which includes all of **d2**'s time—is combined with **b1**'s time, because **b1**'s clock is combined.
- But **c2**'s time from when **b2** calls it is *not* combined with **b2**'s time, because **b2**'s clock is separate.

Ctrl S

Sort

The **Sort** command rearranges the information displayed in the Areas window. You can sort alphabetically (by Name) or numerically (by Address). Sorting by Address lists the areas in an order more consistent with the order in which they appear in your source code.

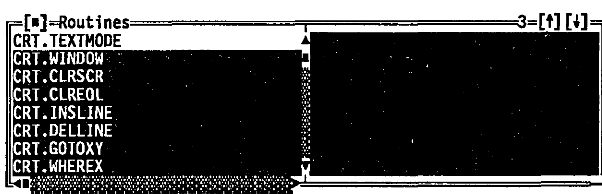
Routines

The Routines window is where Turbo Profiler displays a list of all routines that you can use as area markers. Use it when you can't remember the name of a routine, or to see which routines have areas set on them. You can use the Inspect command to go to other modules by "inspecting" a routine in a particular module.

This menu option gives you easy access to information related to a symbol.

The information displayed is basically a list of all global symbols available from debug information included in the executable file. These symbols include all routine and procedure names in standard libraries for Turbo C and Turbo Pascal, as well as the names of routines in any third-party libraries you might be using (provided you link to those libraries with symbolic debug information turned on).

Figure 4.27
The Routines window

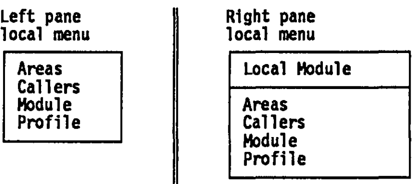


The Routines window is divided into two panes, left and right. The left pane lists routines global to your whole profiled program, and the right pane lists routines that are local to the current module of the program you're profiling.

Local routines include nested routines and procedures in Pascal and static routines in C. Global routines with source code available appear highlighted in the right pane. (By default, an underscore (_) precedes all global variables in Turbo C programs.)

Both panes of the Routines window have local menus.

Figure 4.28
The Routines window local menus



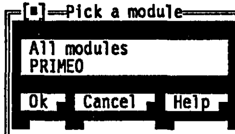
To alternate between the window's panes, press *Tab*. To activate the current pane's local menu, press *Alt-F10*. To activate a local menu item directly (without bringing up the menu), press the *Ctrl-(letter)* hot key, where *letter* is the menu item's highlighted letter.

Ctrl L

Local Module (in right pane)

When you choose Local Module in the Global Routines pane (the right pane), this Pick a Module dialog box pops up, listing all modules in your program.

Figure 4.29
The Pick a Module dialog box



After you highlight a module and choose OK, the profiler displays that module's local symbols in the right pane of the Routines window.

[Ctrl][A] **Areas (in both panes)**

The **A**reas command opens an Areas window and positions that window's highlight bar on the current routine (the one that's highlighted in the Routines window).

[Ctrl][C] **Callers (in both panes)**

The **C**allers command opens a Callers window and shows the current routine's callers.

[Ctrl][M] **Module (in both panes)**

The **M**odule command opens a Module window and shows the source code for the current routine.

[Ctrl][S] **Profile (in both panes)**

The Profile command opens the Execution Profile window and shows the profile statistics for the current routine.

Disassembly (CPU)

The Disassembly window (labeled "CPU" when it's on the screen) displays the current area in the Module window as disassembled source code. You use the Disassembly (CPU) window to help determine if you want to rewrite parts of your program in assembly language.

Figure 4.30
The Disassembly (CPU) window

```
[■]=CPU 80286================================3=[↑][↓]=
 OVRDEMO.49:  Write1;
►7EB6:0076 9A2500C37E    call   far OVRDEMO1.WRITE1
 OVRDEMO.50:  Write2;
►7EB6:007B 9A2500C07E    call   far OVRDEMO2.WRITE2
 OVRDEMO.51:  until KeyPressed;
►7EB6:0080 9AFA02C67E    call   far CRT.KEYPRESSED
 7EB6:0085 08C0          or     al,al
 7EB6:0087 74ED          je     OVRDEMO.49 (0076)
 OVRDEMO.52: end.
►7EB6:0089 89EC          mov    sp,bp
 7EB6:008B 5D            pop    bp
 7EB6:008C 31C0          xor    ax,ax
```

The left part of each disassembled line shows the instruction's address, either as a hexadecimal *Segment:Offset* value or, if the segment value is the same as the current CS register, as a CS:*Offset* value. If the window is wide enough (zoomed or resized), it also displays the bytes that make up the instruction. The disassembled instruction appears to the right of each line.

In the Disassembly (CPU) window, global symbols appear simply as the symbol name. Static symbols appear (generically) as

```
ModName.SymbolName      /* Turbo C */
ModName#SymbolName      { Turbo Pascal }
```

where *ModName* is the module name and *SymbolName* is the static symbol name. Line numbers appear (also generically) as

```
ModName.LineNumber      /* Turbo C */
ModName#LineNumber      { Turbo Pascal }
```

where *ModName* is the module name and *LineNumber* is the decimal line number.

In the Disassembly (CPU) window, you can use the *F2* function key to set area markers for each machine instruction you want to monitor. Any instructions marked in this fashion that have no symbol name appear in the Areas window as hex addresses in *Segment:Offset* form.

```
Goto...
Origin
Follow
Previous
View source
Mixed          Yes
```

With the Disassembly (CPU) window's local menu commands, you can go immediately to any of these locations:

- a specified address, to examine code
- the current program location (CS:IP)
- the destination address of the current instruction
- the previous instruction pointer address
- the address in the source code

You can also choose a local menu item to activate the Module window and move its cursor to the source for the current instruction, or to display disassembled instructions and source code three different ways.

To activate the window's local menu, press *Alt-F10*. To activate a local menu item directly (without bringing up the menu), press the *Ctrl-(letter)* hot key, where *letter* is the menu item's highlighted letter.

Ctrl G

Goto

When you choose **G**oto, a dialog box pops up and requests the address you want to go to. Enter a hexadecimal address, using the hex format of the language your program is in.

You can enter addresses outside of your program to examine code in the BIOS ROM, inside DOS, and in resident utilities.

The **P**revious command restores the Disassembly (CPU) window to the position it had before you chose **G**oto.

Ctrl O

Origin

You choose **O**rigin to position the window's highlight bar at the current program location as indicated by the CS:IP register pair. This command is useful when you have been looking at your code and want to get back to the address of the current instruction pointer (CS:IP), where your program is stopped.

The **P**revious command restores the Disassembly (CPU) window to the position it had before you chose **O**rigin.

Ctrl F

Follow

The **F**ollow command positions the Disassembly (CPU) window's highlight bar at the destination address of the currently highlighted instruction. The window scrolls to display the code at the address where the currently highlighted instruction will transfer control. For conditional jumps, the window shows the address as if the jump occurred.

You can use this command with CALL, JMP, conditional jump (JZ, JNE, LOOP, JCXZ, etc.) and INT instructions.

The **Previous** command restores the Disassembly (CPU) window to the position it had before you chose **Follow**.

Ctrl P

Previous

When you issue a command that changes the instruction pointer address (such as **Goto**, **Origin**, or **Follow**), the **Previous** command goes back to the address displayed before you issued that address-changing command. If you move around with the arrow keys and the *PgUp* and *PgDn* keys, the profiler does not remember the window's position, but you can always return to the origin, which is the current CS:IP.

Repeated use of the **Previous** command switches the Disassembly (CPU) window back and forth between two addresses.

Ctrl V

View Source

The **View Source** command opens a Module window and shows the source code for the current routine.

Ctrl M

Mixed

There are three ways to display disassembled instructions and source code in the Disassembly (CPU) window. You choose the window's display format with the local menu's **Mixed** command, which toggles between three choices: *No*, *Yes*, and *Both*.

- *No* means that no source code is displayed, only disassembled instructions.

  In *No* mode, global label names are still used in place of addresses for calls, jumps, and references to data items.
- *Yes* means that source code lines appear before the first disassembled instruction for that source line.

  The profiler automatically sets the window to *Yes* if your current module is a high-level language source module.
- *Both* means that source code lines replace disassembled lines for those lines that have corresponding source code; otherwise, the disassembled instruction appears.

  The profiler sets the window to *Both* if your current module is an assembler source module.

Use *Both* if you're profiling an assembler module and want to see the original source code line, instead of the corresponding disassembled instruction.

Run menu

```
Run                  F9
Program reset   Ctrl-F2
Arguments...
```

The **Run** menu provides three commands for running your program: **Run**, **Program Reset**, and **Arguments**. Control returns to the profiler when one of the following events occurs:

- Your program finishes running.
- Your program encounters an area marker whose operation is Stop.
- You interrupt execution with the interrupt key.

(Usually, the interrupt key is the *Ctrl-Break* key combination; you can change this to another key with TFINST, the profiler installation program. See Appendix B for details.)

You can run your program even if the Module window is closed (as long as there's a program loaded into Turbo Profiler).

When you choose **Run** | **Run** or **Run** | **Program Reset**, any statistics collected for a previous execution are reset. If you want to save a set of statistics, use **Statistics** | **Save** before you run or reset.

Run

The **Run** command runs your program and collects performance statistics.

If you set the Display Swapping option in the Display Options dialog box to Always, your program's output replaces the Turbo Profiler environment screen until the program finishes or you interrupt it.

If you set Display Swapping to None, the Turbo Profiler environment stays onscreen while your program runs, but the word RUNNING appears in the upper right corner of the screen.

Program Reset

Ctrl F2

The **Run** | **Program** Reset command reloads your program from disk. Use this command if you've run your program too far during a profiling session and need to restart execution at the start of the program.

If you select **Run** | **Program** Reset when the Module or Disassembly (CPU) window is active, the display in that window won't return to the start of the program. Instead, the cursor stays exactly where it was when you chose **Program** Reset.

Arguments

The program you're profiling might expect command-line arguments. With **Run | Arguments**, you can store your program's command-line arguments within Turbo Profiler. Then, when you choose **Run | Run** or **Run | Program Reset**, the profiler passes the arguments to your program just as if you had typed them in at the command line. You can change these arguments from within Turbo Profiler and rerun your program.

Enter the arguments exactly as you would at the DOS command line. (Do not enter the program name.)

Statistics menu

```
Callers        Disabled
Files           Enabled
Interrupts     Disabled
Overlays       Disabled
-----------------------
Profiling Options...
Accumulation    Enabled
Delete All
-----------------------
Save...
Restore...
```

The Statistics menu contains commands to

- specify the type of data the profiler will collect (callers, files, interrupts, overlays)
- set the profile mode to active or passive
- determine the number of program runs and areas
- turn automatic data collection on and off
- erase profile statistics
- save profile statistics to a file
- restore previously saved statistics

You can save all statistical information from the current profile to a .TFS file. Then, whenever you want to study the profile results, you can load that .TFS file—recovering all the saved statistics without having to rerun the profile.

This feature is most useful if your programs take a long time to run or profile. You can save multiple versions of profiles under different conditions, then restore each of the resulting profiles for quick comparison at a later date. Or, you can write a macro to automatically run five profiles with different options or area markers, then save the results in five different .TPS files.

You can then run the macro overnight and come back to your results in the morning.

Callers

You must run your program and accumulate some statistics before these windows show any information.

When you enable the **C**allers option, the profiler gathers statistics about which routines call other routines. To specify which routines you want call histories for, you choose the **C**allers command on the Module window's local menu or the **O**ptions command on the Areas window's local menu, then select the appropriate radio buttons under Callers and Areas.

After running your program and gathering the profile information, use the Callers window to look at the call-history statistics.

Gathering caller information consumes memory and slows down your program's run speed. If you don't need caller information, set **C**allers to *Disabled*.

Files

When you enable the **F**iles option, the profiler gathers statistics about which files your program opens, and which read and write operations take place.

After running your program and gathering the profile information, use the Files window to look at your program's file activity.

Gathering file-activity information consumes memory and slows down your program's run speed. If you don't need file-activity information, set **F**iles to *Disabled*.

Interrupts

When you enable the **I**nterrupts option, the profiler collects statistics about which interrupts your program calls. The profiler keeps separate statistics for DOS, Video, Disk BIOS and Keyboard interrupts.

After running your program and gathering the profile information, use the Interrupts window to look at which interrupts your program called.

Gathering interrupt information consumes memory and slows down your program's run speed. If you don't need interrupts information, set **I**nterrupts to *Disabled*.

Overlays

The Overlays option toggles whether statistics are collected for the overlays your program loads.

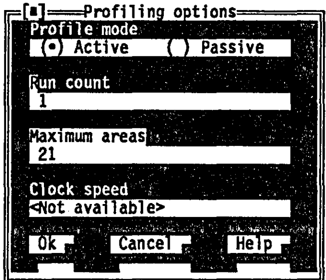
If your program does not contain overlays, an error message is displayed and the option remains disabled.

Gathering overlay information consumes memory and slows down the speed at which your program runs. If you don't need overlay information, you should disable this option.

Profiling Options

The **S**tatistics | **P**rofiling Options command opens the Profiling Options dialog box, shown here:

Figure 4.31
The Profiling Options dialog box



With the Profiling Options dialog box, you can set any of the following options:

- Profile Mode specifies which type of analysis the profiler will perform. The default is Active.

  Active analysis means the profiler will collect full statistical information for each marked routine: basic clock timing information, how many times it was called, and which routines called it.

  Passive analysis means the profiler will only collect basic clock timing information for each marked routine.
- Run Count sets how many times your program will run while the profiler collects statistics. The default is 1.
- Maximum Areas specifies the maximum number of areas you can divide the current program into. The default is twice the number of routines in the program being profiled.
- Clock Speed defines the speed of the timing clock, in ticks per second. The default is 100 ticks per second. Clock speed is available only in passive mode, not in active mode.

With the options in this dialog box, you can tailor the profiling session to meet your unique programming needs.

Active analysis mode provides the most detailed analysis of your program at the cost of slowing program execution speed. On the other hand, passive mode allows your program to run at almost full speed, but does not provide any information about how many times a routine was called or which routines called it.

If there are not many clock ticks during the time your profiled program runs, the data collected might not accurately reflect the time spent in various parts of the program. Running the program several times helps improve the accuracy by increasing the total number of data points collected. Speeding up the clock is another way to increase the total number of data points; this increases the accuracy of the timing statistics for each region at the expense of slowing down program execution speed.

The profiler doesn't actually time each area, but uses the interrupt timer to increment a timer count. When the program terminates, the profiler converts the values in the timer counts to execution times, based on the current setting of the Clock Speed option in the Profiling Options dialog box.

Accumulation

The **S**tatistics | **A**ccumulation option turns automatic data collection on and off, which means you can (1) collect data for a subset of all marked areas without removing any area markers and (2) manually turn data collection on after your program begins running.

To collect data for a subset of all marked areas, do this:

1. In the Areas window's local menu, choose **O**ptions to open the Area Options dialog box.
2. For those areas whose statistics you want, change the area marker from Normal to Enable (to start data collection) or to Disable (to stop data collection).
3. Set **S**tatistics | **A**ccumulation to *Disabled*.
4. Run your program. The profiler will not start collecting data until it trips an area marker that's set to Enable.

To turn on data collection manually after your program has started running, do this:

1. Set area markers.

2. Set **Statistics | Accumulation** to *Disabled*.
3. Run your program from the profiler (press *F9*).
4. When the program is in the appropriate run-time state, interrupt it.
5. Enable the collection of profile data (set **Statistics | Accumulation** to *Enabled*).
6. Resume program execution (press *F9* again).

Turbo Profiler starts accumulating statistics immediately for the marked areas.

When you would disable accumulation

Sometimes, when many different places in your program call a routine or family of routines, you want to know only the time spent in the routine, when a specific part of your code calls it or the time spent in the routine after a specific event.

To monitor only certain calls to a routine, use **Statistics | Accumulation** to disable data-collection at the start. Mark an area that enables collection just before the call to the routine that you want to collect statistics for (set the area marker to Enable. Mark another area that disables collection after the routine returns. You can also set Enable and Disable areas in unrelated parts of the code; do this when you want to collect statistics only after a certain event.

Example #1: Collecting only for a specific call to a routine

Suppose that you're only interested in calls to **abc** when it is called from **xyz**, but not at any other time.

```
=>    main()                         /* normal area marker at routine */
      {
        :
        abc();     /* don't want to collect statistics for this call
                                                                   */
        :
        xyz();
      }
=>    xyz()                          /* normal area marker at routine */
      {
        :
e>      abc();        /* want to collect statistics for this call */
d>      :
```

```
        }
=>      abc()                          /* normal area marker at routine */
        {
           :
        }
```

Notice the e> that enables collection and the d> that disables collection. You must disable **Statistics | Accumulation** before running your program, or the profiler will erroneously collect statistics for the first call to **abc** in **main.**

Example #2: Collecting after a certain event has occurred

Suppose that routine **xyz** behaves differently depending on some global state information controlled by the two routines **bufferon** and **bufferoff.** You are only interested in the time spent in **xyz** when *bufferflag* equals 1.

```
=>      main()                         /* normal area marker at routine */
        {
           :
           xyz();                       /* no statistics collected here */
           :
           bufferon();
           :
           xyz();             /* will collect statistics for this call */
           :
           bufferoff();
           :
           xyz();                       /* no statistics collected here */
        }
=>      bufferon()                     /* normal area marker at routine */
        {
           :
           bufferflag = 1;
e>      }
d>      bufferoff()                    /* normal area marker at routine */
        {
           :
           bufferflag = 0;
        }
=>      xyz()                          /* normal area marker at routine */
        {
           :
        }
```

Notice that the e> to enable collection and the d> to disable collection are not near the calls to **xyz**. Once again, you must disable data-collection at the start (by setting **S**tatistics | **A**ccumulation to *Disabled*), or the first call to **xyz** will erroneously contribute to the collected statistics.

Delete All

The **S**tatistics | **D**elete All command erases all statistics collected for the current profiling session—essentially wiping the data slate clean so you can start afresh. **D**elete All removes all profile data from the open profile report windows (Execution Profile, Callers, Interrupts, Files, and Overlays), but it does not delete the profiling options you've set.

Save

The **S**tatistics | **S**ave command saves the following data, settings, and options:

- all statistics for which collection was enabled when you ran the current profile (execution times and counts, callers, file activity, interrupts, overlays)
- all area information (area names, operations, callers, separate versus combined timing) displayed in the Execution Profile window

Once you've saved statistics to a file, you can recover them at any time with the **S**tatistics | **R**estore command.

When you choose **S**tatistics | **S**ave, this dialog box comes up:

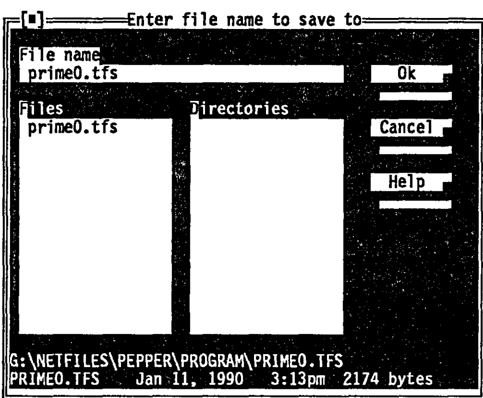


Figure 4.32
The Save dialog box

The Name input box lists a default .TPS file name (*progname*.TFS, where *progname* is the current program's name).

Saving Files

To save the current profile statistics to the default file, choose OK.

To save them to a different file,

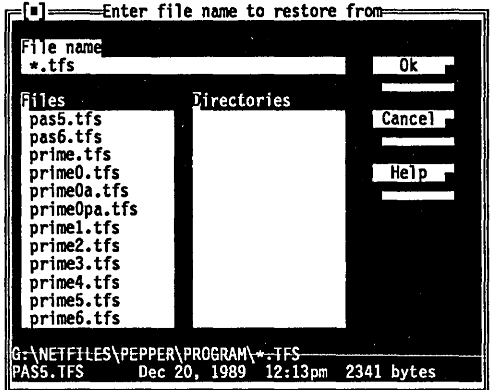
1. Activate the File | Name input box.
2. Type in the different file name (including disk drive and path, if you so choose).
3. Choose OK (or press *Enter*).

If the .TFS file already exists, a message box asks if it's all right to overwrite the file.

Restore

When you choose **Statistics | Restore**, this dialog box comes up:

Figure 4.33
The Restore dialog box



The Restore dialog box works just like the profiler's other file-loading dialog boxes. You can

- enter a file name or a specification (with DOS wildcards) in the File Name input box
- choose a different disk drive or directory from the directory tree
- choose a file name from the Files list box
- choose OK to complete the transaction (or choose Cancel to leave the dialog box without loading a file)
- choose Help to open a window of information about how to use the dialog box

After you type in or choose a .TFS file name and load that file, Turbo Profiler restores all the saved options, settings, and resulting statistical information to the environment screen.

Print menu

```
Statistics
Module...
Options...
```

Turbo Profiler's **Print** menu enables you to print the contents of any open profiler window to a new or existing disk file, or directly to the printer.

Statistics

The **Print | Statistics** command prints the contents of all open profiler windows (*except* the Module window) to the printer, or to the destination file named in the Printing Options dialog box.

Before you choose **Print | Statistics**, open the Printing Options dialog box (choose **Print | Options**) and verify that the current printing options (dimensions, output location, character set used, and—if you're printing to a file—destination file name) are what you want.

```
Append
Overwrite
Cancel
```

If you choose to print statistics to an existing disk file, a menu pops up so you can choose whether to append the existing file, overwrite it, or cancel the printing operation.

Module

From the Pick a Module dialog box accessed by choosing **Print | Module**, you specify which of your program's modules you want printed to the printer or to the disk file named in the Printing Options dialog box.

When you choose **Module**, this Pick a Module dialog box comes up:

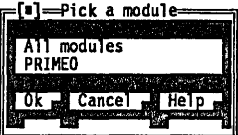


Figure 4.34
The Pick a Module dialog box

You can choose a specific module by name, or choose **All** Modules to print all your program's available source code. When the profiler prints a module, it produces an *annotated source listing*—which lists execution time and counts data next to each source line or routine you've marked as an area, as shown in the following figure.

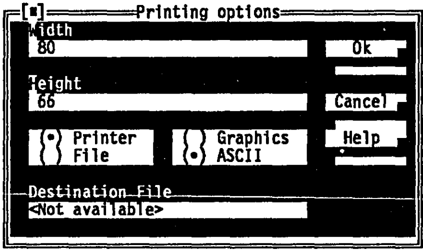
Figure 4.35
Annotated source listing for PRIME1

```
Program: C:\TPROFILE\PRIME1.EXE  File PRIME1.C

Time    Counts
                #include <stdio.h>

0.0090 999      prime(int n)
                {
                        int i;

0.0117 999              for (i=2; i<n; i++)
1.1456 78022                    if (n % i == 0)
0.0080 831                              return 0;
0.0017 168              return 1;
0.0101 999      }

0.0000 1        main()
                {
                        int i, n;

0.0000 1                n = 1000;
0.0000 1                for (i=2; i<=n; i++)
0.0255 999                      if (prime(i))
4.1670 168                              printf("%d\n", i);
0.0000 1        }
```

Options

When you choose **O**ptions from the **P**rint menu, this Printing Options dialog box comes up:

Figure 4.36
The Printing Options dialog box



- Width is the number of characters printed per line (default = 80).
- Height is the total number of lines per page (default = 66).
- The Printer/File radio buttons let you choose between sending the printed statistics to the current printer or to a file. The default is Printer.
- The Graphics/ASCII radio buttons let you toggle between printing characters from the IBM extended character set (including semigraphic characters) and printing only ASCII characters. The default is ASCII.
- Destination File is the disk drive (optional), path name (optional), and file name (required) of the printed disk file.

Options menu

```
Macros              ▶
Display options...
Path for source...
Save options...
Restore options...
```

With the **Options** menu, you can

- record a menu-command macro
- remove one or all menu-command macros
- set display options that control Turbo Profiler's overall appearance and operation
- specify directories (other than the current one) where Turbo Profiler will search for source code
- save your window layout, macros, options set in other menus, and some other miscellaneous options to a configuration file
- restore the settings and options previously saved in a configuration file

Macros

The **Options** | **Macro** command leads to another menu, shown here.

```
Create...       Alt =
Stop recording  Alt -
Remove
Delete all
```

With the commands on this Macro menu, you can define new macros or delete ones you've already assigned to a key. Mouse actions are not included in a macro.

To begin a macro recording session,

1. Choose **Options** | **Macro** | **Create**.

   A prompt asks for which key to assign the macro to. Type in a keystroke or combination of keys (for example, *Alt-M*). The message `RECORDING` appears in the upper right corner of the screen while you record the macro.

2. Type the keystrokes you want to record. The profiler responds normally to these keystrokes and mouse actions, as if you weren't recording a macro.

   When you finish recording keystrokes, choose **Options** | **Macros** | **Stop** Recording (or press the key you assigned the macro to—*Alt-M* in this example) to stop the recording session.

How macro recording and playback works

With the profiler's macro recording facility, you can record your frequently used keystroke sequences. During profiling, for example, you often repeat the same sequence of commands to get to a certain place in your program. This can be very tedious.

But, with **Options | Macro**, you can define a macro that records all the keys you press from the moment you first start the profiler until you have your program in the desired state. At that point, you can stop recording. If you must get back to the same place in your program, simply replay the macro.

You can't use **Options | Macro** to record keystrokes that must be typed as input to your program. You can only record Turbo Profiler commands.

To record your entire profiling session,

1. Start Turbo Profiler from DOS.
2. Choose **Options | Macros | Create** (or press *Alt=*) to define the macro. A prompt appears, asking which key you want the macro assigned to.
3. Choose a key that hasn't been assigned to a routine yet, such as *Shift* and a routine key (*Shift-F10*, for example).
4. Load your program. Turbo Profiler will automatically restore area settings from your program's .TFA file.
5. If you want to use a configuration different from the default, load the appropriate config file (choose **Options | Restore**).
6. Run your program: Stop execution if necessary.
7. Stop recording the macro: Choose **Options | Macros | Stop** Recording (or press *Alt-*).
8. Save the macro to a configuration file: Choose **Options | Save** Options, choose Macros in the Save Configuration dialog box, type the name of the corresponding configuration file (if you don't want to use the one listed in the dialog box), then choose OK or press *Enter.*
9. Continue running your program.

If your program requires you to type things to get to the next part of the recorded command sequence, you still must enter those keystrokes manually (you can do this while the macro is running). For programs that don't require any input, this command-recording mechanism can completely automate the restart procedure, saving many keystrokes.

When you save a macro to a configuration file, the profiler saves the total environment configuration, including opened and zoomed windows. So, if you record a macro that opens a window, and you don't close the window before saving the macro, the next

time you restore that configuration file, the window opens automatically—even if you don't execute the macro.

Create (Alt=)

The **C**reate command starts recording keystrokes that you want assigned to a macro key. *Alt=* is the hot key for **C**reate.

Stop Recording (Alt-)

The **S**top Recording command stops recording the keystrokes you're assigning to a macro key. Use this command after choosing **O**ptions | **M**acros | **C**reate. *Alt-* is the hot key for **S**top recording.

Remove

The **R**emove command removes a macro assigned to a single key. A prompt asks you to press the key of the macro you want removed.

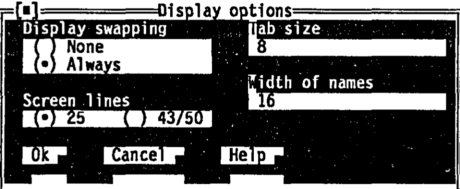
Delete

The **D**elete command removes all keystroke macro definitions and restores all keys to their original (default) meanings.

Display Options

The **O**ptions | **D**isplay Options command opens the Display Options dialog box, shown here.

Figure 4.37 The Display Options dialog box



With the Display Options dialog box, you can do any of the following:

- specify whether Turbo Profiler will swap screens while your program runs
- set how many columns each tab stop occupies in the Module window
- set the Turbo Profiler screen to 25-line or 43/50-line mode

- specify how wide your program's routine names display in the Execution Profile and Areas windows.

Display Swapping

The Display Swapping radio buttons provide two options for how Turbo Profiler swaps the User screen back and forth with the Turbo Profiler environment: *None,* and *Always.*

- *None* means don't swap between the two screens.

  Use this option if you're profiling a program that does not send any output to the User Screen.
- *Always* means swap to the User Screen every time your program runs.

  Use this option if your program does writing to the screen.

Screen Lines

You use Screen Lines to specify whether Turbo Profiler's screen uses the normal 25-line display or the 43- or 50-line display available on EGA and VGA display adapters.

One or both of these buttons will be available, depending on the type of video adapter in your PC. The 25-line mode is the only screen size available to systems with a monochrome display or Color Graphics Adapter (CGA).

Tab Size

With the Tab Size input box, you set how many columns each tab stop occupies, from 1 to 32 columns. You can reduce the tab column width to see more text in source files with a lot of tab-indented code.

Width of Names

The Width of Names input box is where you specify how wide routine names display in the Execution Profile, Callers, and Areas windows.

Path for Source

By default, Turbo Profiler looks for your program's source code in these places, in this order:

1. In the directory where the original compiler found the source files
2. In the current directory
3. In the directory that contains the program you're profiling

With the **O**ptions | **P**ath for Source command, you can add a list of directories that Turbo Profiler searches before it searches in the current directory.

Enter the new to-be-searched directories in this format:

```
Directory; Directory; Directory
```

For example,

```
C:\Borland\TC; C:\Borland\TASM
```

Save Options

With **O**ptions | **S**ave Options, you can save all your current profiler options to a configuration file on disk. Then, whenever you want to reset the profiler options to those saved settings, you can load that configuration file with **O**ptions | **R**estore Options.

When you choose **O**ptions | **S**ave Options, the following Save Configuration dialog box comes up:

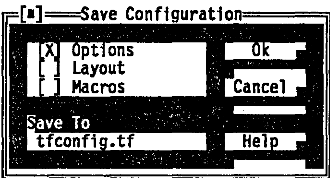


Figure 4.38
The Save Configuration dialog box

With this dialog box, you can save your current profiler setup's options, layout, and macros. Options, Layout, and Macros are check boxes; you can save one, two, or all three types of information to a configuration file.

- Options are menu options not saved in a .TFA or .TFS file (such as **O**ptions | **P**ath for Source, command-line options, and settings in the Display Options dialog box).
- Layout includes which windows are currently open, plus their order, position, and size.

- Macros are all keystroke macros currently defined.
- Save To lists the default configuration file, TFCONFIG.TF. To save your options there, choose OK (or press *Enter*).

  To save them to a different file, type in the different file's name (including disk drive and path, if you want), then choose OK (or press *Enter*).

Once you've saved options to a configuration file, you can recover them at any time with **O**ptions | **R**estore.

Restore

Options | **R**estore restores your profiling options from a disk file. You can have multiple configuration files, containing different macros, window layouts, and so on.

When you choose **O**ptions | **R**estore, this dialog box comes up:

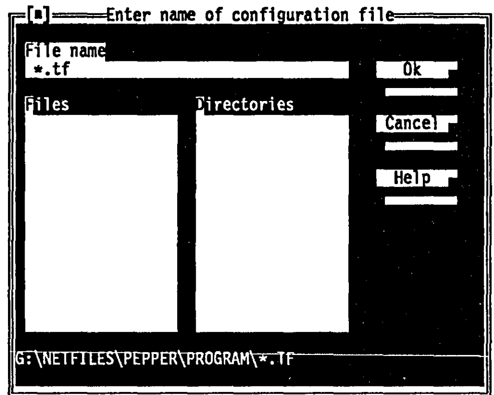


Figure 4.39
The Restore dialog box

The Restore dialog box works just like the profiler's other file-loading dialog boxes. You can

- enter a file name or a specification (with DOS wildcards) in the File name input box
- choose a different disk drive or directory from the directory tree
- choose a file name from the Files list box
- choose OK to complete the transaction (or choose Cancel to leave the dialog box without loading a file)
- choose Help to open a window of information about how to use the dialog box

After you type in or choose a configuration file name and load that file, Turbo Profiler restores all the saved options, settings, layout, and macros to the current Turbo Profiler environment.

(You can only restore a configuration file that was created by the **O**ptions | **S**ave Options command.)

Window menu

```
Zoom              F5
Next              F6
Next pane        Tab
Size/move    Ctrl-F5
Iconize/restore
Close         Alt-F3
Undo close    Alt-F6
--------------------
User screen   Alt-F5
1 Module
2 Profile
```

The **W**indow menu contains commands to

- manipulate Turbo Profiler's windows
- navigate within and through the windows
- toggle windows to icons, and vice versa
- close and reopen windows
- go to your program's output screen
- make an open window active

The commands in the top portion of the **W**indow menu are for moving about within the profiler's windowed environment and for rearranging the windows to your satisfaction. Most Turbo Profiler windows have all the standard window elements (scroll bars, a close box, zoom icons, and so on). Refer to page 65 for information on these elements and how to use them.

Zoom

The **Z**oom command zooms the active window (the one with a double-line border) to full-screen.

Next

The **N**ext command activates the window whose number succeeds the number of the current window.

Next Pane

In windows with multiple panes,the **N**ext Pane command moves the cursor to the next pane.

Size/Move

The **S**ize/Move command activates Turbo Profiler's window-arranging mode. You move the current window with ←, →, ↑ and ↓ arrow keys. Shifted arrow keys expand or contract the window. The legend in the status line explains which key combinations do which action. The hot key for this command is *Ctrl-F5*.

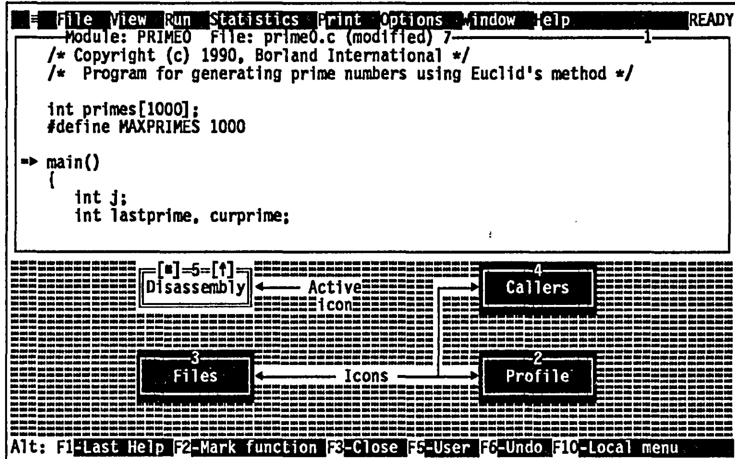
Iconize/Restore

The **I**conize/Restore command shrinks the active window to an icon or restores the active icon to a window. Figure 4.40 shows windows and icons on the Turbo Profiler screen.

Turbo Profiler's iconize feature is a handy tool for keeping several windows open without cluttering up the screen. A window icon is

a small representation of an open window, as shown in the following figure.

Figure 4.40
Windows and window icons



To make a window into its icon, choose **Iconize/Restore** from the **Window** menu, or click the iconize box in the window's top frame. To restore an icon to its previous size, choose **Iconize/Restore** again, or click in the icon's zoom box.

Close

The **Close** command temporarily removes the current window from the Turbo Profiler screen. To redisplay the window just as it was, choose **Undo** Close.

Undo Close

The **Undo** Close command reopens the most recently closed window and makes it the active window.

To go to one of the open windows listed in the bottom portion of the **Window** menu, choose the window from the list (click it or press the listed number). For a full explanation of how to manage windows, see page 67.

User Screen

Choose **Window** | **User** Screen (or press *Alt-F5*) to view your program's full-screen output.

[Alt][F5]

Press any key to return to the windowed environment.

The open window list

At the bottom of the Window menu is a numbered list of open windows. Press the number corresponding to one of these windows to make it the active window.

Help menu

```
Index            Shift-F1
Previous topic   Alt-F1
Help on help
```

The **Help** menu gives you access to online help in a special window. There is help information on virtually all aspects of the environment and Turbo Profiler. (Also, one-line menu and dialog hints appear on the status line whenever you select a command.)

To open the Help window, do one of these actions:

- Press *F1* or *Alt-F1* at any time (including from any dialog box or when any menu command is selected).
- Click Help whenever it appears on the status line or in a dialog box.

To close the Help window, press *Esc*, click the close box, or choose **Window | Close**.

Help screens often contain *keywords* (highlighted text) that you can choose to get more information. Press the arrow keys to move to any keyword; then press *Enter* to get more detailed help on the chosen keyword. (As an alternative, use the arrow keys to move to any keyword, then press *Enter* to choose it). You can press *Home* and *End* to go to the first and last keywords on the screen, respectively. With a mouse, you can click any keyword to open the help text for it.

Index

The **Help | Index** command opens a dialog box displaying a full list of help keywords (the special higlighted text in help screens that let you quickly move to a related screen).

You can page down through the list. When you find a keyword that interests you, choose it by using the arrow keys to move to it and press *Enter*. (You can also double-click it.)

Shift F1

The hot key for **Help | Index** is *Shift-F1*.

Previous Topic

The **Help | Previous** Topic command opens the Help window and redisplays the text you last viewed.

Turbo Profiler lets you back up through 20 previous help screens. You can also click the *PgUp* command in the status line to view the last help screen displayed.

Alt F1 *Alt-F1* is the hot key for **H**elp | **P**revious Topic.

Help on Help The **H**elp | **H**elp on Help command opens up a text screen that explains how to use the Turbo Profiler help system.

A P P E N D I X

A

Turbo Profiler's command-line options

This is the generic command-line format for running Turbo Profiler:

```
TPROF [tprof_options] [progname  [program_args]]
```

where *tprof_options* is a list of one or more command-line options for the profiler (see Table A.1). *progname* is the name of the program you want to profile. *program_args* is a list of one or more command-line arguments for the profiled program.

You can type `TPROF` without a program name or any arguments; if you do, you must then load the program you want to profile with Turbo Profiler's environment.

Here are some example Turbo Profiler command lines:

`tprof -sc prog1 a b`	Starts the profiler with the **–sc** option and loads program PROG1 with two command-line arguments, *a* and *b*.
`tprof prog2 -x`	Starts the profiler with default options and loads program PROG2 with one argument, *–x*.

The command-line options

All of Turbo Profiler's command-line options start with a hyphen (-). At least one space or tab separates each option from the TPROF command and any other command-line components.

To turn an option off at the command line, type a hyphen *after* the option. For example, **-vg-** explicitly turns the *graphics save* option off. Normally you'll only turn an option off if it's permanently enabled in the profiler's configuration file, TFCONFIG.TF. (You can modify the configuration file with the TFINST installation program described in the Introduction to this manual.)

Table A.1 summarizes Turbo Profiler's command-line options; we cover these options in greater detail in the following pages.

Table A.1 Turbo Profiler command-line options

Option	What it does
-c*file*	Reads in configuration file *file*.
-do	Runs the profiler on a secondary display.
-dp	Shows the profiler on one display page, the output of the profiled program on another.
-ds	Maintains separate screen images for the profiler and the program being profiled.
-h	Displays a help screen.
-?	Also displays a help screen.
-i	Enables process ID switching.
-m*N*	Sets the working heap size to *N* kilobytes.
-p	Enables mouse support.
-r	Enables profiling on a remote system over a serial link.
-rp*N*	Sets the remote link port to port *N*.
-rs*N*	Sets the remote link speed.
-sc	Ignores case when you enter symbol names.
-sd	Sets one or more source directories to scan for source files.
-vg	Saves complete graphics image on program screen.
-vn	Disables 43/50 line display.
-vp	Enables EGA palette save for program output screen.
-y*N*	Sets overlay area size to *N* kilobytes.
-ye*N*	Sets EMS overlay area size to *N* 16K pages.

Configuration file (-c)

This option tells Turbo Profiler to use the indicated configuration file. The default is TFCONVIG.TF; if you want to load a different one, you must use **–c**, followed immediately (no space) by the name of the configuration file you want to use.

Display update (-d)

All **–d** options affect the way Turbo Profiler updates the display.

–do Runs the profiler on a secondary display. You can view the program's screen on the primary display while Turbo Profiler runs on the secondary display.

–dp This is the default option for color displays. Shows the profiler on one display page, and the output of the program being profiled on another.

Using two display pages minimizes the time it takes to swap between two screens. You can only use this option on a color display, because only color displays have multiple display pages. You can't use this option if the profiled program uses multiple display pages.

–ds This is the default option for monochrome displays. Maintains separate screen images for the profiler and the program being profiled.

Each time you run the program or reenter the profiler, Turbo Profiler loads an entire screen from memory. This is the most time-consuming method of displaying the two screen images, but it works on any display and with programs that do unusual things to the display.

Help (-h and -?)

Both of these options display Turbo Profiler's command-line syntax and options.

Figure A.1
Turbo Profiler DOS-level help

```
Syntax: TA [options] [program [arguments]]   -x-  = turn option x off

  -c<file>     Use configuration file <file>
  -do,-dp,-ds  Screen updating: do=Other display, dp=Page flip, ds=Screen swap
  -h,-?        Display this help screen
  -i           Allow process id switching
  -m<#>        Set heap size to # kbytes
  -p           Use mouse
  -r           Use remote analysis
  -rp<#>       Set COM # port for remote link
  -rs<#>       Remote link speed: 1=slow, 2=medium, 3=fast
  -sc          No case checking on symbols
  -sd<dir>     Source file directory <dir>
  -vg          Complete graphics screen save
  -vn          43/50 line display not allowed
  -vp          Enable EGA/VGA palette save
  -y<#>        Set overlay area size in Kb
  -ye<#>       Set EMS overlay area size to # 16Kb pages
```

Process ID switching (-i)

Use this option to enable process ID switching.

Modify heap size (-m)

This option sets the working heap to *N* Kbytes; the syntax is

```
-mN
```

The default working heap size is 40K; the high boundary is 64K. If your program needs memory, use **-m** to reduce the amount of heap Turbo Profiler uses. You can also use **-m** to increase the amount of heap when profiling small programs. This option lets Turbo Profiler store transient information, such as command history lists, in the high end of the heap.

If TPROF runs out of memory, you get an error message whenever you try to open windows or perform other memory-consuming tasks. Statistics such as Callers may also be truncated.

Mouse support (-p)

This option enables mouse support.

Remote profiling (-r)

All **–r** options affect Turbo Profiler's remote profiling link.

–r Enables profiling on a remote system over a serial link. Uses the default serial port (COM1) and speed (115

Kbaud), unless these have been changed using TFINST.

–rp*N* Sets the remote link port to port *N*. Set *N* = 1 for COM1; *N* = 2 for COM2.

–rs*N* Sets the remote link speed to the value associated with *N*, as shown here:

N	Speed
1	9600 baud
2	40 Kbaud
3	115 Kbaud

Source code and symbols (-s)

All **–s** options affect the way Turbo Profiler handles source code and program symbols.

–sc Ignores case when you enter symbol names, even if your program has been linked with case-sensitivity enabled.

Without the **–sc** option, Turbo Profiler ignores case only if you've linked your program with the "case ignore" option enabled.

Note: The **–sc** option has no effect if you're profiling a Turbo Pascal program. Turbo Pascal is not case-sensitive.

–sd Sets one or more source directories to scan for source files; the syntax is

Like this, with no space between ***–sd*** *and the directory name.*

```
-sddirname
```

dirname can be a relative or absolute path and can include a disk letter. To set multiple directories, use the **–sd** option for each one. (You can only specify one directory name with each **–sd** option.) Turbo Profiler searches directories in the order specified.

If the configuration file specifies a directory list, the profiler appends the ones specified by the **–sd** option to that list.

Video hardware

(-v)

All **-v** options affect how Turbo Profiler handles the video hardware.

-vg Saves the program screen's complete graphics image. Requires an extra 8K of memory, but allows you to profile programs that use special graphics display modes. Try this option if your program's graphics screen becomes corrupted when running under Turbo Profiler.

-vn Disables the 43/50 line display mode. Specify this option to save some memory. Use **-vn** if you're running on an EGA or VGA and know you won't switch into 43- or 50-line mode once Turbo Profiler is running.

-vp Enables you to save the EGA/VGA palette for the program output screen. Use this option for programs that output to special EGA/VGA graphics modes.

Overlay area size

(-y)

The **-y** options are used to set the size of the overlay area size, either in main memory or in EMS memory.

-y*N* This option sets the overlay area size in main memory. The syntax is as follows, where *N* is the number of kilobytes you want to reserve:

```
-yN
```

Normally, Turbo Profiler uses a 80K code area size. The smallest area size that you can set is 20K. The largest is 250K.

Use this option if you do not have enough memory to load your program under Turbo Profiler, or if you are debugging small programs and want to improve Turbo Profiler's performance. The smaller the code area size, the more often Turbo Profiler loads program overlays from disk, and the slower it responds. With a larger code area, there is less memory available for the program you are profiling, but Turbo Profiler runs faster.

–ye*N* This option sets the overlay area size in EMS memory. Use this option if you need to free up some EMS memory for the program you are profiling. The syntax is as follows, where *N* is the number of 16K EMS pages you want to reserve:

```
-yN
```

For example, **–ye**4 sets the overlay area to four pages. The default is twelve 16K EMS pages.

A P P E N D I X

B

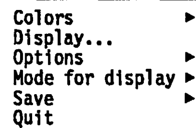
Customizing Turbo Profiler

Turbo Profiler is ready to run as soon as you make working copies of the files on the distribution disk. However, you can change many of the default settings by running the customization program called TFINST. You also can change some of the options using command-line options when you start Turbo Profiler from DOS. If you find yourself frequently specifying the same command-line options over and over, you can make those options permanent by running the customization program.

The customization program lets you set the following items:

- Window and screen colors and patterns
- Display parameters: screen swapping mode, integer display format, beginning display (source or assembler code), screen lines, tab column width, maximum tiled Watches size, fast screen update, 43-/50-line mode, full graphics saving, User screen updating, and log list length
- Your editor startup command and directories to search for source files and the Turbo Profiler help and configuration files
- User input and prompting parameters: history list length, beep on error, mouse, keystroke recording, and control-key shortcuts
- Source profiling: language options and case sensitivity
- NMI intercept, DOS process ID switching, expanded memory specification (EMS) for symbol table, and remote profiling
- Display mode

Running TFINST



To run the customization program, enter `TFINST` at the DOS prompt. As soon as TFINST comes up, it displays its main menu. You can either press the highlighted first letter of a menu option or use the ↑ and ↓ keys to move to the item you want and then press *Enter*. For instance, press *D* to change the display settings. Use this same technique for choosing from the other menus in the installation utility. To return to a previous menu, press *Esc*. You may have to press *Esc* several times to get back to the main menu.

Getting out

Choose **File | Quit** (*Alt-X*) at the menu bar to exit TFINST.

Setting the screen colors

Choose **Colors** from the main menu to bring up the **Colors** menu. It offers you two choices: **Customize** and **Default Color Set**.

Customizing screen colors

If you choose **Customize**, a third menu appears, with options for customizing windows, dialog boxes, menus, and screens.

Windows

To customize windows, choose the **Windows** command. This opens a fourth menu, from which you can choose the kind of window you want to customize: **Text**, **Statistics**, and **Disassembly** (the CPU window). Choosing one of these options brings up yet another menu listing the window elements, together with a pair of sample windows (one active, one inactive) in which you can test various color combinations. The screen looks like this:

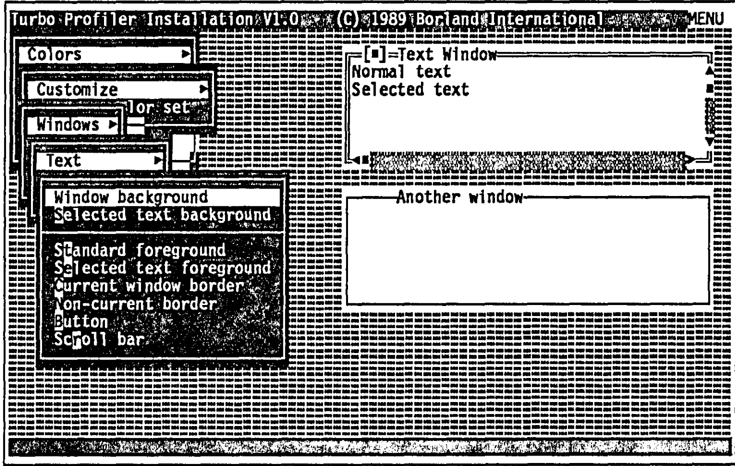


Figure B.1 Customizing colors for windows

When you select an item you want to change, a palette box pops up over the menu. Use the arrow keys to move around in the palette box. As you move the selection box through the various color choices, the window element whose color you are changing is updated to show the current selection. When you find the color you like, press *Enter* to accept it.

Turbo Profiler maintains three color tables: one for color, one for black and white, and one for monochrome. You can only change one set of colors at a time, based on your current video mode and display hardware. So, if you are running on a color display and want to adjust the black-and-white table, first set your video mode to black and white by typing `MODE BW80` at the DOS prompt, and then run TFINST.

Dialog boxes and menus

If you choose **D**ialogs or **M**enus from the **C**ustomize menu, a screen appears with a menu listing dialog box or menu elements, and a sample dialog box or menu for you to experiment with.

As with the **W**indows menu, choosing an item from the current menu opens a palette from which you can choose the color for that item.

Screen

```
Pattern for background ►
------------------------
Pattern background
Pattern foreground
Window move background
------------------------
Window move foreground
```

Choosing **Screen** from the **Customize** menu opens a menu from which you can access another menu with screen patterns and palettes for screen elements, as well as a sample screen background on which to test them.

The default colors

If you choose **Default** Color Set from the **Colors** menu, facsimiles of an active text window and an inactive window appear onscreen, which show you the default colors for their elements. A dialog box lets you select text, statistics, or low-level windows to view.

Setting Turbo Profiler display parameters

Choose **Display** from the main menu to bring up the Display Options dialog box.

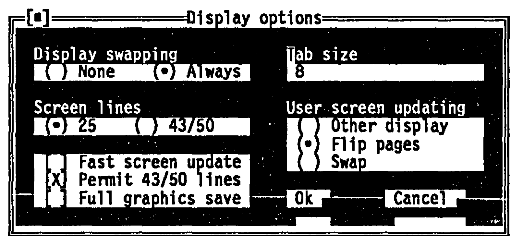


Figure B.2
The Display Options dialog box

These display options include some you can set from the DOS command line when you start up Turbo Profiler, as well as some you can set only with TFINST. See page 148 for a table of Turbo Profiler command-line options and corresponding TFINST settings.

Display Swapping

You use the Display Swapping radio buttons to control how Turbo Profiler switches between its own display and the output of the program you're profiling. You can toggle between the following settings:

None Don't swap between the two screens. Use this option if you're profiling a program that does not output to the User screen.

Always Swap to the User screen every time the user program runs. Use this option if your program writes to the User screen.

This is the default option.

Screen Lines

Use these radio buttons to toggle whether Turbo Profiler should start up with a display screen of 25 lines or a display screen of 43 or 50 lines.

☞ Only the EGA and VGA can display more than 25 lines.

Fast Screen Update

The Fast Screen Update check box lets you toggle whether your displays will be updated quickly. Toggle this option off if you get "snow" on your display with fast updating enabled. You need to disable this option only if the "snow" annoys you. (Some people prefer the snowy screen because it gets updated more quickly.)

Permit 43/50 Lines

Turning this check box on allows big (43-/50-line) display modes. If you turn it off, you save approximately 8K, since the large screen modes need more window buffer space in Turbo Profiler. This may be helpful if you are profiling a very large program that needs as much memory as possible to execute in. When the option is disabled, you will not be able to switch the display into 43-/50-line mode even if your system is capable of handling it.

Full Graphics Saving

Turning this check box on causes the entire graphics display buffer to be saved whenever there is a switch between the Turbo Profiler screen and the User screen. If you turn it off, you can save approximately 12K of memory. This is helpful if you are profiling a very large program that needs as much memory as possible to execute. Generally the only drawback to disabling this option is a

small number of corrupted locations on the User screen that don't usually interfere with profiling.

Tab Size

In this input box, you can set the number of columns between tab stops in a text or source file display. You are prompted for the number of columns (a number from 1 to 32); the default is 8.

User Screen Updating

The User Screen Updating radio buttons set how the User screen is updated when Turbo Profiler switches between its screen and your program's User screen. There are three settings:

Other Display Runs Turbo Profiler on the other display in your system. If you have both a color and monochrome display adapter, this option lets you view your program's screen on one display and Turbo Profiler's on the other.

Flip Pages Puts Turbo Profiler's screen on a separate display page. This option works only if your display adapter has multiple display pages, like a CGA, EGA, or VGA. You can't use this option on a monochrome display. This option works for the majority of profiling situations; it is fast and disturbs only the operation of programs that use multiple display pages—which are few and far between.

Swap Uses a single display adapter and display page, and swaps the contents of the User and Turbo Profiler screens in software. This is the slowest method of display swapping, but it is the most protective and least disruptive. If you are profiling a program that uses multiple display pages, use this option. Also use the Swap option if you shell to DOS and run other utilities or if you are using a TSR (such as SideKick Plus) and want to keep the current Turbo Profiler screen as well.

Turbo Profiler options

```
Directories...
Input & prompting...
Miscellaneous...
```

The **Options** command in the main menu opens a menu of options, which in turn open dialog boxes for you.

The Directories dialog box

This dialog box contains input boxes in which you can enter:

Editor program name Specifies the DOS command that starts your editor. This lets Turbo Profiler start up your favorite editor when you are profiling and want to change something in a file. Turbo Profiler adds to the end of this command the name of the file that it wants to edit, separated by a space.

Source directories Sets the list of directories Turbo Profiler searches for source files.

Turbo directory Sets the directory that Turbo Profiler searches for its help and configuration files.

The User Input and Prompting dialog box

This dialog box lets you set options that control how you input information to Turbo Profiler, and how Turbo Profiler prompts you for information:

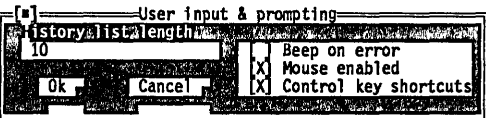


Figure B.3
The User Input and Prompting dialog box

History List Length

This input box lets you specify how many earlier entries are to be saved in the history list of an input box.

Beep on Error

By default, Turbo Profiler gives a warning beep when you press an invalid key or do something that generates an error message. The Beep on Error check box lets you change this default.

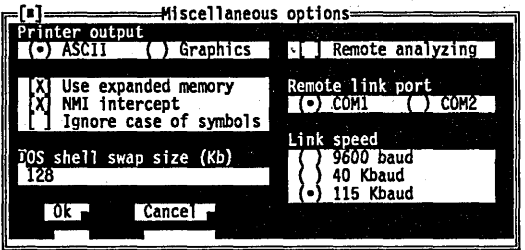
Mouse Enabled This check box controls whether Turbo Profiler defaults to mouse support.

Control Key Shortcuts This check box enables or disables the control-key shortcuts. When control-key shortcuts are enabled, you can invoke any local menu command directly by pressing the *Ctrl* key in combination with the first letter of the menu item. However, in that case, you can't use those control keys as WordStar-style cursor-movement commands.

The Miscellaneous Options dialog box

The Miscellaneous Options dialog box contains options controlling interrupts, EMS memory, DOS shell swapping, and remote profiling.

Figure B.4
The Miscellaneous Options dialog box



Printer Output This option lets you toggle whether to print both high and standard ASCII characters, or just the straight standard ASCII character set.

Use Expanded Memory Use this check box to toggle whether Turbo Profiler uses EMS memory for symbol tables. You can enable this option even if your program uses EMS as well.

NMI Intercept If your computer is a Tandy 1000A, IBM PC Convertible, or NEC MultiSpeed, or if Turbo Profiler hangs loading your system, run TFINST and turn off the NMI Intercept check box. Some computers use the NMI (nonmaskable interrupt) in ways that

conflict with Turbo Profiler, so you must disable Turbo Profiler's use of this interrupt in order to run the program.

Ignore Case of Symbol

If this check box is turned on, Turbo Profiler defaults to treating uppercase and lowercase the same. If it is off, case sensitivity is in effect.

DOS Shell Swap Size (Kb)

In this input box you can set the number of kilobytes for to increase the amount of memory set aside, so you can use the **File | DOS Shell** command even when a large program is loaded.

Remote Analyzing

This check box lets you toggle between enabling and disabling the remote link.

Warning!

Usually you won't want to turn this check box on, since that will mean that Turbo Profiler will start up every time using the remote link.

Remote Link Port

The Remote Link Port radio buttons let you choose between using the COM1 or COM2 serial port for the remote link.

Link Speed

The Link Speed radio buttons let you choose one of the three speeds that are available for the remote link: 9600 baud, 40,000 baud, or 115,000 baud.

Setting the mode for display

```
Default
Color
Black and white
Monochrome
LCP
```

Choosing **Mode** for Display from the main menu opens a menu from which you can select the display mode for your system.

Default

Turbo Profiler detects the kind of graphics adapter on your system and selects the display mode appropriate for it.

Color

If you have an EGA, VGA, CGA, MCGA, or 8514 graphics adapter and choose this as your default, the display will be in color.

Black and White

If you have an EGA, VGA, CGA, MCGA, or 8514 graphics adapter and choose this as your default, the display will be in black and white.

Monochrome

Choose this if you are using a color monitor with a Hercules or monochrome text-only adapter.

LCD

Choosing this instead of **B**lack and White if you have an LCD monitor makes your display much easier to read.

Command-line options and TFINST equivalents

Some of the options described in the previous section can be overridden when you start Turbo Profiler from DOS. The following table shows the correspondence between Turbo Profiler command-line options and the TFINST program command that permanently sets that option.

Table B.1 Command-line options and TFINST equivalents

Option	TFINST menu path and dialog box
	Display | Display Options
–do	`(•) Other Display`
–dp	`(•) Flip Pages`
–ds	`(•) Swap`
	Options | Input and Prompting | User Input and Prompting
–p	`[X] Mouse Enabled`
–p-	`[ ] Mouse Enabled`
	Options | Miscellaneous | Miscellaneous Options
–r	`[X] Remote Profiling`
–r-	`[ ] Remote Profiling`
	Options | Miscellaneous | Miscellaneous Options
–rp1	`(•) COM1`
–rp2	`(•) COM2`
	Options | Miscellaneous | Miscellaneous Options
–rs1	`(•) 9600 Baud`
–rs2	`(•) 40 KBaud`

Table B.1: Command-line options and TFINST equivalents (continued)

–rs3	`(•) 115 KBaud`
	Options | Miscellaneous | Miscellaneous Options
–sc	`[X] Ignore Case of Symbol`
–sc-	`[ ] Ignore Case of Symbol`
–sd	**Options | Directories | Directories**
	`Source Directories`
	Display | Display Options
–vn	`[ ] Permit 43/50 Lines`
–vn-	`[X] Permit 43/50 Lines`

For a list of all the command-line options available for TFINST.EXE, enter the program name followed by `-h`:

When you're through...

Saving changes

```
Save configuration file...
Modify TPROF.EXE
```

When you have all your Turbo Profiler options set the way you want, choose **S**ave from the main menu to determine how you want them saved.

Save Configuration File

If you choose **S**ave Configuration File, a dialog box opens, initialized to the default configuration file TFCONFIG.TF. You can accept this name by pressing *Enter*, or you can type a new configuration file name. If you specify a different file name, you can load that configuration by using the **–c** command-line option when you start Turbo Profiler. For example,

```
tprof -cmycfg myprog
```

You can also use the Turbo Profiler **O**ptions | **R**estore Configuration command to load a configuration once you have started Turbo Profiler.

Modify TPROF.EXE

If you choose **M**odify TPROF.EXE, any changes that you have made to the configuration are saved directly into the Turbo Profiler executable program file TPROF.EXE. The next time you enter Turbo Profiler, those settings will be your defaults.

If at any time, you want to return to the default configuration that Turbo Profiler is shipped with, copy TPROF.EXE from your

master disk onto your working system disk, overwriting the TPROF.EXE file that you modified.

Exiting TFINST

To get out of TFINST at any time, choose **Quit** from the main menu.

A P P E N D I X

C

Remote profiling

If your program requires a lot of memory, you may not be able to run both Turbo Profiler and your program on the same computer. Turbo Profiler's TFREMOTE utility solves this problem by letting you run Turbo Profiler on one system and the program you're profiling on another system.

Here are some examples of when you'd use remote profiling:

- When you attempt to load your program, Turbo Profiler gives one of these error messages:
  - "Not enough memory to load symbol table"
  - "Not enough memory"
- Your program loads properly under Turbo Profiler, but there's not enough memory left for it to operate properly.

Before resorting to remote profiling, you might see if Turbo Profiler is taking advantage of any EMS memory installed on your system.

In this appendix, you'll see how to profile very large programs by using a second PC connected to your main PC.

Remote hardware requirements

To use remote profiling, you need the following equipment:

- a development system with a serial port (this is the *local system*, where you'll run Turbo Profiler)

- another PC with a serial port and enough memory and disk space to hold the program you want to profile (this is your *remote system*)
- a null modem or serial printer cable to connect the two systems

Make sure that the cable connecting the two systems is set up properly: You can't use a straight-through extension-type cable. At the very least, the cable must swap the transmit and receive data lines (lines 2 and 3 on a 25-pin cable).

Use the cable to connect the two serial ports.

Installing TFREMOTE

Copy TFREMOTE.EXE onto the remote system. Put any files required by the profiled program on the remote system. (This includes data-input files, configuration files, help files, and so on.)

To put files on the remote system, you can use floppy disks or the TDRF Remote File Transfer Utility on the Turbo Debugger disks. (It's described in the MANUAL.DOC file for Turbo Debugger.)

If you want, you can put a copy of the program you want to profile onto the remote system. This is not essential: Turbo Profiler sends it over the remote link if necessary.

Once you start TFREMOTE and TPROF in remote mode, the Turbo Profiler commands work exactly the same as on a single system; there is nothing new to learn.

Because the program you're profiling is actually running on the remote system, any screen output or keyboard input to that program happens on the remote system. The **Window | User Screen** command has no effect when you're running on the remote link.

The remote system's CPU type appears as part of the CPU window title, with the word REMOTE before it.

To send files over to the remote system while running Turbo Profiler, go to DOS (choose **File | DOS Shell**) and then use TDRF to perform file-maintenance activities on the remote system.

To return to Turbo Profiler, type `EXIT` at the DOS prompt and continue profiling your program.

Starting the remote link

Before you start TFREMOTE on the remote system, set the current directory (the remote system's) where you want it. This is important because TFREMOTE puts the program to be profiled into the directory that is current when you start TFREMOTE.

If the remote system's serial port is set up as COM1, type

```
TFREMOTE -rp1 -rs3
```

to start TFREMOTE.

If the remote system's serial port is set up as COM2, type

```
TFREMOTE -rp2 -rs3
```

to start TFREMOTE.

Both of these commands start the link at its maximum speed (115 Kbaud). This speed works with most PCs and cable setups. (See page 154 for how to start the link at a slower speed if you experience communication difficulties.) Use the command-line option **–rs1** on a PS/2.

TFREMOTE signs on with a copyright message, then indicates that it is waiting for you to start Turbo Profiler on the other end of the link. To stop and return to DOS, press *Ctrl-Break*.

Starting Turbo Profiler on the remote link

To start Turbo Profiler on the remote link, use one of the following DOS command lines:

- serial port COM1: `tprof -rp1 -rs3` *`filename`*
- serial port COM2: `tprof -rp2 -rs3` *`filename`*

When the link starts successfully, the message "Waiting for handshake" appears on the remote system, and the activity indicator on the local system displays READY. Turbo Profiler's normal display then follows on the local system. Use the command-line option **–rs1** on a PS/2.

TPROF and TFREMOTE use the same command-line options to set the speed and serial port: For them to work properly, you must set them both to the same speed (with the **–rs** option).

Turbo Profiler also has the **–r** command-line option, which starts the remote link using the default speed and serial port. Unless

you've changed the defaults using TFINST, **-r** specifies COM1 at 115 Kbaud (the fastest baud speed.)

Here's a typical Turbo Profiler command line to start the remote link:

```
tp -rs3 myprog
```

This begins the link on the default serial port (usually COM1) at link speed 3 (115 Kbaud), and loads the program MYPROG into the remote system (if it's not already there).

Loading the program to the remote system

Turbo Profiler is smart about loading the to-be-profiled program onto the remote disk. It looks at the date and time of the copy of the program on the local system and the remote system. If the local copy is later (newer) than the remote copy, Turbo Profiler presumes you've recompiled or relinked the program and sends it over the link. At the highest link speed, this happens at a rate of about 11K per second. A typical 60K program takes about six seconds to transfer, so don't be alarmed if there's a delay when you want to load a new program.

To indicate that something's happening, the screen on the remote system adds up the bytes of the file as Turbo Profiler transfers them.

TFREMOTE command-line options

Here are the TFREMOTE command-line options. You can start an option with either a hyphen (-) or a slash (/).

Table C.1 TFREMOTE command-line options

Option	What it does
-?	Displays a help screen
-h	Displays a help screen
-rs1	Slow speed, 9600 baud
-rs2	Medium speed, 40 Kbaud
-rs3	High speed, 115 Kbaud (default)
-rp1	Port 1, (COM1) (default)
-rp2	Port 2, (COM2)
-w	Writes options to executable program file

Started with no command-line options, TFREMOTE uses the default port and speed built into the executable program file

(COM1 and 115 Kbaud), unless you've changed them with the **–w** option.

You can make TFREMOTE's command-line options permanent by writing them back into the TFREMOTE executable program image on disk. To do this, specify **–w**, along with the other options you want to make permanent, on the command line. TFREMOTE prompts for the name of the executable program to write to; when you enter a new (nonexistent) executable file name, TFREMOTE then creates the file. If you press *Enter*, it overwrites the currently running program (TFREMOTE).

Here's an example. If you type this command line at the DOS prompt,

```
tfremote -w -rs2 -rp2
```

Enter the name of the program to modify: `tfrmt40k.exe`. TFREMOTE will create a copy of TFREMOTE.EXE named TFRMT40K.EXE, where the default speed is 40 Kbaud (**-rs2**) and the default port is COM2 (**-rp2**).

For a list of all TFREMOTE.EXE command-line options available, type this at the DOS command line:

```
TFREMOTE -h
```

If you are running on DOS version 3.0 or later, the prompt indicates the path and file name from which you executed TFREMOTE. You can accept this name (press *Enter*), or enter a new executable file name.

If you're using DOS 2.xx, you must supply the full path and file name of the executable program.

Getting it all to work

Because the remote profiling setup involves two different computers and a cable between them, you might run into difficulty getting everything to work together.

If you experience problems, try these troubleshooting techniques:

1. Check your cable hookups.
2. Try running the link at the slowest speed (use the **–rs1** command-line option when starting up both TFREMOTE and TPROF).

3. If it works OK using **–rs1**, go on to try **–rs2** (the middle speed).

Some hardware and cable combinations don't always work properly at the highest speed. If you can only get remote profiling to work at a lower speed, consider trying a different cable or different computers.

TFREMOTE messages

Here is a list of the messages you might receive when you're working with TFREMOTE.

***nn* bytes downloaded**
TFREMOTE is sending a file to the remote system. This message shows the progress of the file transfer. At the highest link speed (115 Kbaud), transfer speed is about 11K per second.

Can't create file
TFREMOTE can't create a file on the remote system. This can happen if there isn't enough room on the remote disk to transfer the executable program across the link.

Can't modify exe file
You specified a file name to modify that is not a valid copy of the TFREMOTE utility. You can only modify a copy of the TFREMOTE utility with the **–w** option.

Can't open exe file to modify
TFREMOTE can't open the file name you specified to be modified. You have probably entered an invalid or nonexistent file name.

Download complete
Your file has been successfully sent to TFREMOTE.

Download failed, write error on disk
TFREMOTE can't write part of a received file to disk. This usually happens when the disk fills up. You must delete some files before TFREMOTE can successfully download the file.

Enter program file name to modify
If you are running on DOS version 3.0 or later, the prompt indicates the path and file name from which you executed TFREMOTE. You can accept this name (press *Enter*), or enter a new executable file name.

If you're running DOS version 2.xx, you must supply the full path and file name of the executable program.

Interrupted
You pressed *Ctrl-Break* while waiting for communications to be established with the other system.

Invalid command line option
You gave an invalid command line option when you started TDRF from the DOS command line.

Link broken
The program communicating with TFREMOTE has stopped and returned to DOS.

Link established
A program on the other system has just started to communicate with TFREMOTE.

Loading program *name* from disk
Turbo Profiler has told TFREMOTE to load a program from disk into memory in preparation for profiling.

Program load failed, EXEC failure
DOS could not load the program into memory. This can happen if the program has become corrupted or truncated. Delete the program file from the remote system's disk: This forces Turbo Profiler to send a new copy over the link. If this message happens again after deleting the file, you should relink your program using TLINK on the local system and try again.

Program load failed; not enough memory
The remote system doesn't have enough free memory to load the program you want to profile. This only happens with very large programs, because TFREMOTE only takes about 15K of memory.

Program load failed; program not found
TFREMOTE could not find the program on its disk. This should never happen since Turbo Profiler downloads the program to the remote system if TFREMOTE can't find it.

Program load successful
TFREMOTE has finished loading the program Turbo Profiler wants to profile.

Reading file *name* from Turbo Profiler
This appears on your remote screen so that you know when a remote file is being sent to Turbo Profiler.

Unknown request: *message*
TFREMOTE has received an invalid request from the local system (where you're running Turbo Profiler). If you get this message, check that the link cable is in good working order. If you keep getting this error, try reducing the link speed (use the **–rs** command-line option).

Waiting for handshake (press Ctrl-Break to quit)
TFREMOTE has started and is waiting for a program on the local system to start talking to it. To return to DOS before the other system initiates communication, press *Ctrl-Break*.

A P P E N D I X

D

Virtual profiling on the 80386 processor

Turbo Profiler lets you use the full power of systems that have the 80386 processor. Virtual profiling lets the program you're profiling use the full address space below 640K, just as if no profiler were loaded. (Turbo Profiler is loaded into extended memory, above the 1MB address point.)

You profile exactly as you would normally use Turbo Profiler, except that your program loads and runs at exactly the same address that it does when it's not being profiled. This is extremely useful both for profiling programs that are large, and for finding bugs that go away if the program is loaded higher in memory, as it is when it is being profiled normally.

Virtual profiling also lets you watch for reads or writes to arbitrary memory or I/O locations, all at full or nearly full processor speed. This gives you some of the power of a hardware profiler at no additional cost.

Equipment required for virtual profiling

You must have a computer based on the 80386 or 80486 processor in order to use the virtual profiler. You must also have 700K of available extended memory. If you have used up your extended memory for RAM disks, caches, and so forth, you may want to make a special CONFIG.SYS or AUTOEXEC.BAT file that

removes some of these programs when you want to use virtual profiling.

Installing the virtual profiler device driver

Before starting the virtual profiler, you must make sure that you have installed its device driver, TDH386, in your CONFIG.SYS file. (TDH386 is included in the Turbo Debugger package.) Do this by including a line similar to the following in CONFIG.SYS:

```
DEVICE = TDH386.SYS
```

If you have placed the TDH386.SYS device driver somewhere other than in the root directory, make sure that you include that directory path as part of the device driver file name.

Normally, the virtual profiler lets you have up to 256 bytes of DOS environment strings. If this is not enough, or if you don't need that much and would like to conserve as much memory as possible, use the **–e** option in CONFIG.SYS to set the number of bytes of environment. For example,

```
DEVICE = TDH386.SYS -e2000
```

reserves 2000 bytes for your DOS environment variables.

Starting the virtual profiler

You start the virtual profiler much as you would normally start Turbo Profiler, with a command line like this:

```
TF386 [options] program [program options]
```

In other words, you simply enter `TF386` instead of `TPROF`. TF386 then takes care of finding the Turbo Profiler executable program and loading it into extended memory.

If you have other programs or device drivers that use extended memory, such as RAM disks, caches, or whatever, you must tell TF386 how much extended memory to set aside for these other programs. Do this by using the **–e** command-line option. Follow the **–e** with the number of kilobytes (K) of extended memory used by other programs. For example,

```
TF386 -e512 myprog
```

This command line informs TF386 that you want to reserve the first 512K of extended memory for other programs. (It isn't necessary to do this if your machine supports the XMS standard; TF386 allocates memory from the XMS device driver if one is present.)

Since you probably always reserve the same amount of extended memory for other programs, TF386 gives you a way to permanently set the amount of extended memory to reserve. Use the **–w** option with the **–e** option to specify that you want the **–e** value to be permanently set in the TF386 executable program file.

You'll then be prompted for the name of the executable program. If you are running on DOS 3.0 or later, the prompt indicates the path and file name that you executed TF386 from. You can accept this name by pressing *Enter*, or you can enter a new executable file name. The new name must already exist and be a copy of the TF386 program that you have already made.

If you are running on version 2.x of DOS, you will have to supply the full path and file name of the TF386 executable program.

Here is a complete list of command-line options for TF386.EXE:

–? Accesses help on TF386.

–e#### Specifies the number of kilobytes of extended memory being used by other programs or by the program you're profiling. (You don't need this option if your system supports the XMS standard.)

–f#### Enables EMS emulation through paging (in extended memory) and sets the page frame segment to #### (in hex). The last three digits must be 000 (like C000 or E000). Note that this option only applies to Turbo Profiler's EMS calls. If you don't use this option when you load TF386, TF386 will not be able to use EMS.

–f- Disables EMS emulation (presumably to override a previous command-line option).

–w Modifies TF386.EXE with the new default value of **–e** or **–f**. You can enter a new executable file name that does not already exist, and TF386 will create the new executable file.

Note that TF386.EXE options must appear first in the command line before any Turbo Profiler options or the program name. For example,

```
TF386 -e1024 -fD000 -w
```

reserves 1024K of extended memory, enables EMS emulation with a page frame of D000, and modifies TF386.EXE with these values.

For a list of all the command-line options available for TF386.EXE, just type `TF386 -?` or `TF386 -h` and press *Enter.*

Note: If you have an 80386-based machine and want to read the command-line options for TF386.EXE, TDH386.SYS must be loaded.

Differences between normal and virtual profiling

Most things work exactly the same whether you are profiling normally or using the 80386 virtual profiling capability. The following items behave differently:

- When you use the **File | DOS** Shell command to run a DOS command, the program you're profiling is never swapped to disk. This means you may not always have enough memory to run other programs from the DOS prompt.
- Your program can use nearly all of the 80386 instructions, with the exception of the privileged protected-mode instructions: **CLTS, LMSW, LTR, LGDT, LIDT, LLDT.**
- Even though you can use all the 80386 extended addressing modes and 32-bit registers during virtual profiling, you can't access memory above the 1MB point. If you try to do so, an exception interrupt will be generated, and Turbo Profiler will regain control.
- You can't use virtual profiling if you're already running a program or device driver that uses the virtual and protected modes of the 80386 processor. This includes programs such as:
  - DesqView operating environment
  - Microsoft Windows-386 operating environment
  - QEMM.SYS, the QuarterDeck EMS simulator
  - CEMM.SYS Compaq EMS simulator

- 386^MAX

  If you normally use one of these or similar programs, you will have to stop them or unload them before using TF386.

TF386 error messages

TF386 generates one of the following messages when it can't start, and then returns to the DOS prompt. You must correct the condition before you can start TF386 successfully.

TF386 error: 80386 device driver missing or wrong version
You must install the TFH386.SYS device driver in your CONFIG.SYS file before you invoke TF386 from the DOS command line.

TF386 error: Can't enable the A20 address line
TF386 can't access the memory above 1MB. This may happen if you're running on a system that is not exactly IBM compatible.

TF386 error: Can't find TPROF.EXE
TF386 could not find T.EXE.

TF386 error: Couldn't execute TPROF.EXE
TF386 could not run TPROF.EXE.

TF386 error: Environment too long; use –e#### switch with TFH386.SYS
You need to change the **–e** option as described on page 160.

TF386 error: Not enough Extended Memory available
TF386 ran out of memory. You need to get more memory for your machine or free up memory (by reducing a RAM disk, for example).

TF386 error: Wrong CPU type (not an 80386)
You are not running on a system with an 80386 or 80486 processor.

The following errors might occur if you're trying to modify TF386 with the **–w** option:

TF386 error: Cannot open program file

TF386 error: Cannot read program file

TF386 error: Cannot write program file

TF386 error: Program file corrupted or wrong version

TDH386.SYS error messages

There are only two possible error messages associated with the TFH386.SYS driver:

Wrong CPU type: TDH386 driver not installed

Invalid command line: TDH386 driver not installed

A P P E N D I X

E

Prompts and error messages

Turbo Profiler displays messages and prompts at the current cursor location. This chapter describes the prompts and error and information messages Turbo Profiler generates.

We tell you how to respond to both prompts and error messages. All the prompts and error messages are listed in alphabetical order, with a description provided for each one.

Turbo Profiler prompts

Turbo Profiler displays a prompt in a dialog box when you must supply additional information to complete a command. The prompt describes the information that's needed. The contents may show a history list (previous responses) that you have given.

You can respond to a prompt in one of two ways:

- Enter a response and accept it by pressing *Enter.*
- Press *Esc* to cancel the dialog and return to the menu command that opened it.

Some prompts only present a choice between two items (like Yes/No). You can use *Tab* to select the choice you want and then press *Enter,* or press *Y* or *N* directly. Cancel the command by pressing *Esc.*

For a more complete discussion of the keystroke commands to use when a dialog box is active, refer to Chapter 4.

Here's an alphabetical list of all the prompts and messages generated by dialog boxes:

Enter code label to position to
Enter the address you wish to examine in the Code pane. The Code pane shows the disassembled instructions at the specified address.

Enter command line arguments
Enter the command-line arguments for the program you're profiling. You can modify the current command-line arguments or enter a new set.

You will then be prompted whether you want to reload your program from disk. Some languages or programs, such as programs written in C, require you to reload the program before the arguments take effect.

Enter file name to restore areas from
Enter the name of the file to restore areas from. If you specify an extension to the file name, it will be used. Otherwise the extension .TFA will be used.

Enter file name to restore from
Enter the name of the file to restore the statistics to. If you specify an extension to the file name, it will be used. Otherwise the extension .TFS will be used.

Enter file name to save areas to
Enter the name of the file to save the current areas to. If you specify an extension to the file name, it will be used. Otherwise the extension .TFA will be used.

Enter file name to save to
Enter the name of the file to save the current statistics to. If you specify an extension to the file name, it will be used. Otherwise the extension .TFS will be used.

Enter file name to write to
Enter the name of a file to send the report to. If the file already exists, it will be overwritten.

Enter name of configuration file
Enter the name of a configuration file to read or write. If you are reading from a configuration file, you can enter a wildcard mask and get a list of matching files.

Enter maximum number of areas
Enter the maximum number of areas that you wish to divide the program into. Since each area takes up some memory, try to set the maximum to as low a value as is reasonable.

Enter maximum symbol width to display
Enter the width for symbol names on reports and for symbol names displayed in windows.

Enter new directory
Enter the new drive and/or directory name that you want to become the current drive and directory.

Enter new line number
Enter a new line number to position the text file to. The first line in the file is line 1. If you specify a line number that is greater than the last line in the file, the file is positioned to the last line.

Enter new page height
Enter the number of lines on a page to be sent to the printer or a disk file. A normal 11-inch sheet of paper has 66 lines.

Enter new page width
Enter the width of the page for reports sent to the printer or a disk file. A normal sheet of paper has 80 columns.

Enter program name to load
Enter the name of the program to load. If the program has the .EXE extension, you don't have to specify it; if the program has any other extension, you must supply it.

If you supply a wildcard specification or accept the default *.EXE, a list of matching files is displayed for you to select from.

Enter program run count
Enter the number of times that you want to run the program and accumulate performance statistics. The more times you run your program, the more accurate the clock timings will be.

Enter routine name to add
Enter the name of the function you wish to include, exclude, or set.

Enter search string
Enter a character string to search for. You can use a simple wildcard matching facility to specify an inexact search string; for example, use * to match zero or more of any characters, and ? to match any single character.

Enter source directory list

Enter the directory or directories to search for source files.

If you want to enter more than one directory, separate the different directory paths with a space or a semicolon (;). These directories will be searched, in the order that they appear in this list, for your source files.

Enter tab column spacing

Enter a number between 1 and 32 that specifies how far apart tab columns will be when Turbo Profiler displays files in a Module window.

Usually, you use tabs for each level of control-structure nesting in your source code. If your source code has deeply nested control structures, you can use a small tab value (like 2 or 3) so that your source doesn't disappear off the right side of Module windows.

Pick a caller

Pick a routine from the list of callers. You will then be positioned to that routine in the window that you picked from the previous menu.

Pick a method name

You have specified a routine name that can refer to more than one method in an object. Pick the correct one from the list presented, with the arguments you want.

Pick a module

Select a module name to view in the Module window. You are presented with a list of all the modules in your program. Either use the cursor keys to move to the desired module, or start typing the name of the module. As you type the module name, the highlight bar will move to the first module that matches the letters you typed. When the highlight bar is on the desired module, press Enter.

Pick a source file

Pick a new source file to display in the Module window. The list shows all the source files that make up the module.

Pick a symbol name

Pick a symbol from the list of displayed symbols. You can start to type a name, and you will be positioned to the first symbol, starting with what you have typed so far.

Pick a window
Pick the window you want to make the active window by moving the highlight bar to it and pressing *Enter* or choosing the OK button.

You can close a window by moving the highlight bar to that window name and pressing the *Del* or *Ctrl-Y*.

Pick interrupt
Pick an interrupt from the list of interrupts built in to Turbo Profiler.

Turbo Profiler error messages

Turbo Profiler uses error messages to tell you about things you haven't quite expected. Sometimes the command you have issued cannot be processed. At other times the message warns that things didn't go exactly as you wanted.

Error messages are normally accompanied by a beep. You can turn off the beep in the customization program, TFINST.

Already recording, do you want to abort?
You are already recording a keystroke macro. You can't start recording another keystroke macro until you finish the current one. Press *Y* to stop recording the macro; *N* to continue recording the macro.

Ambiguous symbol *symbol name*
You have entered a member function or data item name and Turbo Profiler can't tell which of the multiple instances of this member you mean.

This can happen when a member name is duplicated in two multiply inherited classes. Use the classname:: override to name explicitly the member you want.

Bad configuration file name
You have specified a nonexistent file name with the **–c** command-line option when you started Turbo Profiler. The built-in default configuration values are used instead.

Bad interrupt number entered
You have entered an invalid interrupt number. Valid interrupt numbers are 9 to FF.

Bad line number *line number*
The line number that you have entered does not exist or does not correspond to a line of source code.

You can specify only source line numbers that are in functions, not those outside functions.

Bad module name *module name*
The module name that you have entered does not exist.

Can't execute DOS command processor
Either there was not enough memory to execute the DOS command processor, or the command processor could not be found (the COMSPEC environment variable is either absent or incorrect). Make sure that the COMSPEC environment variable correctly specifies where to find the DOS command processor.

Can't open printer
There was an error sending to the printer. Check that the printer is online and not out of paper.

Can't swap user program to disk
The program being profiled could not be swapped to disk. There is probably not enough room on the disk to swap the program. You will not be able to edit any files or execute DOS commands until some more room is made available.

Clear all existing statistics
When you change the clock tick rate, all existing statistics become meaningless and must be cleared. If you do not confirm the statistics clear, the clock speed will not be changed.

Edit program not specified
You tried to use the Edit local menu command from a Module or Disk File window, but you cannot edit the file because Turbo Profiler does not know how to start your editor.

Use the configuration program TFINST to specify an editor.

Error reading areas file
An error occurred while you were restoring the areas. Make sure that the disk is ready.

Error reading statistics file
An error occurred while you were restoring the collected statistics. Make sure that the disk is ready.

Error saving configuration
Your configuration could not be saved to disk. The disk might be full, or there might be no more free directory entries in the root directory.

You can use the File | DOS Shell command to go to DOS and delete a file or two to make room for the configuration file.

Error swapping in user program, program reloaded
An error occurred while you were reloading your program that was swapped to disk. This usually means that the swap file was accidentally deleted.

You will have to reload your program using the Run | Program Reset command before you can continue profiling.

Error writing areas file
An error occurred while you were writing to the area file that stores information on marked areas in your program. Your disk is probably full.

Make sure that the disk is ready and that there is enough room on the disk.

Error writing statistics file
An error occurred while you were writing to the statistics file that stores your program statistics. Your disk is probably full.

Make sure that the disk is ready and that there is enough room on the disk.

Help file TFHELP.TFH not found
You asked for help but the disk file that contains the help screens could not be found. Make sure that the help file is in the same directory as Turbo Profiler.

Invalid areas file
The file name you specified to restore areas from is not formatted correctly. Make sure you specified a file name that was created by Turbo Profiler.

Invalid statistics file
The file you specified to restore statistics from has an invalid format. Make sure the file name you specified was created using the Statistics | Save command.

Maximum number of areas has been reached
There is no more room to add areas. Use the Options | Number of Areas command to increase the amount of memory set aside for areas.

Maximum number of interrupts being monitored
You can't watch any more interrupts; you have already told Turbo Profiler to watch as many interrupts as it is capable of doing. You will have to use the local menu Remove command to remove an existing interrupt before you can add any more.

No help for this context
You pressed *F1* to get help, but Turbo Profiler could not find a relevant help screen. Please report this to Borland technical support.

No caller information for this function
The highlighted line is in a function for which no caller information was collected. Caller information is collected only if it is explicitly requested.

No file name was given
You have indicated that you wish to output a file, but you have not specified a file name. You must either specify a file name or switch to another output location before you can leave the dialog.

No modules with statistics
There are no modules with any statistics collected, so there is nothing to print.

No previous search expression
You have used the Next command from the local menu of a text pane, without previously issuing a Search command. First use Search to specify what to search for, then use Next to look for subsequent instances.

No program loaded
You tried to issue a command that requires a program to be loaded. There are many commands that can only be issued when a program is loaded, for example, the commands in the Run menu. Use the File | Open command to load a program before issuing these commands.

No source file for module *module name*
The source file cannot be found for the module that you wish to view. The source file is searched for first in the current directory, and then in any directories specified in the configuration file and then in any directories specified by the command line **–sd** option.

Not a code address
You have entered an address that is not a code address in your program. You can only set profiling areas on code addresses.

Not enough memory for selected operation
You issued a command that has to create a window, but there is not enough memory left for the new window. You must first remove or reduce the size of some of your windows before you can reissue the command.

Not enough memory to load program
Your program's symbol table has been successfully loaded into memory, but there is not enough memory left to load your program. If your system has EMS memory, make sure that Turbo Profiler is set to use it for the symbol table. You can use TFINST to set this option.

If you don't have EMS or your program doesn't load even with EMS, you can hook two systems together and run Turbo Profiler on one system and the program you're analyzing on the other. See Appendix C for more information on how to do this.

Not enough memory to load symbol table
There is not enough room to load your program's symbol table into memory. The symbol table contains the information that Turbo Profiler uses to show you your source code and program variables. If you have any resident utilities consuming memory, you may want to remove them and then restart Turbo Profiler. You can also try making the symbol table smaller by having the compiler generate debug symbol information only for those modules you are interested in analyzing.

When this message is issued, your program itself has not yet been loaded. This means you must free enough memory for both the symbol table and your program.

Overlay not loaded
You have attempted to examine code in an overlay that is not loaded into memory. You can only examine code for overlays that are already in memory.

However, you can still look at the source code for a module in a Module window.

Overwrite existing macro on selected key
You have pressed a key to record a macro, and that key already has a macro assigned to it. If you want to overwrite the existing macro, press *Y*; otherwise, press *N* to cancel the command.

Overwrite *file name*?
You have specified a file name to write to that already exists. You can choose by entering *Y* to overwrite the file, replacing its previous contents, or you can cancel the command by entering *N* and leave the previous file unchanged.

Path not found
You entered a drive and directory combination that does not exist. Check that you have specified the correct drive and that the directory path is spelled correctly.

The current drive and directory are left as they were before you issued the command.

Path or file not found
You specified a non-existent or invalid file name or path when you were prompted for a file name to load. If you do not know the exact name of the file you want to load, you can pick the file name from a list by pressing *Enter* when the dialog box first appears.

Possibly you entered wildcard specification that is not valid. Only the normal DOS wildcard characters * and ? can be used.

Premature end of string in *symbol name*
The symbol name that you have entered is incomplete. If you specify a module name, it must be followed by either a line number or local symbol name.

Press key to assign macro to
Press the key that you want to assign the macro to. Then, press the keys to do the command sequence that you want to assign to the macro key. The command sequence will actually be performed as you type it. To end the macro recording sequence, press the key you assigned the macro to. This macro will be recorded on disk along with any other keystroke macros.

Press key to delete macro from
Press the key for the macro that you want to delete. The key will then be returned to its original pre-macro function.

Program already terminated, Reload?
You have attempted to run or step your program after it has already terminated. If you choose *Y*, your program will be reloaded. If you choose *N*, your program will not be reloaded, and your run or step command will not be executed.

Program does not have overlays
The program you are profiling does not have any overlays, so you can't open an Overlay window.

Program has invalid symbol table
The program that you wish to load has a symbol table with an invalid format. Re-create your .EXE file and reload it.

Program has no symbol table
The program you want to analyze has been successfully loaded, but it does not contain any debug symbol information. Relink the program so that it has a symbol table.

Program linked with wrong linker version
The program you wish to load was linked with an old version of the linker. You must use the latest version of the linker for profiling programs.

Program not found
The program you wish to load does not exist. Check that the name you supplied to the File | Open command is correct and that you supplied a file name extension if it is different from .EXE.

Program out of date on remote, send over link?
You have specified a program to analyze on the remote system, but it either does not exist on the remote, or the file is newer on the local system than on the remote system.

If you press *Y*, the program is sent across the link. If you press *N*, the program is not sent, and the File | Open command is aborted.

You'll usually respond with *Y*. If you are running the link at the slowest speed (using the **-rs1** command-line option), you might want to abort the command with *N* and transfer the file to the remote system using a floppy disk.

Reload program so arguments take effect?
With most programs, you must reload after changing their arguments. Always press *Y* at this prompt, unless you know what you're doing.

When you press *Y*, a Run | Program Reset command is automatically performed for you.

Reload program so new area count takes effect?
In order for Turbo Profiler to reallocate the memory used for statistics areas, your program must be unloaded from memory and then reloaded and executed from the beginning again.

Press *Y* to make this happen, or press *N* if you can wait for the next manual program load for the new area size to take effect.

Run out of space for keystroke macros
There is not enough memory to record all your keystroke macro.

Search expression not found
The specified text string or byte list is not present in the file. Since the search proceeds forward from the current cursor position, you should return to the top of the file via the *Ctrl-PgUp* hot key, then repeat the search.

Symbol name not found
The symbol name that you supplied is not a valid symbol name.

Symbol not a routine name
The symbol name that you supplied is not a valid name of a routine.

Symbol not found
You have entered an expression containing an invalid symbol name. A valid symbol name consists of either:

1) a global symbol name.

2) a module name, followed by #, followed by a local symbol name.

3) a module name, followed by a #, followed by a decimal line number.

Symbol *SymbolName* is a data symbol
The program symbol name that you have entered refers to data in the program, and not to code. You can only specify code addresses to be profiled.

Syntax error in symbol *SymbolName*
You have entered an invalid symbol name. A valid symbol name consists of either:

1) a global symbol name.

2) a module name, followed by a #, followed by a local symbol name.

3) a module name followed by a # followed by a decimal line number.

Tab width must be between 1 and 32
You have entered an invalid value for the tab width. Tab columns must be at least 1 column wide, but no more than 32 columns.

Too many files match wildcard mask
You specified a wildcard file mask that included more than 100 files. Only the first 100 file names are displayed.

Video mode switched while flipping pages
You have started Turbo Profiler with a display updating mode that does not allow display pages to be saved, and the program that you are profiling has switched into a graphics mode.

Turbo Profiler has changed the display mode back to text display, so the screen contents of the program you are profiling have been lost.

To avoid this, start Turbo Profiler with display-swapping enabled (**-ds** command-line option).

I N D E X

43/50-line mode
  disabling *136*
8514 graphics adapter *147*
43- and 50-line displays *124*
386^MAX *162*
* (asterisk)
  search wildcard *80*
» in dialog boxes *70*
–? option (display help) *133*
  TF386 virtual profiler *161, 162*
? (search wildcard) *80*
80x87 numeric coprocessors *See* numeric coprocessros
80386 processor
  extended address modes *162*
  instructions
    TF386 virtual profiler and *162*
  profiling *159-164*
    device driver *160*
  registers *162*
≡ (System) menu *71*
→ (arrows) in dialog boxes *69*

A

About Turbo Profiler command *73*
Accumulation option *114*
  partial statistics and *45*
  start and stop points
    maximum *45*
  status of *45*
activating
  menu bar *64*
active
  analysis *See also* passive analysis; profiling, analysis modes
  functions *See* functions, active
  windows *See* windows, active
active analysis
  area markers and *51*
  disk I/O and *51*
  passive analysis vs. *51*
  setting *50*
adapters
  video *See* video adapters
Add Areas command *10, 81, 102*
  Module window *81*
Add command *97*
Address option *89*
addresses
  jumping to *81, 108*
addressing modes, 80386 processor *162*
algorithms
  analysis
    line-count information and *48*
    statistics for *48*
  multipass *60*
All Areas command
  Remove Areas *82*
All Callers option
  Area Options dialog box *103*
  Stack Trace dialog box *84*
All option *89*
All Routines command *84*
  Add Areas *81*
area files
  writing to, problems with *171*
area markers *See also* areas
  active analysis and *51*
  defined *43*
  lines
    all *81*
    current *81*
    in current routine *81*
    removing *82*
  normal
    defined *35*

program execution speed and *52*
removing *91*
removing all *82*
return statements and *35*
routine entry
defined *35*
routines *78*
all *81*
current *81*
current module *81*
removing *82*
Area Options dialog box *103*
areas *See also* area markers
adding *102*
call paths for *84*
current
as disassembled source code *78*
changing *33*
settings for *82*
specifying profiling action *83*
statistics *56*
erasing *56, 57*
default *43*
default statistics *43*
defined *43*
execution counts and times and *34*
function-entry markers *See* area markers, function entry
how they work *30*
isnpecting *102*
markers *See* area markers
maximum *113*
measuring efficiency *20*
names *101*
normal markers *See* area markers, normal
profiler behavior when entering *34*
program size and *44*
removing *82, 102*
settings
considerations *43*
creating *10*
default *44*
saving *40*
statistics *78, See* statistics
.TFA files and *40*
Areas command *84*
Routines window *106*
Areas window *78, 101*
area markers
removing *91*
Callers option and *92*
arguments
command-line options *175*
defined *5*
program
setting *110*
Arguments command *110*
arrays
accessing for optimimum speed *58, 61*
sorting *58*
arrows (→) in dialog boxes *69*
ASCII
high
printing *16*
high versus standard, printing *146*
assembler
instructions
protected-mode *162*
assembly language
assessing value of *106*
asterisk (*)
search wildcard *80*
AUTOEXEC.BAT
virtual profiling and *160*

B

bar
magnitude *12*
title *66*
beep on error, setting *145*
Beep on Error check box (TFINST) *145*
Bentley, John *7, 28*
Borland compilers
Turbo Profiler and *42*
Both option *88*
Disassembly (CPU) window *109*
bubble sort
quicksort vs. *58*
buffers
data *59*
overlays *See* overlays, buffers
bugs
finding
memory allocation and *159*

buttons
  choosing *69*
  in dialog boxes *69*
  radio *70*
bytes
  searching for, problems with *176*

C

C
  functions
    terminology *5*
  profiling *8*
C++ *See* object-oriented programs
C++ programs
  expressions, problems with *169*
–c option (load configuration file) *133*
  problems with *169*
cable
  null modem
    remote profiling and *152*
call history
  dynamic
    algorithm analysis and *48*
    program structure analysis and *48*
    program testing and verification and *48*
call paths
  sorting *94*
call stack
  active routines and *35*
  setting *45*
  size *84*
Call Stack option *45*
Called option *94*
Callers command *83*
  call stack and *45*
  logging call paths and *36*
  Routines window *106*
Callers option
  Area Options dialog box *103*
  Areas window and *92*
  setting *92*
  Statistics window *112*
Callers window *56, 78, 91*
  restructuring programs and *93*
calls
  information on
    passive analysis and *51*
  overhead *31*
Cancel button *69*
case and switch statements
  verifying *49*
case sensitivity
  enabling *147*
  option *135*
Change Dir command *75*
Change Dir dialog box *75*
character strings
  searching for *167*
characters
  graphic
    printing *16*
  high versus standard ASCII, printing *146*
check boxes
  Beep on Error (TFINST) *145*
  choosing *69*
  Control Key (TFINST) *146*
  Fast Screen (TFINST) *143*
  Full Graphics Saving (TFINST) *143*
  Ignore Case of Symbol (TFINST) *147*
  Mouse Enabled (TFINST) *146*
  NMI Intercept (TFINST) *146*
  Permit 43/50 Lines (TFINST) *143*
  Remote Analyzing (TFINST) *147*
  Use Expanded Memory (TFINST) *146*
chevron symbol (») *70*
clock
  combined and separate *46, 83*
Clock Speed command *113*
close boxes *66*
Close command *128*
code
  area size, setting *136*
Collection command *56*
  file activity *100*
  interrupts *97*
color monitors
  customizing *140-142*
color tables *141*
Colors menu (TFINST) *140*
combined clock
  timer data and *46, 83*
Combined option
  Area Options dialog box *103*

command line
  options *131-137*
    configuration file (–c) *133*
    display update (–d) *133*
    help (–h and –?) *133, 155*
    modify heap (–m) *134*
    mouse support (–p) *134*
    overlays (–ψ) *136*
    process ID switching (–i) *134*
    remote profiling (–ρ) *134, 153*
    saving (–ω) *154, 155*
    source code and symbols (–σ) *135*
    syntax *131*
    table *132*
    TFREMOTE *153, 154, 155*
    toggling *132*
    video hardware (–ϖ) *136*
  remote profiling *154*
  viewing from IDE *128*
command-line arguments
  passing to your program *110*
command-line options *131-137*
  arguments *175*
  overriding *148*
  saving *139*
  TF386 virtual profiler *161*
    help with *162*
  TFINST vs. *148-149*
commands *See also* specific command names
  choosing with a mouse *64*
  choosing with keyboard *64*
  prompts and *165*
communications, remote systems *147*
Compaq EMS simulator *162*
compatibility *3*
compiling
  for profiling *42*
COMSPEC environment variable (DOS) *170*
configuration files
  changing default name *149*
  creating *19*
  directory paths
    setting *145*
  loading *133, 166*
  problems with *169*
  saving *149*
    problems with *171*
  saving macros to *122, 125*
  saving options to *125*
  virtual profiling and *160*
Control Key check box (TFINST) *146*
control-key shortcuts
  enabling *146*
coprocessors *See* numeric coprocessors
copyright information *73*
Count option *95*
counting
  sampling vs. *37*
Counts option *88*
CPU window *See* Disassembly (CPU) window
Create command *121, 123*
current instruction pointer
  returning to *108*
Current option *89*
Current Routine command
  Add Areas *81*
  Remove Areas *82*
cursor
  moving *80*
customizing Turbo Profiler *139-150*

D

–d option (display update) *133*
data
  caching *59*
  collecting and displaying *See* statistics
  evaluation order *59*
  sets
    size and profiling *41*
  structures
    optimizing *58*
debugging
  information
    global symbols *104*
    profiling and *42*
    Turbo C identifiers and *91*
default buttons *69*
Default Color Set command (TFINST) *142*
default settings *139*
  restoring *149*
Delete All command
  interrupts *98*
  Macros menu *123*
  statistics *117*

DesqView *162*
Detail command *57, 100*
Detail option
  Display Options dialog box *101*
device drivers
  TDH386.SYS
    installing *160*
  TFH386.SYS
    error messages *164*
  virtual profiling and *162*
  XMS *161*
dialog boxes *68*, *See also* specific dialog box names
  Accept Color Set (TFINST) *142*
  arrows in *69*
  Change Dir *75*
  check boxes in *69*
  customizing *141*
  Directories (TFINST) *145*
  Display Options
    TFINST *142*
  entering text *70*
  escaping out of *165*
  hot keys *65*
  Miscellaneous Options (TFINST) *146*
  prompts in *165-169*
Dialogs command (TFINST) *141*
directories
  changing *75*
  default *145*
  how searched *42*
  new *76*
  paths
    problems with *174*
    setting *168*
  source files *79, 85, 125, 135*
Directories dialog box (TFINST) *145*
Disable option *103*
disabling statistics collection *83*
Disassembly (CPU) window *78, 106*
  restoring *108*
disks
  distribution *3*
  writing to, problems with *170, 171*
display
  options *142-144*
Display command *57*
  Display Options dialog box *101*
  Execution Profile window *88*
  Interrupts window *98*
  Overlays window *95*
Display menu (TFINST) *142*
Display option *101*
Display Options command
  Options menu *123*
Display Options dialog box *12, 88, 101, 123*
  TFINST *142*
Display Swapping options *124*
Display Swapping radio buttons
  TFINST *142*
displays *See also* screens
  43 and 50 lines *124*
  buffer, saving *143*
  color *148*
  customizing
    color tables *141*
  dual *77*
  EGA and VGA *124*
  modes *141*
    defaults, setting *147*
    problems with *177*
  options *147*
    colors *140-142*
  pages *144*
  problems with *148*
  swapping *144*
  updating *143*
distribution disks
  copying *3*
DOS
  command processor, problems with *170*
  COMSPEC environment variable *170*
  output
    viewing from IDE *128*
  running programs from
    TF386 virtual profiler and *160, 162*
  shelling to
    display swapping and *144*
  versions
    TF386 virtual profiler and *161*
  wildcards *75*
DOS Shell command *77, 170*
DOS Shell Swap Size input box (TFINST) *147*

dual monitors
  DOS command line and *77*
dual screens *133*
Duration option *101*

E

–e option (TF386 virtual profiler) *161*
Edit command *170*
  problems with *170*
editor
  installing *86*
efficiency
  measuring *20*
EGA *See* Enhanced Graphics Adapter
ellipsis mark (...) *64, 68*
EMS
  emulation and TF386 virtual profiler *161*
  enabling *146*
  simulators *162*
  usage *77*
Enable option *103*
enabling statistics collection *83*
Enhanced Graphics Adapter (EGA) *See also* graphics; video adapters
  43-line mode
    disabling *136*
  line display *143*
  palette
    saving *136*
  screen *124*
Eratosthenes
  Sieve of *28*
errors
  messages *169-177*
    beep, enabling *145*
    memory *151*
    TF386 virtual profiler *163-164*
    TFREMOTE *156-158*
Esc hot key *69*
Every Line command *10*
Every Line in Module command
  Add Areas *81*
  Remove Areas *82*
example programs
  PLOST*.* *32*
execution counts and times
  areas and *34*
  conditional statements and *49*
  default behavior *43*
  passive analysis and *51*
  program structure analysis and *48*
  program testing and verification and *48*
  resource monitoring and *48*
Execution Profile window *55, 78, 86*
  description *11*
  displaying data in *12*
  Module window and *87*
execution timing
  statistics for *48*
exiting
  TFINST *150*
expressions
  entering
    problems with
      invalid variables and *176*
  optimizing *60*
extended memory *159*
  TF386 virtual profiler and *159, 160*
    problems with *163*

F

–f option (TF386 virtual profiler) *161*
far heap
  profiling and *38*
Fast Screen Update check box (TFINST) *143*
features *1*
  environment *63*
File
  command *85, 86*
  dialog box *86*
  menu *73*
files
  access
    monitoring *45*
    profiling purposes and *48*
    tracking *78*
  activities
    displaying *57*
  area
    problems with *171*
  AUTOEXEC.BAT
    virtual profiling and *160*

configuration *See* configuration files
disk
problems with *171*
disks
problems with *170*
distribution
list of *3*
executable program *167*
TF386 virtual profiler and *161*
HELPME!.DOC *145*
information on *76*
loading *21*
cancelling *74*
opening
problems with *136, 174*
wildcard masks and *177*
opening and loading *73*
overwriting *174*
README *3*
source
current routine *106*
directories *85, 125*
inspecting *94*
list of *85*
loading *168*
problems with *172*
options *135*
setting directory path *145*
viewing *88*
viewing with statistics *18*
where searched for *42*
statistics *See* statistics
SWAP.$$$ *171*
TDH386.SYS *160, 162*
.TFA *See* .TFA files
.TFS *See* .TFS files
TPROF.EXE *149*
Files option *45*
Statistics menu *112*
Files window *56, 78, 98, 99*
Filter command *49, 56, 89*
filters *See* statistics
Follow command
Disassembly (CPU) window *108*
43/50-line mode
disabling *136*
43- and 50-line displays *124*
frequency collisions
solving *37*
Frequency option *89, 94*
Full Graphics Saving check box (TFINST) *143*
function-entry area markers *See* area markers, function
functions *See also* routines
C and Pascal
terminology *5*

G

Get Info command *76*
global menus *See* menus
global symbols
list of available *104*
Goto command
Disassembly (CPU) window *108*
Module window *81*
Graph option *101*
graphics *See also* graphics adapters
color tables *141*
display buffer, saving *143*
image
saving *136*
palette
EGA
saving *136*
problems with
snow *143*
graphics adapters *147, See also* graphics
display options *148*
display pages *144*
EGA *143*
Hercules *148*
monochrome text-only *148*
VGA *143*
graphs
file activity *101*

H

–h option (help) *133, 155*
hardware
adapters
display options
setting *143*

requirements
TF386 virtual profiler *159*
hardware requirements *3*
heap *See also* memory
far
profiling and *38*
modifying *134*
size
default *134*
Help
button *69*
menu *129*
window
closing *129*
keywords in *129*
opening *129*
help
accessing *129*
problems with *171, 172*
command-line options
TF386 virtual profiler *162*
TFINST *149*
help on help *130*
index *129*
keywords *129*
option *133*
previous topic *129*
status line *68*
TFREMOTE *155*
Help on Help command *130*
HELPME!.DOC
setting directory path for *145*
Hercules graphics adapter *148*
History List Length input box (TFINST) *145*
history lists *70*
choosing from *75*
length, setting *145*
History option *96*
hot keys
dialog boxes *65*
enabling *146*
Esc *69*
menus *65*
using *65*

I

–i option (process ID switching) *134*
I/O
disk
active analysis and *51*
passive analysis and *51*
keyboard
profiling and *41*
IBM
graphic characters
printing *16*
IBM PC Convertible and NMI *146*
Iconize/Restore command *127*
icons
restoring *127*
identifiers
Turbo C *91*
underscores and *91*
Ignore Case of Symbol check box (TFINST) *147*
Immediate Callers option
Area Options dialog box *103*
Stack Trace dialog box *84*
Index command *129*
input boxes
DOS Shell Swap Size (TFINST) *147*
History List Length (TFINST) *145*
Tab Size
TFINST *144*
Inspect Areas command *102*
Inspect command *94, 96*
INSTALL.EXE *3*
installation *3*
TDH386.SYS device driver *160*
TF386 virtual profiler *159*
TFREMOTE (remote profiling utility) *152*
instructions
current
pointer
returning to *108*
displaying *109*
pointer
address of *109*
integrated environment *63-130*
DOS screen and *128*
interrupts
adding to statistics collection *97*
amount of time in *97*
display formatting options *98*
exception, TF386 virtual profiler and *162*

execution timing and resource monitoring and *48*
monitoring *45*
names *97*
NMI *146*
number of calls to *97*
passive analysis and *51, 52*
pick list *98*
removing from statistics collection *98*
removing from window *57*
statistics *97*
subroutines *97*
Interrupts option *45*
Statistics menu *112*
Interrupts window *56, 78, 96*
I/O
options *145*
watching, TF386 virtual profiler and *159*

K

keyboard
choosing buttons with *69*
choosing commands with *64*
input
profiling and *41*
keys
cursor-movement
TFINST *140*
keystrokes
recording *176*
problems with *169*
keywords
Help windows *129*

L

labels
moving cursor to *80*
LCD screens *148*
Line command *80*
line counts
algorithm analysis and *48*
program verification and testing and *48*
line numbers *167*
lines
jumping to *81*
marking *10, See* area markers
moving cursor to *80*
Lines in Routine command
Add Areas *81*
Remove Areas *82*
link
remote
speed *135*
Link Speed radio buttons (TFINST) *147*
list boxes *71*
file names *75*
searching incrementally *129*
Load command
cancelling *74*
local menus *See* menus
Local Module option *105*
local system
remote profiling and *151*
Longest option *88*
loops
optimizing *58, 59*

M

–m option (modify heap) *134*
macros *121, 121-123*
creating *121, 123*
deleting *123*
recording
problems with *173, 176*
removing *123*
saving *122*
stop recording *123*
Macros command *121*
magnitude bar *12*
maps
memory
profiler use *38*
markers
area *See* area markers
math coprocessors *See* numeric coprocessors
Maximum Areas command *113*
memory *136, 143, See also* heap
accessing
TF386 virtual profiler and *162*
addresses
high *159*
allocation
problems with *173*

TF386 virtual profiler and *160*
EMS *See* EMS
error messages *151*
heap
size *134*
overlays and *95*
problems with *173*
profiler use of *38*
stop and start points and *45*
usage *77*
watching
TF386 virtual profiler and *159*
menu bar *See* menus
menus
accessing *63*
customizing *141*
hot keys *65*
local *See* menus, pop-up
opening *64*
Options *121-127*
pop-up *64*
TFINST *140*
with arrows (▸) *64*
with ellipsis marks (...) *64, 68*
Menus command (TFINST) *141*
Microsoft
Windows *162*
Microsoft mouse
compatibility *63*
Miscellaneous Options dialog box (TFINST) *146*
Mixed command
Disassembly (CPU) window *109*
Mode for Display menu (TFINST) *147*
modem
remote profiling and *152*
Modify TPROF.EXE command (TFINST) *149*
Module command *84, 90*
Print menu *119*
Routines window *106*
Module option *89*
Module window *78, 79*
area markers
removing *91*
Execution Profile window and *87*
printing from *14*
source code and
inspecting *94*
modules
current
statistics *56*
defined *5*
loading *168*
viewing
problems with *172*
source code in *167*
Modules with Source command
Add Areas *81*
Remove Areas *82*
monitors *See* displays
mouse
choosing buttons with *69*
choosing commands with *64*
support *63*
disabling/enabling *134, 146*
Mouse Enabled check box (TFINST) *146*

N

Name input box *74*
Name option *89*
NEC MultiSpeed and NMI *146*
New Directory dialog box *76*
Next command *81, 127*
problems with *172*
Next Pane command *127*
NMI, systems using *146*
NMI Intercept check box (TFINST) *146*
No option
Disassembly (CPU) window *109*
None option
Area Options dialog box *103*
Stack Trace dialog box *84*
normal area markers *See* area markers, normal
Normal option *103*
null modem cable
remote profiling and *152*
numeric coprocessors
profiling tutorial and *9*

O

.OBJ files
Turbo C and *91*
object-oriented programs
expressions, problems with *169*

profiling *52*
OK button *69*
OOP *See* object-oriented programs
Open command *73*
Operation
command *82, 83*
option *103*
optimizers
profiling and *2*
options *145*
customizing *139*
display *142-144*
swapping *142*
input *145*
restoring *118, 126*
restoring defaults *149*
saving *125*
setting *10*
Options command
Areas window *103*
Print menu *120*
Options menu *121, 121-127*
TFINST *145*
Origin command
Disassembly (CPU) window *108*
OS Shell command
TF386 virtual profiler and *162*
output
to DOS
viewing from IDE *128*
user *128*
overhead
calculating *31*
Overlay command *57*
Overlay window *54*
overlays
area size *136*
buffer management
overlay event history and *54*
demonstration *95*
execution timing and resource monitoring
and *48*
history *96*
memory and *95*
monitoring *45*
problems with *173*
profiling
tips and techniques *54*
statistics *95*
demonstration *95*
Overlays option *45*
Statistics window *113*
Overlays window *56, 78, 95*
demonstration *95*

P

–p option (mouse support) *134*
palette *See* graphics
panes
next *127*
parameters *See* arguments
Pascal
functions
terminology *5*
procedures
terminology *5*
profiling *8*
passive analysis *See also* active analysis; profiling, analysis modes
active analysis vs. *51*
caller information and *51*
disk I/O and *51*
execution counts and *51*
interrupts and *51, 52*
program execution time and *51*
setting *50*
Path for Source command *125*
paths
call
logging *36*
sorting *94*
setting *135*
Per Call option *88*
performance analyzer *See* profilers
Permit 43/50 Lines check box (TFINST) *143*
Pick a Caller dialog box *94*
Pick a Module dialog box *85, 89, 106, 119*
Pick command *98*
PLOST.C and PLOSTPAS.PAS *32*
pointers
arrays and *58, 61*
instruction
address of *109*

current
returning to *108*
pop-up menus *See* menus
ports
remote
setting *135*
serial *147*
Previous command
Disassembly (CPU) window *108, 109*
Previous Topic command *129*
PRIME4*.*
listing *24*
PRIMEn*.* (example program) *8*
Print menu *119*
Printer Options radio buttons (TFINST) *146*
printing
high versus standard ASCII *146*
Module window contents *14*
Printing Options dialog box *120*
procedures *See* functions
Pascal
terminology *5*
process ID switching *134*
Profile command
Routines window *106*
Profile mode command *113*
profile report windows *See* windows, profile report
profiling
80386 processors *159-164*
analysis modes *See also* active analysis; passive analysis
active *113*
choosing *48, 50*
compared *114*
current *77*
default *113*
passive *113*
control
TF386 virtual profiler and *162*
defined *1*
end results *47*
type of statistics to collect for *48*
far heap and *38*
large programs *151, 159*
display modes and *143, 144*
object-oriented programs *52*
optimizers and *2*
passes *113*
preparing programs for *41*
program speed and *52*
refining the process *49*
remote *134, 151-158*
commands *152*
DOS version and *155*
hardware requirements *151*
local system *151*
remote system *151*
remote systems
defaults, setting *147*
starting *154*
troubleshooting *155*
User Screen window and *152*
when to do *151*
resetting program *110*
sampler screen *15*
saving profiles *47*
slow programs and *52*
small programs *136*
speeding up *52*
starting *110*
steps *9, 40*
Profiling Options dialog box *50, 113*
program execution
stopping *83*
Program Reset command *110*
program source windows *See* windows, program source
Programming Pearls *7*
programs *144*
compiling
for profiling *42*
current *76*
example *8*
execution speed
profiling and *52*
execution time
passive analysis and *51*
file access
monitoring *45*
keyboard input
profiling and *41*
loading *159, 167*
problems with *136, 172, 175*

symbol tables and *173*
object oriented *See* object-oriented programs
optimizing *57*
preparing for profiling *41*
profiling
starting a run *110*
with no debug symbol information *175*
with out-of-date debug symbol information *175*
reloading
problems with *171*
restructuring
Callers window and *93*
running
from DOS *162*
nonmaskable interrupts and *147*
size
areas and *44*
slow *52*
source
location *85*
viewing *78, 79*
speed
statistics collection and *47*
stopping during a profiling session *43*
structure analysis
statistics for *48*
swapping to disk
problems with *170*
testing and verifying
line-count information and *48*
statistics for *48*
timing
statistics for *48*
unfamiliar
studying *50*
prompts
commands and *165*
dialog boxes *165-169*
responding to *165*
setting *145*
propagation of time *104*

Q

QuarterDeck EMS simulator *162*
question mark (?) search wildcard *80*
quicksort
bubble sort vs. *58*
Quit command *78*
TFINST *150*

R

–r option (remote profiling) *134, 153*
radio buttons *70*
Display Swapping
TFINST *142*
Link Speed (TFINST) *147*
Printer Output (TFINST) *146*
Remote Link Port (TFINST) *147*
Screen Lines
TFINST *143*
User Screen Updating (TFINST) *144*
README file *3*
recursive routines
when to use *60*
Refresh Desktop command *73*
registers
80386 processor, virtual profiling and *162*
Remote Analyzing check box (TFINST) *147*
Remote Link Port radio buttons (TFINST) *147*
remote links
defaults, setting *147*
problems with *175*
remote profiling *See* profiling, remote
remote system
remote profiling and *151*
Remove Areas command *82, 102*
Remove command *57, 91, 98*
current areas *56, 57*
interrupts *57*
Macros menu *123*
undoing *91*
report windows
summary *39*
requirements
hardware *3*
software *42*
resizing windows *18*
resonance *53*
resources
monitoring
statistics for *48*
Restore command *118*

Restore dialog box *126*
Restore Options command *126*
Restore Standard command *73*
return points
  caution *35*
return statements
  area markers and *35*
Routine window *78*
routines *See also* functions
  accessing
    problems with *168*
  active
    call stack and *35*
  available *78*
  calling other routines
    tracking *45*
  calling sequence *78, 112*
  combined clock and *46*
  defined *5*
  jumping to *81*
  marking *See* area markers
  optimizing *57, 60*
  overhead *31*
  recursive
    when to use *60*
  reducing calls to *24*
  timer data *46*
Routines in Module command
  Add Areas *81*
Routines in Modules command
  Remove Areas *82*
Routines window *104*
Run command *110*
Run Count command *113*
Run menu *110*
running
  TF386 virtual profiler *159-164*
  TFINST *139-150*

S

–s option (source code and symbols) *135*
sample programs
  PRIMEn*.* *8*
sampling
  counting vs. *37*
Save command *117*
Save Configuration dialog box *19, 125*
Save Configuration File command (TFINST) *149*
Save dialog box *117*
Save menu (TFINST) *149*
Save Options command *125*
Screen command (TFINST) *142*
Screen Lines command *124*
Screen Lines radio buttons
  TFINST *143*
screens *See also* displays
  background, customizing *142*
  color
    using *133*
  colors, customizing *140-142*
  dual *133*
  EGA/VGA *124*
  LCD *148*
  lines per, setting *143*
  monochrome
    using *133*
  problems with
    snow *143*
  repainting *143*
  swapping *133, 142, 144*
    problems with *162*
  two *77*
  updating *144*
scroll bars *66*
searches
  for text *80*
  in list boxes *129*
  repeat *81*
  wildcards *80*
separate clock
  timer data and *83*
Separate option
  Area Options dialog box *103*
serial links, remote *147*
shortcuts *See* hot keys
Sieve of Eratosthenes *28*
Size/Move command *127*
snow *143*
Sort command *89, 94*
  Areas window *104*
Sort option
  Display Options dialog box *101*

sorts
  bubble vs. quicksort *58*
source code *See* files, source; programs
stack
  call
    size *84*
Stack command *84*
Start Time option *101*
starting Turbo Profiler
  on remote systems
    problems with *147*
startup information *3*
statements
  execution
    verifying *49*
  return *See* return statements
statistics
  accumulation
    disabling *115*
  accuracy *53*
  areas *102*
  automatic collection
    turning on and off *114*
  collecting *56*
  collection *11, 43, 45*
    automatic *77*
    disabling *43, 83*
    enabling *83*
    normal *83*
    program speed and *47*
    type to collect *48*
  collection options *103*
  current
    removing *89*
  current area *56*
  current routine *106*
  default *43*
  displaying *12*
    filtering display *47*
  erasing *91, 117*
  file activity *99*
    graph view *101*
    time in seconds *101*
  files
    writing to, problems with *171*
  filtering *46, 49, 56, 89*
    temporary *90*
  how taken *37*
  interrupts *See* interrupts
  limiting *46*
  overlays *95*
    demonstration *95*
  partial *45*
  printing *16, 119*
  problems with *171*
  program execution speed and *52*
  removing *88*
  restoring *118*
  saving *17, 117*
  sorting *88, 89*
  start and stop points
    maximum *45*
  time
    average *88*
    filters and *89*
    longest *88*
  types of *45*
  viewing *78, 87, 88*
    choices *88*
    number of passes *88*
    time *88*
    with source code *18*
Statistics command *119*
Statistics menu *111*
status line *68*
Stop option
  Areas Options dialog box *103*
  Files window *100*
Stop Recording command *123*
strings
  character
    searching for *167*
structure analysis
  statistics for *48*
Subroutines command *97*
support
  technical *5*
switch statements *See* case and switch statements
symbol names, problems with *169*
symbol tables *175*
  invalid *175*
  loading
    problems with *173*

memory allocation 146
symbols
accessing 168
disassembled 107
problems with 176
System menu *See* ≡ (System) menu

T

Tab Size input box 124
TFINST 144
tabs
setting 124
tabs, setting 144
problems with 176
Tandy 1000A and NMI 146
TD286 protected-mode profiler
instructions 162
TDH386.SYS 160, 162
TDREMOTE
running
problems with 175
TDRF (remote file transfer utility) 152
technical support 5
terminology 5
text
boxes 70
Get Info 76
editors 145
problems with 170
entering in dialog boxes 70
searching for 80
problems with 176
TF386 virtual profiler 159-164
command-line options 161
syntax 162
error messages 163-164
installation
device driver 160
system requirements 159
problems with 162
starting 160
problems with 163
.TFA files
areas and 40
TFCONFIG.TF 19
TFH386.SYS
error messages 164
TFINST 139-150
command-line options vs. 148-149
exiting 150
main menu 140
options, saving 149
TFREMOTE (remote profiling utility) 151
customizing 155
error messages 156-158
installing 152
options *See* command line, options
starting 153
.TFS file
saving to 111
.TFS files
creating 17, 117
This Line command
Add Areas 81
Remove Areas 82
This Module command 84
This Routine command 84
time
propagation 104
time-and-counts profile listing 14
Time option 88, 98
timer
combined clock 46, 83
data
grouping 46
inaccurate results and 37
separate clock 83
setting 113
sound routines and 37
Timer command 83
Timing option
Area Options dialog box 103
title bars 66
TPROF.EXE 149
TSR programs
display swapping and 144
Turbo C
identifiers 91
Turbo language products
Turbo Profiler and 42
Turbo Profiler
leaving 77, 78
Turbo Profiler Statistics file *See* .TFS files

tutorial *7-28*
  numeric coprocessors and *9*

U

underbars *See* underscores
underscores
  identifiers and *91*
  Turbo C identifiers and *91*
Undo Close command *128*
Use Expanded Memory check box (TFINST) *146*
User Screen
  swapping *124*
User screen
  display buffer *143*
  updating *144*
User Screen command *128*
User Screen Updating radio buttons (TFINST) *144*
User Screen window
  remote profiling and *152*
utilities
  TDRF *152*
  TFREMOTE *See* TFREMOTE (remote profiling utility)

V

–v option (video hardware) *136*
version number information *73*
VGA *See* Video Graphics Array Adapter
video adapters *147, See also* Enhanced Graphics Adapter; graphics drivers; Video Graphics Array Adapter
  display pages *144*
  options *136*
Video Graphics Array Adapter (VGA) *See also* graphics; video adapters
  50-line mode
    disabling *136*
  line display *143*
  screen *124*
View Source command *109*

W

–w option (save option settings) *154, 155*
  TF386 virtual profiler *161, 163*
warning beeps, enabling *145*
When Full command *100*
Width of Names input box *124*
wildcards
  DOS *75, 177*
  in searches *80*
Window menu *127*
windows
  active *67*
    defined *65*
    shrinking *127*
    zooming *127*
  Areas *101*
    Callers option and *92*
  Callers *91*
  closing *66, 67, 128*
  customizing *140*
  Disassembly (CPU) *106*
  Execution Profile *86*
    Module window and *87*
  Files *98, 99*
  Help *129*
  Interrupts *96*
  linking *18*
  Module
    Execution Profile and *87*
  moving *67, 127*
  next *127*
  next pane *127*
  open
    list of *129*
  opening *67, 128*
  Overlays *95*
    demonstration *95*
  printing contents of *119*
  problems with *173*
  profile report *55, 56*
  program report *55*
  program source *55*
  report *See* report windows
  resizing *18, 67*
  restoring *73*
  Routine *104*
  saving configuration *19*
  scrolling *66*
  sizing *127*

User screen *128*
zooming *11, 66, 67, 127*
Windows command (TFINST) *140*
WordStar-style cursor-movement commands *146*
Wrap option *100*

X

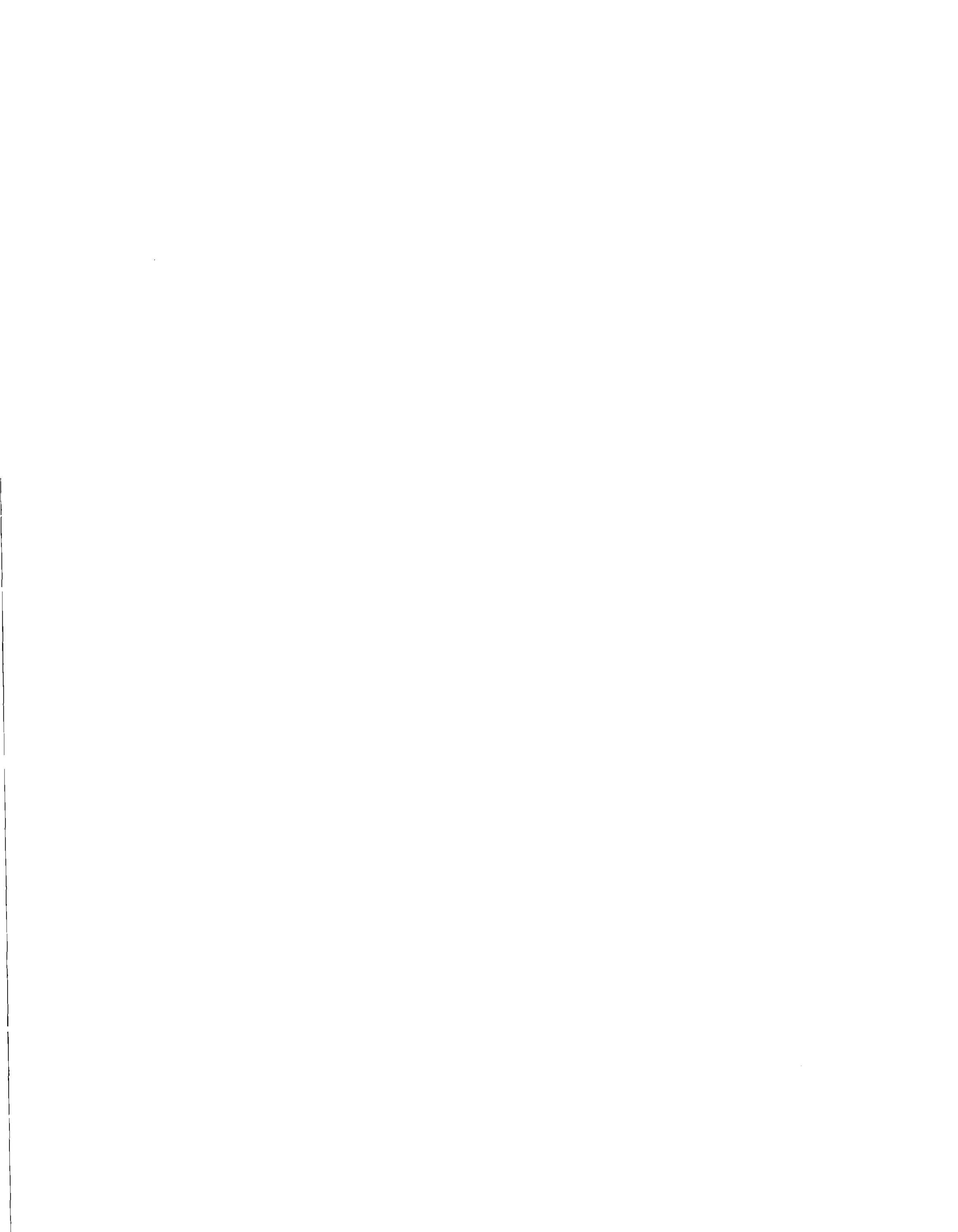
XMS standard *161*

Y

–y option (set overlay area size) *136*
Yes option
Disassembly (CPU) window *109*

Z

Zoom command *11, 127*
zoom icon *66, 67*



1.0

USER'S GUIDE

TURBO PROFILER™

1800 GREEN HILLS ROAD, P.O. BOX 660001, SCOTTS VALLEY, CA 95067-0001, (408) 438-5300 ■ PART # 15MN-PFL01-01 ■ BOR 1483
UNIT 8 PAVILIONS, RUSCOMBE BUSINESS PARK, TWYFORD, BERKSHIRE RG10 9NN, ENGLAND
43 AVENUE DE L'EUROPE—BP 6, 78141 VELIZY VILLACOUBLAY CEDEX FRANCE