User's Guide

VERSION 1.5

Borland®
Turbo Debugger® GX
for OS/2®

User's Guide

Borland®

Turbo Debugger® GX for OS/2®

Version 1.5

Borland International, Inc.
100 Borland Way, Scotts Valley, CA 95066-3249

PRINTED IN THE UNITED STATES OF AMERICA

1E0R0294
9495969798-987654321
H1

Contents

Tables

Figures

Introduction

Turbo Debugger GX is a state-of-the-art, source-level debugger with a graphical user interface (GUI). It's designed for programmers using Borland C/C++ and Turbo Assembler to produce programs that run under OS/2. Multiple views with pop-up menus, a SpeedBar displaying buttons for common actions, and integrated window management provide a fast, interactive environment. An online context-sensitive Help system provides you with help during all phases of operation.

Here are some of Turbo Debugger GX's features:

- Full Borland C, C++, and TASM expression evaluation
- Extensive set of views to support all levels of debugging
- Window management facility
- Comprehensive Online Help
- Control Panel with both menu and SpeedBar access to debugging commands
- High-level and low-level code access
- Logging facility
- Powerful control point facility that supports breakpoints, datapoints, messagepoints, and exceptionpoints
- Support for debugging multithreaded applications
- Special tools for debugging of Presentation Manager programs
- Support for hardware debugging registers

Hardware and software requirements

Turbo Debugger GX runs on any IBM PC-compatible computer that has OS/2 version 2.0 or higher installed. A mouse is recommended. To see the amount of hard-disk space required for Turbo Debugger GX, run the Borland C++ installation program.

➡ Turbo Debugger GX doesn't require a numeric processor chip.

➡ Turbo Debugger GX works with Borland C++ for OS/2 and Turbo Assembler for OS/2. If you want to do source debugging, your application file must be either an executable (.EXE file) or a dynamic-link library (DLL) compiled with full debugging information turned on.

When you run Turbo Debugger GX, you'll need your application's .EXE file and original source files. Turbo Debugger GX searches for source files in the following places in this order:

1. In the directories specified in the File | Properties dialog box
2. In the directory containing the .EXE file
3. In the current directory

You can override the File | Properties setting by starting Turbo Debugger GX with the **–s** option, which specifies the source directories. (See page 8 for more information on command-line options.)

Differences between Turbo Debugger GX and Turbo Debugger

Turbo Debugger GX works similarly to Turbo Debugger for DOS and Turbo Debugger for Windows. You'll find that many of the views, commands, and keystrokes you're accustomed to with the DOS or Windows debugger work with the OS/2 debugger. You'll also find that local menus are accessible from the views in much the same way (by right-clicking or pressing *Ctrl+F10*).

There are differences in functionality, some of which are due to the OS/2 environment. They include the following differences:

- A graphical user interface that includes a *SpeedBar*, a series of buttons you can select to perform common functions, like running, stopping, reloading, or stepping. Because the debugger is always in graphics mode, you can see your application running in another window instead of having to switch between the full-screen debugger and the application.
- Dialog boxes that aren't modal. A nonmodal dialog box is like any other window: it stays around until you close or minimize it, and you can switch to another window while the dialog box is displayed. You press *Enter* in a nonmodal dialog box to get text entries to take effect. Radio-button and check-box selections take effect immediately.
- Windows that can move anywhere on the screen and resize to the full screen size. Each view has its own window, and there's a separate window called the Control Panel for the main menu and the SpeedBar.
- The ability to duplicate any view by choosing New View from the view's local menu. For example, you can open multiple Source views and look at more than one module or DLL at the same time, as long as the module or DLL is used by the currently loaded process.

See Appendix A for detailed information comparing various tasks you can perform with Turbo Debugger and Turbo Debugger GX.

- Dual form views that show a list of items or the details on one item. Using the detail form of the view, you can set all the options for an item from the list. The views that can switch between list form and detail form are the Breakpoint view, the Datapoint view, the Messagepoint view, the Exceptionpoint view, the Variable view, and the Watch view.

There are also many similarities in functionality. You'll find that many of the shortcut keys are the same, and most of the views between Turbo Debugger and Turbo Debugger GX will be familiar. The table in Appendix A lists some typical tasks and shows how to do them with both products.

Files distributed with Turbo Debugger GX

The Turbo Debugger GX part of the Borland C++ package includes this manual and a set of files on disk. The files include

- The files needed to run the program (TD.EXE, associated DLLs, and a Help file)
- Online text files
- Utility program
- Sample program files

The installation program (described on page 7) copies these files into various default directories on your hard drive. (You can specify different directories during installation.) For a complete list of files associated with Turbo Debugger GX, see the README.TD online text file.

For a list of the files on your distribution disks, see the FILELIST.DOC file on the Installation disk.

Program files

The installation program copies the program files into the BIN subdirectory of your main Borland C++ directory. The following files are included:

- TD.EXE
- TDDEBUG.DLL
- TD-LANG.STR
- TDHELP.HLP

Online text files

By default, the installation program copies the Turbo Debugger GX online text files into the DOC subdirectory of the main Borland C++ directory on your hard drive. These files include README.TD, MANUAL.TD, and UTILS.TD. In addition, there's an overall README file for the entire Borland C++ package that resides in the main Borland C++ directory.

The README.TD file

It's important that you take the time to look at the README.TD file before you do anything else with Turbo Debugger GX. This file contains last-minute information that might not be in the manual or the Online Help.

The MANUAL.TD file

Be sure to read the MANUAL.TD file for late-breaking changes and additions to the manual. If there are no changes to report, this file won't be on the disk.

The UTILS.TD file

Turbo Debugger GX comes with the TDUMP utility. By default, it's in the BIN subdirectory of the main Borland C++ directory along with the Turbo Debugger GX program files.

To get a list of the command-line options available for TDUMP, type the program name on the OS/2 command line and press *Enter*.

TDUMP.EXE displays the contents of object modules and .EXE files in a readable format.

Sample programs

A number of sample programs are distributed in the Borland C++ package. The two programs associated with Turbo Debugger GX are TDDEMO.EXE and TDDEMOPM.EXE.

TDDEMO

This program is a simple OS/2 character-mode application that displays text to, and reads text from, a single window. It's the sample program used in Chapter 3, "A quick example." TDDEMO takes lines of text as input. When the user presses *Enter* on an empty line, the program calculates the number of letters, words, and lines, and how many times each letter occurred, and categorizes words according to length. It then displays all this information on the screen.

TDDEMOPM

This program does the same work as TDDEMO, except that it accepts input in one window and displays the output in two other windows after each line is entered. It uses some standard Presentation Manager (PM) window types to do its work.

The window on top, the one the user enters text in, is a multiline edit window. It uses a standard PM multiline entry field control (WC_MLE) to display text and process the entries the user makes.

The two windows below this one are used by TDDEMOPM to display program output. They are standard PM list boxes that use the list box control WC_LISTBOX.

Typefaces, icons, and conventions

This section explains the meaning of the special typefaces and icons used in this manual.

`Monospaced type` This typeface represents text as it appears onscreen or in a program. It is also used for anything you must type literally (such as `TD` to start up Turbo Debugger GX).

ALL CAPS All capital letters are used for the names of files and C++ constants.

[ ] Square brackets [ ] in text, syntax statements, or OS/2 command lines enclose optional items. *Text of this sort should not be typed verbatim.*

Boldface Boldface type indicates

- C++ predefined types, functions, preprocessor directives, reserved words and keywords
- Command-line switches (such as **–s**)

Italics Italic type indicates C++ variable names, data members, user-defined types, and classes. This typeface is also used to emphasize certain words, such as new terms.

Keycaps This typeface indicates a key on your keyboard. For example, "Press *Esc* to exit a menu."

Key1+Key2 Key combinations produced by holding down one or more keys simultaneously are represented as *Key1+Key2*. For example, you can reset the program by holding down the *Ctrl* key and pressing *F2*. This key combination is represented as *Ctrl+F2*.

Choice1 | Choice2 This command sequence represents a choice from the menu bar followed by a choice from the drop-down menu. For example, instead of saying "Choose File, then choose Load Process from the File menu," we say "Choose File | Load Process."

This icon indicates material you should take special notice of.

This icon indicates a reference to the Help system, where you can find complete, up-to-date information on Turbo Debugger GX.

Using this manual

This manual covers the basics of using Turbo Debugger GX. It does not cover all features or discuss debugging tasks in detail—you can find that information in the Online Help system. This manual discusses general aspects of the user interface, tells how to use Online Help, how to install, start, and exit the debugger, and shows how to use some of the debugger's features on a sample program. Once you have the debugger running, you can use the extensive Online Help facility to get complete explanations of features or debugging tasks.

If you're an experienced Turbo Debugger for DOS or Turbo Debugger for Windows user, see Appendix A for a list of debugging tasks and how to perform them with Turbo Debugger GX.

The manual contains the following chapters and appendixes:

Chapter 1: Getting started discusses how to install Turbo Debugger GX, how to enter and exit the debugger, and how to use the Online Help system.

Chapter 2: The Turbo Debugger GX environment discusses some aspects of debugging and provides an overview of the Turbo Debugger GX environment.

Chapter 3: A quick example shows how to use Turbo Debugger GX to perform some debugging tasks on a sample program.

Appendix A: Turbo Debugger GX for experienced Turbo Debugger users lists some typical debugging tasks and shows how to do them with both DOS or Windows Turbo Debugger and Turbo Debugger GX.

C H A P T E R 1

Getting started

Your Borland C++ package contains a set of distribution disks and manuals, including the *Turbo Debugger GX for OS/2 User's Guide* (this book). The distribution disks contain all the programs, files, and utilities needed to debug programs written using Borland C++ for OS/2 and Turbo Assembler for OS/2. The online text files README, MANUAL.TD, and UTILS.TD contain documentation on subjects not covered in this manual.

If you aren't familiar with Borland's no-nonsense license statement, now is the time to read the agreement. Mail your filled-in product registration card, so you'll be notified about updates and new products as they become available.

Installing Turbo Debugger GX

When you installed Borland C++ on your system, INSTALL.EXE (the installation program on your distribution disks) copied files from the distribution disks to your hard disk. If you left the defaults on, the installation program also created a Borland C++ folder on the desktop and put the icons for Borland C++, Resource Workshop, and Turbo Debugger GX into it.

If you chose not to install Turbo Debugger GX when you installed Borland C++, you can install it now:

1. Insert the Installation disk in one of your floppy drives (for example, drive A).
2. In an OS/2 window, type `A:INSTALL.EXE` and press *Enter*.
3. In the Installation dialog box, click the Installation Options button.
4. Specify Turbo Debugger GX as the only program to install, then click OK.
5. In the Installation dialog box, click Install to start installation.

See page 3 for more information on the Turbo Debugger GX files.

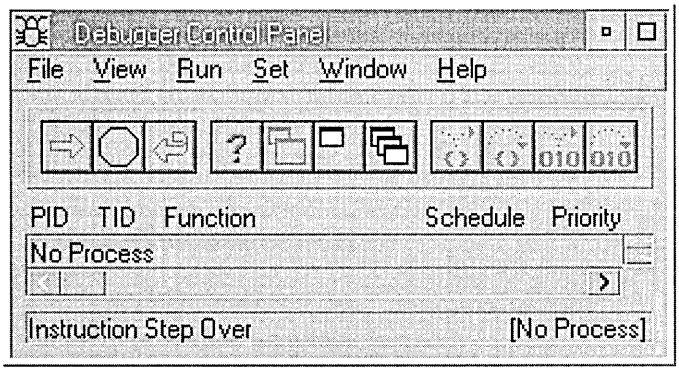
By default, the installation program copies the Turbo Debugger GX program files and utilities to the BIN subdirectory of the main Borland C++ directory, online text files to the DOC subdirectory, and examples to the EXAMPLES subdirectory.

Before installing the files, you can change default directories by clicking the Directory Options button and entering new directories in the Borland C++ Directory Options dialog box.

Entering and exiting Turbo Debugger GX

When you've installed Turbo Debugger GX and it appears as an icon in an OS/2 folder, double-click the icon to start Turbo Debugger GX and display the Debugger Control Panel.

Figure 1.1
Control Panel view



See page 17 for a description of the Control Panel view.

From the Control Panel, you can choose File | Load Process to load an application program so you can debug it.

When you're finished debugging that application, you can choose File | Unload Process to unload the current process, and then load in another application. You can also exit the program by choosing File | Exit, pressing *Alt+X*, pressing *Alt+4* in the Control Panel, choosing Close from the System Menu, or double-clicking on the system menu icon (at the top left corner of the Control Panel's title bar).

Using command-line options

There are a number of command-line options you can use when starting Turbo Debugger GX. You can enter these command-line options two different ways:

- Start Turbo Debugger GX from the OS/2 command line (for example, `td -m`).
- Right-click on the Turbo Debugger GX icon to display the icon's pop-up menu, then

  1. Choose the arrow to the right of Open to display the Settings notebook.
  2. On the Program page, enter any Turbo Debugger GX parameters in the Parameters field.
  3. Close the notebook when you're done.

The command-line format is as follows:

```
TD [option[optionarg] ... option[optionarg]] [progname [progargs]]
```

Brackets indicate that an argument is optional. All Turbo Debugger GX command-line arguments are optional.

- *option* is one of the command-line options listed in Table 1.1. Options must be preceded by either a dash (-) or a slash (/).
- *optionarg* is the argument to a switch, such as the path name that follows the **–s** option.
- *progname* is the file name or full path to the file name of the application program you intend to debug.
- *progargs* are arguments to the application program.

The following table lists the Turbo Debugger GX command-line options. For more information on these options, access the Turbo Debugger GX Help system and use the Search button to find *command-line option*.

Table 1.1
Turbo Debugger GX command-line options

Option	Description
–c*filename*	Indicates the path and filename of the configuration file to be used when Turbo Debugger GX starts up. By default, Turbo Debugger GX uses TD.INI as its configuration file. If you specify a **–c** command-line option, Turbo Debugger GX reads and writes to the configuration file specified.
–h or **–?**	Opens a window displaying a panel of Help text describing these command-line options.
–m	Enables monochrome screen colors for plasma screens and other monochrome video adapters.

Table 1.1: Turbo Debugger GX command-line options (continued)

–r<*expression*>	Run to *expression* on start up. You must also specify an application to be debugged (a *progname* command-line argument). This switch causes Turbo Debugger GX to run the startup code of the application and position the program counter at *expression* after it loads the application. Note that **main** is the default.
–s*dirlist*	Indicates where to find the source files for your application. You can enter one search path or multiple search paths separated by semicolons.

Using the Help system

This section provides a detailed overview of the Turbo Debugger GX Online Help system. The Help system is the principal source of information about Turbo Debugger GX. You can go there to get detailed procedural and descriptive help on debugging tasks or to get context-sensitive Help on elements of the user interface (like views, menu choices, dialog boxes, list boxes, and entry fields).

Online Help is available from any view, menu, or dialog box, and provides three kinds of information:

- Context-sensitive Help for all individual elements of the Turbo Debugger GX environment: menu choices, views, entry fields, check boxes, and radio buttons in views and dialog boxes. You can select any menu choice or any element of a dialog box or view (entry field, check box, or radio button), then press *F1* to get Help for that item.
- Task-oriented information on debugging tasks, such as essential information to get you started, how to set breakpoints, how to go on a bug hunt, and debugging tips and techniques.
- Information on the Turbo Debugger GX environment, such as views, menus, and keyboard shortcuts. (Most of this information is available as context-sensitive Help, but you can also access it from within the Help system.)

To access the Help system, do any of the following actions:

- Choose Help from the Control Panel's menu bar.
- Click the Help button on the Control Panel's SpeedBar.
- Press *F1* anywhere in Turbo Debugger GX.
- Press *Shift+F1* (to get the Help index).
- Press *Ctrl+H* in a view or dialog box, or choose Help from the local menu.

How you access the Help system depends on what kind of information you want.

- If you need general help, you can access the Help menu or the Help Contents panel and choose the topic you need.
- If you know what you're looking for but not where it is in the Help system, you can display the Help index and search for the item.
- If you want help with an element of the user interface, you can set the focus to that control and press *F1* for context-sensitive Help.
- If you want task-oriented information, such as how to set breakpoints, you can display the Contents panel and select Essentials or Tasks, or you can use the Search button in the Help window to find the topic.

Accessing the Control Panel Help menu

If you choose Help from the Control Panel menu bar, you see the Help menu.

Table 1.2 The Help menu

Menu choice	Description
Contents	Table of contents for the Help system. Each topic preceded by a [+] can be expanded into subtopics. A [-] means the subtopics for that topic are already expanded. Double-clicking on a topic brings up a Help panel for it.
Index	Alphabetical list of topics for the Help system. Double-clicking on a topic or subtopic brings up a Help panel for it.
Essentials	Information to help you get started with the debugger. Also available from Contents.
Tasks	A list of debugging tasks, such as compiling your program for debugging, executing your program under the debugger, and setting and using control points. Also available from Contents.
Menus	A list of all the global and local menu choices available in the debugger. Also available from Contents.
Views	A list of all the views, including the Control Panel. Also available from Contents.
Glossary	A glossary of debugging terms. Also available from Contents.
Using Help	Help on how to use the Help system.
Product Information	A panel showing the Turbo Debugger GX name and version.

Using a Help window

Picking any Help menu choice except Product Information displays a Help window.

To get information on how to use Help, choose Help from the window's menu bar. At the bottom of the window are some pushbuttons, which work as follows:

Contents	Display the Contents panel.
Index	Display the Help index.
Print	Print selected Help panels.
Search	Search for a topic in the Help system.
Previous	Go to the previously viewed Help panel. If this is the first panel you displayed, clicking this button exits you from Help.
Forward	Display the next Help panel.

Displaying the Contents panel

To display the Contents panel from the debugger, choose Help | Contents from the Control Panel menu bar or click the Help button on the SpeedBar.

If you're already in a Help window, you can go to the Contents panel by clicking the Contents button at the bottom of the Help window, pressing *Ctrl+C*, or choosing Options | Contents with your mouse or by pressing *Alt+O+T*.

The Contents panel shows the same topics as those shown in Figure 1.1.

Displaying the Help index

To display the Help index, choose Help | Index from the Control Panel, or press *Shift+F1* from anywhere within the debugger.

If you're in a Help window, you can click the Index button at the bottom of the window, press *Ctrl+I*, choose Options | Index with your mouse, or press *Alt+O*.

The Help index is an alphabetic list of topics in the debugger's Online Help. You might want to display it if you know what you're looking for, but you aren't sure where it is. You can scan down the list or search for a topic. When you find the topic you want, double-click it or press *Enter* to display the associated Help panel.

Getting context-sensitive Help

To get Help on a part of the user interface, such as a menu choice or a dialog box entry field, select it, then press *F1*. A Help panel comes up showing information on the area you clicked.

For example, if you press *F1* while File | Load Process is selected, you get information on that menu choice. If you press *F1* when the Source view is active, you get information on that view.

For Help on a menu in its entirety (rather than a single menu choice), go into Help and find the menu name in the Contents or the Index about that menu, then double-click the menu name to display a Help panel.

Printing Help information

You can print Help information from a Help window as follows:

1. Select the panels you want to print by doing one of the following:
   - If you want to print one panel, display a single Help panel.
   - If you want to print more than one panel, display the Contents panel and select the topics you want to print. Each topic represents a single Help panel. (The Print facility calls each Help topic a *section* and calls this process *marking sections*.) To select topics, press *Ctrl* and click each topic you want to print. (To deselect topics, repeat this process.)
2. Select the Print button, press *Ctrl-P*, or Services | Print (with your mouse) to display the Print dialog box.
3. Select the radio button for what you want to print, then click Print.

Choosing All Sections isn't recommended because it will print all the panels in the Help system.

C H A P T E R 2

The Turbo Debugger GX environment

Debugging is the process of finding and correcting errors (*bugs*) in your programs. It's not unusual to spend more time finding and fixing bugs in your program than writing the program in the first place. Debugging is not an exact science; the best debugging tool you have is your own feel for where a program has gone wrong. Nonetheless, you can always profit from a systematic method of debugging.

The debugging process can be broadly divided into four steps:

1. Realizing you have a bug
2. Finding where the bug is
3. Finding the cause of the bug
4. Fixing the bug

What Turbo Debugger GX can do for you

Turbo Debugger GX helps with the two hardest parts of the debugging process: finding where the bug is and finding the cause of the bug. It does this by controlling program execution so you can examine the state of the program at any given spot. You can even test new values in variables to see how they affect your program. With Turbo Debugger GX, you can perform *stepping*, *viewing*, *inspecting*, *changing*, and *watching*.

Stepping into You can execute your program one line or one instruction at a time, stepping into each function call.

Stepping over You can execute your program one line or one instruction at a time, but step over any function calls. If you're sure your procedures and functions are error-free, stepping over them speeds up debugging.

Viewing You can have Turbo Debugger GX open a special window to show you the state of your program from various perspectives: variables and their values, breakpoints, datapoints, messagepoints, exceptionpoints, the contents of the stack, an event log, a data file, a source file, disassembled code, memory, the heap, registers, numeric processor information, or program output.

Inspecting You can look at the contents of variables and expressions, including complex data structures like arrays and structures.

Changing You can replace the current value of a global or local variable with a value you specify.

Watching You can isolate program variables and keep track of their changing values as the program runs.

You can use these tools to dissect your program into discrete sections, confirming that one section works before moving to the next. In this way, you can work through any program, no matter how large or complicated, until you find where a bug is hiding. You might find there's a function that inadvertently reassigns a value to a variable, or gets stuck in an endless loop. Whatever the problem, Turbo Debugger GX helps you find where it is and what's at fault.

Turbo Debugger GX enables you to debug object-oriented C++ programs. It's smart about classes, and it correctly handles late binding of member functions so that it executes and displays the correct code.

Turbo Debugger GX also enables you to debug both Presentation Manager and OS/2 line-mode programs.

What Turbo Debugger GX won't do

With all these features, you might be thinking that Turbo Debugger GX has it all. However, there are at least three things Turbo Debugger GX *won't* do for you:

- Turbo Debugger GX doesn't have a built-in editor to change your source code. You can use the Borland C++ editor or your favorite text editor for this purpose.
- Turbo Debugger GX can't recompile your program for you. You need the original program compiler to do that.
- Turbo Debugger GX can't come up with strategies for finding bugs. It's a powerful tool, but is only that—a tool.

How Turbo Debugger GX does it

Here's the good news: Turbo Debugger GX gives you all this power and sophistication, and at the same time it's easy to use.

Turbo Debugger GX accomplishes this blend of power and ease by offering an environment featuring a graphical user interface. The next section describes the advantages of the Turbo Debugger GX GUI environment.

The Turbo Debugger GX environment

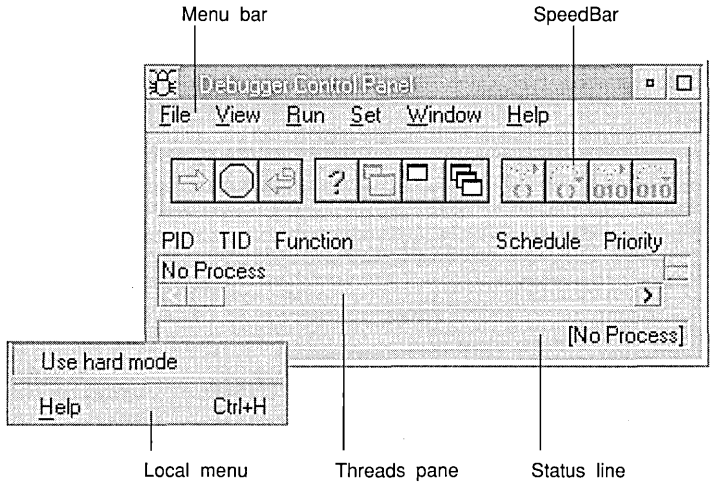
Turbo Debugger GX has been designed for intuitive use. To this end, Turbo Debugger GX provides you with the following features:

- A Control Panel view, from which you can control all aspects of a debugging session.
- Global and local menus that make it easier to access menu commands.
- Online Help, available from any view, menu, or dialog box, that provides context-sensitive and task-oriented information. (See page 10 for a description of the Help system.)
- Dialog boxes you can use to change preferences, look at variables, set control points, and load processes.
- Views that show you different aspects of your code and data, and tell you what's going on in memory, with the processor, and with the operating system.

Using the Control Panel

The first thing you see when you start Turbo Debugger GX is the Control Panel view.

Figure 2.1
Control Panel view



You use the Control Panel to oversee, manage, and control the debugging process. From the Control Panel, you can perform the following tasks:

See the Online Help for complete information on the Control Panel and on these tasks.

- Load and unload applications
- Open views
- Run and step through applications
- Set control points (breakpoints, messagepoints, datapoints, and exceptionpoints)
- Manage all the views
- Monitor the status of threads

As you can see in Figure 2.1, the Control Panel view contains the following elements:

- Menu bar
- SpeedBar
- Threads pane
- Status line
- Local menu

The menu bar

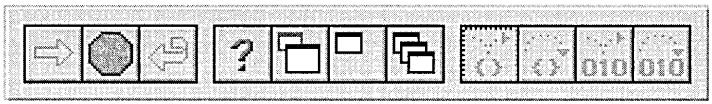
There is one menu bar in the debugger, the one at the top of the Control Panel. The menu bar has the following choices:

Table 2.1
Menu bar choices

Choice	Description
File	Use the File menu to load or unload a process, set debugger properties, or exit the debugger.
View	Use the View menu to select any of the debugger views. With these menu choices, you get access to views that show things like source code, disassembled code, control points, program data, memory, and what's going on with the CPU. The views are described later, starting on page 22.
Run	Use the Run menu to run your application in different ways, such as stepping through your program one source line at a time, running to a certain point, or simply running the program. You can also stop or reset the program from this menu. There are SpeedBar buttons that correspond to the following choices on this menu: Run, Stop, Reset, Statement Into, Statement Over, Instruction Into, and Instruction Over.
Set	You can use this menu to set four kinds of control points (breakpoints, datapoints, C++ exceptionpoints, and messagepoints) and to set a watch to monitor changes in an expression.
Window	Use this menu to control your debugger views. You can switch to the window of the application (the *user* window), move from view to view, hide or show all the views (except the Control Panel), save or restore the positions you've put the views in, and choose from a list of open views.
Help	Use this menu to access Online Help. (Note that you can also use the SpeedBar button to access the Help.) See page 10 for more information on using Online Help.

See the Menus topic in Online Help for a complete description of these menu choices.

The SpeedBar



Use context-sensitive Help for a complete description of each of these buttons.

The SpeedBar gives you quick access to typical debugging tasks. If you run the mouse across the SpeedBar, the function of each button appears on the status line at the bottom of the Control Panel. The following table describes what these buttons do:

Table 2.2: SpeedBar buttons

Button	Description
Start Program Running	Runs the currently loaded process. Same as Run|Run.
Stop Program	Stops the currently loaded process if possible and returns control to the debugger. If the Stop button is disabled, it isn't possible to stop the process at this time. Same as Run|Stop.
Reset Program	Reloads the current process so you can run it again from the beginning. Any watches and breakpoints you set in an earlier run remain set. Same as Run|Reset.
Show Help	Displays the Contents screen of Online Help. Same as Help|Contents.
Raise Program Window	Switches to the application's active window. Same as Window|User Window.
Hide Debugger Windows	Hides all open views. You typically do this before minimizing the Control Panel. Same as Window|Hide Views.
Show Debugger Windows	Shows all the views hidden by the Hide Views command. Same as Window|Show Views.
Statement Step Into	Steps through the application one source statement at a time, and into any functions that are called. Same as Run|Statement Into.
Statement Step Over	Steps through the application one source statement at a time, but steps over any function calls (doesn't step into the function, but rather runs the function until it returns). Same as Run|Statement Over.
Instruction Step Into	Steps through the application one assembly instruction at a time, and steps into any routines that are called. Same as Run|Instruction Into.
Instruction Step Over	Steps through the application one assembly instruction at a time, but steps over any routine calls (doesn't step into the routine, but rather runs it until it returns). Same as Run|Instruction Over.

The Threads pane

The Threads pane shows information about the threads that make up your application. It's most useful with multithreaded applications. Using this pane, you can

- Reset all views to show information about a particular thread you want to debug
- Get information about the current thread and process, such as the process ID (PID), the thread ID (TID), which function is currently active, and the schedule and priority of the current thread

A typical use of this pane is to select a thread so you can get information about it (such as register settings or variable values), and to change the Source view to show where that thread stopped. To select a thread, click the scroll buttons on the right side of the pane till you see the thread you want. Then, double-click the thread. (All the views will then take on the context of that thread.)

The status line

The status line at the bottom of the Control Panel displays error messages, shows the status of the current process (suspended, running, no process), and describes what each SpeedBar button does.

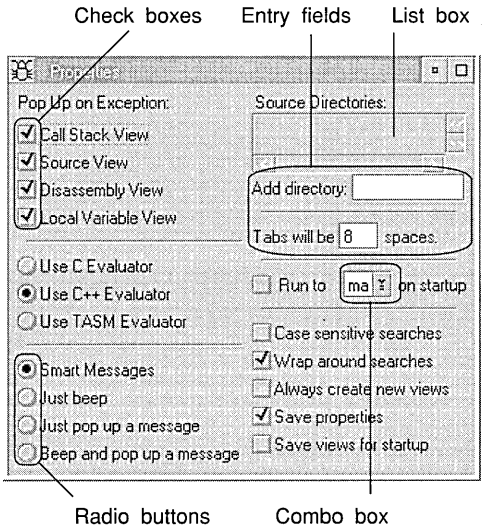
Because the status line displays error messages, keep this part of the Control Panel visible at all times, even when working in other views.

Using dialog boxes

A dialog box is a window you can enter information in for some task you want to perform. A dialog box, unlike a view, just lets you enter or change information. (A view shows updated information about your application and might let you enter information as well.)

For example, if you choose File | Properties, you see the Properties dialog box.

Figure 2.2
Properties dialog box



See the Online Help for more information on this dialog box.

This dialog box contains radio buttons, check boxes, a list box, and three entry fields. Because the dialog box is just another window, there is no OK or Cancel button. Depending on the kind of change you make, it either takes effect immediately or takes effect when you press *Enter*.

For example, if you type a source directory name in the Add Directory entry field, you must press *Enter* for it to take effect. However, if you click any of the radio buttons or check boxes, your selections take effect immediately.

This behavior is common to all dialog boxes and views.

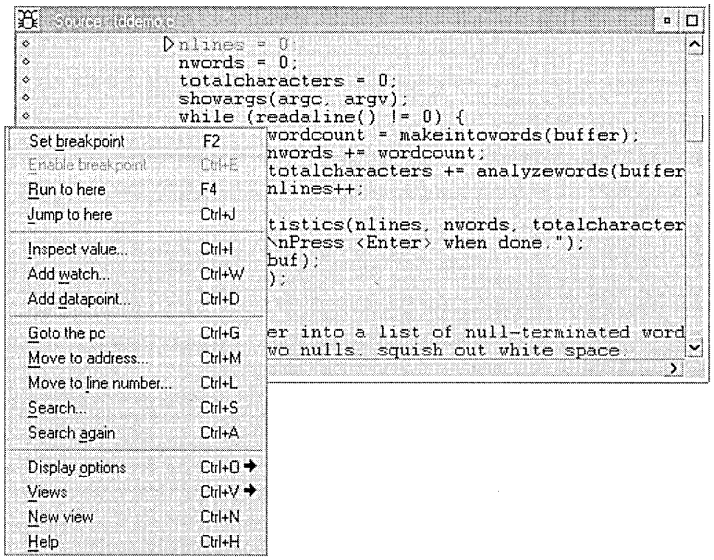
Working with views

A view is a window that shows information about your application. Some views also let you enter information. These views follow the same rules as dialog boxes for entering information (see the previous section). There are eighteen different views, all available from the Control Panel's View menu.

Local menus

All views have local menus that you can pop up by right-clicking the mouse or by pressing *Ctrl+F10*. For example, right-clicking in the Source view displays the following local menu:

Figure 2.3
Source view local menu



The local menu choices have shortcut keys, indicated by an underlined letter in the choice. For example, to search for a string in the Source view, you can press *Ctrl+S* without having to display the local menu first.

List views and Detail views

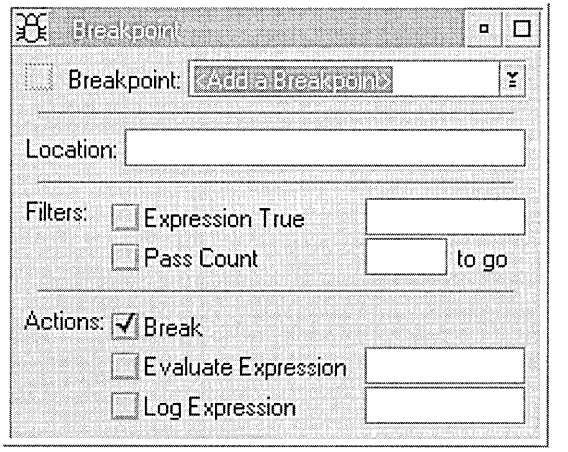
Some of the views have two forms, a List view that shows all items that have been set and a Detail view that shows information about each item. The views that have these two forms all manage lists of items, such as variables or control points. The following views have both forms:

- Breakpoint view
- Datapoint view
- Exceptionpoint view
- Messagepoint view
- Variable view
- Watch view

The view initially displays in a default form (if there are items to display, the list form; if there are no items, the detail form). To switch to the other form, either press *Ctrl+S* or right-click in the view to display the local menu, then choose the first menu choice. The wording of this menu choice changes depending on which form of the view is displayed.

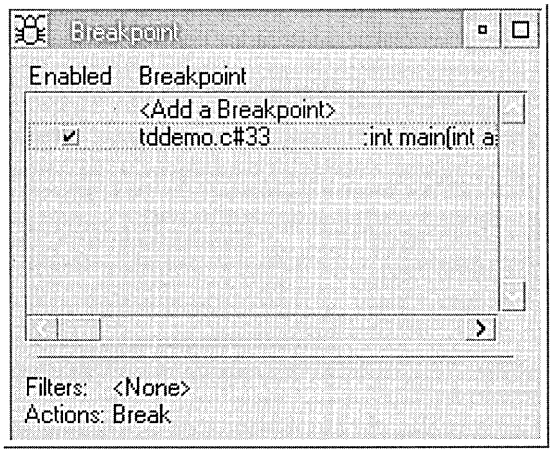
For example, when you first load your application, there are no breakpoints set. Choosing View | Breakpoint in the Control Panel displays the Breakpoint Detail view, which you can use to set a breakpoint.

Figure 2.4
Breakpoint detail view



Enter the name of a function (for example, **main**) in the Location entry field and press *Enter*. (The Location entry field takes a program location, such as function name that evaluates to a program location.) Next, right-click to display the local menu, then choose Show Breakpoint List to display the List view. You see the breakpoint you just set.

Figure 2.5
Breakpoint List view

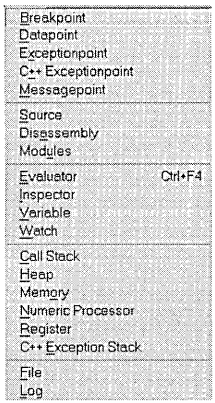


If you close the Breakpoint view, then choose View | Breakpoint again, notice that the Breakpoint view comes up in list form. That's because there's a breakpoint to display. If you right-click in the Breakpoint view, you see that the first local menu choice has changed to *Show Breakpoint Details*.

If you close the Breakpoint view again, then choose Set | Breakpoint from the Control Panel, notice that the Breakpoint view comes up in detail form, letting you set a breakpoint immediately.

The views

The View menu is displayed when you choose View from the Control Panel. The View menu is divided into five sections that reflect the functionality of the views.



- The first section groups choices for all the *control point* views. (A control point is a name referring to breakpoints, datapoints, exceptionpoints, and messagepoints, all of which can be used to control program execution.) Use these views to set various kinds of control points that, when encountered, can log information about the control point or return control to the debugger.
- The second section groups choices for views that show you information about your application's code. For example, the Source view shows your application's source code and where the program has executed to, and the Modules view lists the source files contained in your executable program.
- The third section groups choices for views that show you information about your program's data. For example, the Variable view displays all local or global variables.
- The fourth section groups choices for views that show you hardware-related information, such as the contents of memory or the CPU registers.

- The fifth section contains choices for the Log view and the File view, both used for auxiliary functions (logging information and looking at files that do not contain debug information).

The Breakpoint view

Choosing View | Breakpoint from the Control Panel displays the Breakpoint view. You can also choose Set | Breakpoint to display this view in Detail form.

Breakpoints stop the processing of your program and give control of it to you. Use the breakpoint view to set, remove, modify, enable, and disable breakpoints, and to see a list of the breakpoints that have been set in your program.

When you set or change a breakpoint, you can also set filter conditions and actions, which customize the conditions under which a breakpoint is activated and specify the actions that take place when the breakpoint is activated.

The local menu

Right-click or press *Ctrl+F10* to display the local menu.

With the Breakpoint view local menu, you can perform actions with breakpoints, such as setting or removing them. You can also use the shortcut keys you see on the menu directly from the Breakpoint view, without displaying the menu.

The List and Detail views

Press *Ctrl+S* to change view forms.

This view has two forms, a Detail view form and a List view form. The Detail view shows details about a particular breakpoint (if any exist); you use this form to set a new breakpoint or change settings for an existing breakpoint. The List view shows all breakpoints that have been set.

See page 23 for more information on displaying forms.

You can set a simple breakpoint without using the Breakpoint view. There are two ways to do this:

- Select a line of code in the Source view or the Disassembly view, then press *F2*.
- Double-click the mouse either inside a line of disassembled code or near the diamond in the left margin of a line of source code (if you've displayed the diamonds when using the Display Options | Show Attributes submenu).

You can get the following additional information from the Help system:

- For other methods of setting breakpoints, see the Online Help topic "Setting Breakpoints".
- For a complete discussion of breakpoints, see "Breakpoints" and its subtopics in the Online Help under the "Setting and Using Control Points" task.
- For information on the Breakpoint view itself, click in the view, then press *F1* or choose Help from the view's local menu.
- For information on a local menu choice, right-click to display the menu, then select the menu choice and press *F1*.

The Datapoint view

Choosing View | Datapoint from the Control Panel displays the Datapoint view. You can also choose Set | Datapoint to display this view in detail form.

Use this view to set a *datapoint* or see all the datapoints you've set. A datapoint (also known as a watchpoint) is a variable or expression whose memory location the debugger watches during program execution. When the value in that memory location matches a condition, such as being equal to or less than a certain value, the debugger performs the action you've indicated, such as breaking and returning control to the debugger.

A datapoint has characteristics similar to a breakpoint (see the Breakpoint view description starting on page 25). For additional information about datapoints, refer to the topic in the Online Help.

The local menu

You use the Datapoint view to perform the following actions on datapoints:

- Set new datapoints
- Adjust the filters and actions associated with those datapoints
- Enable or disable datapoints
- Remove existing datapoints
- Look at the datapoints that have been set

All these choices are available from the Datapoint view local menu (right-click or press *Ctrl+F10* or *Shift+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Datapoint view, without actually displaying the menu.

The List and Detail views

Press *Ctrl+S* to change view forms.

This view has two forms, a Detail view form and a List view form. The Detail view shows details about a particular datapoint (if any exist); you

use this view to set a new datapoint or change settings for an existing datapoint. The List view shows all datapoints that have been set.

For more information on the Datapoint view itself, click in the view, then press *F1* to display a Help screen or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*. For more information on working with datapoints, see "Datapoints" in the Online Help under the "Setting and Using Control Points" task.

The Exceptionpoint view

Choosing View | Exceptionpoint from the Control Panel displays the Exceptionpoint view.

Use this view to change settings for an *exceptionpoint* or see all the exceptionpoints. An exceptionpoint tells the debugger what to do when it intercepts a particular *exception* or signal sent to your application. (An exception is an asynchronous notification from OS/2 that an event has occurred, such as a divide-by-zero exception or a guardpage exception.)

When the exception comes in, the debugger performs the action you've indicated, such as breaking and returning control to the debugger, then passing the exception to the application when you run the application again.

An exceptionpoint has characteristics similar to a breakpoint (see the Breakpoint view description starting on page 25). For further information about exceptionpoints, refer to the topic in the Online Help.

The local menu

You can use the Exceptionpoint view to perform the following actions on exceptionpoints:

- Indicate whether an exceptionpoint pauses program execution
- Adjust the filter conditions and actions associated with exceptionpoints
- Look at the list of exceptionpoints

All these choices are available from the Exceptionpoint view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Exceptionpoint view, without actually displaying the menu.

The List and Detail views

Press *Ctrl+S* to change view forms.

This view has two forms, a List view form and a Detail view form. The List view shows all exceptionpoints. The Detail view shows details about a particular exceptionpoint; you use this view to change settings for an exceptionpoint.

For more information on the Exceptionpoint view itself, click in the view, then press *F1* to display a Help screen or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

The C++ exceptionpoint view

Choosing View | C++ exceptionpoint from the Control Panel displays the C++ exceptionpoint view.

Use this view to customize the action that the debugger should take when a C++ exception is thrown. By default, the debugger stops on all C++ exception throws.

When a C++ exception is thrown by your program, the debugger performs the actions you've indicated, such as breaking and returning control to the debugger, then with the throw when you run the application again.

A C++ exceptionpoint has characteristics similar to a breakpoint (see the Breakpoint view description starting on page 25). For further information about C++ exceptionpoints, refer to the topic in the Online Help.

When the debugger stops on a C++ exception, it displays a dialog box that displays the C++ exception's type and value. With this dialog box, you can choose to run to either the catch or stack-unwinding destructors associated with this C++ exception.

The local menu

You can use the local menu of the C++ exceptionpoint view to enable, disable, remove, and add C++ exceptionpoints.

The List and Detail views

Press *Ctrl+S* to change view forms.

This view has two forms, a List view form and a Detail view form. The List view shows all C++ exceptionpoints. The Detail view shows details about a particular C++ exceptionpoint; you use this view to change settings for a C++ exceptionpoint.

For more information on the C++ exceptionpoint view itself, click in the view, then press *F1* to display a Help screen or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

The Messagepoint view

All these choices are available from the Exceptionpoint view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Exceptionpoint view, without actually displaying the menu.

Choosing View | Messagepoint from the Control Panel displays the Messagepoint view. You can also choose Set | Messagepoint to display this view in detail form.

Use this view to track PM messages sent to the window functions in your application. You can also have your application break and return control to the debugger or perform some other action when it encounters a message for one of your window functions.

When you designate a window message to be tracked, you're setting a *messagepoint*. A messagepoint has characteristics similar to a breakpoint (see the previous section). For additional information about messagepoints, see the topic in the Online Help.

The local menu

You can use the Messagepoint view to perform the following actions on messagepoints:

- Set new messagepoints
- Adjust the filter conditions and actions associated with those messagepoints
- Enable or disable messagepoints
- Set messagepoints on your own custom messages
- Remove existing messagepoints
- Look at the messagepoints that have been set on window functions

Most of these choices are available from the Messagepoint view local menu (right-click or press *Ctrl+F10* or *Shift+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Messagepoint view, without actually displaying the menu.

The List and Detail views

Press *Ctrl+S* to change view forms.

This view has two forms, a Detail view form and a List view form. The Detail view shows details about a particular messagepoint (if any exist); you use this view to set a new messagepoint or change settings for an existing messagepoint. The List view shows all messagepoints that have been set.

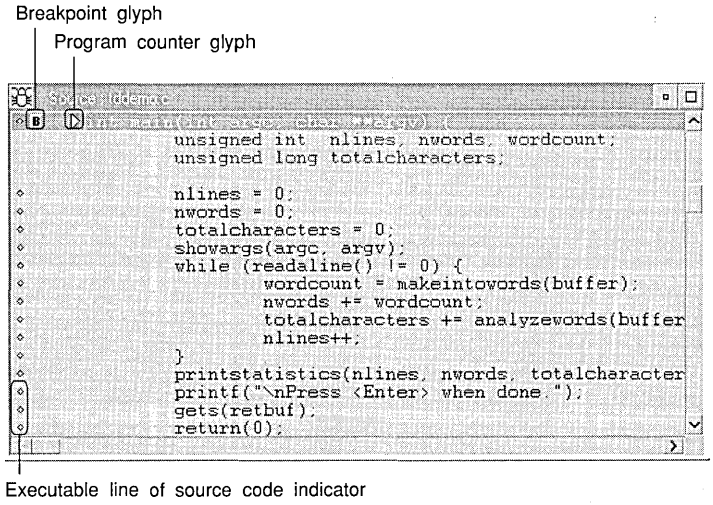
For more information on the Messagepoint view itself, click in the view, then press *F1* to display a Help screen or choose Help from the view's local menu. For information on a local menu choice, select it and press *F1*. For more information on working with messagepoints, see "Messagepoints" in the Online Help under the "Setting and Using Control Points" task.

The Source view

If you load an application that has debugging information and source code, the debugger displays the source code for the current module in the Source view. You can also display the Source view by choosing View | Source from the Control Panel.

The following figure shows the Source view opened on TDDEMO, one of the sample programs distributed with Turbo Debugger GX. For demonstration purposes, a breakpoint has been set and the program has been run to **main**.

Figure 2.6
The Source view



- The diamond at the left of a line of source code indicates that the line is executable (not a declaration or comment) and has a valid address. It's a location where you can set a breakpoint.
- If you set a breakpoint on a line of source code, a breakpoint glyph (a blue *B* in a box) appears to the right of the diamond marking that line of source code.
- The program counter glyph indicates the line of code that will execute next when you run your program. The first line of that code is also selected.

You're likely to spend much of your time in the Source view when you're debugging an application.

The local menu

From this view, you can do the following things:

- Set, delete, enable, and disable breakpoints
- Run or jump to the current insertion point position
- Add datapoints and watches
- Inspect variables and expressions
- Move around in the source code by searching, moving to an address or line number, or returning to the program counter

All these choices are available from the Source view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Source view, without actually displaying the menu.

For more information on the Source view, click in the view, then press *F1* to display Help or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

The Disassembly view

Choosing View | Disassembly from the Control Panel displays the Disassembly view.

This view shows your disassembled source code. You use it to see the assembly language instructions that correspond to your source code. You must use this view if the program you're debugging wasn't compiled with debugging information or doesn't have source code available.

- If you set a breakpoint on a line of disassembled code, a breakpoint glyph (a blue *B* in a box) appears to the left of the line of code.
- The program counter glyph indicates the instruction that will execute next when you run your program. That line of code is also selected.

The local menu

In this view you can do things such as the following:

- Set, delete, enable, and disable breakpoints
- Run or jump to the current insertion point position
- Move around in the code by moving to an address or jump target, or returning to the program counter

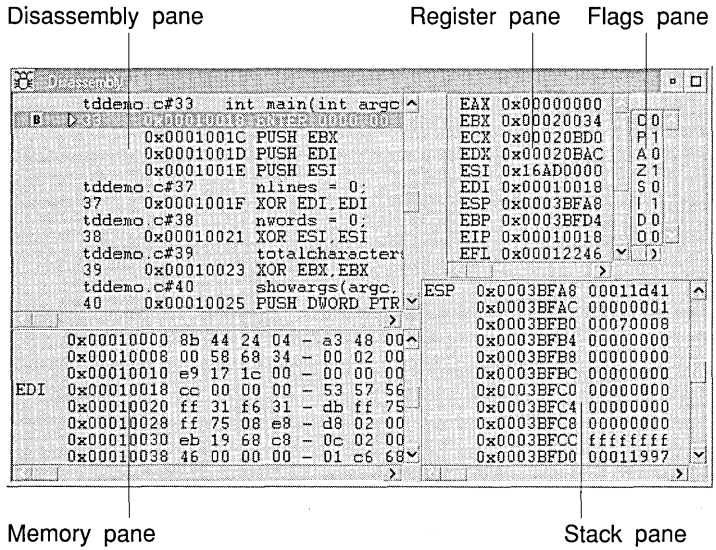
All these choices are available from the Disassembly view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Disassembly view, without actually displaying the menu.

Displaying panes

The Disassembly view can represent five views as panes. By default, this view has one pane, the Disassembly pane. Using the Display Options local

menu choice (*Ctrl+O*), you can add a Memory pane, a Stack pane, a Registers pane, and a Flags pane.

Figure 2.7
The five panes of the Disassembly view



The local menu that comes up for all these panes is the Disassembly local menu. If you want to perform a pane-specific task that is only available from the corresponding view's local menu (such as clearing register ESP in the Register pane), open the corresponding view and perform the task there.

Note, however, that many pane-specific tasks can be performed directly in the Disassembly view without using the local menu. For example, to change a register value, you can double-click the register and enter the new value in the dialog box that appears. You can perform this type of task in the pane without opening the associated view.

For more information on the Disassembly view, click in the view, then press *F1* to display Help or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

The Modules view

Choosing View | Modules from the Control Panel displays the Modules view.

Use this view to display source modules in addition to the one currently displayed in the Source view. This view initially displays the name of your application's .EXE file and any DLLs used by your .EXE.

Each DLL or .EXE has a ⊞ preceding it. Click this icon to see all the source modules for the DLL or .EXE. If you double-click one of the modules, the

debugger loads it into the Source view, where you can do things like set control points, set watches on expressions, and so on.

If you want to see more than one module at a time, double-click on the module you want to examine in the Module view, and a new Source view will open with the desired source code.

For more information on the Modules view, click in the view, then press *F1* to display Help, or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

The Evaluator view

Choosing View | Evaluator from the Control Panel displays the Evaluator view.

Use this view to change the values of variables and expressions and to evaluate expressions that cause side effects (like function calls). It's especially useful for changing the values of complex variables (like *letterinfo* from TDDEMO).

The local menu

In this view you can do the following things:

- Enter a new expression to evaluate
- Inspect the value of an expression (open an Inspector view on the current expression)
- Display the stack concurrently with the expression (a Display Option—*Ctrl+O*)

These choices are available from the Evaluator view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Evaluator view, without actually displaying the menu.

For more information on the Evaluator view, click in the view, then press *F1* to display Help or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

The Inspector view

Choosing View | Inspector from the Control Panel menu bar displays the Inspector view. You can also display this view by choosing Inspect Value from the local menu of the Source view, the Variable view, or the Watch view (or pressing *Ctrl+I* in any of these views). Note that in the Source view, you must have the insertion point on the variable you want to inspect.

Use this view to display or change the current value of a selected variable or expression. Double-clicking on a variable or expression in the Source view automatically displays it in the Inspector view. The Inspector is useful

for taking a quick look at a variable or expression or seeing the elements of a complex variable or expression. You can also use the Inspector to change the value of a simple variable (or a single element of a complex variable).

The local menu

In this view you can do the following things:

- Enter a new expression
- Change the value of an expression
- Show type information
- Change the form of the data display for an expression

All these choices are available from the Inspector view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Inspector view, without actually displaying the menu.

For more information on the Inspector view, click in the view, then press *F1* to display Help or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

The Variable view

Choosing View | Variable from the Control Panel displays the Variable view.

Use this view to display a list of variables whose values you want to see. (To change which variables display, press *Ctrl+O* or choose Display Options from the local menu.)

The local menu

In this view you can do the following things:

- Change between list form and detail form
- Inspect a value
- Add a watch
- Add a datapoint

All these choices are available from the Variable view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Variable view, without actually displaying the menu.

The List and Detail views

This view also has a Detail view. To see details on a variable, first select the variable in the List view, then press *Ctrl+S* to display the Detail view.

The Detail view shows the variable's address, its type, and its value. If the variable is a complex type, such as an array or structure, you also see a list showing each element and its value.

While in the Detail view, you can see details on other variables in the list by clicking the drop-down button to the right of the variable name (in the combo box at the top of the Detail view), then choosing a variable from the list that appears.

For more information on the Variable view itself, click in the Variable view, then press *F1* to display Help. For information on viewing variables, see the Help topic "Viewing Program Data in the Variable View" or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

The Watch view

Choosing View | Watch from the Control Panel displays the Watch view. You can also display this view by choosing Set | Add Watch from the menu bar or choosing Add Watch from the local menu of the Source view or the Variable view (or pressing *Ctrl+W* in either of these views). Note that in the Source view, you must have the insertion point on a variable.

You use this view to track the values of variables and expressions as they change, or to change their values yourself. Using this view, you can watch more than one expression or variable at a time and get a quick picture of what's going on in your application.

The local menu

In this view you can do the following things:

- Change between list form and detail form
- Add, remove, or disable a watch
- Change the value of a variable or expression
- Inspect the value of a variable or expression

All these choices are available from the Watch view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the menu directly from the Watch view, without displaying the menu.

The List and Detail views

Press *Ctrl+S* to change view forms.

This view has two forms, a List view form and a Detail view form. The List view shows all variables and expressions you are watching. The Detail view shows details about a particular variable or expression and allows you to change its value in memory. Note: you can't change the value of a constant expression.

For more information on the Watch view, click in the view, then press *F1* to display Help or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

The Call Stack view

Choosing View | Call Stack from the Control Panel displays the Call Stack view.

This view shows the current state of the stack. If you haven't run your application yet, no routines are listed. You can add this view to the Disassembly view as a pane. You can also open a stack pane in several other views, such as: Variable, Watch, Evaluator, Inspector, and Memory.

For more information on the Call Stack view, click in the view, then press *F1* to display Help or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

The Heap view

Choosing View | Heap from the Control Panel displays the Heap view.

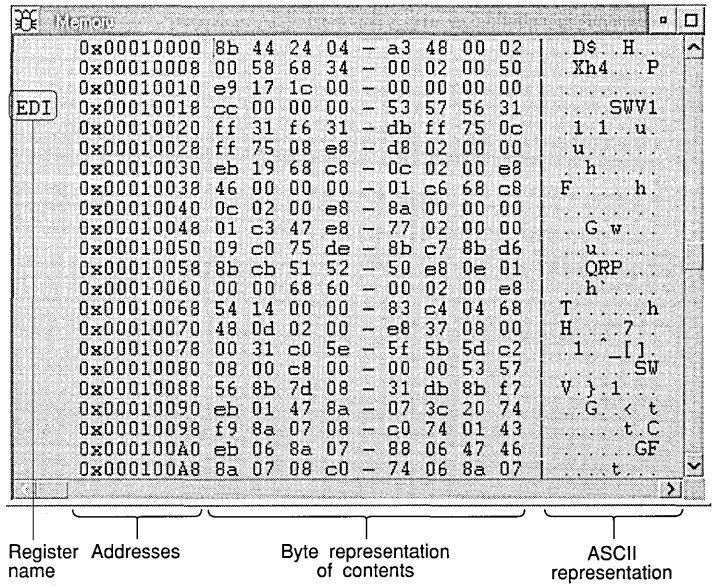
Use this view to look at your application's heap. The Heap view represents each memory object in the heap as a line in the Heap view. For each object there is an index (a line number), an address where the object starts, the object's size in bytes, and an indication of whether the object is being used.

For more information on the Heap view, click in the view, then press *F1* to display Help, or choose Help from the view's local menu. For information on looking at memory, see the Help topic "Viewing Memory". For information on a menu choice, select it and press *F1*.

The Memory view

Choosing View | Memory from the Control Panel displays the Memory view.

Figure 2.8
The Memory view



Use this view to look at the contents of memory. When it first opens, you see memory contents displayed as hexadecimal bytes and their ASCII representation at the right side of the window. You can change the form of the display (for example, to **short**) using the local menu Display Options choice (press *Ctrl+O*).

You can also display this view as a pane in the Disassembly view. (See page 31 for a description.)

The local menu

In this view you can do the following things:

- Go directly to an address
- Search memory for an expression
- Clear an area of memory
- Move an area of memory to another location (nondestructive copy)
- Change the contents of an area of memory
- Read an area of memory into a file
- Write from a file into an area of memory
- Go to an area of memory indicated by the four bytes at the current text selector location

All these choices are available from the Memory view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you

see on the local menu directly from the Memory view, without actually displaying the menu.

For more information on the Memory view, click in the view, then press *F1* to display Help, or choose Help from the view's local menu. For information on looking at memory, see the Help topic "Viewing Memory." For information on a menu choice, select it and press *F1*.

The Numeric Processor view

Choosing View | Numeric Processor from the Control Panel displays the Numeric Processor view.

Use this view to look at or change the state of the numeric processor. You must instruction-step through code that uses the numeric processor in order to see anything meaningful in this view since the numeric stack is usually left clean at the end of each high-level statement. This view indicates

- Contents of the registers
- Control word and control flag settings
- Status word and status flag settings
- NPX Tag word
- Addresses pointed to by the instruction and data pointers
- Current instruction being executed

You can do the following in this view:

- Change control flag values (*Ctrl+G* or double-click)
- Change status flag values (*Ctrl+S* or double-click)
- Change register values (use the entry field)
- Change the value of the control or status word (use the entry field)
- Change the value of the NPX tag word (use the entry field)
- Choose hexadecimal or decimal as the display form (*Ctrl+O*)

For more information on the Numeric Processor view, click in the view, then press *F1* to display Help or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

The Register view

Choosing View | Register from the Control Panel displays the Register view.

Use this view to look at the contents of the CPU registers and flags. You can also display the different panes in this view in the Disassembly view. (See page 31 for a description.)

The local menu

In this view you can do the following things:

- Change a register value
- Clear, increment, or decrement a register
- Toggle the value of a flag

All these choices are available from the Register view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Register view, without actually displaying the menu. In addition, you can double-click a register or a flag to change its value.

For more information on the Register view, click in the view, then press *F1* to display Help, or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

The C++ exception stack view

The C++ exception stack view displays the state of all pending C++ exceptions. This view displays the most recent C++ exception thrown as the first entry in the list.

Use this view to control how *events* associated with C++ exceptions are handled by the debugger. Events are C++ exception catches or destructor invocations caused by stack unwinding. The check boxes for each entry in this view controls whether or not the debugger stops for destructors or catches for the C++ exception throw associated with the entry. By default, the debugger stops on both of these events.

The File view

Choosing View | File from the Control Panel displays the File view.

Use this view to look at files that do not contain Debug information. Typically, you'll use this view on files you can't load into the Source view, such as header files or source files from programs other than the one you're debugging. The default form is hexadecimal bytes with ASCII displayed on the right side. You can change to ASCII form by pressing *Ctrl+O* and choosing Show ASCII.

The local menu

In this view you can do the following things:

- Go to a location in the file by entering a C expression (such as a string in quotation marks)
- Change display form (hexadecimal with ASCII on the side is the default)

These choices are available from the File view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the File view, without actually displaying the menu.

For more information on the File view, click in the view, then press *F1* to display Help, or choose Help in the view's local menu. For information on a menu choice, select it and press *F1*.

The Log view

Choosing View | Log from the Control Panel displays the Log view.

Use this view to examine the event log. The event log is a dynamic listing of the control points that your program encounters during execution. To indicate that a control point is to be logged, check the Log Expression check box on a control point's Detail view.

For example, to log a breakpoint, choose Set | Breakpoint, enter the breakpoint information, and then check the Log Expression check box at the bottom of the Breakpoint detail view. (This check box is also on the detail views for messagepoints, datapoints, and exceptionpoints.) Whenever your program encounters this breakpoint, the debugger logs its action to the event log.

The local menu

In this view you can perform the following actions:

- Erase the contents of the event log
- Open a log file to store the contents of the event log
- Disable and enable event logging

All these choices are available from the Log view local menu (right-click or press *Ctrl+F10* to display it). You can also use the shortcut keys you see on the local menu directly from the Log view, without actually displaying the menu.

For more information on the Log view, click in the view, then press *F1* to display Help, or choose Help from the view's local menu. For information on a menu choice, select it and press *F1*.

C H A P T E R 3

A quick example

This chapter gives you enough information to debug your first program. Once you've learned the basic concepts described here, the graphical environment and context-sensitive Help system assist you in learning as you go along.

This chapter leads you through the basic features of Turbo Debugger GX. After describing the demo program, it shows you how to do the following procedures:

- Run and stop program execution
- Examine the contents of program variables
- Look at complex data objects, such as arrays and structures
- Change the value of variables

The demo program

This tutorial uses the TDDEMO.C demo program to introduce the two main things you need to know to debug a program: how to stop and start your program, and how to examine your program's variables and data structures. The demo program itself isn't meant to be very useful—some of its code and data structures exist solely to show you the capabilities of Turbo Debugger GX.

The demo program prompts you for lines of text, then counts the number of words and letters you entered. It finishes by displaying some statistics about the nature of the text entered, including the average number of words per line and the number of times each letter occurred.

Make sure your current directory contains the two files needed to debug the demo: TDDEMO.C and TDDEMO.EXE.

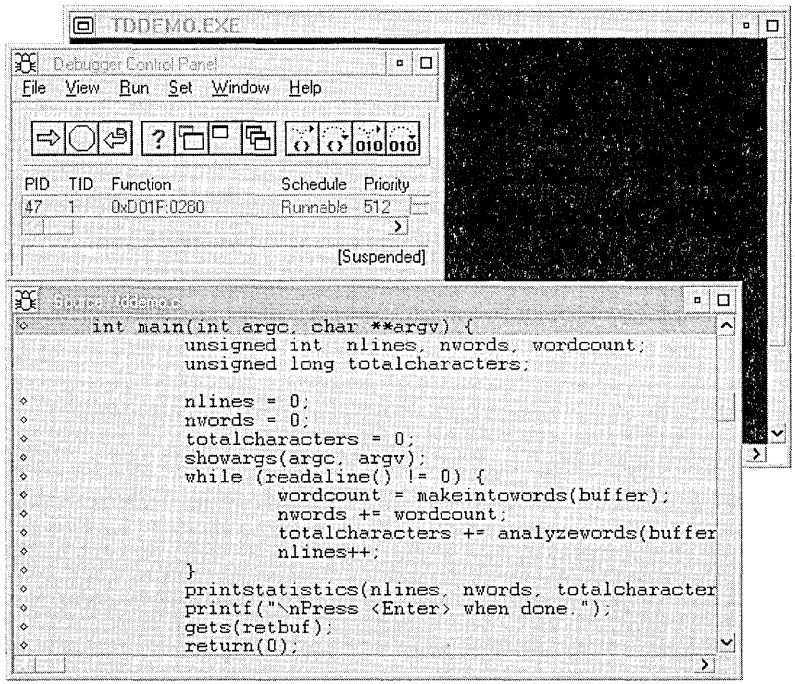
Getting in

To start the program, run the debugger, then:

1. Choose File | Load Process from the Control Panel.
2. If necessary, change to the directory containing TDDEMO.C and TDDEMO.EXE. (The default directory is \BORLANDC\EXAMPLES\TD.)
3. Enter TDDEMO.EXE as the file to open, then click OK.

Turbo Debugger GX loads the demo program, opens the Source view and the application's Program Window, and positions the text selector in the Source view at the start of the program.

Figure 3.1
Debugger views after loading TDDEMO



By default, the debugger doesn't run your program's startup code. You can change this default setting in the Properties dialog box by specifying **main** in Run To __ On Startup and checking that button to enable it, or by running the debugger with the **–r** command-line switch (see Table 1.1 on page 9).

The application's user screen appears with TDDEMO because TDDEMO is a character-mode program that requires an OS/2 window, which OS/2 starts automatically before the program is run. If TDDEMO were a

Presentation Manager program (like TDDEMOPM), the user screen wouldn't appear until you actually ran the program.

Getting out

To exit from the tutorial and Turbo Debugger GX at any time, press *Alt+X*. If at any point you want to reload the program and start at the beginning, press *Ctrl+F2* or click the Reset Program button on the SpeedBar.

Getting Help

Press *F1* whenever you need help with the current view, menu choice or dialog box. You can learn a lot by working your way through the menu system and pressing *F1* at each menu choice to get a summary of what it does. You can also learn a lot by reading the online *User's Guide*, which consists of all the subtopics under the Tasks topic in the Help Contents panel. You can read the *User's Guide* online, taking advantage of its hyperlinks and modular design, or you can print individual topics or sections for reading offline. See Chapter 1 for information about printing Help topics.

Using the debugger

The Control Panel's menu bar, SpeedBar, and status line, and the various views and their local menus are the keys to using the debugger effectively. For more information, see Chapter 2, "The Turbo Debugger GX environment."

Using TDDEMO

If you haven't loaded TDDEMO yet, do so now. The text selector in the Source view is on the first executable line of your program, the **main** function. Since you haven't run your program yet, the program counter doesn't show. Press the SpeedBar's Statement Step Into button (or *F7*) to run the startup code for the program. The program counter now appears to the left of **main**, indicating that the debugger has run the startup code and is ready to start execution with this line.

Ctrl O

Look at the left margin of the Source view. You see diamonds indicating lines that generated executable code. To see line numbers, click the Source view, press *Ctrl+O*, and choose Show Line Numbers. Now line numbers appear in the left margin.

To position the text selector on a line in the Source view, press *Ctrl+L*, type the line number, and press *Enter*.

As you can see from the Run menu, there are a number of ways to control the execution of your program. Let's say you want to run the program until it reaches line 40.

F4

F7

First, position the text selector on line 40, then press *F4* to run the program up to (but not including) line 40. Now press *F7*, which executes one line of source code at a time and enters into any functions called; in this case, it

executes line 40, a call to the function **showargs**. The cursor immediately jumps to line 167, where the definition of **showargs** is found.

Ctrl F8

Continuing to press *F7* would step through the function **showargs** and then return to the line following the call—line 41. Instead, press *Ctrl+F8*, which causes **showargs** to execute and then return, at which point the program stops. This command, too, returns to line 41 and is very useful when you want to run past the end of a function.

F8

If you had pressed *F8* (or used the Statement Step Over button on the SpeedBar) instead of *F7* on line 40, the program counter would have gone directly to line 41 instead of into the function. *F8* is similar to *F7* in that it executes a function or source line, but skips any function calls.

Figure 3.2
Program stops on return from function showargs

```
32        */
33 ◆    int main(int argc, char **argv) {
34              unsigned int  nlines, nwords, wordcount;
35              unsigned long totalcharacters;
36
37 ◆            nlines = 0;
38 ◆            nwords = 0;
39 ◆            totalcharacters = 0;
40 ◆            showargs(argc, argv);
41 ◆
42 ◆                    wordcount = makeintowords(buffer);
43 ◆                    nwords += wordcount;
44 ◆                    totalcharacters += analyzewords(buf
45 ◆                    nlines++;
46 ◆            }
47 ◆            printstatistics(nlines, nwords, totalcharac
48 ◆            printf("\nPress <Enter> when done.");
49 ◆            gets(retbuf);
```

Ctrl F9

To execute the program until a specific program location is reached, you can directly name the function or line number, without moving the text selector to that line in a source file and then running to that point. Press *Ctrl+F9* (or choose Run | Execute To from the Control Panel) to specify a label to run to. A dialog box appears. Type `readaline` and press *Enter*. The program runs, then stops at the beginning of function **readaline**.

Setting breakpoints

F2

Another way to control where your program stops running is to set breakpoints. The simplest way to set a breakpoint is with the *F2* key. Move the text selector to line 45 and press *F2*. Turbo Debugger GX puts a blue *B* in a box to the left of the line, indicating there is a breakpoint set on it.

You can also use the mouse to toggle breakpoints by clicking near the diamond to the left of a line of source code.

Figure 3.3
A breakpoint set at line 45

```
35                  unsigned long totalcharacters;
36
37 ◇                nlines = 0;
38 ◇                nwords = 0;
39 ◇                totalcharacters = 0;
40 ◇                showargs(argc, argv);
41 ◇                while (readaline() != 0) {
42 ◇                        wordcount = makeintowords(buffer);
43 ◇                        nwords += wordcount;
44 ◇                        totalcharacters += analyzewords(buf
45 ◇ B                      printline;
46 ◇ ≡              }
47 ◇                printstatistics(nlines, nwords, totalcharac
48 ◇                printf("\nPress <Enter> when done.");
49 ◇                gets(retbuf);
50 ◇                return(0);
51          }
52
```

Notice the small box containing horizontal lines to the left of line 46. This stack glyph indicates the next line that will execute after a return from a procedure call. It appears at the end of this **while** loop because you previously ran the program to **readaline**, which is called by this **while** statement.

F9

Now press *F9* to execute your program without interruption. The focus switches to the program's display. The demo program is now running and waiting for you to enter a line of text. Click the application window, and type `abc`, a space, `def`, and then press *Enter*. The display returns to the Source view with the arrow on line 45, where your breakpoint has stopped the program.

Ctrl E

Now press *Ctrl+E* to disable the breakpoint. You see the capital *B* change to a lowercase *b*, indicating that the breakpoint is still set (preserving any filters, conditions, and actions) but is disabled.

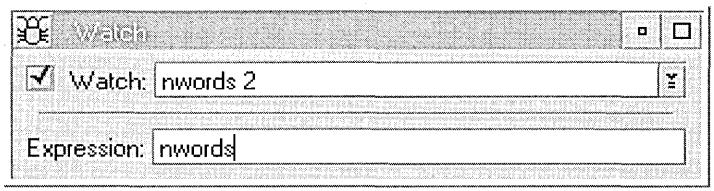
See page 25 for more information on breakpoints. The Online Help also provides a complete description of setting and using breakpoints, under "Tasks."

Using watches

Ctrl F7

The Watch view shows the value of variables you specify. For example, to watch the value of the variable *nwords*, move the text selector to the variable name on line 43, choose Add Watch from the Source view local menu (or press either *Ctrl+F7* or *Ctrl+W*), then press *Enter* to accept that expression.

Figure 3.4
A variable in the Watch view



The symbol *nwords* now appears in the Watch view, along with its value. As you execute the program, Turbo Debugger GX updates this value to reflect the variable's current value.

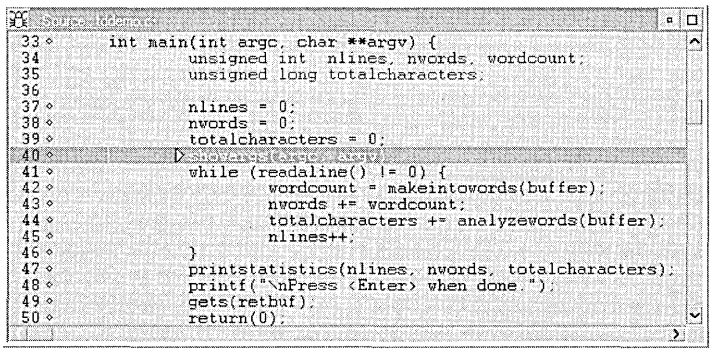
If you pass out of the variable's scope (for example, if you continue statement-stepping-into and step into the **readaline** function), the Watch view shows the variable as undefined. As soon as the variable is back in scope (for example, you statement-step through **readaline**, enter another line of characters, and step back into the **while** loop containing *nwords*), you can see its value again.

Examining simple C data objects

Once you have stopped your program, there are a number of ways of looking at data using the Inspector view. This facility lets you examine data structures in the same way you visualize them when you write a program.

With the Inspector view (available from all local menus and from the View menu), you can examine any variable you specify. Suppose you want to look at the value of the variable *nlines*. Double-click *nlines* in the Source view: an Inspector view pops up with *nlines* in it.

Figure 3.5
An Inspector window



The address, type, and name of the variable are listed on the first line and its value on the second. Because *nlines* has been optimized into a register variable, its address is the EDI register.

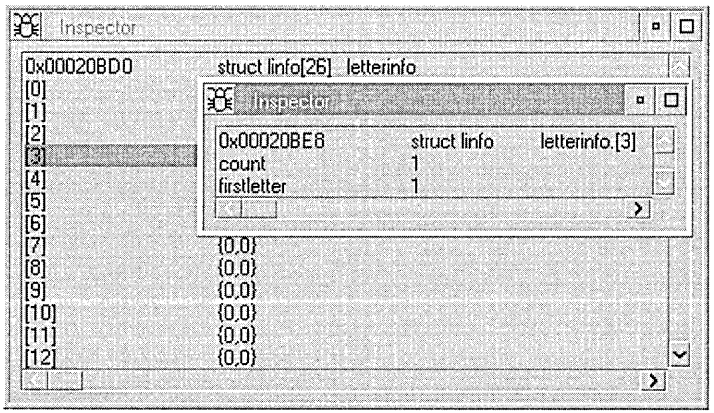
To examine a data item that isn't conveniently displayed in the Source view, choose View | Inspector. The Inspector view appears, asking you to enter the expression to inspect. Type `letterinfo` and press *Enter*. The

Inspector view lists the values of the *letterinfo* array elements. The first line of the list shows the address, type, and name of the data you're inspecting. Scroll through the 26 elements that make up the *letterinfo* array. The next section shows you how to examine this compound data object.

Examining compound data objects

A compound data object, such as an array or structure, contains multiple components. Double-click the fourth element of the *letterinfo* array (the one indicated by `[3]`). A new Inspector view appears, showing the contents of that element in the array.

Figure 3.6
Inspecting a structure



When you double-click one of the member names, it appears in yet another Inspector view. If one of these members was in turn a compound data object, you could double-click it and dig down further into the data structure.

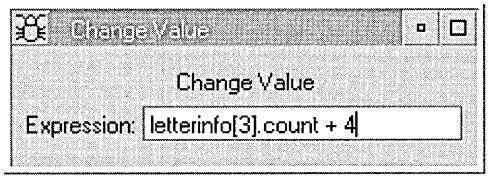
Now return to the Source view by clicking on it.

Changing data values

So far, you've learned how to look at data in the program. Now you'll see how to *change* the value of data items.

Use the mouse to go to line 39 in the source file. Double-click the variable *totalcharacters* to inspect its value. With the Inspector window open, right-click to bring up the Inspector's local menu, then choose the Change Value option. (You could also have done this directly by pressing *Ctrl+G*.) A dialog box appears, asking for the new value.

Figure 3.7
The Change Value dialog box

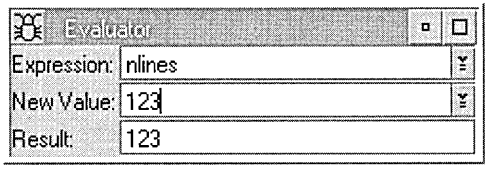


At this point, you can enter any C expression that evaluates to a number. Type `totalcharacters + 4` and press *Enter*. The value in the Inspector window now shows the new value, `10`.

You can also use the Inspector view to change the value of a structure or array member. For example, if you double-click *letterinfo*, then double-click the fourth element of the array (element *[3]*), you get an Inspector view for *count* and *firstletter*. If you double-click *count*, you get an Inspector view for that single element. If you then press *Ctrl-G*, you can change the value of that element.

To change a data item that isn't displayed in the Source view, choose View | Evaluator (or press *Ctrl+F4*). A dialog box appears. In the Expression entry field, enter the name of the variable to change. Type `nlines`, press *Enter*, then press *Tab* to move to the New Value entry field. Type `123` and press *Enter*. The Result field shows `123`.

Figure 3.8
The Evaluator view



You can use the Evaluator view to change values of complex data types, but you can change only one element at a time. For example, to change the fourth element of *letterinfo* (*letterinfo[3]*), you must bring up the Evaluator and change *letterinfo[3].count* first, then change *letterinfo[3].firstletter*.

Conclusion

That's a quick introduction to using the Turbo Debugger GX with a character-mode program written using Borland C++ for OS/2. If you're interested in Presentation Manager debugging, try using the demo program TDDEMOPM and playing around both with the features mentioned in this chapter and with the Messagepoint and Exceptionpoint views.

Another view you might find useful for general debugging is the Disassembly view, which shows disassembled code and can simultaneously show you registers, flags, the stack, and memory contents. You can also use this view on code you have no source or no debugging

information for. (Note that you have to select the local menu display options to select these displays.) See page 31 for more information, or see the Online Help topic "Disassembly View."

For more information on debugging tasks, see the Online Help topics "Essentials" and "Tasks." For more information on the Turbo Debugger GX environment, see Chapter 2, "The Turbo Debugger GX environment" or the Online Help topics "Menus" and "Views."

A P P E N D I X A

Turbo Debugger GX for experienced Turbo Debugger users

The following table lists major tasks you can perform when you're debugging, and shows you the commands or keystrokes to accomplish each task in Turbo Debugger for DOS (or Windows) and in Turbo Debugger GX.

Table A.1: Turbo Debugger GX task list

Task	TD for DOS or TDW	Turbo Debugger GX
Access the local menu of a view	Right mouse click or *Ctrl+F10*	Right mouse click or *Ctrl+F10.*
Add a breakpoint and change characteristics	Breakpoints|At (*Alt+F2*) View|Breakpoints|Add	Set|Breakpoint. View|Breakpoint.
Animate	Run|Animate	Run|Animate.
Back Trace	Run|Back Trace (*Alt+F4*)	No equivalent.
Change directories	File|Change Dir	File|Load Process.
Check the value that a function is about to return	Data|Function Return	No equivalent.
Close a file	File|Open	File|Unload Process.
Copy from current window to Log window	Edit|Copy to Log Edit|Dump Pane to Log	No equivalent.
Delete all breakpoints	Breakpoints|Delete All	Breakpoint view local menu| All Breakpoints|Remove all Breakpoints.
Display Breakpoints view	View|Breakpoints	View|Breakpoint.
Display CPU view	View|CPU	View|Disassembly. View|Call Stack. View|Memory. View|Register.
Display Dump view	View|Dump	View|Memory.

Table A.1: Turbo Debugger GX task list (continued)

Display Execution History view	View|Execution History	No equivalent.
Display File view	View|File	View|File.
Display Log view	View|Log	View|Log.
Display Module (source) view	View|Module (*F3*)	View|Source.
Display Numeric Processor view	View|Numeric Processor	View|Numeric Processor.
Display Registers view	View|Registers	View|Register.
Display Stack view	View|Stack	View|Call Stack.
Display Variables view	View|Variables	View|Variable.
Display Watches view	View|Watches	View|Watch.
Display Windows Messages view	View|Windows Messages	View|Messagepoint.
Enable/Disable all breakpoints	No equivalent	Breakpoint view local menu| All Breakpoints|Enable/ Disable All Breakpoints.
Enable/Disable breakpoint	Not available	With cursor on breakpoint position in Source or Disassembly view, choose local menu|Enable/Disable breakpoint.
Evaluate or modify data	Data|Evaluate/Modify (*Ctrl+F4*)	View|Evaluator (*Ctrl+F4*).
Execute to a specified location	Run|Execute to (*Alt+F9*)	Run|Execute To (*Ctrl+F9*).
Execute until current routine returns	Run|Until Return (*Alt+F8*)	Run|Return from Function (*Ctrl+F8*).
Exit program	File|Exit (*Alt+X*)	File|Exit (*Alt+X*).
Inspect a variable	Position cursor in Module window, then press *Ctrl+I* or choose local menu|Inspect Or, choose View|Inspect	Position cursor in Source view, then double-click, press *Ctrl+I*, or choose local menu|Inspect Value. Or, choose View|Inspector.
Look at window messages returned to application	Windows Messages window, lower pane View|Log (if sent to Log window)	In Messagepoint view details pane, choose Log Expression. Then View|Log to see window messages.
Look at Windows local heap, module list, or global heap	View|Log|(local menu)|Display Windows Info	No exact equivalent. View|Heap and View|Module are similar to Windows local heap and module list. No equivalent for global heap.

Table A.1: Turbo Debugger GX task list (continued)

Open a file	File|Open	File|Load Process.
Pick a module to view	View|Module (*F3*)	View|Module (*F3*).
Reload application program	Run|Program Reset (*Ctrl+F2*)	Run|Reset (*Ctrl+F2*). Press SpeedBar button.
Run application program	Run|Run (*F9*)	Run|Run (*F9*). Press SpeedBar button.
Run to current location	Run|Go To Cursor (*F4*)	Source View local menu| Run to Here (*F4*).
Set arguments before running application	Run|Arguments	Run|Arguments.
Set characteristics of breakpoint	Breakpoints|At (*Ctrl+F2*) View|Breakpoints|Set options	Set|Breakpoint. Pick it from list of breakpoints.
Set message breakpoint	View|Windows Messages	Set|Messagepoint.
Set watchpoint, tracepoint, or hardware breakpoint	Breakpoints|Changed Mem. Glb. Breakpoints|Expn. True Global Breakpoints|Hdw. Breakpoint	Set|Datapoint.
Step into routine by instructions	Run|Trace Into (*F7*) in CPU window	Run|Instruction Into (*F11*). Press SpeedBar button.
Step into routine by statements	Run|Trace Into (*F7*) in Module window	Run|Statement Into (*F7*). Press SpeedBar button.
Step over routine by instructions	Run|Step Over (*F8*) in CPU window	Run|Instruction Over (*F12*). Press SpeedBar button.
Step over routine by statements	Run|Step Over (*F8*) in Module window	Run|Statement Over (*F8*). Press SpeedBar button.
Toggle breakpoint on and off at cursor in source or assembly language view	Breakpoints|Toggle (*F2*)	*F2* or double-click (if on left line marker). Local menu|Set/Remove Breakpoint.
Watch data	Data|Add Watch (*Ctrl+F7*) View|Watches Or, in Module window, press *Ctrl+W* or choose Watch from local menu	Set|Add Watch (*Ctrl+F7*). View|Watch or, in Source view, press *Ctrl+W* or choose Add Watch from local menu.

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Borland

Corporate Headquarters: 100 Borland Way, Scotts Valley, CA 95066-3249, (408) 431-1000. Offices in: Australia, Belgium, Canada, Chile, Denmark, France, Germany, Hong Kong, Italy, Japan, Korea, Latin America, Malaysia, Netherlands, New Zealand, Singapore, Spain, Sweden, Taiwan, and United Kingdom • Part # BCP1415WW21774 • BOR 7004