User's Guide

VERSION

4.5

Borland®

Turbo Profiler®

Turbo Profiler User's Guide Borland

User's Guide

VERSION 4.5

Borland®
Turbo Profiler®

Borland International, Inc., 100 Borland Way
P.O. Box 660001, Scotts Valley, CA 95067-0001

Printed in the U.S.A.

1E0R1094
9495969798-9 8 7 6 5 4 3 2
H1

Contents

Tables

Figures

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.

Turbo Profiler is a performance analyzer, a software tool that measures your program's performance by finding

- Where your program spends its time
- How many times a line executes
- What lines have been executed
- How many times a routine is called, and by which routines
- What files your program accesses most and for how long

Turbo Profiler also monitors critical computer resources, such as

- 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, 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 by providing the following features:

- Interactive profiling quickly reveals inefficient code in a program.
- Lets you read and edit any text file during profiling sessions.
- Profiles any size program that runs under DOS or Windows.
- Handles programs written using Borland's C++ compilers and Turbo Assembler.
- Provides an easy-to-use interface with multiple overlapping windows, mouse support, and context-sensitive help.

- Reports execution time and execution count for routines and program lines.
- Tracks which blocks of code have and haven't been executed.
- 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. Uses an event list to log the number of bytes read or written.
- Supports complete tracking of overlays.
- Allows remote serial and network profiling.

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.

Turbo Profiler helps you detect the least efficient part of your code and helps point to algorithms that 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 program bottlenecks without a profiler is like trying to find bugs without a debugger; Turbo Profiler reduces both the time and effort it takes to improve your program's performance.

Hardware and software requirements

Turbo Profiler has the same hardware and software requirements as your Borland language product. For a complete list of requirements, refer to the *User's Guide* for the Borland language you're using.

New features for version 4.5

Several new features have been added to Turbo Profiler 4.5: greater capacity, session state saving, and DPMI profiling.

Turbo Profiler for Windows (TPROFW.EXE) can profile larger programs and provide statistics for more areas than previous versions. Turbo Profiler also provides *session state saving*; the profiler saves your settings so you can easily resume a profiling session. In addition, TPROF.EXE (the DOS profiler) now uses the DOS protected mode interface (DPMI), which lets you profile larger DOS programs.

What's in this manual

Chapter 1, "A sample profiling session," is a tutorial that takes you through a simple profiling session. The tutorial 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, "The Turbo Profiler environment," explains in detail each menu item and dialog box option in the Turbo Profiler environment.

Chapter 3, "Profiling strategies," provides general guidelines and tips for conducting a fruitful profiling session.

Chapter 4, "Inside the profiler," uses analogy to explain how Turbo Profiler gathers execution-time and execution-count data while your program runs.

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 and TPROFW.

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, "Turbo Profiler for Windows," describes how to run Turbo Profiler for Windows and how to use its special features.

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

Typeface conventions

This manual uses the following special fonts:

`Monospace`	This type represents text that you type or text as it appears onscreen.
Italics	These are used to emphasize and introduce words, and to indicate variable names (identifiers), function names, class names, and structure names.
Bold	This type indicates reserved keywords words, format specifiers, and command-line options.
Keycap	This type represents a particular key you should press on your keyboard. For example, "Press *Del* to erase the character."
Key1+Key2	This indicates a command that requires you to press *Key1* with *Key2*. For example, *Shift+a* (although not a command) indicates the uppercase letter "A."

ALL CAPS	This type represents disk directories, file names, and application names. (However, header file names are presented in lowercase to be consistent with how these files are usually written in source code.)
Menu | Choice	This represents menu commands. Rather than use the phrase "choose the Save command from the File menu," Borland manuals use the convention "choose File | Save."

Software Registration and Technical Support

The Borland Assist program offers a range of technical support plans to fit the different needs of individuals, consultants, large corporations, and developers. To receive help with this product, send in the registration card and select the Borland Assist plan which best suits your needs. North American customers can register by phone 24 hours a day by calling 1-800-845-0147. For additional details on these and other Borland services, see the *Borland Assist Support and Services Guide* included with this product.

Chapter

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 John Bentley's "Programming Pearls" column (July 1987) in *Communications of the ACM.*

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.

All the tutorial examples were run on a 486 machine with an SVGA video adapter.

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 an integer and 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 (actually, 2 is the only even prime number). 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, and PRIME5) is a variation on its predecessor.

About the sample programs

The Turbo Profiler package comes complete with the sample programs used in this chapter. Both the source code and the executable code are provided; Turbo Profiler requires both to analyze a program. Each of the sample programs was compiled with full symbolic information, since the profiler also requires this information.

To ensure that your own programs contain full symbolic information, you must compile them with the appropriate compiler options turned on, as shown in the following list:

- **Borland's line of C++ compilers:** If you are compiling in the IDE,
  1. Choose Options | Project.
  2. Choose the Compiler | Debugging topic, then check Include Debug Information.
  3. Choose the Linker | General topic, then check Include Debug Information.

  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.

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 "primeness" 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*.

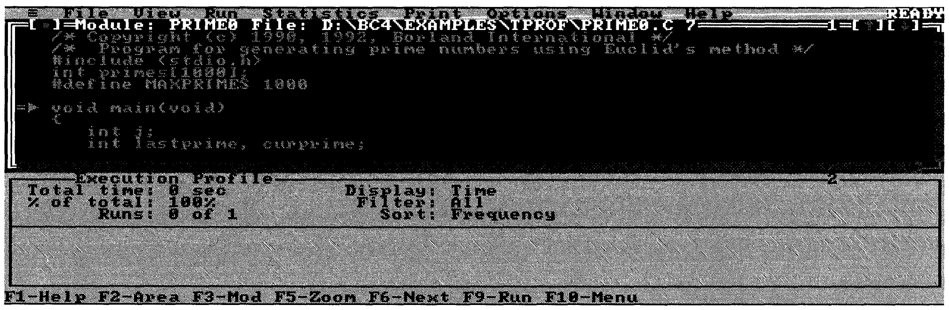
Load PRIME0 into Turbo Profiler by typing

```
TPROF PRIME0
```

and pressing ***Enter***.

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 PRIME0).

Figure 1.1 Turbo Profiler with PRIME0 loaded



```
 ≡  File  View  Run  Statistics  Print  Options  Window  Help                READY
╔═[■]═Module: PRIME0 File: D:\BC4\EXAMPLES\TPROF\PRIME0.C 7══════════1═[↑][↓]═╗
    /* Copyright (c) 1990, 1992, Borland International */
    /*  Program for generating prime numbers using Euclid's method */
    #include <stdio.h>
    int primes[1000];
    #define MAXPRIMES 1000

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

  Execution Profile                                                        2
 Total time: 0 sec            Display: Time
 % of total: 100%              Filter: All
       Runs: 0 of 1              Sort: Frequency

F1-Help F2-Area F3-Mod F5-Zoom F6-Next F9-Run F10-Menu
```

For a more detailed description of the profiler's environment, see Chapter 2.

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 might want to 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.

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, every line in the program must be marked as an area.

By default, Turbo Profiler marks every line in a small program. To verify that this is true, you can check the Module window to see that all executable lines are tagged with a marker symbol (=►).

1 Press *Alt+F10* to open the Module window SpeedMenu.

2 Choose Add Areas from the SpeedMenu. This menu lists area boundaries for you to choose from.

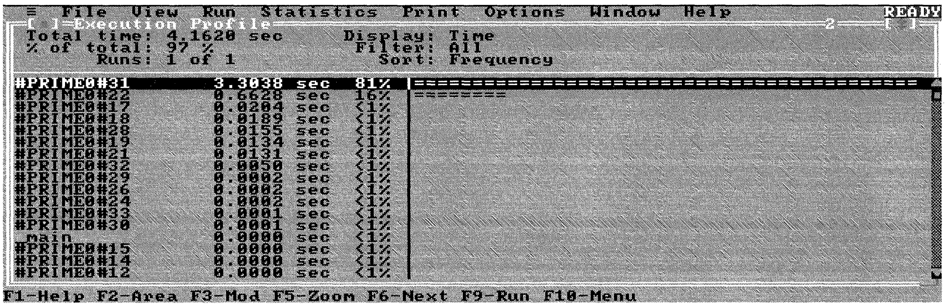
3 Choose Every Line in Module. This sets area markers for all lines in the module, then returns the cursor to the Module window.

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 your 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 similar to this:

Figure 1.2 Program statistics, PRIME0


≡ File View Run Statistics Print Options Window Help READY
F1-Help F2-Area F3-Mod F5-Zoom F6-Next F9-Run F10-Menu

```
[ ]=Execution Profile=================================2===[ ]
 Total time: 4.1620 sec        Display: Time
 % of total: 97 %               Filter: All
       Runs: 1 of 1               Sort: Frequency

#PRIME0#31          3.3038 sec   81% |=========================================
#PRIME0#22          0.6628 sec   16% |========
#PRIME0#17          0.0204 sec   <1% |
#PRIME0#18          0.0189 sec   <1% |
#PRIME0#28          0.0155 sec   <1% |
#PRIME0#19          0.0134 sec   <1% |
#PRIME0#21          0.0131 sec   <1% |
#PRIME0#32          0.0050 sec   <1% |
#PRIME0#29          0.0002 sec   <1% |
#PRIME0#26          0.0002 sec   <1% |
#PRIME0#24          0.0002 sec   <1% |
#PRIME0#33          0.0001 sec   <1% |
#PRIME0#30          0.0001 sec   <1% |
 main               0.0000 sec   <1% |
#PRIME0#15          0.0000 sec   <1% |
#PRIME0#14          0.0000 sec   <1% |
#PRIME0#12          0.0000 sec   <1% |
```

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 fields for 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* displaying a proportional graph of the execution time spent in that area

The line

```
#PRIME0#31   3.3038 sec  81%  |====================================
```

tells you that the thirty-first line of code in module PRIME0 executed for about 3.3 seconds—which was 81% of the total execution time for all marked areas. The magnitude bar automatically shows line 31's time full-scale because line 31 is the most time-consuming of the marked areas.

Actual time and percentage statistics will vary from system to system.

Displaying statistics

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

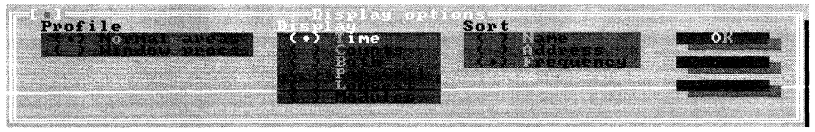
```
Display...
Filter        All ►
-------------------
Module
Remove
```

1 Press *Alt+F10* to bring up the SpeedMenu for the Execution Profile window.

2 Choose Display on the SpeedMenu.

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

Figure 1.3 The Display Options dialog box



```
[■]═══════════════════════Display options═══════════════════════
 Profile                  Display               Sort
 (•) Normal areas         (•) Time              ( ) Name            OK
 ( ) Window procs         ( ) Counts            ( ) Address
                          ( ) Both              (•) Frequency
                          ( ) Per call
                          ( ) Longest
                          ( ) Modules
```

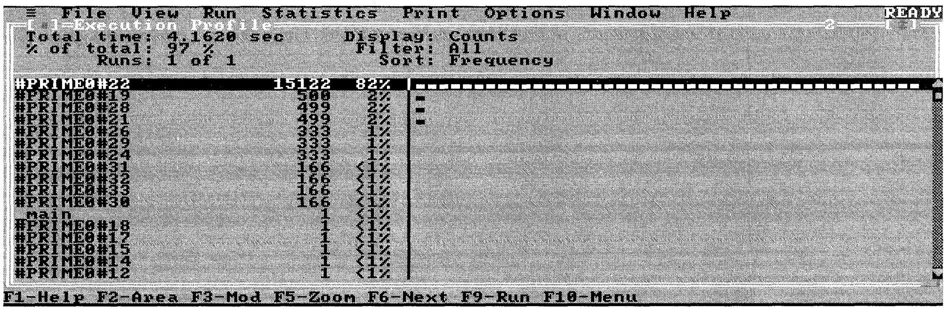
- Time (the default setting) shows the total time spent in each marked area.
- Counts displays the number of times program control entered each area.
- Both shows time and counts data on the same screen.
- Per Call displays the average amount of time per call.
- Longest shows the longest time spent in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

4 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.)

5 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 the Execution Profile window



```
 ≡  File  View  Run  Statistics  Print  Options  Window  Help              READY
╔═[■]═Execution Profile═══════════════════════════════════════════════2═══[↕]═╗
║ Total time: 4.1620 sec      Display: Counts
║ % of total: 97 %             Filter: All
║       Runs: 1 of 1             Sort: Frequency
║
#PRIME0#22                  15122  82% |■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
#PRIME0#19                    500   2% |■
#PRIME0#28                    499   2% |■
#PRIME0#21                    499   2% |■
#PRIME0#26                    333   1% |
#PRIME0#29                    333   1% |
#PRIME0#24                    333   1% |
#PRIME0#31                    166  <1% |
#PRIME0#32                    166  <1% |
#PRIME0#33                    166  <1% |
#PRIME0#30                    166  <1% |
  main                          1  <1% |
#PRIME0#18                      1  <1% |
#PRIME0#17                      1  <1% |
#PRIME0#15                      1  <1% |
#PRIME0#14                      1  <1% |
#PRIME0#12                      1  <1% |
F1-Help F2-Area F3-Mod F5-Zoom F6-Next F9-Run F10-Menu
```

This display of PRIME0's statistics shows that line 22 is the most frequently called line in PRIME0.

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 *Down* to get to it, then press *Enter*, or press *B*, the hot key for this option.)

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.

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 the time-and-counts statistics to a file, you must first 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 ***Down*** to turn the setting to File).

3 Tab to the Destination File input box and type
```
PRIME0SC.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 Print | Module. In the Pick a Module dialog box, press ***Down*** to highlight the module name PRIME0, then press ***Enter*** (or choose OK).

To inspect the file PRIME0SC.LST, choose View | Text File, and at the File Name prompt, type

```
PRIME0SC.LST
```

This is what you see if you're profiling the C program PRIME0.C. The times in your file will probably vary from the ones shown here because of the differences in computer systems.

```
Turbo Profiler  Version 4.5  Tue Aug 20 15:16:47 1994

Program: D:TPROFPRIME0.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", last prime, 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.

When you have finished examining the listing, press ***Alt+F3***, or click the close box, to close the File window.

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 extended ASCII characters in the printed report. (If your printer does not support extended ASCII characters such as ╓ 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 Turbo Profiler.

Choose Statistics | Save 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.

To restore the statistics you saved for PRIME0, open PRIME0 in Turbo Profiler and choose Statistics | Restore. The File Name input box lists *.TFS as the default. Press ***Enter*** to go to the Files list box, then highlight PRIME0.TFS and choose OK to recover the data from this file.

The File Name input box lists *.TFS as the default. Press Enter to go to the Files list box, then highlight PRIME0.TFS and choose OK to recover the data from this file.

Analyzing the statistics

In this section you will 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 Zoom from the Window menu or press *F5*) and look at the statistics for lines 22 and 31 (the **if** and **printf** statements).

We cover modifications to the **printf** statement in program PRIME5.

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's 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:

   1 Resize the window to full-screen height and half-screen width.

   2 Move the resized window to the right.

   When you've done steps 1 and 2, 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.

2 Open the SpeedMenu (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.

   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 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 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 10.

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 Turbo Profiler.

1 Choose Options | Save 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 SpeedMenu (press *Alt+F10*), choose Display.

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. 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 20 milliseconds per call.

Note 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", last prime, curprime);
```

to simply

```
printf("%d\n", 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's 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)

You're finished with PRIME0 now, so load PRIME1 (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 *Up* and *Down* keys to highlight PRIME1.EXE.
4 Press *Enter*. Turbo Profiler loads PRIME1 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.

Run PRIME1 in Turbo Profiler (press *F9*) and look at the statistics. Then choose Display from the Execution Profile window SpeedMenu to open the Display Options dialog box and turn on the Both radio button. Press *Enter*, then zoom the Execution Profile window (*F5*).

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).

Notice in particular that the test for prime numbers (line 9 in PRIME0) now executes 78,022 times instead of 15,122. This may be surprising 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 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 sample programs.

Modifying the program and reprofiling

Earlier, we pointed 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.

You need to set areas for all lines in the module, so bring up the SpeedMenu in the Module window, choose Add Areas | Every Line, then press *Enter*.

Press ***F9*** to start profiling. 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 Display from the Execution Profile SpeedMenu) and set Display to *Both*. Choose 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 is the expense of the new root routine. Line 7 inside the routine executes 5,456 times, consuming the most time of any routine.

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.

When a 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 magnitude 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.

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*.

In the Module window SpeedMenu, 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 25% 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.

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

if (n % 2 == 0)
   return (n==2);

if (n % 3 == 0)
   return (n==3);

if (n % 5 == 0)
   return (n==5);

for (i=7; i*i <= n; i+=2)
   if (n % i == 0)
      return 0;

return 1;
```

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 nearly half a second off the execution time.

Eliminating CR/LF pairs (PRIME5)

Here's one last change. Instead of printing a carriage return/linefeed pair after each prime number, try printing just a space. This is the only change made in program PRIME5.

Load PRIME5, 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 newline characters is expensive. The distribution of profiles is fairly even for execution times and counts. We'd be hard-pressed to squeeze more out of this program without substantially changing the algorithm.

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 Turbo 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 2.

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 newline characters with calls to position the cursor.

Note There are a number of articles on the subject of profiling, but not many books. John 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.

Chapter

2

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 or later
- 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 both global and SpeedMenus. *Global menus* are ones you access via the menu bar, and *SpeedMenus* 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 `WAIT`ing 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's 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 Once the global menu is active, use the arrow keys to select the menu you want to display. Then press *Enter*.

Note 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, the command displays a dialog box when you choose it. 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.

SpeedMenus

In addition to the global menus that you access through the menu bar, each of Turbo Profiler's windows has its own unique SpeedMenu. When you're in a window, press *Alt+F10* to bring up the SpeedMenu. For more information on accessing SpeedMenus, refer to the discussion on page 28.

Choosing menu commands with the mouse

To use the mouse to choose commands from global menus, 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.)

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.

Table 2.1 Menu hot keys

Press this shortcut...	To accomplish this...
Ctrl and the highlighted letter of the SpeedMenu command	Carry out the SpeedMenu 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. Any command you choose or text you type applies only to the active window.

Note The active window is the one that you're currently working in.

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 the one on top of all the others (the frontmost one).

There are several types of windows. Most of them have these things: 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).

Window management

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

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 2.2 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, if active, click the window's close box.

Table 2.2 Manipulating windows (continued)

To accomplish this...	Use one of these methods...
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 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 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 left border character that isn't a scroll bar, close box, or zoom or iconize box, or
	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, or
	Choose Size/Move from the Window menu (or press *Ctrl+F5*), press *Shift+Arrow* to change the size of the window, then press *Enter*, or
	Drag any part of the right or bottom border that isn't a scroll bar to resize the window.
Zoom the active window	Click the zoom box, or
	Double-click the window's title bar, or
	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 unzoom box.([↕]) When it is iconized, it has only a zoom box ([↑]). In its restored state, it has both an up and a down arrow.
Move from pane to pane	Press *Tab*, *Shift+Tab*, or *Shift+Arrow*, or
	Choose Window I 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 onscreen 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 status line changes as you switch windows or activities. You can click any of the shortcuts to carry out the command.

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.

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 six basic types of controls: radio buttons, check boxes, action buttons, text boxes, list boxes, and standard buttons.

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

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.

≡ menu (System)

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

With the commands in the ≡ menu, you can

- Repaint the screen
- Restore your original window configuration
- Activate the Turbo Profiler information box

Repaint Desktop

Choose Repaint 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, window 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 Restore Standard, Turbo Profiler puts all the windows back the way they were when you first started the profiler.

About

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

File menu

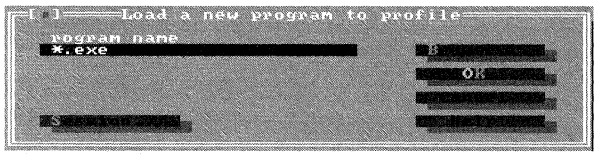
The File menu contains commands for

- Opening and loading a program to be profiled
- Changing the current directory
- Obtaining information about your program and system memory allocation
- Opening up a DOS shell
- Quitting the profiler

Open

The File | Open command, used to load an explicit file into the Module window, opens a two-tiered set of dialog boxes. The first being the Load aNew Program to Profile dialog box.

Figure 2.1 **The Load A New Program to Profile dialog box**

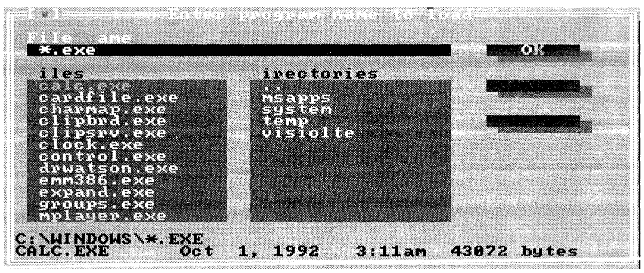


TRPOF.EXE's Load a New Program to Debug dialog box contains an additional button, Session, to support its remote profiling feature. For more information on remote profiling, and the Session button, see Appendix C.

If you know the name of the program you want to load, enter the executable name into the Program Name input box and press ***Enter***.

To search through directories for your program, click the Browse button to open the second dialog box (the Enter Program Name to Load dialog box):

Figure 2.2 The Enter Program Name to Load dialog box



```
[■]═══════Enter program name to load
File  ame
 *.exe                                              OK
 iles                     irectories
 calc.exe                 ..
 cardfile.exe             msapps
 charmap.exe              system
 clipbrd.exe              temp
 clipsrv.exe              visiolte
 clock.exe
 control.exe
 drwatson.exe
 emm386.exe
 expand.exe
 groups.exe
 mplayer.exe
C:\WINDOWS\*.EXE
CALC.EXE        Oct  1, 1992    3:11am   43072 bytes
```

The Files list box displays the files in the currently selected directory. By entering a file mask into the File Name input box (such as **.EXE*), you can specify which files should be listed. You can also use the File Name input box to change disk drives.

To "walk" through disk directories, double-click the entries listed in the Directories list box (the .. entry steps you back one directory level). Once you've selected a directory, choose a file to load from the Files list box. To quickly search for a file, type a file name into the Files list box. Turbo Profiler's incremental matching feature moves the highlight bar to the file that begins with the letters you type. Once you've selected a file, press OK. This action returns you to the Load a New Program to Profile dialog box.

To support remote debugging, TPROF.EXE contains a buttons in the Load a New Program to Profile dialog box. The Session radio buttons specify whether or not the program you're debugging is on a local or remote system. If it's located on a remote system, select the Remote Windows radio button; if it's not on a remote system, select Local. See Appendix B for complete instructions on remote debugging.

Note Before loading a program into the profiler, be sure to compile your source code into an executable file (.EXE or .DLL) with full debugging information. Although you can load programs that don't have debug information, you will not be able to use the Module window to view the program's source code. (The profiler cannot reference the source code of executable modules that lack debug information. If you load a module that doesn't contain debug information, Turbo Profiler opens the Disassembly window to show the disassembled machine instructions of that module.)

When you run a program under the control of Turbo Profiler, the program's executable files (including all .DLL files) and original source files must be available. In addition, all .EXE and .DLL files for the application must be located in the same directory.

Session Saving

When you exit Turbo Profiler, it saves to the current directory a *session-state file* that contains information about the profiling session you're leaving. When you reload your program from that directory, Turbo Profiler restores the history lists from the last profiling session.

By default, all history lists are saved to the session-state file. Session-state files are named *XXXX*.TP and *XXXX*.TPW by TPROF.EXE and TPROFW.EXE, respectivley, where *XXXX* is the name of the program you're profiling. If no program is loaded when you exit Turbo Profiler, then *XXXX* is named either TPROF or TPROFW.

The Options | Set Restart Options command opens the Restart Options dialog box, from where you can set how Turbo Profiler handles the session-state files. In this dialog box, the Restore at Restart check box specifies whether you want to save the profiler's history lists. The Use Restart Info radio buttons specify how you want to handle the file:

Table 2.3 Turbo Profile session-state saving options

Option	Description
Always	Always use the session-state file.
Ignor if old	Don't use the session-state file if you've recompiled your program.
Prompt if old	Prompts if you want to use the session-state file if you've recompiled your program.
Never	Do not use the session-state file.

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.

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 describes how Turbo Profiler gained control: it can be any one of the following messages:

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

- Mode is the profiling mode (active, passive, or coverage); 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 shows the use of memory:
  - 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
- DOS version shows the current DOS version on your system.
- Current date and time is taken from the system clock.

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

Note 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 screen in the system, Window | User Screen and *Alt+F5* are disabled during remote profiling. (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.

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 the Statistics | Save and Options | Save commands before exiting. If you don't, you'll lose the options you've set.

Note Each time you exit Turbo Profiler, it remembers the areas you set up for the current program by saving the settings in a .TFA file. Then, the next time the program is loaded, the area settings are automatically put into effect.

View menu

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

Table 2.4 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 Borland's line of Pascal compilers, Borland's C and C++ compilers, 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	The current profile area in the Module window, as disassembled source code
Text File	Contents of any text file you specify
Coverage	In its default setting, lists the code blocks which haven't yet been executed

SpeedMenus

Each Profiler window has its own *SpeedMenu* (actually, some windows have more than one SpeedMenu, depending on the number of panes in the window). A SpeedMenu contains commands and settings specific to the window pane.

To activate a SpeedMenu, press *Alt+F10* (if there is more than one window pane, press *Tab* to alternate between the panes). When the SpeedMenu pops up, use the arrow keys to select the command you want and press *Enter*, or press the highlighted letter. Once you choose a SpeedMenu command, Turbo Profiler either carries it out, displays a dialog box, or displays another menu.

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

To pop up the active menu's SpeedMenu using a mouse, click the mouse's right button. Then, select the command you want by clicking on the menu item.

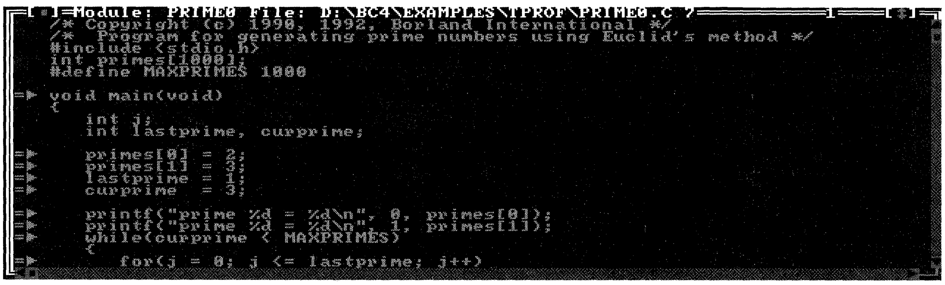
Module

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 other windows.

When you choose View | Module, a list box appears that lists all the source modules linked with the program currently loaded into the Module window. Highlight the new module you want to display, and press OK to load it into the Module window.

If the `Modified` appears in the title bar of the Module window, it indicates that the source code to the file you're viewing has changes since the program was last compiled.

Figure 2.3 The Module window (zoomed)



```
[■]=Module: PRIME0 File: D:\BC4\EXAMPLES\TPROF\PRIME0.C 7=====1=====[↕]
   /* Copyright (c) 1990, 1992, Borland International */
   /*  Program for generating prime numbers using Euclid's method */
   #include <stdio.h>
   int primes[1000];
   #define MAXPRIMES 1000

=► void main(void)
   {
      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, both the .EXE file and the original source file must be 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. 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.

2 In the directories (if any) you've listed under Options | Path for Source (or stated in the command-line option using the **–sd** switch).

3 In the current directory.

4 In the directory that contains the .EXE file of the program you're profiling.

Press *Alt+F10* or click the right mouse button to bring up the Module window's SpeedMenu. With the SpeedMenu 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 specified when you run TFINST.

Line

Ctrl L

To move swiftly to a particular line of code in the Module window, choose Line from the SpeedMenu. The dialog box that pops up requests the line number you seek; type in the new line number, 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.

Search

Ctrl S

To search for a character string in the current module, choose Search. 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 the * matches 0 or more characters.

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

Next

Ctrl N

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

Goto

Ctrl G To position the Module window's cursor on a particular routine or other code label in your program's source code, choose Goto. 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*).

Note Use the hex format of your program language.

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

Add Areas

Ctrl A Add Areas on the Module window's SpeedMenu leads to another menu.

- All Routines 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).
- Modules with Source adds area markers for all routines in modules whose source code is available.
- Routines in Module adds area markers for all routines in the current module (the one in the Module window).
- Every Line in Module adds area markers for all lines in the current module.
- Lines in Routine adds area markers for all lines in the current routine (whichever routine the cursor is on in the Module window).
- Current 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.

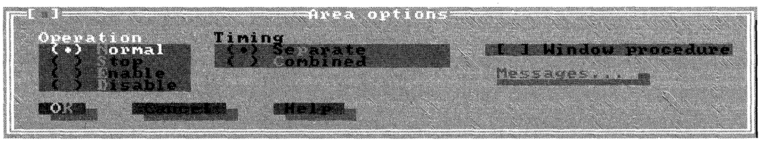
Remove Areas

Ctrl R Choosing Remove Areas on the Module window's SpeedMenu displays another menu. This menu is almost identical to the Add Areas menu just described. Except for the All Areas command, which removes all markers, each command on the Remove Areas menu erases area markers the same way as the respective Add Areas command adds area markers.

Operation

Ctrl O The Operation command opens the Area Options dialog box, which contains settings for the area marker on the current line in the Module window.

Figure 2.4 The Area Options dialog box



Operation: (•) Normal ( ) Stop ( ) Enable ( ) Disable

Timing: (•) Separate ( ) Combined

[ ] Window procedure

You can specify two options with the radio buttons in this dialog box: Operation and Timing. In addition, the Window Procedure check box enables message tracking for Windows programs.

- *Operation* specifies what profiling action will occur for the current area. *Window Procedure*, when checked, specifies that the current area marks a procedure specific to Windows.

  When you mark an area, a marker symbol signifying the chosen operation appears to the left of that area in the Module window. A different symbol is used to refer to each type of area marker: A Normal area is marked with **=➤**, a Stop area is marked with **s➤**, an Enable area is marked with **e➤**, and a Disable area is marked with **d➤**.

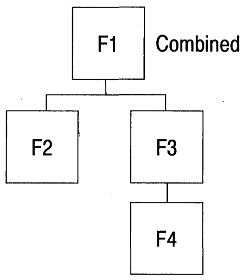
  - *Normal* collects profile statistics for this area as specified in the Statistics menu (callers, file activity, interrupts, overlays, and so on) and Area Options dialog box, which you reach through the SpeedMenus of the Module and Areas windows.
  - *Stop* stops program execution at this marker.
  - *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. Data collection resumes once program control passes an Enable marker.

Note Coverage mode provides only one type of area marker, denoted by a single ➤ character. The marker indicates that the block has not been executed.

- *Messages* becomes active when the Window Procedure check box is checked. Choosing Messages displays the Window Procedure Messages dialog box.
- *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 in the current routine to that routine's statistics.
  - *Combined* sums the timer ticks of the marked routine with the timer ticks of *all* the children of that routine. You can specify combined time for an area only if that area's Callers setting is *Immediate* or *All*.

    When you set a routine's Timing to Combined, Turbo Profiler does not display timing statistics for the children of that routine—their times are reported as part of the routine whose timing is set to Combined.

The following illustration shows how Combined timing works.



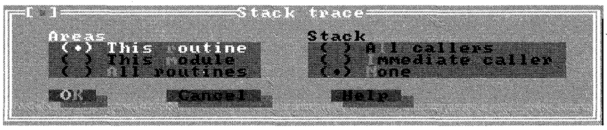
In this illustration, the routine *F1* has it's Timing set to Combined. Turbo Profiler collects the timing information for all the routines (*F1*, *F2*, *F3*, and *F4*), and sums them into the timing information collected for the routine *F1*.

Callers

Ctrl C

The Callers command on the SpeedMenu leads to the Stack Trace dialog box.

Figure 2.5 The Stack Trace dialog box



```
[■]═══════════Stack trace═══════════
  Areas                  Stack
   (•) This Routine       ( ) All callers
   ( ) This Module        ( ) Immediate caller
   ( ) All routines       (•) None

  OK        Cancel        Help
```

You specify how callers are handled with two sets of radio buttons, Areas and Stack.

- Areas specifies which areas you want call paths recorded for.
  - This Routine sets only the current routine (the one the cursor is on in the Module window) to the setting specified in Stack.
  - This Module sets all routines in the current module to the setting specified in Stack.
  - All Routines sets all routines in all program modules to the option specified in Stack.
- 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 Caller 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.

When OK is selected from the Stack Trace dialog box, the areas specified by the Areas options are set according to the selected Stack option. Changes made in this dialog box are reflected in the Areas window.

Module

Ctrl M

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

Most modules have only 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 the file names for the source code modules that 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 Module 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 Options | Path for Source (or stated in the command-line option using the **–sd** switch).

3 In the current directory.

4 In the directory that contains the .EXE file of the program you're profiling.

File

Ctrl F

The File command on the Module SpeedMenu 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.

The File command searches for the source code in the same order as the Module command in the previous section.

Edit command

Ctrl E

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 using TFINST, whenever you choose Edit from the Module window's SpeedMenu, 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).

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 for the areas you've chosen.
- Runs: the current profile run (if you're collecting and averaging statistics from more than one run).
- The options you've chosen from the SpeedMenu (display format, filter status, and sort order).
- Total Ticks: the total number of timer ticks which occurred during the program run. Timer Ticks displays only during passive mode profiling.

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 SpeedMenu (press *Alt+F10*). Through this SpeedMenu, you can

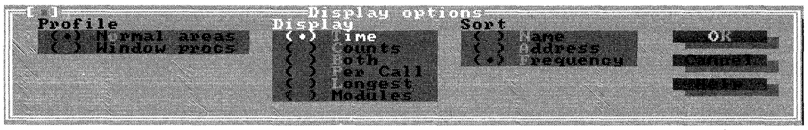
- Select what type of modules will be profiled: Window procedures or normal areas.
- Choose any one of six different ways to display profile statistics in the Execution Profile window.
- Sort the displayed statistics.
- Temporarily remove one or more areas' statistics from the display.
- Examine the source code for an area.
- Delete an area's statistics from memory and erase the associated area marker.

Display

Ctrl D

When you choose Display from the Execution Profile window's SpeedMenu, the Display Options dialog box comes up.

Figure 2.6 The Display Options dialog box



```
[■]═══════════════════Display options═══════════════════
 Profile                Display            Sort
  (•) Normal areas       (•) Time           ( ) Name          OK
  ( ) Window procs       ( ) Counts         ( ) Address
                         ( ) Both           (•) Frequency     Cancel
                         ( ) Per Call
                         ( ) Longest                          Help
                         ( ) Modules
```

You can specify three options with the radio buttons in this dialog box: Profile, Display, and Sort.

- Profile specifies which areas are displayed in the Execution Profile window.
  - Normal Areas displays all marked areas, including any Window procedures that are marked.
  - Window Procs filters the displayed areas to show only the classes and Windows messages that are specified in the Windows Procedure Messages dialog box.

    See Appendix D for a complete description of the Windows Procedure Messages dialog box.
- 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.
- Module displays, for each module in the program, the time program control was in that module. This setting is useful only if Turbo Profiler is in passive mode, which means it's recording with every clock tick which module the program is in. This option is helpful as a first cut at profiling a large program.

- 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

Ctrl F The Filter command on the SpeedMenu leads to the three-item menu shown here.

- All 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 Filter | Module or Filter | Current), choose Filter | All to restore all profile statistics to the window.

- Module 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 ***Up*** and ***Down*** 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.

- Current temporarily removes the highlighted area's statistics from the Execution Profile window.

  Choose Filter | Current if you want to throw out one area's statistics and see what happens to the remaining percentages. The Current 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.

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.

Note Don't confuse Filter | Current with the Remove command on the Execution Profile window's SpeedMenu.

Module

Ctrl M The Module command on the SpeedMenu takes you to the line of source code in the Module window for which the statistics are highlighted in the Execution Profile Window. Note that in addition to the *Ctrl+M* hot key, you can press *Enter* from the Execution Profile window to execute the same command.

Suppose you highlight the statistics for routine **frank** in the Execution Profile window, then choose Module from the SpeedMenu to activate the link. Turbo Profiler activates the Module window and places the cursor on the first line of **frank** 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 unidirectional: 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.)

If, when you choose the Module command, 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 47 for details about the Disassembly window.)

Position

Ctrl P The Position command is identical to the Module command, with the exception that the Position command does not activate the Module window; the Execution Profile window remains active. Another shortcut for this command is the *Spacebar*.

Remove

Ctrl R The Remove command removes area marker settings from the currently highlighted line in the Execution Profile window.

Warning The Remove command *erases* statistical data. Use it with discretion.

Once you remove the line's area markers with the Remove 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 Remove action, you must

1 Activate the Module window and bring up its SpeedMenu.

2 Place the cursor on the line whose marker you removed.

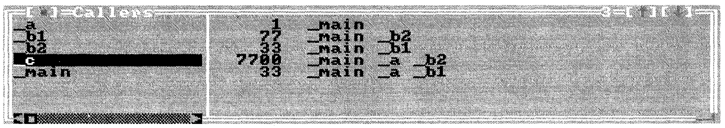
3 Choose Add Areas | This Line.

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. A call path is a list of all the routines that were called to execute the currently selected routine. The call path starts with the original calling routine. You must set the Statistics | Callers menu item to *Enabled* before the profiler will record any call-path information.

Figure 2.7 The Callers window, showing calls in CALLTEST



```
_a              1  _main
_b1            77  _main _b2
_b2            33  _main _b1
_c           7700  _main _a  _b2
_main          33  _main _a  _b1
```

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.

Note An underscore precedes the identifier names in this Callers window because Borland's C and C++ compilers add the underscore to all symbol names appearing in .OBJ files and symbolic debugging information.

Although the Callers window displays the call-path information, you must specify what type of call path recording you want. This is done 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 SpeedMenu.

2 Choose Callers 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 44 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 SpeedMenu.

2 Choose Options to see the Area Options dialog box.

3. 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.
4. Press *Enter* or choose OK to go back to the Areas window.

Figure 2.7 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**
- 77 calls from **main** to **b2** to **c**
- 33 calls from **main** to **b1** to **c**
- 7,700 calls from **main** to **a** to **b2** to **c**
- 33 calls from **main** to **a** to **b1** 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 SpeedMenus. In the Callers window's right pane, the Inspect SpeedMenu item brings up a subsequent menu containing the commands areas, module, and profile.

Inspect (left pane)

Ctrl I

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.

Inspect (right pane)

Ctrl I

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 (Areas, Module, or Profile) from the list. This brings up the Pick a Caller dialog box, which lists all callers on the current call path.
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 isn't already open, the profiler opens it automatically, then goes to the caller's location in that window.

Sort (right pane)

Ctrl S

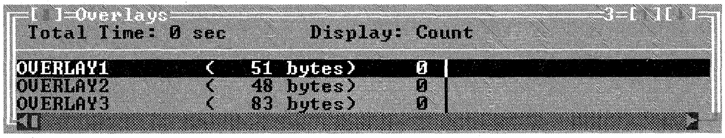
With the local Sort command in the Callers window's right pane, you can sort the list of call paths in two ways:

- Called sorts the call paths in the same order that program control traversed them at run time.
- Frequency 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 Borland's Pascal compilers, Borland's C and C++ compilers, and Turbo Assembler programs. You must set the Statistics | Overlays menu item to Enabled before the profiler will record any overlay information (if your program has overlays, Turbo Profiler automatically enables the Overlay option).

Figure 2.8 The Overlays window


```
Total Time: 0 sec        Display: Count

OVERLAY1          (  51 bytes)    0 |
OVERLAY2          (  48 bytes)    0 |
OVERLAY3          (  83 bytes)    0 |
```

The information listed in this window can include:

- How many times your program loads each overlay into memory.
- When each overlay was loaded.
- The sequence in which your program loads the overlays.
- The size of the overlay

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.

Note Press any key to halt the execution of the program OVRDEMO.

If you have one of Borland's Pascal compilers, there's a program, OVRDEMO, that gives a live demonstration of how the Overlays window works. Load this program into the profiler. Then set area markers for every line in the module OVRDEMO, enable Statistics | Overlays, and run the program. (You'll need the files OVRDEMO.PAS, OVRDEMO1.PAS, OVRDEMO2.PAS, and OVRDEMO.EXE to profile this program.)

The Overlays window's SpeedMenu provides two commands, shown here.

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.

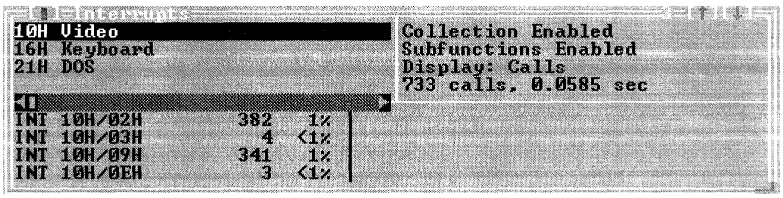
Inspect

Ctrl I 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 2.9 The Interrupts window


```
10H Video                          Collection Enabled
16H Keyboard                       Subfunctions Enabled
21H DOS                            Display: Calls
                                   733 calls, 0.0585 sec

INT 10H/02H          382   1%
INT 10H/03H            4  <1%
INT 10H/09H          341   1%
INT 10H/0EH            3  <1%
```

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

- The top left pane displays the list of specific interrupts to be profiled (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.

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)

Both active panes of the Interrupts window have SpeedMenus.

Collection (top pane)

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

Subfunctions (top pane)

Ctrl S The Subfunctions command enables or disables collection of statistics for subfunctions of the current interrupt (this is particularly useful for DOS INT 21H calls). Subfunction numbers are determined from the value in the AH register when the interrupt is called.

Add (top pane)

Ctrl A The Add command adds an interrupt, by number, to the list in the pane's left display area. Type the interrupt number in hexadecimal notation. For example, type 21 for INT 21H (if you type 33, turbo Profiler adds INT 33H to the list).

Pick (top pane)

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

Remove (top pane)

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

Delete All (top pane)

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

Display (bottom pane)

Ctrl D The Interrupt window's bottom pane has a one-item SpeedMenu; its command, Display, leads to a subsequent menu. From this second menu, you can choose to display interrupt statistics in one of four different formats, as shown in Table 2.5:

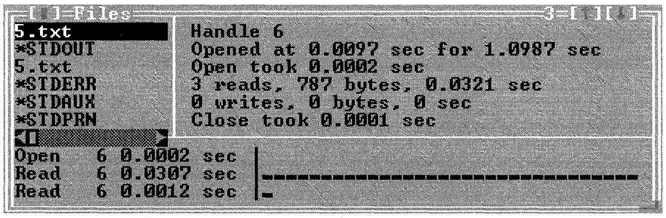
Table 2.5 Summary of interrupt statistic formats

Format	Function
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. For Turbo Profiler to record any file-activity information (such as read, write, open, or close), Statistics | Files must be set to *Enabled*.

Figure 2.10 The Files Window



```
[■]=Files==========================================3=[↑][↓]=
5.txt          Handle 6
*STDOUT        Opened at 0.0097 sec for 1.0987 sec
5.txt          Open took 0.0002 sec
*STDERR        3 reads, 787 bytes, 0.0321 sec
*STDAUX        0 writes, 0 bytes, 0 sec
*STDPRN        Close took 0.0001 sec
Open     6 0.0002 sec |
Read     6 0.0307 sec |▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬▬
Read     6 0.0012 sec |▬
```

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 the file was opened
- How long the file was open
- The time required to open the file
- 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 SpeedMenu.

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 SpeedMenus.

Collection (top pane)

Ctrl C The Collection 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.

Detail (top pane)

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

When Full (top pane)

Ctrl W The When 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.

Display (bottom pane)

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

You can specify two options with the radio buttons in 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.

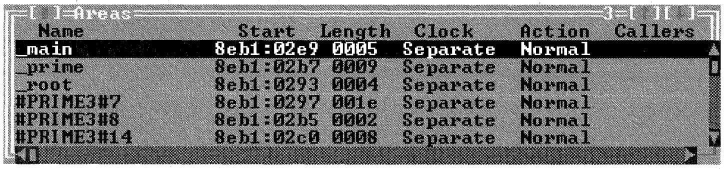
  Both options display the execution time in seconds; however, Graph also graphs the display and Details tells when, in the course of program execution, the file activity took place.

- 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. The Areas window is used to inspect areas that have been set and to adjust the behavior of individual areas.

Figure 2.11 The Areas window



```
[ ]=Areas=========================================3=[ ][ ]=
 Name                Start   Length  Clock     Action  Callers
_main               8eb1:02e9 0005  Separate  Normal
_prime              8eb1:02b7 0009  Separate  Normal
_root               8eb1:0293 0004  Separate  Normal
#PRIME3#7           8eb1:0297 001e  Separate  Normal
#PRIME3#8           8eb1:02b5 0002  Separate  Normal
#PRIME3#14          8eb1:02c0 0008  Separate  Normal
```

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
```

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.

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

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 passes the marker).
- Callers: whether the profiler tracks the area's immediate caller only, all callers, or no callers.
- Winproc: "Yes" if the area is a Windows procedure, otherwise it is blank. This column is pertinent only if a Windows program is being profiled with TPROFW, or with TPROF acting as a remote Windows profiler.

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

- Add or remove areas
- Inspect areas
- Change options for individual areas
- Sort the displayed information

Add Areas

Ctrl A Choose Add Areas to add area markers. When selected, this command leads to another menu that contains the commands All Routines, Module, and Routine.

- All Routines places markers at each routine in the current module.
- Module leads to the Pick a Module dialog box. This command lets you place markers in a program module other than the one present in the Module window.

  For more information on the Pick a Module dialog box, refer to the "Module" section on page 32.
- Routine, when selected, leads to the Enter Routine Name to Add text box. Type the name of the routine that you want to select, and choose OK.

Remove Areas

Ctrl R Remove Areas is used to erase markers that have been set.

- All Areas removes the markers from all areas in the program, including the modules not currently displayed in the Module window.
- Module leads to the Pick a Module dialog box. From this dialog box, choose a module whose area markers you want to delete.
- This Area removes the area marker currently highlighted in the Areas window.

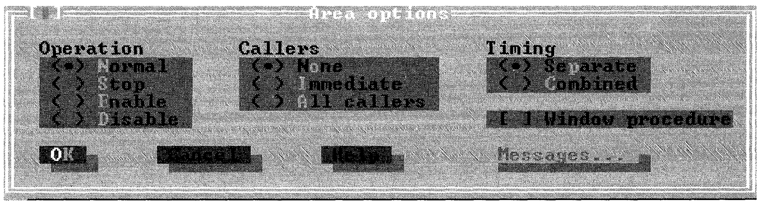
Inspect

Ctrl I When you choose Inspect, the profiler switches to the Module window and places the cursor on the first line of source code corresponding to the current area highlighted in the Areas window. If the area highlighted does not correspond to a program source line, the CPU window is opened instead.

Options

Ctrl O When you choose Options from the Areas window's SpeedMenu, the Area Options dialog box comes up.

Figure 2.12 The Area Options dialog box



```
Area options

Operation        Callers            Timing
(•) Normal       (•) None           (•) Separate
( ) Stop         ( ) Immediate      ( ) Combined
( ) Enable       ( ) All callers
( ) Disable                         [ ] Window procedure

OK   Cancel   Help   Messages...
```

You can specify three options with the radio buttons in this dialog box: Operation, Callers, and Timing.

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

  See page 30 for a complete discussion on the Operation option.
- Callers specifies the depth of call-path information.

- 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 execution time of the current area's child routines to a higher-level area or keep it separate.

  The timing option is described in detail on page 31.

- Window Procedure, when checked, specifies that the current area marks a procedure used by Windows.

- The Messages box becomes active when the Window Procedure check box is checked. When Messages is chosen, the Window Procedure Messages dialog box is displayed.

For a complete discussion of the Window Procedure Messages dialog box, refer to Appendix D.

Sort

Ctrl S 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.

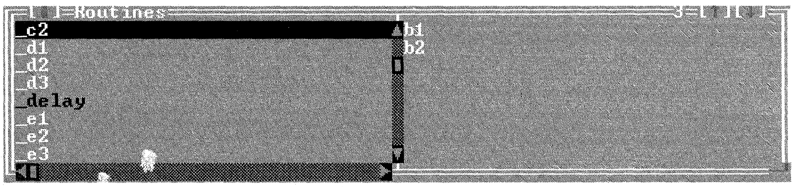
Routines

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 when you want to see which routines have markers set on them. You can use the Inspect command on the Areas SpeedMenu to go to other modules by "inspecting" a routine in a particular module.

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 Borland's line of C++ or Pascal compilers, 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).

Note The Routines menu gives you easy access to information related to symbols.

Figure 2.13 The Routines window


```
[■]═1═Routines═══════════════════════════3═[↑][↓]═
 _c2                                      b1
 _d1                                      b2
 _d2
 _d3
 _delay
 _e1
 _e2
 _e3
```

The Routines window is divided into two panes. 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 area markers appear highlighted in the right pane. (By default, an underscore (_) precedes all global variables in Borland C and C++ programs.)

Both panes of the Routines window have SpeedMenus.

Local Module (right pane)

Ctrl L When you choose Local Module in the Local Routines pane, the Pick a Module dialog box pops up, listing all modules in your program.

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

Areas (both panes)

Ctrl A The Areas 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).

Callers (both panes)

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

Module (both panes)

Ctrl M The Module command opens a Module window and positions the cursor on the source code for the current routine.

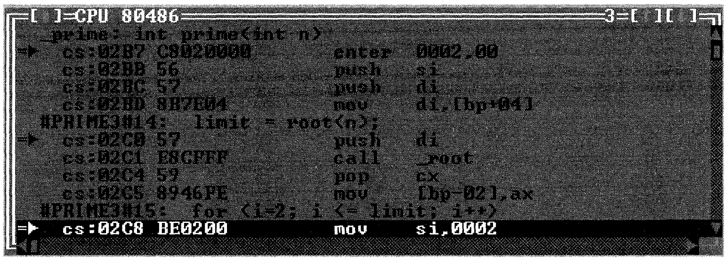
Profile (both panes)

Ctrl S 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. The title bar of the Disassembly window indicates your system processor type and the word `Protected` appears if your program is a protected-mode program. You use the Disassembly (CPU) window to help determine if you want to rewrite parts of your program in assembly language.

Figure 2.14 The Disassembly (CPU) window



```
[ ]=CPU 80486========================================3=[ ][ ]
 _prime: int prime(int n)
=▶  cs:02B7 C8020000        enter  0002,00
    cs:02BB 56              push   si
    cs:02BC 57              push   di
    cs:02BD 8B7E04          mov    di,[bp+04]
 #PRIME3#14:  limit = root(n);
=▶  cs:02C0 57              push   di
    cs:02C1 E8CFFF          call   _root
    cs:02C4 59              pop    cx
    cs:02C5 8946FE          mov    [bp-02],ax
 #PRIME3#15:  for (i=2; i <= limit; i++)
=▶  cs:02C8 BE0200          mov    si,0002
```

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

```
THAT#ModName#SymbolName     /* Borland C++ */

ModName.SymbolName     { Borland Pascal }
```

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

```
#ModName#LineNumber     /* Borland C++ */

ModName.LineNumber     { Borland 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 marked instructions that have no symbol name appear in the Areas window as hex addresses in *Segment:Offset* form. (Note that *F2* can also be used to remove a marker from an area.)

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

- A specified address
- 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 SpeedMenu 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.

Goto

Ctrl G When you choose Goto, a dialog box pops up and requests the address you want to go to. Enter a hexadecimal address, using the hex format for your programming language.

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

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

Origin

Ctrl O You choose Origin 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 Previous command restores the Disassembly (CPU) window to the position it had before you chose Origin.

Follow

Ctrl F The Follow 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, and conditional jump (JZ, JNE, LOOP, JCXZ, and so on) instructions.

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

Previous

Ctrl P 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.

View Source

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

Mixed

Ctrl M 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 SpeedMenu'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.

Text File

You can examine or modify any file on your system by using a Text File window. You can view the file only as ASCII text, so files containing binary data may display characters from the extended ASCII character set.

Before you can open a File window, you must choose the View | Text File command from the menu bar. This command brings up a dialog box in which you can use DOS-style wildcards to get a list of file choices, or you can type a specific file name to load.

Once you've chosen your file, Turbo Profiler displays it in the File window.

The File window shows the contents of the file you've selected. The name of the file you're viewing is displayed at the top of the window, along with the line number the cursor is on.

The File window SpeedMenu

The File window SpeedMenu has a number of commands for moving around in a disk file, changing the way the contents of the file are displayed, and making changes to the file.

Goto

Ctrl G

Positions you at a new line number in the file when you enter the new line number to go to. If you enter a line number after the last line in the file, you will be positioned at the end of the file.

Search

Ctrl S

Searches for a character string, starting at the current cursor position. You are prompted to enter the string to search for. If you have marked a block in the file using the *Ins* key, that block will be used to initialize the Search dialog box. This saves you from typing if you want to search for a string that is already in the file you are viewing. The search string can include simple wildcards, with ? indicating a match on any single character, and * matching 0 or more characters.

The search begins from the current cursor position and does not wrap around from the end of the file to the beginning. To search the entire file, start at the first line (press *Ctrl+PgUp* to move to the top of the file).

You can also invoke this command by simply starting to type the string you want to search for. This brings up a dialog box exactly as if you had specified the Search command.

Next

Ctrl N

Searches for the next instance of the character string you specified with the Search command; you can use this command only after first issuing a Search command.

This command is useful when your Search command didn't find the instance of the string you wanted. You can keep issuing this command until you find what you want.

File

Ctrl F Displays the same Program Load dialog box as the View | Text File command, enabling you to load a different file.

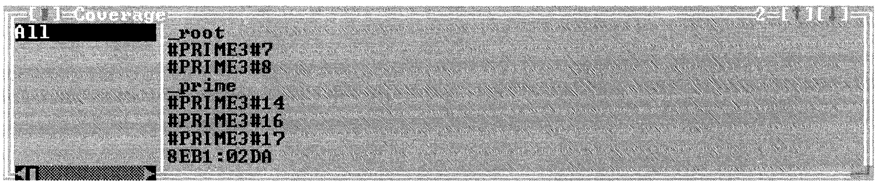
Edit

Ctrl E Lets you make changes to the file you're viewing by invoking the editor you specified with the TFINST installation program. This is done through the Editor Program Name field (located under the Options | Directories dialog box in TFINST). This optio: is not available when profiling Windows applications with TPROFW.

Coverage

The Coverage window has two panes: the left pane displays a list of selected modules, and the right pane shows the blocks contained in those modules. A *block* is a section of code that has only one entry point and one exit point; there are no jumps into or out of a block.

Figure 2.15 The Coverage window



```
[ ]=Coverage=========================================2=[↑][↓]=
All          _root
             #PRIME3#7
             #PRIME3#8
             _prime
             #PRIME3#14
             #PRIME3#16
             #PRIME3#17
             8EB1:02DA
```

Selecting coverage mode (Statistics | Profiling Options) opens the Coverage window, replacing the Execution Profile window. By default, Turbo Profiler selects as many program modules as possible in the left pane and all blocks within those modules are listed in the right pane.

Note The Coverage window and the Execution Profile window are mutually exclusive; only one can be open at any given time.

In its default setting, the Coverage window lists only unexecuted blocks in the right pane. Blocks that have been executed (*hit*) during a program run will be deleted from the listing.

Before a program run, all program blocks are marked in the Module window with a ► character. In the default Coverage mode, Turbo Profiler removes the ► character from each block as it is hit. Successive program runs continue to remove block markers, allowing you to attempt different actions in order to hit the remaining marked blocks.

Note Turbo Profiler automatically marks all program blocks with a single marker.

To restore all program blocks to an unhit status, choose Delete All from the Statistics menu.

Each pane of the Coverage window has its own SpeedMenu.

The left pane's SpeedMenu provides commands for selecting which modules are to be included in the profiling session. The right pane's SpeedMenu provides commands pertaining to how blocks are displayed.

Add All Modules (left pane)

Ctrl A When you choose Add All Modules, the blocks from all program modules contained in the executable program are marked for the profile session. The keyword *All* indicates this selection.

Remove All Modules (left pane)

Ctrl R Remove All Modules removes all modules from the module list, leaving both panes in the Coverage window empty.

Add Module (left pane)

Ctrl M The Add Module command opens the Pick a Module dialog box, allowing you to select from the list of modules in the current program.

For more information on the Pick a Module dialog box and its use, refer to page 32.

Remove Module (left pane)

Ctrl V Choosing Remove Module opens the Pick a Module dialog box. Using this dialog box, you can remove a module from subsequent profile runs.

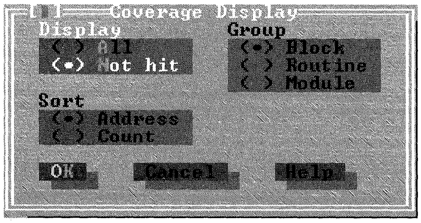
Delete This Item (left pane)

Ctrl D The Delete This Item command either deletes or adds the currently highlighted item for the remaining profile runs. If the item has no dash (—) in front of it, the item is deleted. If the item does have a dash in front of it, it's added.

Display (right pane)

Ctrl D When you choose Display from the right pane's SpeedMenu, the Coverage Display dialog box appears.

Figure 2.16 The Coverage Display dialog box



```
Coverage Display
Display            Group
( ) All            (•) Block
(•) Not hit        ( ) Routine
                   ( ) Module
Sort
(•) Address
( ) Count

OK   Cancel   Help
```

- The Display buttons define what gets displayed in the right pane of the Coverage window.
  - When you choose All, all blocks (both hit and unhit) are displayed in the right pane. Blocks that have been hit show how many times they've been hit, and blocks that haven't been hit display a zero (0) next to their entries. You can set the number of hit counts that the Profiler tracks from the Profiling Options dialog box's Maximum Coverage Count option.

    See Figure 2.18 on page 56 for more information on setting Turbo Profiler's coverage hit count.

    - Not Hit (the default setting) means that only blocks that haven't been hit are displayed.

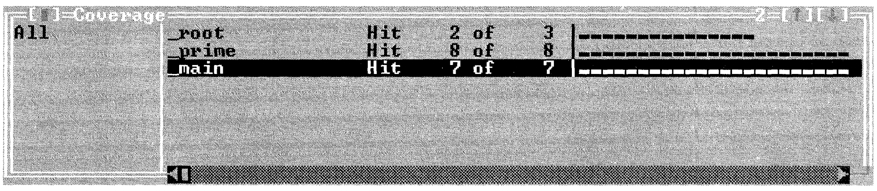
- The Sort buttons define the order of block display. This menu item is available only if All is selected from the Display option and Max Coverage Count in the Profiling Options dialog box is greater than 1.
    - A sort by Address lists blocks in alphabetical order.
    - A sort by Count lists the blocks in order of number of hits, starting with those blocks that have not been hit.

- The Group radio buttons allow you to group blocks individually, or by routine or module.

Note The Group buttons are available only if Display is set to All.

- Selecting Block (the default setting) causes all blocks to be listed in the window pane.
- Selecting Routine causes blocks to be grouped together by routine. With this selection, only routine names from the modules displayed in the left pane are listed in the right pane. However, in addition to each routine name, a count of blocks contained in the routine and the total number of blocks that have been hit is displayed.

Figure 2.17 The Coverage window, listing blocks by routine



```
All          _root        Hit    2 of    3 |
             _prime       Hit    8 of    8 |
             _main        Hit    7 of    7 |
```

- The Module radio button applies only to programs with multiple modules. It works similarly to the Routine button, except that it groups blocks by module instead of by routine name.

Position (right pane)

Ctrl P

Use the Position command to inspect source code for a particular block of code. When in the right pane, select a block by moving the highlight bar with the arrow keys. After you select a block, choose Position to bring the source code into view in the Module window or the CPU window while keeping the Coverage window active.

Pressing *Spacebar* also executes the Position command.

Module (right pane)

Ctrl M

The Module command is much like Position, except that Module activates the Module or CPU window and positions the cursor at the top of the block selected.

Pressing *Enter* also executes the Module command.

Note When Coverage mode is in effect, certain Turbo Profiler functions are no longer pertinent, and respective menu items will be dimmed. Specifically, the following windows are unavailable from the View menu: Execution Profile, Callers, Files, Areas, and Routines. The Statistics menu dims both the Callers and Files functions. Also, the Accumulation toggle switch in the Statistics menu is automatically set to Enabled. Lastly, the SpeedMenu of the Module window dims the functions Add areas, Remove Areas, Operation, and Callers.

Run menu

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
- Your program causes an exception

(Usually, the interrupt key is the *Ctrl+Break* key combination; you can change this to another key with TFINST, the profiler installation program. When profiling a Windows application using TPROFW, use the keystroke sequence *Ctrl+Alt+SysRq* to interrupt the program. See Appendix D 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).

Note When you choose Run | Run or Run | Program Reset after reaching the run count limit, any statistics collected for a previous execution are reset. If you want to save a set of statistics, use Statistics | Save before you select Run or Program Reset, or use Statistics | Profiling Options to set the run count greater than 1.

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.

Program Reset

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

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, passive, or coverage
- 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

The default extension is .TFS, but you can use any extension you want.

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. To automate this process, you can create a macro or DOS batch file to automatically run several profiles with different options or area markers, saving the results to individual .TFS files. Then, using the macro or batch file, you can run the profiles and view your results later.

Note The first four items on the Statistics menu are toggle switches. When selected, the Callers, Files, Interrupts, and Overlay options will toggle between the *Enabled* and *Disabled* modes.

Callers

When you set the Callers option to Enabled, the profiler gathers statistics about which routines call other routines. To specify which routines you want call histories for, you choose the Callers command on the Module window's SpeedMenu or the Options command on the Areas window's SpeedMenu, then select the appropriate radio buttons under Callers and Areas.

Note You must run your program and accumulate some statistics before these windows show any information.

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 Callers to Disabled.

Files

When you enable the Files 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 Files to Disabled.

Interrupts

When the Interrupts option is set to Enabled, the profiler collects statistics about which interrupts your program calls. The profiler keeps separate statistics for DOS, video, disk BIOS, mouse, 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 interrupt information, set Interrupts to Disabled.

Overlays

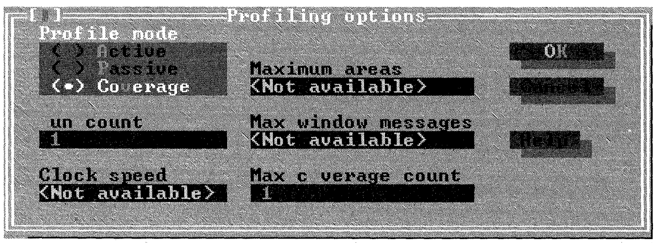
The Overlays option toggles whether statistics are collected for the overlays your program loads. If your program does not contain overlays, an error message is displayed if you try to enable this menu item, and the option remains disabled.

Gathering overlay information consumes memory and slows down the speed at which your program runs. If you don't need overlay information, set this option to Disabled.

Profiling Options

The Statistics | Profiling Options command opens the Profiling Options dialog box, shown here:

Figure 2.18 The Profiling Options dialog box



```
[■]══════════════Profiling options═══════════════════
 Profile mode
  ( ) Active                                   OK
  ( ) Passive          Maximum areas
  (•) Coverage         <Not available>         Cancel

  Run count            Max window messages
  1                    <Not available>         Help

 Clock speed           Max coverage count
 <Not available>       1
```

With the Profiling Options dialog box, you can set any of the following options:

- Profile Mode specifies the analysis mode. The default mode is Active.
  - *Active analysis* means full statistical information is collected for each marked area. This includes basic clock timing information, and for routines, how many times the routine was called and where it was called from.

- *Passive analysis* means only basic clock timing information for each marked area is collected. Passive mode is not available in TPROFW when running Windows in 386 enhanced mode.
- *Coverage analysis*, in its default setting, means that Turbo Profiler tracks the blocks of code that haven't been hit during a succession of program runs.

Note If you change analysis mode from Active or Passive to Coverage (or vice versa), you'll be prompted with `Reload program and change profiling mode?` If you select the YES button in answer to this prompt, the current set of statistics is deleted, and the new analysis mode becomes effective. Be sure to save any vital profile statistics before changing analysis modes, or they'll be lost.

- Run Count sets how many times your program will run while the profiler collects statistics. The default is 1.

Note If Run Count is set to a number higher than 1, Turbo Profiler will reset statistics only after the specified number of runs.

- 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.
- Maximum Areas specifies the maximum number of areas you can divide the current program into. Turbo Profiler sets default areas based on the density of the symbols it finds appended to the executable file. Increasing this setting will decrease the amount of memory left for the profiled program.
- Maximum Windows Messages sets the maximum number of Windows messages that can be marked. The default setting is 20. Increasing this setting will decrease the amount of memory left for the profiled program.
- Maximum Coverage Count sets the maximum number of hit counts that the Profiler will keep track of while in coverage mode. The default for this setting is 1 (a block has either been hit, or it has not). Increasing this setting will decrease the amount of memory left for the profiled program and will increase the time it takes to run.

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.

Note 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 Statistics | Accumulation option turns automatic data collection on and off. This means you can collect data for a subset of all marked areas without removing any area markers, and 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 SpeedMenu, choose Options 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 Statistics | Accumulation 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.

Disabling 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 the routine
- 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 Enabled). Mark another area that disables collection after the routine returns. You can also 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 you're interested in calls to **abc** only when it is called from **xyz,** but not at any other time.

```
=>     main()                 /* normal area marker at routine */
      {
        ⋮
```

```
          abc();                 /* don't want to collect stats 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 interested only 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 use of 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 Statistics | Accumulation to *Disabled*), or the first call to **xyz** will erroneously contribute to the collected statistics.

Delete All

The Statistics | Delete All command erases all statistics collected for the current profiling session—essentially wiping the data slate clean so you can start afresh. Delete All removes all profile data from the open profile report windows (Execution Profile, Callers, Interrupts, Files, Overlays, and Coverage), but it does not delete the profiling options you've set.

Save

The Statistics | Save 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, coverage counts).
- 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 Statistics | Restore command.

When you choose Statistics | Save, the Enter File Name to Save dialog box appears.

The Name input box lists a default .TFS 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 desired file name (including disk drive and path, if you so choose).
3. Choose OK (or press *Enter*).

If the statistics file already exists, a message box asks if it's all right to overwrite the file.

Restore

When you choose Statistics | Restore, the Enter File Name to Restore dialog box appears.

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 load a statistics file, Turbo Profiler restores all the saved options, settings, and resulting statistical information to the environment screen.

Print menu

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* for the Module, Routines, Text File, and Disassembly windows) 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.

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.

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* that lists execution time and counts data next to each source line or routine you've marked as an area, as shown in the following listing:

```
Program: C:TPROFILEPRIME1.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
0.0000 1        }
```

Options

When you choose Options from the Print menu, the Printing Options dialog box appears.

- 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

With the Options menu, you can

- Record a keystroke macro.
- Remove one or all 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 a menu that lets you define new keystroke macros or delete ones that have already been assigned.

The profiler's macro facility gives you the ability to record frequently used keystroke sequences and place the recording into a single *macro* keystroke. For example, during profiling, you may often repeat the same sequence of commands. With macros, you can define a single keystroke that "plays" this sequence of keystrokes. Once defined, you can simply press the macro key to perform the tedious task.

Create

Alt =

When issued, the Create command starts recording keystrokes into an assigned macro key. As an alternative, press the *Alt*= hot key for Create.

When you choose Create to start recording, a prompt asks for a key to assign the macro to. Respond by typing in a keystroke or combination of keys (for example, *Alt+M*). The message `RECORDING` will be displayed in the upper right corner of the screen while you record the macro.

Stop Recording

Alt -

The Stop Recording command terminates the macro recording session. Use the *Alt*+- (*Alt+Hyphen*) hot key to issue this command, or press the macro keystroke that you are defining to stop recording.

Important Do *not* use the Options | Macro | Stop Recording menu selection to stop recording your macro, because these keystrokes will then be added to your macro! (The menu item is added to remind you of the hot key.)

Remove

The Remove command removes a macro assigned to a keystroke. When you choose this menu item, a list box pops up, displaying a list of the currently defined macros. Select the macro that you want to delete, and press *Enter*.

Delete All

The Delete All command removes all macro keystroke definitions and restores all keys to their original (default) meanings.

Recording macros

To record a macro,

1. Choose Options | Macros | Create (or press *Alt*=) to begin the macro definition. A prompt appears, asking which key you want the macro assigned to.
2. Enter a key that isn't already assigned, for example, *Shift+F10*.
3. Once you begin recording the macro, all keystrokes entered become part of the macro definition. However, keystrokes entered as program input and mouse actions will *not* be included in the macro definition.

4 Stop recording the macro: Press *Alt-* or press the keystroke of the macro you are defining (*Shift+F10* in this example).

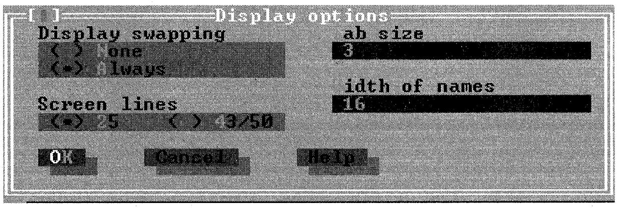
5 Save the macro to a configuration file: Choose Macros from the Save Configuration dialog box (Options | Save Options). Type the name of the configuration file you want (or use one listed in the dialog box), and press OK.

6 Continue profiling.

Display Options

The Options | Display Options command opens the Display Options dialog box, shown here.

Figure 2.19 The Display Options dialog box



```
[ ]              Display options
Display swapping            ab size
 ( )  one                   3
 (•)  lways
                            idth of names
Screen lines                16
 (•)  5     ( )  3/50

 OK        Cancel      Help
```

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 None and Always specify whether Turbo Profiler swaps the user screen back and forth with the Turbo Profiler environment.

- *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.

  When profiling a Windows application, display swapping is automatically set to *Always*, and it cannot be changed.

Screen Lines

You use Screen Lines to specify whether Turbo Profiler's screen uses the normal 25-line display or the 43-line 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 program was originally compiled
2. In the directories (if any) you've listed under Options | Path for Source (or stated in the command-line option using the **–sd** switch)
3. In the current directory
4. In the directory that contains the .EXE file of the program you're profiling

With the Options | Path 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,

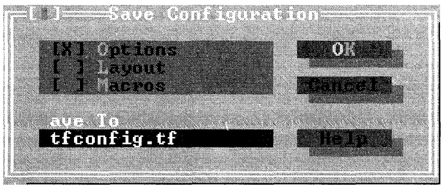
```
C:BorlandTC; C:Borland'TASM
```

Save Options

With Options | Save 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 Options | Restore Options.

When you choose Options | Save Options, the Save Configuration dialog box appears:

Figure 2.20 The Save Configuration dialog box



```
[■]═══Save Configuration═══
 [X] Options          OK
 [ ] Layout
 [ ] Macros           Cancel
 Save To
 tfconfig.tf          Help
```

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 Options | Path 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. To save your options there, choose OK (or press *Enter*).

  To save your options 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 Options | Restore Options.

Restore Options

Options | Restore Options 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 Options | Restore Options, the Restore Options dialog box appears.

The Restore Options 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 restore only a configuration file that was created by the Options | Save Options command.)

Window menu

The Window 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 Window 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 the section "Turbo Profiler windows" earlier in this chapter for information on these elements and how to use them.

Zoom

The Zoom command zooms the active window (the one with a double-line border) to full-screen, or returns the active window to the pre-zoomed size.

Next

The Next command activates the window whose number succeeds the number of the current window.

Next Pane

In windows with multiple panes, the Next Pane command moves the cursor to the next pane.

Size/Move

The Size/Move command activates Turbo Profiler's window-arranging mode. You move the current window with the *Left, Right, Up* and *Down* 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 Iconize/Restore command shrinks the active window to an icon or restores the active icon to a window.

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.

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.

User Screen

Choose Window | User Screen (or press *Alt+F5*) to view your program's full-screen output. 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. For a full explanation of how to manage windows, see page 21.

Help menu

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, you can either

- Press *F1* or *Alt+F1* at any time (including from any dialog box or when any menu command is selected), or
- 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) 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. 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 highlighted 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 pressing ***Enter***. (You can also use the mouse to click it.)

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.

Help on Help

The Help | Help on Help command opens up a text screen that explains how to use the Turbo Profiler help system.

Chapter

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 your screen 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.

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 these questions in this chapter. We do, however, provide some general guidelines, techniques, and strategies to get you moving.

The first time you load a program into Turbo Profiler, it

- Sets the profile mode to Active.
- Automatically scans through your .EXE file to find the main program module.
- Loads the main source module into the Module window.
- Sets area markers for the program.
- 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 lines in the .EXE file that have debug information associated with them.

Note 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 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 presented 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.

Adjusting your program

The first adjustment to your program is to set it up so you can 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.

Carefully think through how you would like to gather profile statistics, and adjust your source code so the statistics gathered are useful and sufficient. Once the source code is modified (if it needs to be), compile the program with debug information turned on. Then, set area markers that tell the profiler where to collect statistics and what kind of statistics to collect.

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

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

  Selecting pertinent input data is important. A string search program evoked 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. (The Run Count setting in the Profiling Options dialog box allows statistics to be accumulated for the number of runs specified.)
- Modify the program to work independently of keyboard input, or disable accumulation of statistics in any areas that require keyboard input.

  If your program requires keyboard input, read data from a file or use a random number generator to stuff numbers into 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.

Compiling 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.

Turbo Profiler works with 16-bit Turbo C++, Borland C++, and Turbo Assembler programs. You must compile your source code with full symbolic debugging information turned on, as follows:

- **Borland C++:** In the Project Options dialog box, check the following two options:
  - Compiler | Debugging | Debug Information in OBJs.
  - Linker | General | Include Debug Information.
- **Turbo Assembler:** Source code must be assembled with the **/zi** command-line option and linked with TLINK, using the **/v** option.

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 directories specified with the Options | Path for Source command (or in the directories specified with the **–sd** command-line option).
3. In the current directory.
4. In the directory containing the executable program being profiled.

Setting profile areas

Once you've adjusted your program so you can concentrate on the troublesome areas and have 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 on 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* is a location in your program where you want to collect statistics: It 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 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 encounters an area marker:

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 (or to DOS, if you are using batch mode execution). 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 SpeedMenus.

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 the 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 Statistics | Profiling 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.

Adding areas

Divide your program into a number of areas by selecting Add Areas from the Module window's SpeedMenu. After this, 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 Every Line in Module as the default area setting.
- If there are many symbols and several modules, Turbo Profiler selects All Routines as the default area setting.

Note 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 for Active and Passive modes:

- 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 Statistics | Callers 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 Statistics | Files option enabled, you can monitor your program's file-access activity.
- The Statistics | Interrupts option, when it is enabled, records your program's interrupts.
- You can monitor your program's overlay file activity by enabling the Statistics | Overlays option.

Once you've enabled the appropriate Statistics menu options, you can open the corresponding profile report windows (through the View menu), then call up each window's SpeedMenu to specify details about how you want the data collected.

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

When should data collection start?

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

With Accumulation 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 routine's 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 the caller.

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 other routines. If you want a routine's time data to include time spent in child routines, choose Combined under Timing from the Areas window local Option command.

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).

Note 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.

See page 83 for more information about filtering displayed 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.

Profiling your program

Once you've selected appropriate areas to monitor, run the profile. You can save the resulting profile with the Statistics | Save command. This command saves the statistics to a .TFS (Turbo Profiler Statistics) file. If you plan to save several different profile results, use a 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.

You might not know if a profile is worth saving until you look at several sets of statistics.

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.

Focusing the profile session

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? (*Program testing and verification.*)
- 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? (*Program structure analysis.*)

The following table relates your profiling session to the type of information you'd like to gather:

Tabie 3.1 Ways of using a profiler

Purpose of profile	Type of information needed
Algorithm testing	Line-count information Dynamic call history
Program testing and verification	Coverage analysis Dynamic call history

Table 3.1 Ways of using a profiler (continued)

Purpose of profile	Type of information needed
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 it's more important to gather information about line count than execution times. You need to do the following:

1 Isolate the algorithm and its supporting routines by marking them as areas.

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, block execution information is more pertinent than execution times. 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. Coverage analysis mode is most useful for this type of verification testing.

If you deal with large pieces of code when you test and verify programs, you won't need as much detail as you would for algorithm analysis. However, it's still useful to know how many times a routine has been called. Organize a program test into groups of routines that create a call hierarchy. With this type of test, coverage analysis can help prove that every path in a switch statement or conditional branch has executed at least once.

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 routines are called at the right time. And, if a block of code does not get hit at the appropriate time, you can investigate why the piece of code does not get executed.

Timing execution and monitoring performance

When 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 per profile run.

Once you've set the area markers in a single module, profiling becomes a matter of successive grouping and refinement. These are the 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 to study 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, you can print all recorded call paths by choosing Print | Statistics with the Callers window open.

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.

Execution times and counts give you 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 to use

One important consideration when you're profiling is whether to use active, passive, or coverage analysis. You set the profiling mode from the Profiling Options dialog box (choose Statistics | Profiling Options).

Once you know your program works correctly, active and passive analysis are important tools for helping to improve the overall performance of your program. Coverage analysis, on the other hand, is a useful tool to use while developing your code.

Does the program call all routines at the appropriate time? Are there any sections of code that do not get executed? These are important questions that can be answered by profiling in coverage mode.

Active analysis

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.

When you profile in active mode, keep in mind how frequently the 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 marked areas won't slow the profile much.

See the section "Speeding up profiling" for other ways to make your profiling sessions go faster.

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 the expensive bookkeeping routines that the profiler performs under active analysis mode.

Passive analysis

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 marked areas (such as execution times, interrupt calls, and file activity). Your profile runs will go much faster in passive analysis mode.

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. To obtain greater statistical accuracy, collect statistics over several program runs (be sure to set the Run Count in the Profiling Options dialog box to coincide with the number of runs you plan to monitor).

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.

Passive versus active analysis

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 gives 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 gives the interrupt's time to the calling routine in active mode, but excludes the interrupt's time in passive mode.

Coverage analysis

Coverage analysis lets you verify program structure and execution sequence. Because coverage analysis doesn't slow down program execution (unless Max Coverage count is set to a high number), you can perform two tasks with one profiling pass. Coverage allows you to count how many times a block has been executed (set the Maximum Coverage Count using Statistics | Profiling Options). With this, you can verify that a block of code does get called and, at the same time, determine if it is getting called too many or too few times.

Speeding 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 SpeedMenu Display 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 Filter | Current command. You can also unmark areas with the local Remove command in the Module, Areas, and Execution Profile windows.

Improving 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), 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. Be sure to set the Run Count option in 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 executed your program the number of times specified in the Run Count option.

Caution! Running your program after the specified number of runs will reset all statistical information in preparation for a new set of runs. Be sure to save (or analyze) the statistics before embarking on a new set of runs.

Resonance

Resonance occurs, for example, if a loop cycles at the same frequency as the timer tick. 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 to 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.

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

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

Also note that if a section of your program disables interrupts, only the first clock tick during execution of that section will be counted, and your system's internal clock will run slower. Avoid setting the clock speed so fast that it causes multiple ticks while interrupts are disabled.

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, 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.

Profiling object-oriented programs

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

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 newfound knowledge to your source code to make your program faster and more efficient than ever.

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

Callers window

Once you have isolated a routine you want to improve, use the Callers window to locate all of the areas in your program that 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 lets you detect excessive overlay calls, which will then become candidates for placement in a non-overlaid module (unit).

Interrupts window

The Interrupts window reveals all 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 which files deserve your attention.

Coverage window

The Coverage window, in its default setting, displays blocks of code that have not been hit during the profiling session runs. This window helps to isolate sections of code that do not get called (dead code) and areas that need to be called. In addition, the Coverage window can be adjusted so that it displays the number of times each block of code has been hit during the program runs.

Filtering collected data

Turbo Profiler's windows provide SpeedMenu 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 SpeedMenu commands for filtering collected statistics

Window	SpeedMenu 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 Current, Module, or All from the Filter menu.)
	Remove	Permanently erases the current area's statistics from the collected data. ***Use with caution!***
Files	Collection (top pane)	When disabled, no file statistics are collected.
	Detail (top pane)	When disabled, displays only file open and close activities. When enabled, also displays file read and write activities.
	Display (bottom pane)	Displays each event either as a bar graph element, or as text showing the exact time and duration of the event.
Interrupts	Remove (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	Display	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.
Coverage	Display	Specifies blocks to display: (1) display all blocks or (2) display only blocks not hit.
	Sort	Determines the order of display for the blocks (either by address or by number of times hit).
	Group	Sets the method of block display as either (1) display all blocks, (2) group blocks by routine, or (3) group blocks by module.

When you choose Remove from the Execution Profile's SpeedMenu 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.

Note When you remove an area from the Execution Profile window, the remaining area's statistics will be recalculated for the current run only. A subsequent run of the program will most likely show the time for the removed area shifted into one or more other areas.

To permanently remove an area (and the time spent in that area) from the statistics, you must set the area to *Disable*. Be sure you don't forget to set a following area to *Enable*.

Revising your program

Here's a general plan of attack for finding routines where simple changes 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 SpeedMenu, set Display to Both or Per 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 the general guidelines just listed, 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

Modifying data structures

Try using more sophisticated data structures or algorithms. For example, 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 need to reorder only the pointers, rather than entire lines of text.

Storing precomputed results

If a set of computed numbers is referenced more than once in a program, it is best to do the computations once, and store the results in a table that can then be accessed. For example, build a sine table, then look up sine as a function of degrees based on an integer index.

Caching 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.

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.

Evaluating data as needed

Structure the order of conditional tests and switches so that those most likely to yield results are evaluated first. This will reduce the number of times that low-incident switches will be tested.

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

Optimizing existing code

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

For 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.

  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 needs to move the successive element down only 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] + items[1] + items[2] + items[3];
  ```

For routines:

- Rewrite frequently called routines as inline routines, or replace their definitions with inline macros.
- Use co-routines for multipass algorithms that operate on large data files. (See the **setjmp** and **longjmp** routines in C.)
- Recode recursive routines to use an explicitly managed data stack.

For 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 show the details of how Turbo Profiler accumulates statistics; how area markers affect data collection, where time and count information gets calculated, and how Turbo Profiler keeps track of routine calls.

Chapter

4

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 both for specifying the type of information to collect and for interpreting the resulting reports.

Consider the source code in 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);
   printf("Leaving Route 66\n");
}

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

In the previous program, setting areas to Routines in Module effectively sets up four time-collection compartments and four count-collection compartments. Turbo Profiler keeps execution times and counts for *main*, *route66*, and *highway80*. In addition, Turbo Profiler keeps a total for both execution times and for counts.

Area boundaries

In this section, you follow program execution and see what the profiler does as each area marker is passed. It's best to think of this process as going through a series of tollbooths. After you pass a tollbooth, you're on a section of road associated with that tollbooth until you come to another tollbooth.

You're in no tollbooth's territory before you reach the first tollbooth. When you pass the first tollbooth, time spent on each section of roadway is tracked by the tollbooth in charge of that section of the road. Note 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.

Time and count collection

Before you enter the main program block, C startup code is executed, which is equivalent to no tollbooth's territory. 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 (tollbooth) 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 (the tollbooth) 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, turn off *route66*'s area marker (position the cursor on *route66*'s area marker and press *F2*) and compare 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 took to execute *route66*.

Showing routine call overhead

You might want to measure the time consumed by calling a routine (for example, **route66**) and ignore the time spent inside the routine. You can also get this kind of information using *passive analysis*, discussed in Chapter 3. The easiest way to get this information is to disable collection at the entry point for *route66*, and then to reenable collection upon return from *route66*.

To disable **route66**, position the cursor on the function header. Next, choose Operation from the Module window's SpeedMenu to open the Area Options dialog box, and set Operation to Disable.

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 operation for that area marker to Enable (using the Area Operations dialog box).

Who pays for loops?

The tollbooth analogy helps explain why passing through an area marker and jumping back to a program statement that precedes that marker (using a loop or goto statement) doesn't change the current area. Even though you're 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.

Knowing this, take a look at the next program:

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

=>   void main()
     {
        printf("Entering main
        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);
     }
```

In program *plost*, we've complicated the routine *lost_in_town* by using a compound statement inside a loop. Assume that three area markers have been set: one for *main*, one for *lost_in_town*, and one for the line 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`, timing information 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 changes only when you trip an area marker. This can produce unexpected results.

For instance, if you set the three markers for program *plost* as already described (one each for the *main* function, the *lost_in_town* function, 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, occur after the `Wrong Turn` marker was tripped.

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*.

Now, consider the following code:

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

Assume that areas are set for All Routines in the module so that routines *main*, *func1*, and *func2* each mark the beginning of an area. 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 *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).

Multiple return statements

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

Note Even though you might have several explicit return points in your function, Borland's C and C++ compilers actually turn 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.

Disabling often-called functions

The easiest way to disable collection for a function is to set a Disable marker at the function header and an Enable marker on the line after the call to the function. However, if a function is called from more than one place, it may be difficult to set Enable markers after each function call. In this case, set the Enable marker at the closing brace of the function, or (better yet) at the actual return statement in the function code.

For example, if you want to overlook the time spent in *func1*, set a Disable marker at the header for *func1*. Then, enable collection again at the return statement for *func1*:

1. Go to the Disassembly window (View | Disassembly).
2. Set an area marker at the `ret` statement of the function.
3. Go to the View | Areas window.

4 Set options to Enable (*Ctrl+O, E*).

Now, whenever *func1* is called, collection will be disabled upon entry to the function, and enabled at the exact point that the function returns.

A simpler yet less accurate way to reenable collection is to set the Enable marker at the closing curly brace of the function in question. The drawback of this method, however, is that timing information will be collected for the time it takes to return from the function.

Logging callers

An active routine is a routine currently on the profiler's routine call stack. In active analysis, 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 Statistics | Callers 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 three menu settings:

- Statistics | Callers (set to either Enabled or Disabled)

- The Callers option, set from the Areas' local window Option selection
- The Stack option, set from the Module's local window Callers selection

While all selections relate to callers, you gain finer control over logging call paths by using the SpeedMenu of the Module window. From this menu, you can set the Areas option to log callers for a single routine, for all the routines in a single module, or for all the routines in the program. As well, the Stack command allows you to 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 Statistics | Callers 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 Statistics | Callers to Enable won't change the value of the Callers options for any of the areas.

Disabling the Statistics | Callers 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 Statistics | Caller.)

Sampling vs. counting

This section relates only to passive mode.

The profiler doesn't 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 value of the clock speed (Statistics | Profiling Options | Clock Speed) and compare the resulting profile to the previous one.

Profiler memory use

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.

Appendix

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, and *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 turn an option off only 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 Appendix B.)

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
–b	Loads Turbo Profiler in batch mode.
–bc*count*	Run in batch mode with run count of *count*.
–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.
–ji	Session-state saving: Don't use session-state file if you've recompiled your program.
–jn	Session-state saving: Don't use session-state file.
–jn	Session-state saving: Prompt for session-state file use if you've recompiled your program.
–jn	Session-state saving: Use session-state file, even if you've recompiled your program.
–p	Enables mouse support.
–r	Enables profiling on a remote system.
–rn*L;R*	Remote network profiling, where *L* is the local machine and *R* is the remote machine.
–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*DIR*	Sets one or more source directories to scan for source files.
–Sn	Don't load symbols.
–vg	Saves complete graphics image on program screen.
–vn	Disables 43/50 line display.
–vp	Enables EGA palette save for program output screen.

Batch mode (–b)

The **–b** command-line argument instructs Turbo Profiler to run in *batch mode*. Turbo Profiler's environment isn't activated in batch mode; instead, the profiler runs the program and saves all statistics to a *filename.TFS* file, where *filename* is the name of the executable file.

The **–b** argument lets you create a DOS batch file of one or more TPROF commands. If the program you're profiling doesn't require any keyboard input, you can run the batch process while you're doing something else.

Note Before profiling in batch mode, an area file must be set up to instruct the profiler what areas to inspect and what profiling mode to use.

To set up the area file, load your program and choose the profiling mode you want to use (Statistics | Profiling Options). Next, mark areas that you want inspected, setting up all Enable and Disable markers as needed. Then, exit Turbo Profiler. This creates a .TFA file that records the selected areas and profiling mode.

Once the area file is set, create a DOS batch file with the desired Turbo Profiler commands. Here's an example:

```
TPROF -b MYPROG arg1 arg2
rename MYPROG.TFS MYPROG1.TFS
TPROF -b MYPROG arg1 arg3
rename MYPROG.TFS MYPROG2.TFS
TPROF -b MYPROG arg2 arg2
rename MYPROG.TFS MYPROG3.TFS
TPROF -b MYPROG arg2 arg3
```

Alternately, you can set run count >1 to accumulate all statistics into the same file.

Notice that you must rename the .TFS file after each TPROF command. If you fail to do so, successive runs of the same program will overwrite the statistics file already created. Once you've written your batch file, run it from the DOS command line.

Configuration file (–c)

This option tells Turbo Profiler to use the indicated configuration file. The default is TFCONFIG.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. It 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 use this option only on a color display, because only color displays have multiple display pages. You can't use this option if the profiled program itself uses multiple display pages.

–ds This is the default option for monochrome displays. It 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.

Session-state saving (–j*n*)

The session-state options specify how you want Turbo Profiler to handle the session-state files. See "Session Saving" on page 25 for more information.

Mouse support (–p)

This option enables mouse support (on by default).

Remote profiling (–r)

All **–r** options affect Turbo Profiler's remote profiling link.

–r	Enables profiling on a remote system. If no other **–r** command-line options are specified, **–r** uses the default serial port (COM1) and speed (115K baud), unless these have been changed using TFINST.
–rn*L;R*	Enables profiling on a remote system over a local area network link. *L* and *R* are optional arguments, specifying the local and remote system names respectively.
–rp*N*	Sets the remote serial link port to port *N*. Set *N* = 1 for COM1; *N* = 2 for COM2, *N*=3 for COM3, and *N*=4 for COM4.
–rs*N*	Sets the remote serial link speed to the value associated with *N*. Set *N*=1 for 9,600 baud, *N*=2 for 19,200 baud, *N*=3 for 38,400 baud, and *N*=4 for 115,000 baud.

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.

–sd*DIR* Sets one or more source directories to scan for source files; the syntax is: **Note:** There shouldn't be a 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 separate directory, or list them together like this

```
sddir1;dir2;dir3
```

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. This option requires extra memory but lets you 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.

Appendix

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 settings using command-line options when you load Turbo Profiler. 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, screen lines, tab column width, fast screen update, 43/50-line mode, full graphics saving, and user-screen updating
- 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, interrupt key, mouse, and control-key shortcuts
- NMI intercept 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 ***Up*** and ***Down arrow*** 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.

Choose Quit (or ***Alt+X***) from the menu 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.

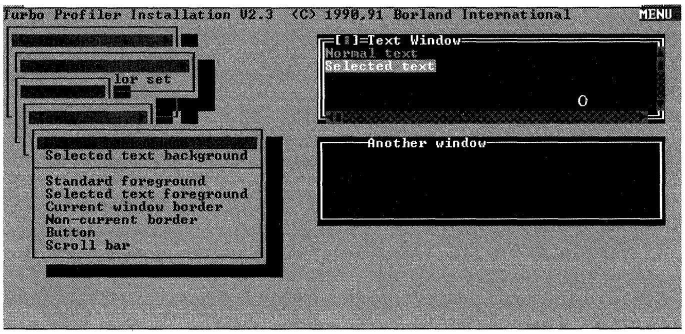
Customizing screen colors

If you choose Customize, a third menu appears, with options for customizing Windows, Dialogs, Menus, and Screens.

Windows

To customize windows, choose the Windows command. This command opens a fourth menu, from which you can choose the kind of window you want to customize: Text, Statistics, or 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.

Note Turbo Profiler maintains three color tables: one for color, one for black and white, and one for monochrome. You can change only 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 Dialogs or Menus from the Customize 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 Windows menu, choosing an item from the current menu opens a palette from which you can choose the color for that item.

Screen

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.

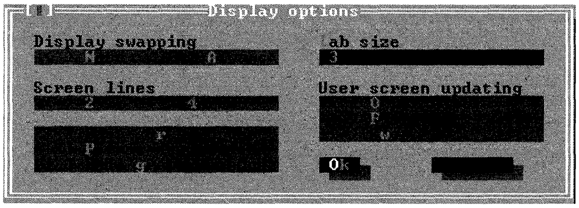
The default colors

If you choose Default Color Set from the Colors menu, facsimiles of an active text window and an inactive window appear onscreen. The facsimiles 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



Note 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 110 for a table of Turbo Profiler command-line options and corresponding TFINST settings.

Display Swapping

You use the Display Swapping radio buttons to control whether 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.

Note Only EGA and VGA monitors 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, which 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 that the user screen might show a small number of corrupted locations that usually don't 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's 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) and want to keep the current Turbo Profiler screen as well.

Turbo Profiler options

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:

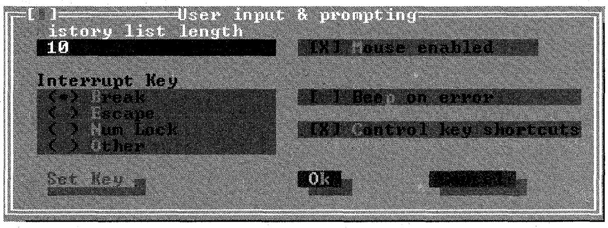


Figure B.3 The User Input and Prompting dialog box

```
[ ]──────────User input & prompting─────────────
 istory list length
 10                          [X] ouse enabled

Interrupt Key
 (•) reak                    [ ] Bee  on error
 ( ) scape
 ( ) um Lock                 [X] ontrol key shortcuts
 ( ) ther

 Set Key                     Ok
```

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.

Interrupt Key

The Interrupt Key radio buttons let you change the Turbo Profiler interrupt key from its default of ***Break*** to ***Escape, NumLock,*** or another key or key combination. Pushing the Other radio button displays a prompt asking you to press the key or key combination you want to use as the interrupt key.

Mouse Enabled

This check box controls whether Turbo Profiler defaults to mouse support.

Beep on Error

Turbo Profiler can give a warning beep when you press an invalid key or do something that generates an error message. Checking the Beep on Error check box enables the warning beep.

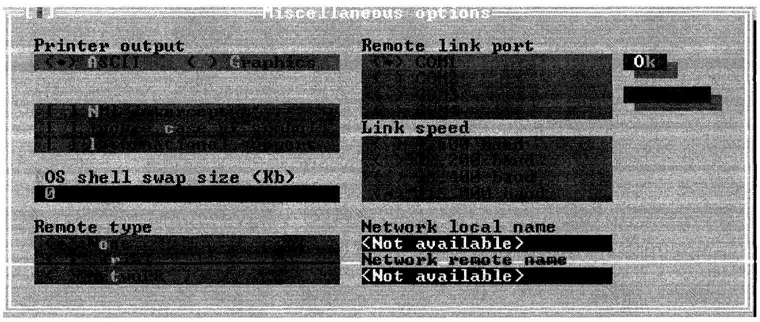
Control Key Shortcuts

This check box enables or disables the control-key shortcuts. When control-key shortcuts are enabled, you can invoke any SpeedMenu 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



```
Miscellaneous options
Printer output                  Remote link port
(•) ASCII   ( ) Graphics        (•) COM1

OS shell swap size (Kb)         Link speed
0

Remote type                     Network local name
                                <Not available>
                                Network remote name
                                <Not available>
```

Printer Output

This option lets you toggle whether to print both extended and standard ASCII characters, or just the straight ASCII character set.

NMI Intercept

The nonmaskable interrupt (NMI) is a hardware interrupt that the processor must deal with immediately. It is typically used to halt processing when there is a memory parity error: an error message like "Memory Parity Error" is displayed and the system hangs.

Another use for this interrupt is to enable a debugger board to perform a breakout when you press the breakout button. Because the NMI defaults to OFF with Turbo Profiler, you will probably want to turn this interrupt on if you use a debugger board.

If your computer is not a Tandy 1000, IBM PC compatible, ACER 1100, or NEC MultiSpeed, you can run TFINST and try turning on the NMI Intercept check box. Some computers use the NMI in ways that conflict with Turbo Profiler, so if you have problems loading in applications under Turbo Profiler after turning this option on, run TFINST again and disable Turbo Profiler's use of this interrupt.

Ignore Case of Symbols

If this check box is turned on, Turbo Profiler will ignore the difference between uppercase and lowercase. If it is not checked, case sensitivity will be in effect.

International support

If this check box is checked, Turbo Profiler will sort all items in list boxes according to the *country* setting in your CONFIG.SYS file (when using DOS), or according to the country checked in the Windows Control Panel (when using Windows). For more information on setting the country code, refer to your DOS or Windows User's Guide.

If the box is not checked, Turbo Profiler will sort entries in list boxes according to the ASCII values of the items in the box (when using DOS), or according to the ANSI values of the items in the box (when using Windows).

DOS Shell Swap Size (Kb)

In this input box you can set the number of kilobytes of memory to be swapped out for the DOS shell. Memory swapping allows you to use the File | DOS Shell command even when a large program is loaded.

DOS Shell Swap Size is not available in TPROFW.

Remote type

The Remote Type radio buttons let you specify the type of remote profile link. None, the default mode, specifies that profiling is local; there is no remote link. The Serial button enables remote serial profiling. The communication port and link speed are defined by Remote Link Port and Link Speed options. The Network button specifies remote LAN profiling.

Warning! Usually you won't want to save a configuration file that specifies remote profiling, since Turbo Profiler will then look for the remote link each time it's loaded.

Remote Link Port

The Remote Link Port radio buttons let you choose either the COM1, COM2, COM3, and COM4 serial ports for the remote serial link.

Link Speed

The Link Speed radio buttons let you choose one of the four speeds that are available for the remote serial link: 9,600 baud, 19,200 baud, 38,400 baud, or 115,000 baud.

Network local name

This text box lets you define the default name of the local system when using remote LAN profiling. By default, the name *LOCAL* is given to the local system. This name should be changed if more than one person is using the network for remote profiling.

Network remote name

This text box lets you define the default name of the remote system when using remote LAN profiling. By default, the name *REMOTE* is given to the local system. This name should be changed if more than one person is using the network for remote profiling.

Setting the mode for display

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 automatically 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 only if you are using a color monitor with a Hercules or monochrome text display adapter.

LCD
Choosing this instead of Black and White if you have an LCD monitor makes your display much easier to read.

When you're through...

After configuring and customizing the way Turbo Profiler looks and behaves, you'll want to save the settings out to disk. You can either modify the Turbo Profiler executable program directly (TPROF.EXE), or you can create a configuration file that gets loaded as you load Turbo Profiler.

Saving changes

When you have all your Turbo Profiler options set the way you want, choose Save from the main menu to determine how you want them saved.

Save Configuration File
If you choose Save 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 Options | Restore Configuration command to load a configuration once you have started Turbo Profiler.

Modify TPROF.EXE
If you choose Modify TPROF.EXE, any changes you've 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.

Note 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.

Command-line options and TFINST equivalents

Some of the options described above 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–	(•) `None`
–r	(•) `Serial`
–rn	(•) `Network`
	Options | Miscellaneous | Miscellaneous Options
–rp1	(•) `COM1`
–rp2	(•) `COM2`
	Options | Miscellaneous | Miscellaneous Options
–rs1	(•) `9,600 baud`
–rs2	(•) `19,200 baud`
–rs3	(•) `38,400 baud`
–rs4	(•) `115,000 baud`
	Options | Miscellaneous | Miscellaneous Options
–sc	[X] `Ignore Case of Symbol`
–sc–	[ ] `Ignore Case of Symbol`
	Options | Directories | Directories
–sd	`Source Directories`
	Display | Display Options
–vn	[ ] `Permit 43/50 Lines`
–vn–	[X] `Permit 43/50 Lines`

TFINST.EXE uses the following syntax:

```
TFINST [options] [exefile]
```

Items enclosed in brackets are optional. For a list of options, see the following table.

Table B.2 TFINST.EXE options

Option	Description
–c*file*	Use configuration file *file*.
–h, –?	Display help screen.
–p	Enable mouse support.
–w	Configure TPROFW.EXE.

Appendix

C

Remote Profiling

Remote profiling is just what it sounds like: You run Turbo Profiler on one computer and run the program you're profiling on another. The systems can be connected by either the serial ports of the systems or through a NETBIOS-compatible local area network (LAN).

Remote profiling is useful in several situations:

- When your program needs a lot of memory, and you can't run both the program and Turbo Profiler on the same computer. When this happens, you'll get `Not enough memory` error messages.
- When your program loads under Turbo Profiler, but there's not enough memory left for it to operate properly. Here, you'll get memory allocation errors during the profiling session.
- When you are profiling a Windows program.

If you're profiling a Windows application, you have the choice of running Turbo Profiler for Windows (TPROFW) and the application on a single machine, or of running Windows, WREMOTE, and the application on one machine and running Turbo Profiler (TPROF) on another.

Although there are many reasons why you'll want to profile a program using two systems, the advantages become even greater when you are developing a Windows application:

- If you have a single monitor, running Turbo Profiler and the application on the same machine means that you must switch between Turbo Profiler's character mode screens and the application's graphics mode screens.

  If you use remote profiling, you can see the application's screens and Turbo Profiler's screens at the same time. (This same result can be achieved if you have two monitors attached to the same system.)

- TFREMOTE and WREMOTE use far less memory than Turbo Profiler, so the program you're profiling will behave more like it does when running normally, without the profiler in the background.

Hardware and software requirements

You choose between a serial or a LAN connection for the remote session. The two setups do use different hardware; however, both share the following requirements:

- A development system with enough memory to load TPROF (this is the *local* system).
- Another PC with enough memory and disk space to hold either TFREMOTE and the DOS program you want to profile or WREMOTE, Windows, and the Windows program you want to profile (this is the *remote* system).

  If you're going to profile a Windows application, the remote machine must be able to run in protected mode, which means that the CPU must be at least an 80286. The amount of memory required depends on the mode in which you're running Windows, but must be at least 1MB.

For a serial connection, you'll need a null modem cable to connect the serial ports of the two systems. Make sure 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).

For a LAN connection, you'll need a LAN running Novell Netware-compatible software (IPX and NETBIOS version 3.0 or later).

Note NETBIOS must be loaded onto *both* the local and remote systems before TPROF, TFREMOTE, TFREMOTE, or WREMOTE can be loaded. This is true for both DOS and Windows profiling.

Profiling remote DOS applications

To profile a remote DOS application, you must run TFREMOTE and the application on one machine and Turbo Profiler on another. In this discussion, the machine running TFREMOTE and the application is called the *remote* machine, and the machine running Turbo Profiler is called the *local* machine.

Setting up the remote system

Copy the remote profiling driver TFREMOTE.EXE onto the remote system, as well as any files required by the program you're profiling. These files can be data input files, configuration files, help files, and so on. If you want, you can also copy your application program onto the remote system. However, Turbo Profiler automatically sends it over the remote link if necessary.

To put files on the remote system, you can use floppy disks or the TDRF remote file transfer utility.

Configuring TFREMOTE

When starting TFREMOTE, you must configure it so it can communicate over the remote link. You do this by starting the driver with specific command-line options. Start an option with either a hyphen (-) or a slash (/).

Table C.1 TFREMOTE command-line options

Option	What it does
-? or **-h**	Displays a help screen
-rn<*remotename*>	Remote LAN profiling
-rp1	Port 1, (COM1); default
-rp2	Port 2, (COM2)
-rp3	Port 3, (COM3)
-rp4	Port 4, (COM4)
-rs1	Slowest speed (9,600 baud)
-rs2	Slow speed (19,200 baud)
-rs3	Medium speed (38,400 baud)
-rs4	High speed (115,000 baud); default
-w	Writes options to the executable program file

Note For a list of all available TFREMOTE command-line options, type the following at the remote DOS prompt:

```
TFREMOTE -h
```

Customizing TFREMOTE

If TFREMOTE is started without command-line options, it assumes remote serial profiling at the default port and speed built into TFREMOTE.EXE (COM1 and 115,000 baud respectively, unless you've changed them with the -**w** option).

You can make TFREMOTE's command-line options permanent by writing them back to disk. To do this, specify -**w** on the command-line along with the other options you want to make permanent. TFREMOTE then prompts for the name of the executable file to write to; if you enter a nonexistent executable file name, TFREMOTE creates the file. If you press ***Enter***, the currently running program (usually TFREMOTE.EXE) is overwritten.

If you are running 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 2.xx, you must supply the full path and file name of the executable program.)

For example, on the remote system, type the following command line at the DOS prompt:

```
TFREMOTE -w -rs3 -rp2
```

When prompted, enter the name of the program to modify, for instance, *tfremot2.exe*. With this, TFREMOTE creates a new remote driver named TFREMOT2.EXE, where the default speed is 38,400 baud (**-rs3**) and the default port is COM2 (-**rp2**).

The remote DOS driver

To begin a remote profiling session, you must first start the driver on the remote system and then load TPROF on the local system.

Before starting TFREMOTE, be sure the directory on the remote system is set to the one that contains the program files. This is essential because TFREMOTE puts the program to be profiled into the directory that is current when you start TPROF. You don't give the program name on the TFREMOTE command line, since TPROF controls the loading of the program.

When loaded, TFREMOTE signs on with a copyright message, then indicates that it's waiting for you to start Turbo Profiler at the other end of the link. To stop and return to DOS, press *Ctrl+Break*.

Starting the remote serial driver

If you're using a null modem cable to connect the two systems, you must use the command-line options **–rs** and **–rp** to indicate the speed and port of the data communications.

If the remote system's serial port is set up as COM1, type

```
TFREMOTE -rp1 -rs4
```

Note that this is the default setting of TFREMOTE, and is the same as issuing the command

```
TFREMOTE
```

if the default settings haven't been changed. If the remote system's serial port is set up as COM2, type

```
TFREMOTE -rp2 -rs4
```

to start TFREMOTE.

If you're using a PS/2, use the command-line option **–rs1**.

All three of these commands start the link at its maximum speed (115,000 baud). This speed works with most PCs and cable setups. However, if you experience communication difficulties, see Table C.1 on page 115 for how to start the link at a slower speed.

Note It's possible that the local and remote systems use different serial ports for the null modem cable connection. In this case, the two systems' serial port settings will not match. However, the communication speed of the two systems must always be the same for the connection to work.

Starting the remote LAN driver

If you're using a LAN to connect the two systems, you must use the **–rn** command-line option to start TFREMOTE. For example, issuing the following command at the DOS prompt will start TFREMOTE over a LAN connection, naming the remote system *remotelink*:

```
TFREMOTE -rnremotelink
```

If a remote name is left out of the command, the default name *REMOTE* is used.

For more on naming remote systems, see the "LAN connection" section that follows.

Establishing the remote DOS link

Once TFREMOTE has been loaded on the remote system, start Turbo Profiler on the local system using command-line options that correspond to the established data link (serial or LAN).

Serial connection

TPROF and TFREMOTE use the same syntax for specifying the speed and port settings for remote serial communications. For the link to work properly, you must set both systems to the same speed (with the **–rs** option).

After loading TFREMOTE on the remote system, run TPROF on the local system to complete the remote link. The following DOS command will load Turbo Profiler, establishing a connection through serial port 2, at the default speed of 115,000 baud:

```
TPROF -rp2 filename
```

When the link is successful, the message `Link established` appears on the remote system, and the activity indicator on the local system displays `READY`. Turbo Profiler's display then appears on the local system.

If the program *filename* is not on the remote system, then Turbo Profiler will send the program over the remote link. For more about loading programs over the link, refer to"Loading programs onto the remote system" on page 121.

Instead of using **–rs** and **–rp**, you can use the **–r** command-line option, which starts the remote serial link using the default speed and serial port. Unless you've changed the defaults using TFINST, **–r** specifies COM1 at 115,000 baud.

LAN connection

TPROF uses the **–rn** command-line option to initiate a remote LAN link. However, the syntax used with Turbo Profiler is slightly different from that used with TFREMOTE. Following is the Turbo Profiler remote LAN syntax:

```
TPROF -rn [Local][;Remote] filename
```

The **–rn** command-line option takes two optional parameters: the local system name and the remote system name, separated by a semicolon. Since both parameters are optional, there are four ways to use the **–rn** command-line option with TPROF. These DOS commands all load Turbo Profiler, specify a remote LAN connection, and load the program **filename** for profiling.

```
TPROF -rn filename
```

Uses the default names *Local* and *Remote* for both the local and remote systems.

```
TPROF -rnLOCAL1 filename
```

Specifies LOCAL1 as the local system name, but uses the default (*Remote*) for the remote system name.

```
TPROF -rn;REMOTE1 filename
```

Uses the default (*LOCAL*) for the local system name, but specifies *REMOTE1* for the remote system.

```
TPROF -rnLOCAL1;REMOTE1 filename
```

Specifies both system names.The handshake should take less than 15 seconds after you enter the TPROF command.

Local and remote system names can be up to 16 characters long.

Note If only one person on a network is using remote profiling, then it isn't necessary to define special local and remote system names. However, if more than one person uses remote profiling on a given network, unique names must be given to all systems.

Profiling remote Windows applications

To profile a remote Windows application, you must run Windows, WREMOTE, and the application on one machine and Turbo Profiler on another. In this discussion, the machine running Windows, WREMOTE, and the application is called the *remote* system, and the machine running Turbo Profiler is called the *local* system.

Setting up the remote system

Copy to the remote system the remote profiling driver WREMOTE.EXE, the configuration program WRSETUP.EXE, and any files required by the program you're profiling. These files include data input files, configuration files, help files, Windows DLL files, and so on. If you want, you can also copy your application program onto the remote system. However, Turbo Profiler automatically sends it over the remote link if necessary.

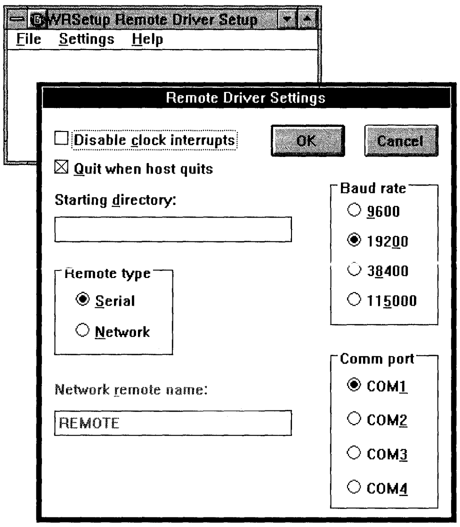
To put files on the remote system, you can use floppy disks or the TDRF remote file transfer utility. TDRF is described in the online text files.

Configuring WREMOTE

Before running WREMOTE for the first time, you should run the WRSETUP program to establish the communication settings.

Set up WREMOTE with WRSETUP. When you run WRSETUP, you see a window displaying the commands File, Settings, and Help. Choosing Settings displays the following screen:

Figure C.1 WRSETUP main window and Settings dialog box



☐ Disable clock interrupts
☒ Quit when host quits

Starting directory:

Remote type: [✓ Serial] Network

Network remote name: REMOTE

Baud rate: 9600 [✓ 19200] 38400 115000

Comm port: [✓ COM1] COM2 COM3 COM4

Serial configuration

If you're using serial communications, select the Serial radio button, and set a baud rate and communications port that works for your hardware setup. The defaults are 19,200 baud and COM1.

LAN configuration

If you're using LAN communications, select the Network radio button, and specify the desired remote system name in the Network Remote Name text box. By default, the remote system name is *REMOTE*. For more on remote system names, refer to "LAN connection" on page 117.

In the Starting Directory text entry box, enter the directory path where Turbo Profiler should look for the program you're profiling. If you want WREMOTE to return control to Windows when you terminate Turbo Profiler on the local machine, select Quit When Host Quits. If you are using the higher transmission speeds (38,400 or 115,000 baud) check the Disable Clock Interrupts box. This will help WREMOTE and Turbo Profiler establish a connection in the Windows environment.

As with any *.INI* file, you can edit the file directly using any word processor that produces ASCII text.

Once you've set your options and closed the WRSETUP window, WRSETUP will save your settings to the file TDW.INI in your Windows directory. The following TDW.INI file sets WREMOTE at 19,200 baud on COM2 with clock interrupts disabled and the program returning control to Windows when Turbo Profiler terminates:

```
[WRemote]
BaudRate=2
Port=2
```

```
Quit=1
Clock=1
```

WREMOTE command-line options

You can use WREMOTE command-line options to override the default settings or the settings listed in the WREMOTE.INI file. Start an option with either a hyphen (–) or a slash (/).

Table C.2 WREMOTE command-line options

Option	What it does
–c<*filename*>	Use <*filename*> as the configuration (.INI) file
–d<*dir*>	Use <*dir*> as the startup directory
–rc0	Clock interrupts enabled
–rc1	Clock interrupts disabled
–rn<*remotename*>	Remote LAN profiling
–rp1	Port 1 (COM1); default
–rp2	Port 2 (COM2)
–rp3	Port 3 (COM3)
–rp4	Port 4 (COM4)
–rq0	Don't quit when Turbo Profiler quits
–rq1	Quit when Turbo Profiler quits
–rs1	Slowest speed (9,600 baud)
–rs2	Slow speed (19,200 baud)
–rs3	Medium speed (38,400 baud)
–rs4	Fast speed (115,000 baud); default

Starting the remote Windows driver

After you start WREMOTE from Windows, the program displays an hourglass at the mouse cursor location, indicating that it's ready for you to start Turbo Profiler at the other end of the link.

To terminate WREMOTE while it is waiting to establish a connection with TPROF, press *Ctrl+Break* on the remote machine.

Establishing the remote Windows link

If you're using a null modem cable to connect the two systems, you may use the command-line options **–rs** and **–rp** to indicate the speed and port of the data communications.

For more on command-line options, see "Serial connection" on page 117.

Both Turbo Profiler and WREMOTE must be set to the same speed to work properly. You can use the **–rs** parameter to set the baud rate for Turbo Profiler, or you can use the **–r** command-line option, which starts the remote serial link using the default speed and serial port. Unless you've changed the defaults using TFINST, **–r** specifies COM1 at 115,000 baud.

LAN connection

TPROF uses the **-rn** command-line option to initiate a remote LAN link. For more on **-rn**, see "LAN connection" on page 117.

Here's a typical Turbo Profiler command to start the remote Windows link:

```
TPROF -rs2 myprog
```

This command begins the link on the default serial port (usually COM1) at the link speed (19,200 baud), and loads the program *myprog* into the remote system if it's not already there.

When Turbo Profiler starts on the local machine, it displays copyright and version information and the following message:

```
Waiting for handshake from remote driver (Ctrl+Break to quit)
```

While waiting for a connection, an hourglass is displayed on the remote system. Turbo Profiler's normal window display comes up on the local machine. Press ***Ctrl+Break*** to exit WREMOTE if the link is not successful.

Loading programs onto the remote system

If a file name is included as a TPROF command-line argument, or if you load a new file into the profiler using the File | Open command, Turbo Profiler will automatically check to see if the program needs to be sent to the remote system.

Windows DLL files are not automatically transferred to the remote system.

Turbo Profiler is smart about loading programs onto the remote system. First, a check is made to see if the program exists on the remote system. If the program doesn't exist on the remote system, it's sent over the link right away. If the program does exist on the remote system, Turbo Profiler looks at the date and time of the program on the local system and compares this with the copy on the remote system. If the program on the local system is later (newer) than the remote copy, Turbo Profiler assumes you've recompiled or relinked the program, and sends it over the link.

At the highest link speed, file transfers move at a rate of about 10K per second. A typical 60K program takes roughly six seconds to transfer. On DOS systems, Turbo Profiler indicates that the system is working by displaying the number of bytes transferred on the remote system.

Remote profiling sessions

Once you start TPROF in remote mode (using either TFREMOTE or WREMOTE), the Turbo Profiler commands work exactly the same as they do 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.

Troubleshooting

Here's a list of troubleshooting techniques you can try if you experience problems with the remote setup:

- Check your cable hookups.
- Check that you're using the correct serial port settings or that you're properly connected to the network.
- Try running the link at a slower speed (using the **–rs** command-line option) until you find a speed that works.
- Some hardware and cable combinations don't always work properly at the highest speed, so if you can get the link to work only at a lower speed, you might want to try a different cable or different computers.
- If you're profiling a Windows program and can't get the connection to work at any speed, use WRSETUP to *Disable clock interrupts* and try running the link at 9,600 baud. If that works, try successively higher speeds.

TFREMOTE messages

***nn* bytes downloaded**
TFREMOTE is receiving a file from the local system. This message shows the progress of the file transfer. At the highest link speed (115,000 baud), transfer speed is about 10K 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 modify a copy of the TFREMOTE utility only with the **–w** option.

Can't open .exe file to modify
TFREMOTE can't open the file name you specified. You've probably entered an invalid 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.

No network present
TFREMOTE is unable to detect a NETBIOS compatible network. Make sure you have loaded NETBIOS (version 3.0 or greater) and are connected to the network.

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 to force Turbo Profiler to send a new copy over the link. If this message appears 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.

Program load failed; program not found
TFREMOTE could not find the program on its disk.

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*.

WREMOTE messages

Can't find configuration file
You used the **–c** command-line option to specify a file that doesn't exist.

Can't load WINDEBUG.DLL
The dynamic link library WINDEBUG.DLL isn't in the current directory. WREMOTE requires this DLL in order to run.

Can't open COMx serial port
WREMOTE is trying to use a COM port that is either in use or doesn't exist.

Invalid switch
You specified an unknown option on the WREMOTE command line.

No network present
WREMOTE is unable to detect a NETBIOS compatible network. Make sure you've loaded NETBIOS (version 3.0 or greater) and are connected to the network.

Appendix

D

Turbo Profiler for Windows

Turbo Profiler for Windows (TPROFW) lets you profile applications you've written for Microsoft Windows, version 3.0 and higher. It runs under Windows on the same machine as the program you are profiling and switches between its own screens and your application's screens, just as Turbo Profiler does.

You profile in Windows much as you would in DOS, except that you can also access information particular to Windows applications, such as

- Messages received and sent by your application's windows
- The complete list of modules loaded by Windows (including dynamic-link libraries)
- Dynamic-link library (DLL) profiling

TPROFW runs in Windows standard mode or 386 enhanced mode, which means that your computer must have an 80286 processor or higher and at least 1MB of memory.

TPROFW.EXE supports several differnt video adapters through the use of several DLLs. After you've installed TPROFW, run TDWINI.EXE to help you select or modify the driver that's used with your setup.

By default, TPROFW uses the SVGA.DLL video driver, which supports most video adapters and monitors. For more information on the available video DLLs, refer tho the entries for DUAL8514.DLL, STB.DLL, SVGA.DLL, and TDWGUI.DLL in the online Help system of TDWINI.EXE.

Like Turbo Profiler, TPROFW can also take advantage of a second monitor attached to your computer, allowing you to view TPROFW screens on one monitor and your application's screens on another. You select this display option by starting TPROFW with the **-do** command-line switch or by running the TFINST utility and setting User Screen Updating to *Other display*.

Installing TPROFW

When you install Turbo Profiler on your system, the installation program puts the following two Windows-related files in the same directory as your Turbo Profiler files:

- TPROFW.EXE, the TPROFW program
- TFWHELP.TFH, the TPROFW help files

The installation process creates an icon for TPROFW and installs it in the Windows Program Manager group for your Borland language. You can run TPROFW by choosing the icon, just as you can with any other Windows application.

Installing TDDEBUG.386

The TDDEBUG.386 file on your installation disks provides the same functionality as the Windows SDK file WINDEBUG.386. In addition, it provides better support than WINDEBUG.386 for the ***Ctrl+Alt+SysRq*** key combination (used to break out of a Windows application and return to TPROFW).

The installation program should copy this file to your hard disk and alter your Windows SYSTEM.INI file so that Windows loads TDDEBUG.386 instead of WINDEBUG.386. If, for some reason, the installation program can't complete this task, you'll have to do it by hand as follows:

1. The installation program will have copied TDDEBUG.386 from the installation disks to your hard disk. The standard directory for this file is C:\WINDOWS. If you move the file to another directory, substitute that directory in the instructions.
2. With an editor, open the Windows SYSTEM.INI file, search for *[386enh]*, and add the following line to the 386enh section:

   ```
   device=c:\windows\ddebug.386
   ```

3. If there's a line in the 386enh section that loads WINDEBUG.386, either comment the line out with a semicolon or delete it altogether. (You can't have both TDDEBUG.386 and WINDEBUG.386 loaded at the same time.)

   For example, if you load WINDEBUG.386 from the C:\WINDOWS directory, the commented-out line would be

   ```
   ;device=c:\windows\windebug.386
   ```

Configuring TPROFW

Just as with Turbo Profiler, you can configure TPROFW two ways: by entering command-line options or by using the TFINST utility.

Using TPROFW command-line options

You can set the configuration of TPROFW by using various command-line options followed by an optional program name with its own command-line options. The program name can be preceded by a path name.

Because TPROFW is a Windows program, you will probably enter any command-line options either by using the Program Manager's File | Run command or by using the Program Manager's File | Properties command to change the command-line property of the TPROFW icon. You can also start Windows and TPROFW from the DOS command line. Follow the Windows command with the TPROFW command, optionally followed by switches or a program name (with or without switches).

The command-line syntax for TPROFW is

```
TPROFW [options] [program-name [program-args]]
```

Table D.1 provides a summary of the command-line options for TPROFW:

Table D.1 TPROFW command-line options

Option	What it does
–?,–h	Displays help on TPROFW command-line options.
–b	Uses batch-mode profiling.
–b*count*	Use batch-mode profiling, and run the progrm *count* number of times.
–c*filename*	Uses configuration file *filename*.
–do	Runs TPROFW on the secondary display.
–ds	Updates screens by swapping pages.
–p	Uses a mouse. If the mouse driver is disabled for Windows, it will be disabled for TPROFW as well, and the **–p** command-line option will have no effect.
–sc	Ignores case for symbol names.
–sd*dir*[;*dir*...]	Sets one or more source file directories.
–t*dir*	Sets the starting directory.

See Appendix A for a complete description of the command-line options.

Note The command-line option **–t** is available only with TPROFW. This option changes TPROFW's starting directory, which is where TPROFW looks for the configuration file and for .EXE files not specified with a full path. The syntax is

```
-tdirname
```

You can set only one starting directory with this option. If you enter this command more than once on the same command line, TPROFW uses only the last entry.

Using TFINST with TPROFW

To use TFINST with TPROFW, start TFINST using the **–w** command-line option. TFINST for TPROFW works just like TFINST for Turbo Profiler, except that the default configuration file is TFCONFIG.TFW and fewer options are available. (See the list of TPROFW command-line options in the previous section.)

For a description of how to use TFINST, see Appendix B.

Using TPROFW

When you load TPROFW, it comes up in full-screen DOS character mode, not in a DLL (unless you're using the TDWGUI.DLL video driver). Unlike other applications that run under Windows, you can't use the Windows shortcut keys (like *Alt+Esc* or *Ctrl+Esc*) to switch out of TPROFW and run another application. However, if the application you are profiling is active (the cursor is active in one of its windows), you can use these keys or the mouse to switch to other programs.

Note If you do use *Ctrl+Esc* to switch out of an application running under TPROFW, you see the application name on the list of tasks. You will never see TPROFW on the task list because TPROFW is not a normal Windows task that you can switch into or out of.

Profiling using TPROFW is pretty much the same as profiling using Turbo Profiler. However, there are some differences:

- Switching from your application to TPROFW is accomplished by using the *Ctrl+Alt+SysRq* key combination. This operation is similar to using *Ctrl+Break* to switch out of a DOS application and back to Turbo Profiler, except that the DOS application terminates, while the Windows application is only suspended.
- If possible, run your application to completion or use the System command to exit it before exiting TPROFW or loading in another program to be profiled. Failing to exit a Windows application properly can leave resources allocated that would otherwise have been deallocated, potentially causing problems with TPROFW or other applications.
- The DOS Shell command from the File menu is not available.
- The Edit command on the Module and Text File SpeedMenus are not available.
- Interrupts, File I/O, and Overlay windows are not available.
- Display Swapping settings are not available in the Display Options dialog box.

Profiling window procedures

TPROFW keeps track of routines inside window procedures by tracking the *message classes* called by the routine and *window messages* sent to the procedures. Before message classes and window messages can be tracked, you must first specify that an area marker represents a window procedure. Specifying a window procedure area marker is done from either the Module window's SpeedMenu Operation command, or the Areas window SpeedMenu Option command.

The Window Procedure Messages dialog box

After an area marker is specified as a window procedure marker, you must then select which messages and classes you want to track for that particular procedure. When *Messages* is selected from either the Module window's SpeedMenu Operation command or the Areas window SpeedMenu Option command, the Window Procedure Messages dialog box is displayed.

With the Window Procedure Messages dialog box, you can select the window messages and message classes that are tracked for the current area marker. TPROFW, by default, tracks all message classes.

The Window Procedure Messages dialog box uses check boxes and text boxes for the following:

- The Window Messages list box displays specially selected window messages.
- Message Name is a text box that accepts a window message name or window message number. Use the Add command to append the specified message to the Window Messages list.

  Turbo Profiler will recognize only window message names that begin with *WM_*. If you wish to track a message other than a *WM_* message, you must provide the window message number (window message numbers are acquired from either your program source files or from the windows header include files).

  Window message names are case sensitive.
- Delete removes the currently highlighted message from the Window Messages list box.
- Remove All deletes all specially selected messages from the Window Messages list box. When this command is selected, the Window Messages list box will be cleared of all entries.
- Add All selects all *WM_* messages from all classes. Each message is listed in the Window Messages list box after this command is chosen. Because so many messages come through, you'll probably want to narrow the focus by selecting only the classes of interest from the list of message names.

  Note Before selecting Add All, ensure the Max Windows Messages setting (Statistics | Profiling Options) has been adjusted to accommodate the number of messages you want to track.

  Including all message classes, there are over 140 *WM_* messages.
- Add appends the message name specified in the Message Name text box to the Window Messages list.
- The Message Classes check boxes allow you to choose specific classes of messages to watch. When a specific Windows message class is selected, all *WM_* messages from that class will be tracked.

Table D.2 describes the window message classes:

Table D.2 Window's message classes

Message class	Description
All Messages	All messages starting with WM_.
Mouse	Messages generated by a mouse event (for example, WM_LBUTTONDOWN and WM_MOUSEMOVE).
Window	Messages from the window manager (for example, WM_PAINT and WM_CREATE).
Input	Messages generated by a keyboard event or by the user's accessing a System menu, scroll bar, or size box (for example, WM_KEYDOWN).
System	Messages generated by a system-wide change, (for example, WM_FONTCHANGE and WM_SPOOLERSTATUS).
Initialization	Messages generated when an application creates a dialog box or a window (for example, WM_INITDIALOG and WM_INITMENU).
Clipboard	Messages generated when one application tries to access the clipboard of a window in another application (for example, WM_DRAWCLIPBOARD and WM_SIZECLIPBOARD).
DDE	Dynamic Data Exchange messages, generated by applications' communicating with one another's windows (for example, WM_DDE_INITIATE and WM_DDE_ACK).
Non-client	Messages generated by Windows to maintain the non-client area of an application window (for example, WM_NCHITTEST and WM_NCCREATE).
Other	Any messages starting with WM_ that don't fall into any of the other categories, such as owner draw control messages and multiple document interface messages.

For a complete list of all WM_ messages, refer to your Borland compiler's online Help.

Note When you've selected the appropriate window messages, profile your program as usual. To view window message statistics, choose Window Procedures from the Display Options dialog box, accessed through the Execution Profile SpeedMenu.

Profiling dynamic-link libraries (DLLs)

A *DLL* (dynamic-linked library) is a Windows library of routines and resources that is linked to your application at run time instead of at compile time. This run-time linking allows multiple applications to share a single copy of routines, data, or device drivers, thus saving on memory usage. When an application that uses a DLL starts up, Windows loads it in memory so the application can access the DLL's entry points (if the DLL isn't already loaded into memory).

TPROFW can load a DLL that doesn't have a symbol table, but only into a CPU window.

When you load an application with DLLs linked to it, TPROFW determines which of these DLLs, if any, have symbol tables (were compiled with the debugging option turned on) and tracks them for you.

TPROFW automatically loads in the symbol table and source of every DLL that's linked to your application, but only if the DLL has a compatible symbol table. A DLL has a symbol table compatible with TPROFW if it was compiled with debugging information turned on and the compiler was one of Borland's C++ Windows compilers, or Turbo Assembler.

Note DLLs that are loaded via the LoadLibrary call will not be automatically tracked by Turbo Profiler. To track these DLLs, you must set a Stop area marker after the LoadLibrary call. When the profiler encounters this area marker, you'll be able to access the DLL through the Module command on the Module window's SpeedMenu, or through View | Module. You'll need to set up a stop area on the last line of the DLL function in order to view the DLL's statistics. If you don't set the stop area, you will not be able to view or analyze the statistics.

Because the symbol table for the DLL is not associated with the symbol table for the executable program, Turbo Profiler will produce a separate statistics file (.TFS) and a separate areas file (.TFA) for *each* DLL profiled.

TPROFW error messages

There is only one error message returned solely by TPROFW. In addition to this error message, Turbo Profiler error messages can also be returned.

Ctrl+Alt+SysRq interrupt. System crash possible. Continue?

You attempted either to exit TPROFW or to reload your application program while the program was suspended as a result of your having pressed *Ctrl+Alt+SysRq*. Because Windows kernel code was executing at the time you suspended the application, exiting TPROFW or reloading the application will have unpredictable results (most likely hanging the system and forcing a reboot).

Appendix

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 of 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 box 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 2.

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 Disassembly pane. The Disassembly 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 from

Enter the name of the file to restore the statistics from. 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 .TFS 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 interrupt number

Enter the number of the interrupt that you wish to track.

Enter name of file to view

Enter the name of a text file that you want to inspect. The file specified will be brought into the File window.

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 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 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.

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 interrupt

Pick an interrupt from the list of interrupts built into Turbo Profiler.

Pick macro to delete

Pick a macro to erase from the list of defined macros.

Reload program and change profiling mode?

When you change the profiling mode from active or passive to coverage, or from coverage to active or passive, all current profile statistics will be erased. If you wish to save the statistics before changing modes, answer NO to this prompt and save the statistics to a .TFS file. Otherwise, answer YES to the prompt.

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 can be accompanied by a beep. You can turn the beep on or off 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 or missing 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 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 find *filename.DLL*

You attempted to load a program that requires one or more DLLs, but TPROFW can't find one of them. Make sure your executable file and the DLLs it requires are in the same directory, then load the program again.

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.

Error loading program

You program could not be loaded. The format of the .EXE does not match the operating system.

Error printing statistics

There was an error sending to the printer. Check that the printer is online and not out of paper.

Edit program not specified

You tried to use the Edit SpeedMenu 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 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 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.

Exception *N*, error code *N*

Turbo Profiler encountered either an invalid memory reference or an invalid instruction in your program. You must correct ther error before continuing profiling.

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 number entered

The line number you specified is either a negative number, or contains an alphabetic character. Make sure you specify a positive integer as a line number.

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.

Invalid switch

An invalid command-line option was encountered during program loading.

Invalid window message number

The window message that you have specified is not a valid name or number. Make sure that the message name is correctly spelled (window message names are case sensitive).

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 SpeedMenu Remove command to remove an existing interrupt before you can add any more.

Maximum number of windows messages has been reached

By default, Turbo Profiler sets the maximum number of window messages. If you attempt to track more than the number of messages specified in the Max Windows Messages text box (found through the Statistics | Profiling Options command), you'll receive this error message.

When cited with this error message, no window messages will be added to the list by Turbo Profiler. Make sure to reset the Max Windows Messages count to reflect the number of messages that you want to watch, and then add the appropriate messages through the Window Procedure Messages dialog box.

NMI interrupt

The program you're profiling has generated an NMI (non-maskable interrupt).

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 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 box.

No modules with statistics

There are no modules with any statistics collected, so there is nothing to print.

No network present

You must load NETBIOS (version 3.0 or above) before running TPROF or TFREMOTE with the **-rn** option.

No previous search expression

You have used the Next command from the SpeedMenu 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 be issued only 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 set profiling areas only on code addresses.

Not available when in coverage mode

This error message appears when you attempt to use a function that can't be used in coverage mode. Most likely, you're trying to set (or remove) an area marker. Since Turbo Profiler automatically sets coverage mode markers, you can't set or remove them manually.

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. Also see the **–m** option in Appendix A.

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. 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.

Out of heap space

Turbo Profiler ran out of memory to collect the information you requested. You'll need to reduce the amount of data that you want to gather in a single run.

Overlay not loaded

You have attempted to examine code in an overlay that is not loaded into memory. You can examine code only for overlays that are already in memory.

However, you can still look at the source code for a module in a Module window. Setting an area's operation to stop will let you view the disassembled overlay.

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.

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.

Procedure stack overflow

Your program has too many nested procedure or function calls. You must remove some of the areas that are set on routines in the deepest calling path. Use the Callers window to find this area.

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 tried to load was linked with a linker whose version is incompatible with that of Turbo Profiler. Either the linker was an old one or you're using an old version of Turbo Profiler.

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 or missing 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 **-rs**1 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.

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.

Stopped by area

Turbo Profiler encountered an area whose operation you set to "stop." You can continue profiling by using the Run | Run command.

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 one of the following:

A global symbol name

A module name, followed by #, followed by a local symbol name

A module name, followed by a #, followed by a decimal line number

Syntax error in symbol *SymbolName*

You have entered an invalid symbol name. A valid symbol name consists of one of the following:

A global symbol name

A module name, followed by #, followed by a local symbol name

A module name, followed by a #, followed by a decimal line number

Too many areas for a Windows program

You've attempted to profile a program with more areas than the 511 supported by Windows. Some of the areas you set will not be profiled. You'll have to reduce the number of areas in order to control which areas are not included.

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.

Unknown control point

Turbo Profiler has encountered an INT 3 instruction in your program that it doesn't recognize. Because Turbo Profiler uses INT 3 instructions to indicate control points, if you've put any in your program yourself, there will be a conflict. It's also possible that Turbo Profiler inserted a control point and then lost track of it.

If you inserted the INT 3 in your program, you'll have to remove it to run Turbo Profiler on your program.

If you didn't put the INT 3 in, then it's one of Turbo Profiler's. Removing the area containing the INT 3 will allow you to continue profiling the program.

Unable to determine procedure type

Turbo Profiler can't determine whether the current area is a near or far procedure. You'll have to remove it from the list of areas for Turbo Profiler to proceed.

Value must be between *nn* and *nn*

You have entered an invalid numeric value for an editor setting (such as the tab width) or printer setting (such as the number of lines per page). The error message will tell you the allowed range of numbers.

Video mode switched while flipping pages

You've 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 situation, start Turbo Profiler with display-swapping enabled (**–ds** command-line option).

Waiting for remote driver. Press Esc to stop waiting

You've started a remote profiling session, and Turbo Profiler is waiting to connect to the remote driver. Press ***Esc*** to about the remote session.

Index

Borland

Corporate Headquarters: 100 Borland Way, Scotts Valley, CA 95066-3249, (408) 431-1000. Offices in: Australia, Belgium, Brazil, Canada, Chile, Denmark, France, Germany, Hong Kong, Italy, Japan, Korea, Latin America, Malaysia, Netherlands, Singapore, Spain, Sweden, Taiwan, and United Kingdom • Part # BCP1245WW2177A • BOR 7778