WorkShop: Getting Started

THE NETWORK IS THE COMPUTER™

SunSoft, Inc.
A Sun Microsystems, Inc. Business
2550 Garcia Avenue
Mountain View, CA 94043 USA
415 960-1300 fax 415 969-9131

Part No.: 802-5659-10
Revision A, December 1996

Contents

Preface

This purpose of this manual to help you become familiar with WorkShop by learning how to perform basic developing operations through WorkShop. In addition to the overview of WorkShop features, this guide show you how to:

- Select a default text editor
- Run a build job
- Debug a program
- Browse source code
- Analyze performance data
- Set some WorkShop resources

Who Should Use This Book

This manual is written for application developers who want to use the main development features of WorkShop. For a discussion of advanced programming topics, see *WorkShop: Beyond the Basics*. For a discussion of WorkShop features with command-line implementations, see *WorkShop: Command-Line Utilities*.

How This Book Is Organized

WorkShop: Getting Started contains the following chapters:

Chapter 1, "About Sun WorkShop," is an overview of the WorkShop programming environment. This chapter also highlights the components available in the various WorkShop products.

Chapter 2, "Starting Sun WorkShop," explains what you need to do to start developing in WorkShop, such as how to start WorkShop, how to select a text editor, and what WorkSets are and how to use them,

Chapter 3, "Building Programs in Sun Workshop," shows you how to build an application with WorkShop default settings or your own build settings, and how to fix build errors.

Chapter 4, "Debugging A Program," highlights the many debugging features offered in WorkShop, and explains how to debug in quick mode. Quick Mode allows you to run your program normally, but keeps debugging ready in the background to take over the process at any time.

Chapter 5, "Browsing Source Code," shows you how to use the Browsing window, the Call Grapher, the Class Grapher, and the Class Browser to examine source files, function call relationships, and class hierarchies.

Chapter 6, "Analyzing Program Performance," explains how to gather and examine the various types of data with the Behaviour Data Collector and the Analyzer to improve the performance of an application.

Chapter 7, "Merging Source Files," shows you how to compare different versions of a source file and merge the changes.

Appendix A, "Setting WorkShop Resources," shows you how to modify the some of the resource settings in WorkShop.

How to Get Help

This release of the WorkShop includes a new documentation delivery system as well as online manuals and video demonstrations. To find out more, you can start in any of the following places:

- **Online Help** – A new help system containing extensive task-oriented, context-sensitive help. To access the help, choose Help ➤ Help Contents. Help menus are available in all WorkShop windows.
- **WorkShop Documentation** – A complete set of online manuals. These manuals make up the complete documentation set for WorkShop and are available using AnswerBook™ or (at the user's option) using the Netscape™ browser. To access the online manuals, choose Help ➤ WorkShop Manuals in the main WorkShop window.

- **Video Demonstrations** – These demos provide a general overview of the WorkShop and describe how to use WorkShop to build targets or debug programs. To access them, choose Help ➤ Demos in the WorkShop main window.
- **Release Notes** – The Release Notes contain general information about WorkShop and specific information about software limitations and bugs. To access the Release Notes, choose Help ➤ Release Notes.

How to Access the AnswerBook Documentation

To access the AnswerBook online documentation for WorkShop, you must run a script to set up your environment.

1. **To start AnswerBook, type the following at a command prompt:**

   ```
   % workshop-answerbooks
   ```

   The script sets the `AB_CARDCATALOG` environment variable and runs `/usr/openwin/bin/answerbook`. The AnswerBook Navigator opens and displays the available AnswerBook documents.

2. **Add the WorkShop AnswerBook documents to your library by clicking the Modify Library button.**
   The AnswerBooks Navigator: Modify Library window is displayed.

3. **Select the AnswerBook documents you wish to add to your library from the list; then click the Apply button.**
   The AnswerBook documents are added to your library.

4. **To view an AnswerBook document, double-click on the title you wish to view.**

Related Books

The Sun WorkShop provides comprehensive documentation. Depending on which version of WorkShop you have, the following books are available in online and printed forms (except where noted). Some documents are available with all WorkShop products, others are not (as noted).

Sun WorkShop Documentation

Available with all WorkShop products.

WorkShop *Roadmap*	(hard copy only) Provides a documentation map to the WorkShop printed and online documentation. Includes a complete list of the documentation included with your WorkShop.
WorkShop Installation and Licensing Guide	Provides instructions about product licensing and installation of Workshop products on Solaris™ 2.x systems. Provides instructions for local or remote installation for single independent license servers, multiple independent license servers, and redundant license servers.
WorkShop Quick Install for Solaris	Provides quick installation instructions for product installation and licensing.
WorkShop: Getting Started	Provides a basic introduction. This book provides the information you need to use the basic WorkShop features.
WorkShop: Beyond the Basics	Contains information about the advanced programming, debugging, browsing, and visualization applications in the WorkShop product suite, including: DMake, LoopTool, Thread Analyzer, WorkShop Browsing, and WorkShop Visual.
WorkShop: Command Line Utilities	Provides reference information for all of the workshop utilities that can be run directly from the command line, including Loop Report, LockLint Utilities, sbquery, and all the dbx commands.
Using XEmacs	(online only) Contains user-level information for XEmacs.
WorkShop Online Help	(online only) Contains extensive task-oriented information for all the tools included with the WorkShop
WorkShop Video Demonstrations	(online only) Three video demonstrations providing information about WorkShop building and debugging as well as general product information.
Release Notes	(online only) Contains any information that was too late to get into the other documentation. To access the Release Notes, open any Help menu and then click on Release Notes.
Manual Pages	(online only) Provide information about the WorkShop command-line utilities.

Sun WorkShop TeamWare Documentation

Available only with Performance WorkShop Fortran and Visual WorkShop C++.

Sun WorkShop TeamWare: User's Guide	Describes how to use all the tools in the TeamWare tool set, for both the command-line interface and the graphical user interface.
Sun WorkShop TeamWare: Solutions Guide	Provides an in-depth case study and eight scenario-based topics to help users take full advantage of TeamWare's features.
Sun WorkShop TeamWare: Online Help	Provides succinct task-oriented information to help you become familiar with the application. Help volume includes video overview.
Manual Pages	(online only) Provide information about the TeamWare command-line commands and utilities.

Sun Visual WorkShop C++ Documentation

Available only with Sun Visual WorkShop C++.

WorkShop documentation	Visual WorkShop C++ contains the entire WorkShop and TeamWare documentation sets.
C++ User's Guide	Describes how to use the Sun C++ compiler to write programs in C++. It covers the C++ compiler options, programs, templates, exception handling, and more. It is intended for the experienced C++ programmer.
C++ Library Reference	Describes how to use the complex, coroutine, and iostream libraries, and it lists the manual pages (man pages) for these libraries.
Tools.h++ User's Guide	Describes how to use the Tools.h++ libraries to make programs more efficient.
Tools.h++ Class Library Reference	Describes how to use the Tools.h++ class library, and also describes a set of C++ classes that can simplify programming while maintaining efficiency.
C++ 4.2 Quick Reference Card	Provides concise descriptions of the C++ compiler flags.
C User's Guide	Describes how to use the Sun ANSI C compiler to write programs in C. It covers the C compiler options, the pragmas, the lint tool, the cscope tool, and more. It is intended for the experienced C programmer.
Making the Transition to ANSI C	(online only) Provides information about the transition from K&R C to ANSI C.
C 4.2 Quick Reference Card	Provides concise descriptions of the C compiler flags.

WorkShop: Visual User's Guide	Explains how to use Visual, an interactive tool for building graphical user interfaces (GUIs) using the widgets of the standard OSF/Motif toolkit or Microsoft Foundation Class. It includes a tutorial as well as reference information for the more advanced user.
Sun WorkShop 2.0 Visual Quick Reference Card	Contains menu shortcuts and icon explanations for Visual.
Numerical Computation Guide	Describes the floating-point software and hardware for the SPARC™, Intel, HP 700, and PowerPC system architectures. It also contains a tutorial on floating-point arithmetic.
Incremental Link Editor	Describes how to use `ild` as an incremental linker to replace `ld` for linking programs. `ild` allows you to complete the development sequence more quickly than is possible with a standard linker.
Performance Profiling Tools	Describes the `prof`(1), `gprof`(1), and `tcov`(1) utilities
Manual Pages	(online only) Provides information about the command-line commands and utilities included with Visual WorkShop C++.

Sun Performance WorkShop Fortran Documentation

Available only with Sun Performance WorkShop Fortran.

WorkShop documentation	Performance WorkShop Fortran contains the entire WorkShop and TeamWare documentation sets.
Fortran User's Guide	Describes how to use the Sun Fortran 77 4.0 and Fortran 90 1.2 compilers, including the compiler command options, debugging and development tools, program profiling and performance tuning, mixing C and Fortran, and making and using libraries. It is intended for programmers with knowledge of Fortran.
FORTRAN 77 Language Reference	Describes and defines the Fortran 77 language accepted by the Sun `f77` compiler under Solaris 1.x and 2.x. It is intended for use by programmers with knowledge of and experience with Fortran.
Fortran Programmer's Guide	Provides the essential information programmers need to develop efficient applications using the Fortran 77 and Fortran 90 compilers. Includes information on input/output, program development, use and creation of software libraries, program analysis and debugging, numerical accuracy, porting, performance, optimization, parallelization, and the C/Fortran interface.

Fortran Library Reference	(online only) Describes the language and routines of the Fortran compilers.
Sun Performance Library 1.2 Quick Reference Card	Provides a quick reference to Sun Performance Library language routines with brief descriptions.
Fortran 90 Handbook	(online only) Contains user-level information about this release of Fortran90.
Fortran 90 Browser	Describes how to use the Sun Fortran 90 Browser, one of the development tools in the `f90` package, to view Fortran 90 source code. It is intended for programmers with knowledge of Fortran 90.
Fortran Quick Reference Card	Lists the `f77` 4.0 compiler's command-line options with brief descriptions.
C User's Guide	Describes how to use the Sun ANSI C compiler to write programs in C. It covers the C compiler options, the pragmas, the lint tool, the cscope tool, and more. It is intended for the experienced C programmer.
Making the Transition to ANSI C	(online only) Provides information about the transition from K&R C to ANSI C.
C 4.2 Quick Reference Card	Describes the C compiler options in a concise and easy-to-read format.
Numerical Computation Guide	Describes the floating-point software and hardware for the SPARC™, Intel, HP 700, and PowerPC system architectures. It also contains a tutorial on floating-point arithmetic.
Incremental Link Editor	Describes how to use `ild` as an incremental linker to replace `ld` for linking programs. `ild` allows you to complete the development sequence more quickly than is possible with a standard linker.
Performance Profiling Tools	Describes the `prof`(1), `gprof`(1), and `tcov`(1) utilities.
Manual Pages	(online only) Provide information about the Fortran command-line commands and utilities included with Performance WorkShop Fortran.

Sun WorkShop Professional Pascal Documentation

Available only with Sun WorkShop Professional Pascal.

WorkShop documentation	WorkShop Professional Pascal contains the entire WorkShop documentation set.

Pascal User's Guide	Describes how to begin writing and compiling Pascal programs for the Solaris computing environment. Pascal is a derivative of the Berkeley Pascal system distributed with UNIX®4.2 BSD. It complies with FIPS PUB 109 ANSI/IEEE 770 X3.97-1983 and BS6192/ISO7185 at both level 0 and level 1, and it includes many extensions to the standard.
Pascal Language Reference	Provides reference material for the Pascal 4.0 compiler, an implementation of the Pascal language that includes all the standard language elements and many extensions. Pascal 4.0 contains a compiler switch, `-xl`, to provide compatibility with Apollo DOMAIN Pascal to ease the task of porting your Apollo Pascal applications to workstations.
Pascal 4.2 Quick Reference Card	Lists all of the Pascal 4.2 compiler options with a brief, one-line description of each option.
Numerical Computation Guide	Describes the floating-point software and hardware for the SPARC™, Intel, HP 700, and PowerPC system architectures. It also contains a tutorial on floating-point arithmetic.
Incremental Link Editor	Describes how to use `ild` as an incremental linker to replace `ld` for linking programs. `ild` allows you to complete the development sequence more quickly than is possible with a standard linker.
Performance Profiling Tools	Describes the `prof`(1), `gprof`(1), and `tcov`(1) utilities.
Manual Pages	(online only) Provide information about the command-line commands and utilities included with WorkShop Professional Pascal.

Sun WorkShop Professional C Documentation

Available with Sun WorkShop Professional C only.

WorkShop documentation	WorkShop Professional C contains the entire WorkShop documentation set.
C User's Guide	Describes how to use the Sun ANSI C compiler to write programs in C. It covers the C compiler options, the pragmas, the lint tool, the cscope tool, and more. It is intended for the experienced C programmer.
Making the Transition to ANSI C	(online only) Provides information about the transition from K&R C to ANSI C.
C 4.2 Quick Reference Card	Describes the C compiler options in a concise and easy-to-read format.

Numerical Computation Guide	Describes the floating-point software and hardware for the SPARC™, x86, HP 700 and PowerPC system architectures. It also contains a tutorial on floating-point arithmetic.
Incremental Link Editor	Describes how to use `ild` as an incremental linker to replace `ld` for linking programs. `ild` allows you to complete the development sequence more quickly than is possible with a standard linker.
Performance Profiling Tools	Describes the `prof`(1), `gprof`(1), and `tcov`(1) utilities.
Manual Pages	(online only) Provides information about the command-line commands and utilities included with WorkShop Professional C.

Ordering Additional Hardcopy Documentation

You can order additional copies of the hard copy documentation by calling SunExpress at 1-800-USE-SUNX or visiting their web page at

`http://sunexpress.usec.sun.com`

Sun on the World Wide Web

World Wide Web (WWW) users can view Sun's Developer Products site at the following URL:

http://sun-www.EBay.Sun.COM:80/sunsoft/Developer-products/ \
products.html

This area is updated regularly and contains helpful information, including current release and configuration tables, special programs, and success stories.

What Typographic Changes Mean

The following table describes the typographic changes used in this book.

Table P-1 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% You have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`machine_name% su` `Password:`
AaBbCc123	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *filename*.
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in *User's Guide*. These are called *class* options. You *must* be root to do this.

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

Table P-2 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

Part 1 — Welcome to WorkShop

About Sun WorkShop 1

This release ofSun WorkShop makes complex development tasks much easier by providing a tightly integrated development environment for building, editing, source browsing, and debugging. It also provides a new, more integrated set of tools and services, including the Visual™ GUI builder that can help you to quickly create new GUIs. New integrated editors make it easier to perform common development tasks, and WorkShop WorkSets help you keep track of the files, programs, directories, and targets associated with your development projects. This chapter presents a general introduction to the features of this release including:

1

Integrated Development Tools

The Sun WorkShop provides an integrated environment for the development and evolution of C++, C, Fortran 90, Fortran 77, and Pascal applications. It provides a high level of integration of core development functions such as editing, source browsing, building, and debugging.

The most common development operations are obvious and easy to perform because the vi, XEmacs, and GNU Emacs editors are the center of an integrated development tool set that includes WorkShop Building, WorkShop Debugging, and WorkShop Browsing. The integrated editors also provide access to common development tasks such as evaluating expressions, setting breakpoints, and stepping through functions, as well as powerful new features such as Fix and Continue. This integration allows programmers to spend most of their time in their editors and makes code development quicker and more efficient.

For an introductory demonstration of how the integrated environment can help make development easier and quicker, choose Help ➤ Demos from the WorkShop main window.

Three Integrated Editors

To increase ease of use and improve developer productivity, the WorkShop uses a new architecture that makes most development tasks accessible from your editor of choice (XEmacs, GNU Emacs, or vi). This “edit server” architecture means that you always view, edit, and operate on source code from a single view—your preferred editor. These editors are really your editors, not emulations. They have the familiar look and feel of editor including your existing keyboard shortcuts. The editors can perform many development functions and share task information with the other integrated development tools.

This means that most common tasks, such as evaluating expressions, setting breakpoints, stepping through functions are available from several different windows, including your editor of choice (vi, XEmacs, or GNU Emacs). Complex application development becomes easier and more efficient.

For more information on using the WorkShop editors, see

- “Selecting and Using Text Editors” on page 13
- “Text Editing” in the online help
- “Video Demonstrations” in the online help.

WorkShop Picklists and WorkSets

Sun WorkShop provides a new method of organizing and accessing the files, targets, programs, experiments and (if Sun WorkShop TeamWare is installed), workspaces associated with a given development project. The WorkShop remembers recent work completed on a given project and populates menu picklists with the files and operations used on that project. Whenever you start the WorkShop, it remembers the last set of operations performed and populates the appropriate menu picklist—whether it's 5 minutes or a week later. You don't have to remember long path names or argument sequences. The WorkShop remembers them for you.

Additionally, sets of picklists can be saved as WorkSets. WorkSets allow you to save sets of picklists associated with a given development project under a single name. By loading a WorkSet file, you can reload the files connected to a development project to the appropriate menu picklist.

For more information on using WorkSets, see

- "Getting Started with WorkSets and Menu Picklists" on page 14
- "Using WorkSets" in the online help

Performance, Debugging, and File Management Tools

This release of WorkShop uses a Tools menu (and button bar) to provide easy access to new performance and debugging tools and their object files. The individual tools in the Tools menu contains picklists for the objects specific to the tool. Users can build a list of objects or files used by a particular tool, thus making it easier to bring up the tool with the object loaded. For example, after you have loaded a design file into Visual once, start Visual with that design file loaded again by choosing the file from the Visual picklist on the main WorkShopTools menu.

By default, the WorkShop main window includes button bar or menu access to the Analyzer and Merging. If you have the Sun Performance WorkShop Fortran, the Tools menu or button bar also provides access to the Sun WorkShop TeamWare file management tools, the F90 browser, and the multithreaded tools—LoopTool and Thread Analyzer. If you have Sun Visual WorkShop C++, you have access to Visual instead of the F90 browser.

For more information about using the tools, or about the picklists, see one of the following:

1

- “Getting Started with WorkSets and Menu Picklists” on page 14
- “Multithreaded Development Tools” on page 7
- *Sun WorkShop TeamWare: User's Guide*
- “Analyzing Performance Data” or “Tuning Multithreaded Programs” in the online help

WorkShop Visual

Available only with Sun Visual WorkShop C++.

Visual helps developers quickly and easily design GUIs, generate portable object-oriented code, and develop Motif or Microsoft Foundation Class GUIs.

Up to 70 percent of your application’s source code base can be GUI code. Visual is an interactive tool that allows you to see what the interface looks like and how it behaves while it is being built. Visual automatically generates the code when the design is complete.

For more information about this release of Visual, see *WorkShop*: *Visual User’s Guide*.

Sun WorkShop TeamWare

Available with Sun Performance WorkShop Fortran and Sun Visual WorkShop C++ only.

Sun WorkShop TeamWare provides services for source code management either visually, through a set of GUIs, or from a command line. TeamWare enables teams to work together more efficiently even when team members are distributed among multiple sites. TeamWare provides structure as well as automated functions that allow a team to work in parallel to coordinate, integrate, and build a product. The services include:

Configuring	For managing and integrating source code configurations and releases
Versioning	For creating and tracking file version histories
Freeze Pointing	For baselining a software configuration or release for later retrieval
Building	For reducing the time required to build large projects by executing build jobs on multiple Solaris™ hosts
Merging	For merging source files and coordinating source changes

For more information about using TeamWare, see the *Sun WorkShop TeamWare: User's Guide* or start Sun WorkShop TeamWare and select Help from the main window.

Multithreaded Development Tools

The Sun Performance WorkShop Fortran and Sun Visual WorkShop C++ include advanced tools for developing multithreaded applications. WorkShop Debugging supports dynamic analysis and control of multithreaded programs. LockLint analyzes source code for potential synchronization errors, such as deadlock and data race conditions. LoopTool displays a graph of loop runtimes and shows which loops were parallelized. Thread Analyzer provides detailed thread-level profiling to help you understand the behavior of your multithreaded programs and tune for better performance. Together they provide powerful support for multithreaded program development.

For more information on using the multithreaded tool set, see:

- *Workshop: Beyond the Basics*
- "Tuning MultiThreaded Programs" in the online help

Internet WorkShop

The Internet WorkShop is an integrated suite of powerful object-oriented tools for developing 3-tier class Inter/Intranet applications. In addition to Visual WorkShop C++ tools and services, the Internet WorkShop provides NEOworks™ software for server development and Java™ WorkShop™ and Joe™ software for client development. NEOworks and Visual WorkShop provide all the necessary tools to develop state-of-the-art networked objects running on the Inter/Intranet. Java applets and applications may be developed using Java WorkShop. Joe provides the capability of integrating Java applets with remote NEO objects running on the Inter/Intranet.

WorkShop Compilers

This release of the WorkShop supports the following five compilers.

1

Compiler C++

Available only with Sun Visual WorkShop C++.

This release implements the complete feature set found in *The Annotated C++ Reference Manual*. It includes support for exception handling, an incremental linker, a fast template instantiation scheme, and an enhanced version of the commercially available Tools.h++ class library.

As an optimizing, native C++ compiler, the version offers significant boosts in both compilation and execution speed.

For more information about the C++ compiler, including a list of the C++ documentation, see *C++ User's Guide.*

Fortran 90 Compiler

Available only with Performance WorkShop Fortran.

This release is a complete implementation of the Fortran 90 ANSI X3.198-1992 standard. This standard has added many powerful features, such as an improved ability to express mathematical formulas more directly in the programming language. In addition, the Fortran 90 compiler works with the rest of the WorkShop to automatically parallelize your code.

For more information about the Fortran 90 compiler, including a list of the Fortran documentation, see the *Fortran User's Guide* or *Fortran Programmer's Guide.*

Fortran 77 Compiler

Available only with the Performance WorkShop Fortran.

This compiler is a complete implementation of the Fortran 77 ANSI X3.9-1978, ISO 1539-1980 standards. It has an improved ability to express mathematical formulas more directly in the programming language, as well as extensions that provide compatibility with VAX VMS Fortran and Cray Fortran.

For more information about Fortran 77 compiler, including a list of the Fortran documentation, see the *Fortran User's Guide* or *Fortran Programmer's Guide.*

C Compiler

Not available with the WorkShop Professional Pascal.

This compiler is fully compliant with the ANSI C language and environment standard, and it also supports traditional K&R C. The C optimizer provides significant performance increases over nonoptimized code. The code optimizer removes redundancies, efficiently allocates registers, and schedules instructions. Also featured is an incremental linker to reduce link time during the debugging phase.

For more information about the C compiler, including a list of the C documentation, see *C User's Guide.*

Pascal Compiler

Available only with WorkShop Professional Pascal.

This compiler is fully compliant with the ANSI/ISO Pascal language and environment standard. The Pascal optimizer provides significant performance increases over nonoptimized code. The code optimizer removes redundancies, efficiently allocates registers, and schedules instructions and reorganizes code to take full advantage of the SPARC™ instruction set. It also supports conformant arrays and 32- and 64-bit IEEE floating-point numbers.

For more information about the Pascal compiler, including a list of the Pascal documentation, see *Pascal User's Guide.*

1

Part 2 — Developing Programs in WorkShop

Starting Sun WorkShop 2

This chapter describes how to begin working in Sun WorkShop. For instructions on installing WorkShop, see the *Sun Developer Products Installation and Licensing Guide* or the WorkShop Roadmap.

This chapter is organized into the following sections:

Opening WorkShop

Once you have installed WorkShop and added it to your command path, you can open it by typing `workshop` at a command prompt.

If this is the first time you have used WorkShop, the About WorkShop window appears in front of the main WorkShop window. If you have not registered, take a few minutes and do so. If you have not read the Overview or seen the video demonstrations, now might be a good time to check them out.

Selecting and Using Text Editors

Unless you have previously set your `EDITOR` environment variable to GNU Emacs or XEmacs, WorkShop uses the `vi` editor (For information on how to set your `EDITOR` environment variable, see the manual page for your command

shell). To use the integrated GNU Emacs or XEmacs editors, you must use the Editor Options dialog box. The editor you choose will remain the default editor for subsequent sessions of WorkShop. To change editors, choose Options ➤ Text Editor Options from the WorkShop main window to open the Editor Options dialog box and choose a new default editor.

The WorkShop implementation of vi includes a reuse button. If the button is enabled, subseqently opened files are displayed in the original vi window. If the button is disabled and you open a new vi file, the new file is displayed in a new vi window. The Re-usable button in the lower-right corner of the window toggles manually between enabling and disabling the reuse feature.

Vi and XEmacs include WorkShop-specific tool bars. GNU Emacs and XEmacs include WorkShop-specific minibuffer commands that help you to browse and debug source code. In addition, you can open WorkShop from an existing Emacs session by typing `M-x workshop-start` in the minibuffer. To use GNU Emacs with WorkShop, you need to set the load path in your `.emacs` file. See the online help section “Starting WorkShop From Emacs” for instructions.

Getting Started with WorkSets and Menu Picklists

WorkSets and menu picklists help you keep track of the files and objects you use for your development projects. WorkShop uses WorkSets and menu picklists to provide quick access the various directories and files associated with a given development project, including:

- source files
- build targets
- programs
- experiments
- source browser directories
- workspaces (if TeamWare is installed
- WorkShop Visual design files

Each type of object is saved to a different menu picklist. For example, if you load a program called `Freeway` into the Debugging window, `Freeway` is added to the Debug menu picklist. The next time you want to debug the `Freeway` program, you can select `Freeway` from the Debug menu picklist

WorkShop uses WorkSets to save complete sets of menu picklists. Whenever you start WorkShop, it either creates a default WorkSet (usually `.default.wst`) or it opens the last WorkSet you had open. By default, when you close WorkShop, it automatically saves your current WorkSet. Consequently, to use WorkShop WorkSets, you do not have to do anything. The WorkShop does it for you.

If you want to create new WorkSets of your own, you can do so using the New WorkSet item in the main WorkShop menu. By saving a WorkSet you can save all the picklists objects associated with a given development project under a single name. Saving your objects and files as a WorkSet makes them easier to access later.

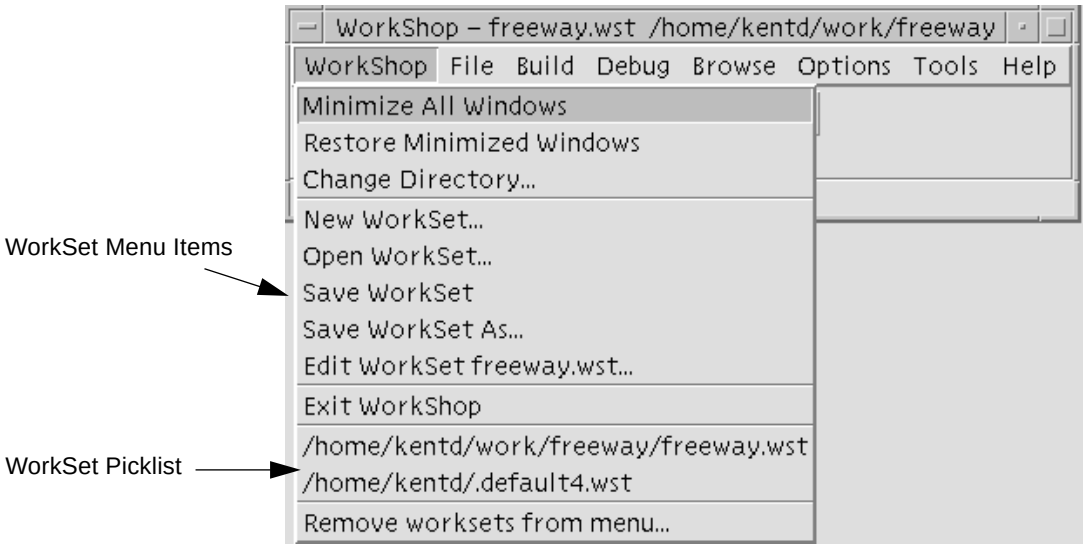


Figure 2-1 WorkShop Menu

For example, suppose you use the WorkShop to do the following:

1. **Create and edit several source files**
2. **Build a program target**
3. **Debug the program**

By default, the files, build target, and program you created are added to the appropriate menu picklist. In this case, the source files are placed on the File menu picklist; the build target is placed on the Build Menu picklist; and, the program is placed on the Debug Menu picklist. To save all of these as a WorkSet, select WorkShop ➤ Save WorkSet As, and then enter a name in the chooser. The WorkShop stores the files, target and program under the name you choose. Loading this WorkSet later reloads the files, target, and program connected to the appropriate menu picklist

If you want to start the WorkShop with a specific WorkSet loaded, you can. To do so, specify the workset name on the command line at startup. For example, the command:

```
workshop freeway.wst
```

starts WorkShop with the all the files, targets, programs, and so on associated with the freeway WorkSet loaded on the appropriate menu picklist.

Adding Items to WorkSets and Menu Picklists

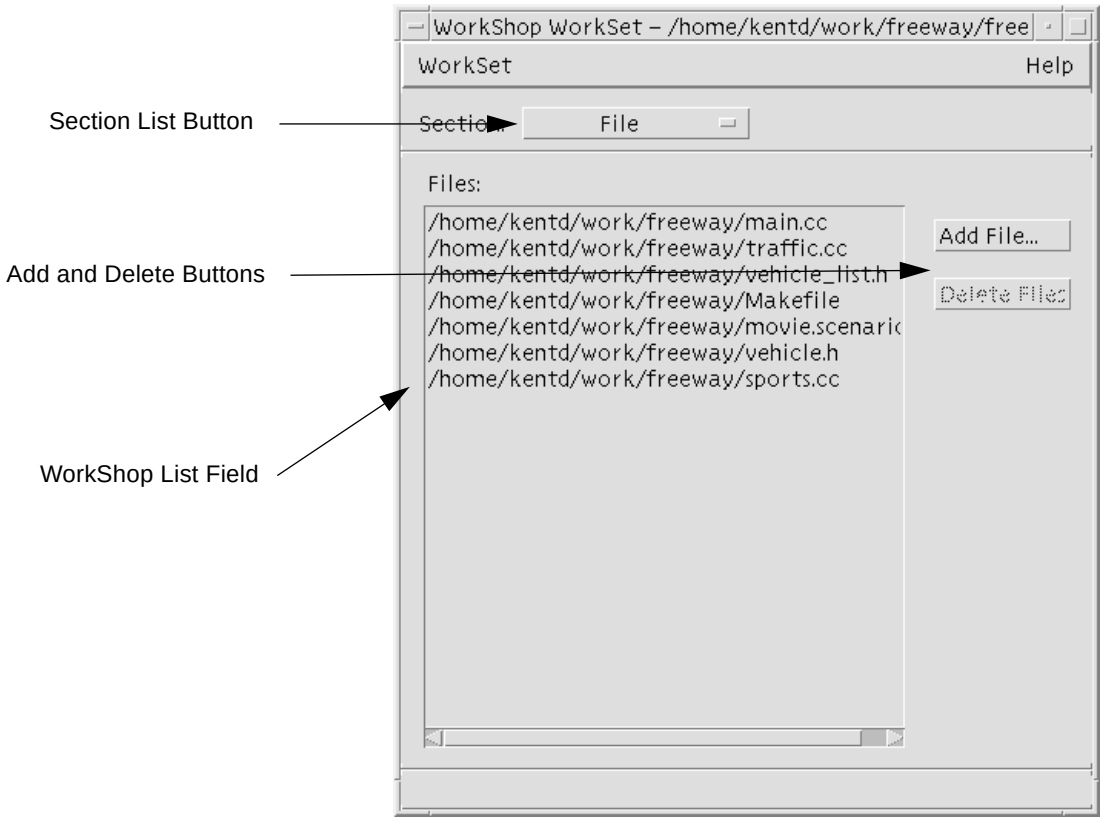
Whenever you start working, WorkShop remembers the last WorkSet you had open and the last set of development tasks you performed. It populates WorkShop menu picklists with the objects (files, programs, and so on) contained in that WorkSet.

Picklist objects (for example, Files on the File menu) can be added to or removed from menu picklists (and therefore to the current WorkSet) either by editing the WorkSet directly, or by adding or removing an object directly from any of the WorkShop menus.

Adding Items to a WorkSet Using the WorkSet Window

To edit the WorkSet directly, select WorkShop ➤ Edit WorkSet *WorkSet_Name*. When the WorkShop WorkSet window appears, use it to add or delete the desired object.

Figure 2-2 WorkShop WorkSet Window

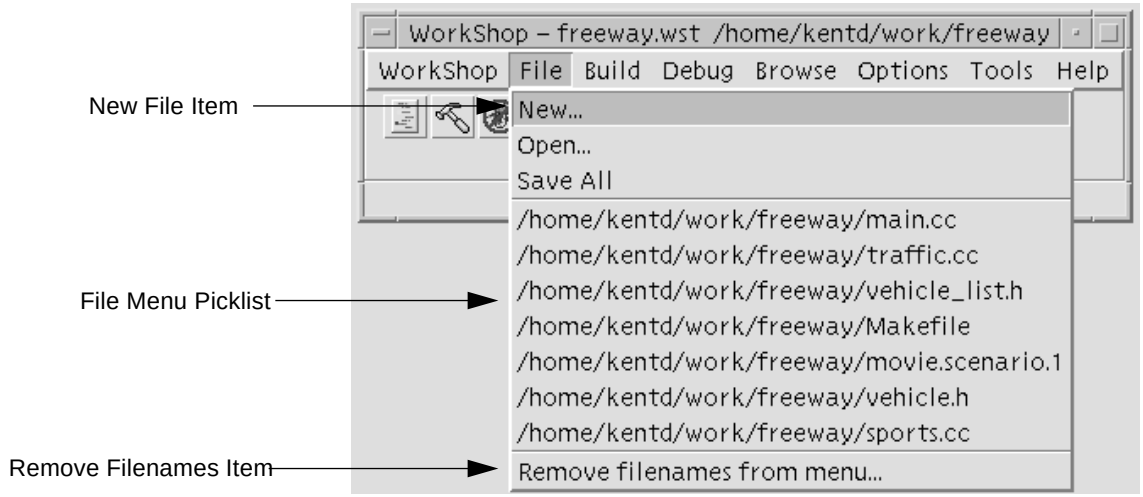


Adding Items to a WorkSet Using a WorkShop Menu

To add an object to a WorkSet using a WorkShop menu, open the menu and select New *menu_item_name*. When you open a new item, it is automatically added to the menu picklist and the current WorkSet. To remove an item from a menu, select Remove *menu_item_name* from Menu. When the file chooser appears, select the item you want to remove.

For more information about creating, editing, or modifying WorkSets, see "Using WorkSets" in the WorkShop online help.

Figure 2-3 File Menu Picklist.



Building Programs in Sun Workshop 3

Sun WorkShop provides you with the ability to run one build job at a time or you can run several build jobs concurrently. This chapter shows you how to quickly build a single application (with or without WorkShop's default build settings) and how to fix build errors using the Building window and the WorkShop editor of your choice.

This chapter is organized into the following sections:

Building a WorkShop Target

A WorkShop target differs from a make target. When you build a program in WorkShop, you are actually building a WorkShop target. A WorkShop target is an object made up of the following items:

- build directory
- build command
- makefile
- make target

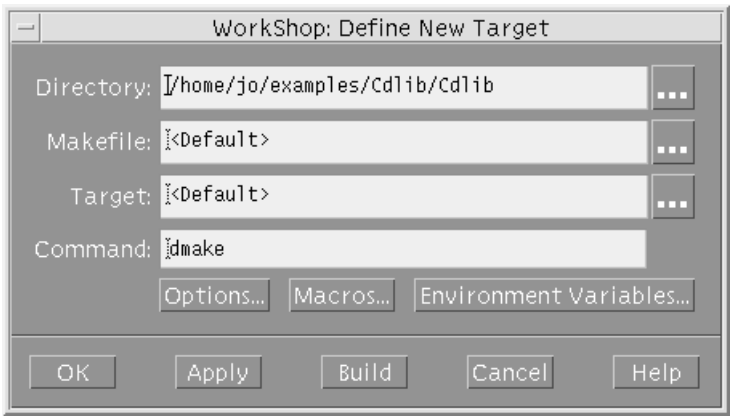
3

A make target is an object that is built from the rules contained in a particular makefile.

Building With Default Values

The quickest way to build a program is to use the default values provided in the Define New Target dialog box. All you need to supply is the build path.

1. **From the WorkShop console, choose Build ➤ New Target to open the Define New Target dialog box.**



   The Define New Target dialog box contains the value `Default` in the Makefile and Target text fields. If you do not specify a particular makefile or make target, WorkShop looks for a makefile in the build directory named `makefile` or `Makefile` and uses the first make target in that makefile. The build command provided by default is `dmake`. It is set to run in serial mode so you can run one job at a time on the local host. For information on building in distributed mode, see "Running a Distributed Build" in the online help or *WorkShop: Beyond the Basics*.

2. **Type the build path in the Directory text field.**
   You can also click the button to the right of the text field to open a directory chooser. Choose a directory in the list and click OK to load it into the Directory text field.

3. **Click Build at the bottom of the dialog box and watch the build output in the Build Output display pane in the Building window.**
   The Building window opens when you click the Build button.

The build output is displayed in the Build Output display. Click on the Stop Build button in the Building window to stop the build process.

Note – The next time you bring up the Building window, the build directory will be set to the last directory in which you ran a build job. You can see the path name in the Build Information field.

Building With NonDefault Values

If you have a specific build command, a makefile with a unique name, or a certain make target, specify it in the Define New Target dialog box.

1. **Type the name of the directory you want to build in and click Apply to apply the change.**
   You can also select another directory from the Set Build Directory dialog box by clicking the button next to the Directory text field.

2. **Type the name of the makefile you want in the Makefile text field.**
   If you want to choose another makefile from the current build directory, click on the button next to the Makefile text field. Choose a makefile from the list in the Set Makefile dialog box and click OK to load it into the Makefile text field.

3. **Type the name of the make target you want in the Target text field.**
   You can also choose another make target in the current makefile by clicking on the button next to the Target text field. Click OK in the Target Chooser dialog box to load the make target into the text field.

4. **Type the name of the build command you want in the Command text field.**

If the build command you specify is something other than `make` or `dmake`, you can include any of its arguments in the Command text field.

Note – If the build command is not in your `PATH`, you might have to specify the full command path.

---

5. **Click Build in the dialog box to start a build with the settings you supplied.**

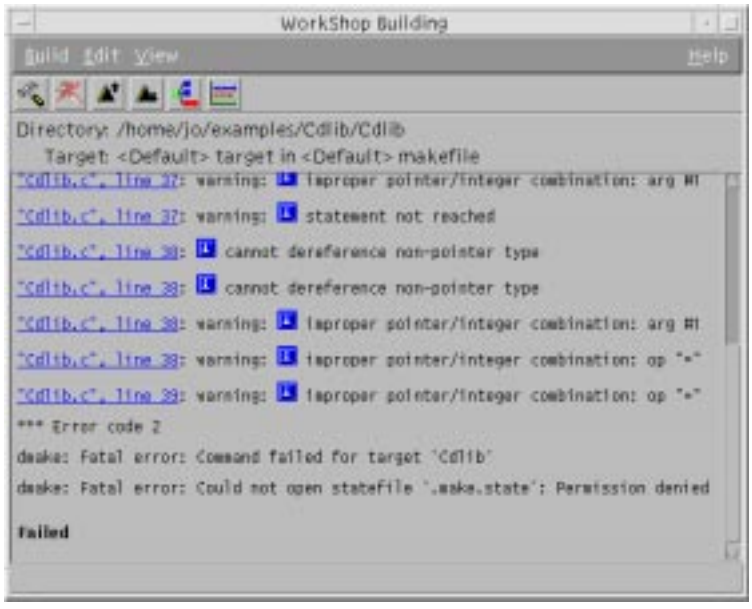
3

Modifying a WorkShop Target

To edit an existing WorkShop target, choose Build ➤ Edit Target and choose a WorkShop target from the list. The Edit Target dialog box opens, displaying the current settings for the build directory, makefile, make target, and build command. Edit any of these fields or change options, macros, or environment variables. Click Build to rebuild the WorkShop target with your new settings. See “Editing a WorkShop Target” in the online help for detailed information.

Fixing Build Errors

The process of fixing build errors has improved due to the integration of the text editor with the build process. When a build fails, the build errors are displayed in the Build Output display of the Building window. Build errors that have links to the source files containing the errors are highlighted and underscored.



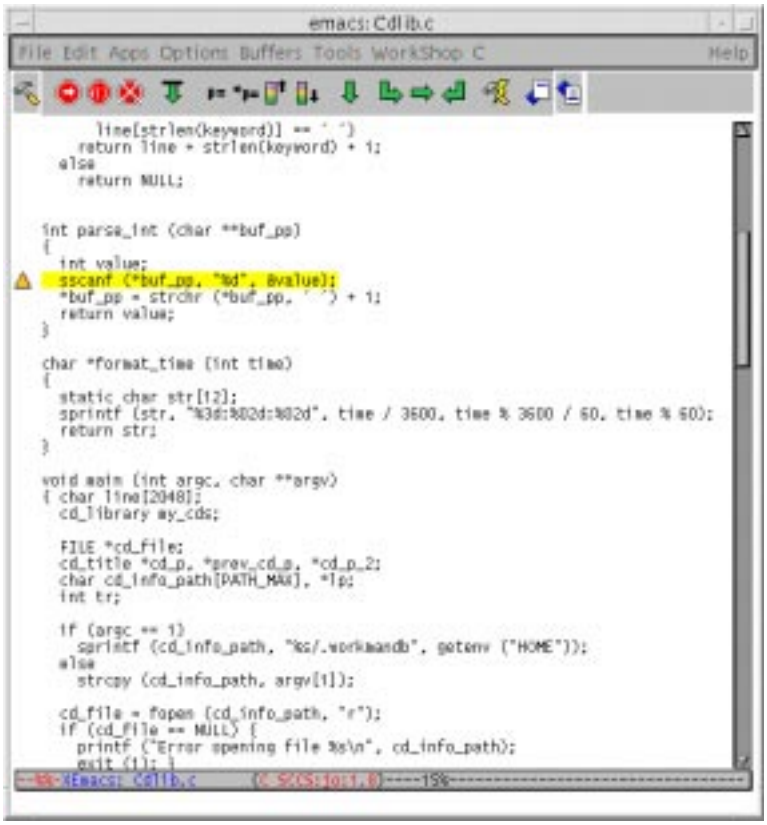
Each error gives the name of the file containing the error, the line number on which the error occurs, and the error message.

In C programs, an additional glyph (the letter “i”) is included in the build error message. Clicking on the glyph opens a pop-up window that defines the associated error message.)

Note – Only Sun compilers produce output that can be converted to hypertext links. If the build command you use does not call Sun compilers, you might lose the link facilities of the Building window.

Clicking on the underscored error immediately starts a text editor that displays the source file containing the error. The source file is shown with the error line highlighted and an error glyph appears to the left of the line.

3



The following steps show how you can use the Building window and the text editor to quickly fix build errors:

1. **Click on a highlighted error in the Build Output display.**
   The editor window opens, displaying the source file containing the error. You do not have to search for the line containing the error—the error line is highlighted in the editor and the cursor is already positioned at the line. The error message is repeated in the footer of the text editor.

2. **In the text editor, make sure the source file can be edited.**
   From the vi editor, choose Version ➤ Checkout. From the XEmacs editor, choose Tools ➤ VC ➤ Check out File *file*. From the GNU Emacs editor, choose Tools ➤ Version Control ➤ Check Out.

3. **Edit the source file containing the error.**

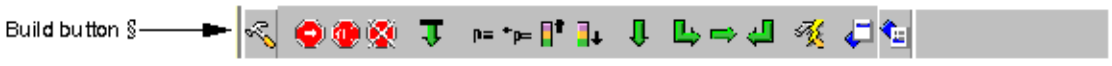
4. **In the Building window, click on the Next Error button in the tool bar to go to the location of the next build error in the text editor.**

As you click Next Error, notice how each successive error in the build output is highlighted and how the corresponding source line in the text editor is also highlighted.

5. **Save your edits to the file.**

6. **Check the file back in.**

7. **Click the Build button in the text editor's tool bar to rebuild.**
You can also build by clicking the Build button in the Building window's tool bar.



You can watch the Build Output display pane to follow the progress of the build.

For detailed information on building in WorkShop, see the online help. You can access the online help by choosing Help ➤ Building from the Building window or by choosing Help ➤ Help Contents ➤ Building a Program from the main WorkShop window.

3

Debugging A Program 4

Sun WorkShop provides an integrated debugging service that can run a program in a controlled fashion and to inspect the state of a stopped program. WorkShop gives you complete control of the dynamic execution of a program, including the collection of performance data.

You can determine where a program crashed, view the values of variables and expressions, set breakpoints in the code, and run and trace a program. In addition, machine-level and other commands are available to help you debug code. You can use standard dbx commands in the Dbx Commands window.

This chapter is organized into the following sections:

Debugging Features

When debugging in WorkShop, you have access to an extensive range of event management, process control, and data inspection features. You can:

- Run a program in Quick Mode with the debugger in the background, ready to take over processing when a program is about to core dump
- Run, stop, and continue execution

- Set breakpoints at lines or in functions; *trace* program execution line by line across a whole program or within a function; *set watchpoints* to stop or trace a program if a specified value or expression changes or meets some other condition
- Set multiple breakpoints or trace tags in *C++* code—in all member functions of the same name across a class, in all members of a specified class, or in all overloaded functions
- Single-step through program code one line at a time; step over or into function calls; step up and out of a function call arriving at the line of the calling function line after the call
- Collect data for use with the Analyzer using the Collector
- Automatically detect runtime errors in an application using runtime checking to detect memory access errors, memory leaks, and memory block usage
- Look up declarations of identifiers and definitions of types, classes, and templates
- Spot check the value of variables or expressions whenever the program is stopped; monitor variables or expressions for changes over time; examine the call stack; move up and down the call stack; and call functions in the program
- Graphically examine program variables including complex structures and arrays, and monitor values during program execution using the Data Grapher and the Data Display window
- Debug multithreaded programs
- Use the Fix and Continue feature to edit a file, recompile it with the same options, install the new code in the program, and continue
- Support C++ by supporting virtual functions, supporting *C++* exceptions, debugging with C++ Templates, using function overloading for argument resolution, and using default arguments
- Use an embedded Korn shell for programmability
- Use the Program Input/Output window to provide an I/O command interface separate from the `dbx` interaction in the Dbx Commands window
- Follow programs as they fork

Quick Start

WorkShop simplifies the debugging task by providing you with an intuitive, easy-to-use interface to ease the task.

The following shows the basic steps to follow when debugging a program. For more detailed information on any of the steps, see the online help for WorkShop.

Table 4-1 Task List: Debugging a Program

Task	Description
Build the application	Compile the application using the `-g` or `-g0` (zero) option, which instructs the compiler to generate debugging information during compilation.
Start debugging	1. Select a debugging state. Choose Debug ➤ Quick Mode to run a program normally, with the option of switching to debugging at any point. Choose Debug ➤ Debug Mode to debug the program using the full functionality of the debugger. For more information on Quick Mode, see “Using Quick Mode” later in this chapter. 2. If you just built your program, it is considered the current program and is the top menu item in the Debug menu picklist. To start debugging the current program, choose Debug ➤ *path name/program name*, or click the Debug button on the WorkShop main window tool bar. 3. If you want to debug a program other than the current program, load a new program using Debug ➤ New Program, attach to a process using Debug ➤ Attach Process, or debug a core file using Debug ➤ Load Core File. 4. Change program information such as arguments, the run directory, and environment variables.

Table 4-1 Task List: Debugging a Program *(Continued)*

Task	Description
	5. Debug multiple processes at once using the Active Debugging Sessions dialog box to manage the sessions. 6. Run your program by clicking either the Start or Go button on the tool bar.
View code	1. Wait until your program stops, or interrupt execution by clicking the Interrupt button or pressing Control + Break. 2. In the editor window, step through your code one line at a time, through functions, around functions, or out of functions. 3. Continue executing your program by clicking Go or choosing Execute ➤ Go.
Set breakpoints	1. Open the Breakpoints window by choosing Windows ➤ Breakpoints or Execute ➤ Set Breakpoints. 2. Set breakpoints to force the debugger to stop execution. You can set simple breakpoints to stop at a line of code or in a procedure or function. 3. Set advanced breakpoints to break in C++ classes, track changes in data, break on a condition, break on special events, or create your own custom breakpoints. 4. Disable and re-enable breakpoints as you move through your program.
Trace code	1. In the Breakpoints window, set a trace handler so that you can view each line of source code as it is executed. 2. Add a filter to your trace to view just calls to a function, every member function of a name, every function in a class, or each exit from a function.

Table 4-1 Task List: Debugging a Program *(Continued)*

Task	Description
	3. Control the speed of the trace using the Debugging Options dialog box.
Check values and data	1. Spot-check the value of expressions using the Data History pane of the Debugging window. 2. Select variables or expressions from the editor window, and click the Evaluate button or choose Data ➤ Evaluate Selected to find their value. 3. Copy a variable or expression into the Expressions field and evaluate it for value and type. 4. Copy a variable or expression into the Expressions field and click Display to open the Data Display window to monitor changes in its value. 5. While the program is stopped, change the value of a variable or expression using the Assign button.
Monitor data values	1. Use the Data Display window to watch the changes in the value of an expression while a program runs. Choose Windows ➤ Data Display. 2. Enter expressions in the window using the Display button in the Data History pane. 3. Follow changes in the value of an expression, compare current and previous values, view static information, and graph arrays.
Examine the call stack	1. View the call stack in the Stack pane of the Debugging window. 2. Navigate around the call stack using the Up and Down buttons, or click the hypertext link. 3. Remove the function you are stopped in from the stack by choosing Stack ➤ Pop.

Table 4-1 Task List: Debugging a Program *(Continued)*

Task	Description
	4. Remove multiple frames by placing your cursor next to the frame you want to return to and choosing Stack ➤ Pop to Current Frame.
Debug multithreaded programs	1. When a multithreaded program is detected, the Threads pane on the Debugging window opens listing information about the threads in the program. 2. Click a thread in the Threads pane to view its context. 3. Hide and expose threads to help in managing them.
Customize debugging sessions	Set new defaults for debugging performance, output, language, and so on by choosing Debug ➤ Debugging Options.

Quick Mode

Quick Mode is a new debugging feature that allows you to run your program normally but keep debugging ready in the background to take over the process at any point. Think of Quick Mode like a safety net. If you fall—your program terminates abnormally—the Debugger is there to save the program before it core dumps, just like the net. If you decide you want to debug the program you are running, you can interrupt and turn control over to the Debugger.

If all you want to do is run your program as quickly as possible, but you are afraid you might still need some debugging, select Quick Mode. There may be a short delay while the Debugging window opens, but otherwise your program runs exactly as if you were running it from the shell. Only when you switch back to Debug mode through a termination or interrupt are the symbols for the program loaded, causing a delay.

By providing this rescue capability, Quick Mode reduces the guesswork when you are testing your program.

Advantages of Quick Mode

Quick Mode offers the following advantages:

- When a program SEGVs or has some other abnormal termination, WorkShop steps in, leaving you with an active program and full debugging functionality. If you had run the program without Quick Mode, WorkShop would have terminated the program, and you would have to debug a core file using a restricted set of debugging actions.
- You can interrupt your program at any point in the process and automatically bring the debugger into control to set breakpoints, watch data, and browse through source code. You don't have to restart your program to get access to debugging functionality.
- When you run in Quick Mode, your program runs quickly, but you have the security of knowing all of the debugging functionality is available if you need it.
- Your only initial time lag comes from opening the Debugging window. Symbol tables aren't loaded.
- Your program never crashes and produces a core dump. The debugger always stops the program first.
- You have full debugger functionality with a program that was about to crash. (When you attach to a core file, you can only use limited debugger functions.)
- When the debugger takes over for a program that was about to crash, you can pop back through function calls.
- You can add breakpoints by interrupting a running program.
  - Choose Execute ➤ Interrupt.
  - Click the Interrupt button.
  - Press Control+Break.

When to Use Quick Mode

Use Quick Mode when you:

- Think you are done debugging
- Can't choose between running the program or debugging it
- Want to avoid waiting for symbols to load

- Want to test a fix you made

As you debug your program, you can toggle between Debug mode and Quick mode to take advantage of the best features of each.

How to Switch to Quick Mode

You can choose to run your program in Quick mode or Debug mode whenever you select a program to run or debug.

You can toggle between Quick mode and Debug mode for the current program from the Debug menu in the WorkShop main window or in the Debugging window.

If you want to change your defaults so that future programs automatically start in Quick mode, choose Debugging Options ➤ Debugging Behavior from the WorkShop main window.

Quick Mode Example

Suppose you made a change to a program and are confident that the change works. After rebuilding the program, you select Quick mode from the Debug menu, so that the program runs with minimal overhead. As this is the first time you have run or debugged a program since opening WorkShop, there is a slight delay as the Debugging window opens to provide the debugging “safety net.”

Your program starts automatically and runs normally until it SEGVs.

Before the program can terminate and create a core dump, WorkShop switches into Debug mode and loads the symbols for your program. You now have access to the full debugging functionality of WorkShop and can debug the program as if you had started debugging in Debug mode. Eventually, after making many changes, fixing them, and continuing, you rebuild your program.

You again select Quick Mode before running the program.

This time, no initial pause is noticeable, but, your program appears to be stuck in an infinite loop. Pressing the Interrupt button stops the program and loads the debugging symbols. You can now view data values, set breakpoints, and do any other needed debugging actions to track down your bug.

Convinced the third time is the charm, you rebuild your program, re-enable Quick mode, and run the program again. Your program runs without a flaw, and you never see the Debugging window at all.

4

Browsing Source Code 5

WorkShop's browsing feature is a powerful tool. Using browsing, you can find *all* occurrences of any symbol or string in a large program, including those found in header files.

This chapter is organized into the following sections:

Browsing uses a "what you see is what you browse" paradigm. The source code that you edit and compile is the same source code that WorkShop uses in its searches.

Browsing can be used with multiple languages. When you browse a program that is written in more than one language, the browsing feature automatically determines the language in which each source file is written. The browsing operations do not change from one language to another.

Understanding Browsing

You browse source code by issuing queries that instruct WorkShop to find all occurrences of the symbol, string constant, or search pattern that you have specified. You then view the occurrences or *matches* of the item you requested, with their surrounding source code.

You can also graph the function relationships in your program. If your source code is written in C++, then you can browse and graph the classes defined in your program.

Browsing responds to queries by searching in a database that contains information about the files you are browsing. You create this database when you compile your source file with the Browsing option.

Pattern Searches or Source Browsing?

Two types of browsing are available through the Browsing window: pattern searches and source browsing.

Click the Pattern Search radio button when you:

- Want to do a quick search (grep-style) for a regular expression
- Do not have a source browsing database in the directory you want to search
- Do not want to graphically view function call relationships or class hierarchies
- Do not want to examine the data or member functions of a class

Click the Source Browsing radio button when you:

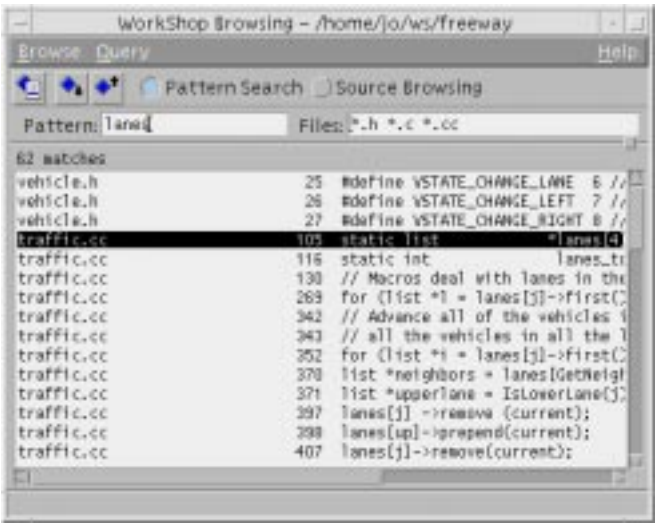
- Have a source browsing database created by adding the `-xsb` option to your compilation command or your makefile
- Want to search for language elements such as functions, classes, structs, unions, and records or for their usage, definitions, or assignments
- Want to graphically view function call relationships or class hierarchies
- Want to examine the data or member functions of a class

If you modify the files in the database, you must recompile them to get the most accurate information when using source browsing.

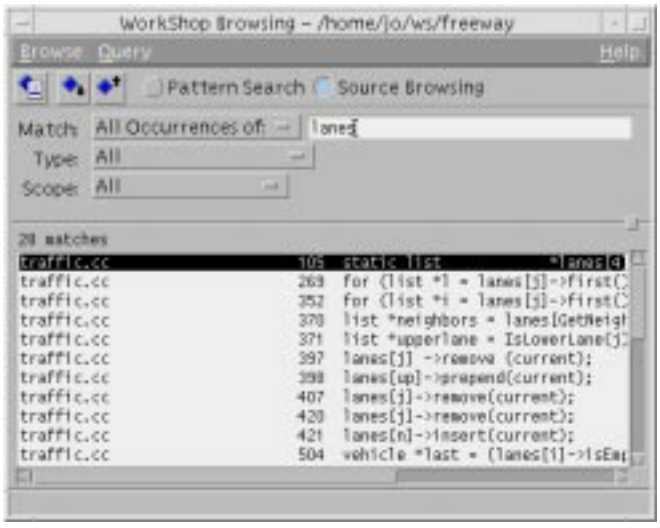
Browsing Features

Browsing consists of five main windows:

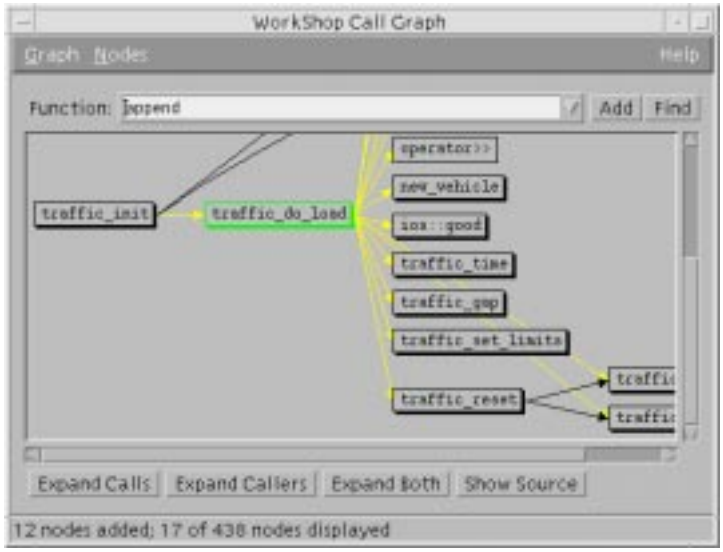
- **Browsing window in Pattern Search mode** – With pattern searching, you can search for regular expressions and simple text strings.



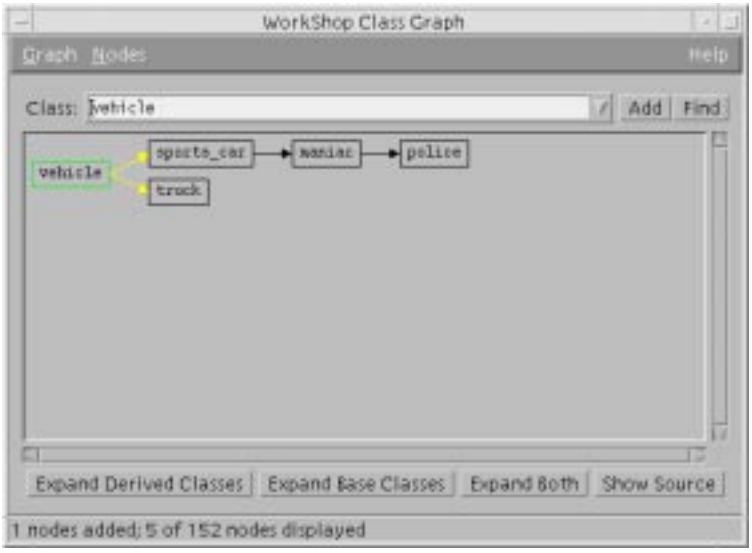
- **Browsing window in Source Browsing mode** – Using source browsing, you can search for functions, classes, structs, and other language elements.



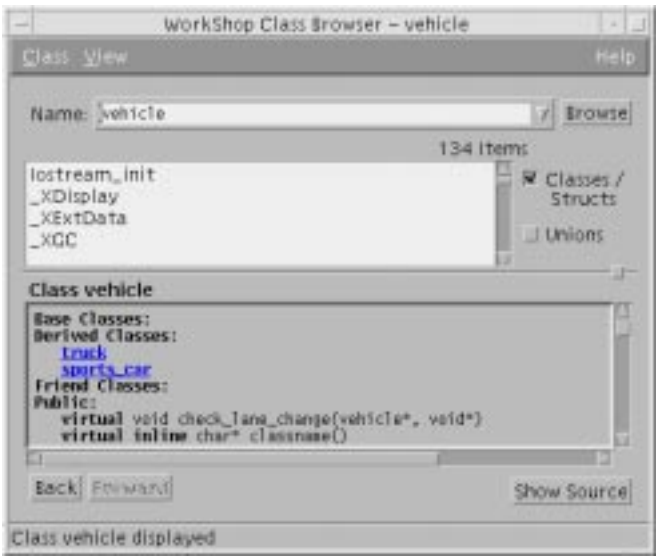
- **Call Grapher** – With the Call Grapher, you can graphically inspect the relationships of the functions in programs.



- **Class Grapher** – With the Class Grapher, you can graphically inspect the inheritance structure of classes in C++ programs.



- **Class Browser** – With the Class Browser, you can browse through C++ classes and their data and function members. You can also view the class interfaces and relationships.



5

Relationship of Browsers and Graphers

Figure 5-1 shows how the Browsing window, Call Grapher, Class Grapher, and Class Browser interrelate.

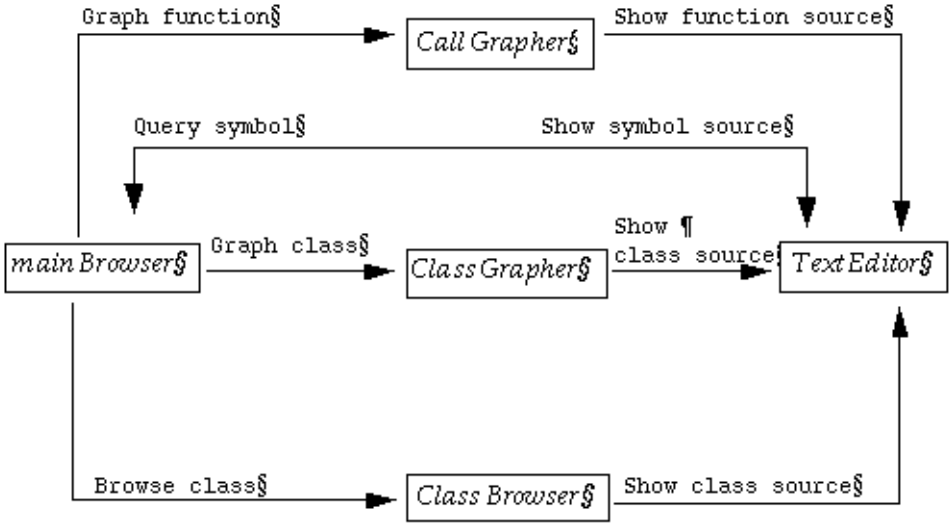


Figure 5-1 How Browsing, the Graphers, and the Class Browser Interrelate

Quick Start

The following five tables show the basic steps involoved in browsing source code. For more detailed information on any of the steps, see the online help for WorkShop.

When you start browsing, first choose the task you want to perform: query a symbol, graph a function, graph a class, or browse a class.

Using Pattern Searching

Table 5-1 Pattern Searching

Task	Description
Browse for a pattern	Choose Browse ➤ Pattern Search.
Check Directory	Look in the Browsing window title bar to be sure you are in the correct browsing directory. If not, change directories. Choose Browse ➤ Change Browsing Directory.
Enter a query	Enter a simple query in the text field and press Return, choose Query ➤ Find Matches, or click the Find Matches button.
Navigate through the matches	Move through the Match list using the mouse, the Next Match and Previous Match buttons or menu items, or press F5 and Shift+F5.
Jump to the source code	Single-click on a match to view the source in the editor window.

Using Source Browsing

Table 5-2 Source Browsing

Task	Description
Build the application	Compile the application using the `-sb` or `-xsb` option, which instructs the compiler to generate a browsing database (`.sb`) during compilation.
Browse for a symbol	Choose Browse ➤ Browse Sources.

Table 5-2 Source Browsing

Task	Description
Check Directory	Look in the Browsing window title bar to be sure you are in the correct browsing directory. If not, change directories. Choose Browse ➤ Change Browsing Directory.
Enter a query	Enter a query in the text field and press Return, choose Query ➤ Find Matches, or click the Find Matches button.
Filter or focus a query	Restrict the number of matches returned by choosing a match, type, or scope.
Navigate through the matches	Move through the Match list using the mouse, the Next Match and Previous Match buttons or menu items, or press F5 and Shift+F5.
Jump to the source code	Single-click on a match to view the source in the editor window.
View relationships	Start the graphers or the Class Browser.

Graphing Calls

Table 5-3 Graphing Function Calls

Task	Description
Start the Call Grapher	Choose Browse ➤ Graph Calls, or select a function in the source code or Debugging window and choose Browse ➤ Graph Function Calls.
Display function relationships	Enter a function in the Function field and click Add or press Return.

Table 5-3 Graphing Function Calls

Task	Description
Navigate the graph	Click on a node to select it. Expand the call, its callers, or both. Expand the entire graph, hide some nodes, or show all connected nodes. Double-click to expand calls.
Jump to the source code	Select a node and click Show Source to view the source in the editor window.

Graphing Classes

Table 5-4 Graphing Class Hierarchies

Task	Description
Start the Class Grapher	Choose Browse ➤ Graph Classes, or select a class in the source code or Debugging window and choose Browse ➤ Graph Classes.
Display a class hierarchy	Enter a class in the Class field and click Add Current Class or press Return.
Navigate the graph	Click on a node to select it. Expand the derived classes, the base classes, or both. Expand the entire graph, hide some nodes, or show all connected nodes. Double-click to expand calls.
Jump to the source code	Select a node and click Show Source to view the source in the editor window.

Browsing Classes

Table 5-5 Browsing for Class Information

Task	Description
Start the Class Browser	Choose Browse ➤ Browse Classes, or select a class in the source code or Debugging window and choose Browse ➤ Browse Classes.
Filter the class list	Click the Classes/Structs and the Unions check boxes to filter the classes and data function members listed.
View information about the class	Click on an underlined base, derived, or friend class.
Jump to the source code	Select a line (or a portion of a line) containing a data member or friend function and click Show Source to view the source in the editor window.

Analyzing Program Performance 6

This chapter describes how to use the Behavior Data Collector and the Analyzer to performance-tune applications in Sun WorkShop. This chapter is organized into the following sections:

Performance Profiling in WorkShop

The Behavior Data Collector, available from the WorkShop Debugging window, gathers performance data during the execution of an application and saves it to an experiment file. The Analyzer, a separate window available from the WorkShop main window, displays collected performance data and, if paging is causing a bottleneck, can create a file to instruct the linker to remap functions in memory.

The UNIX `prof` and `gprof` performance profiling tools generate only user CPU information. With the Collector and Analyzer, you can get information about I/O time, system time, text and data page fault times, program sizes, and execution statistics, in addition to user CPU information.

The following figure illustrates the basic performance-tuning architecture in WorkShop:

6

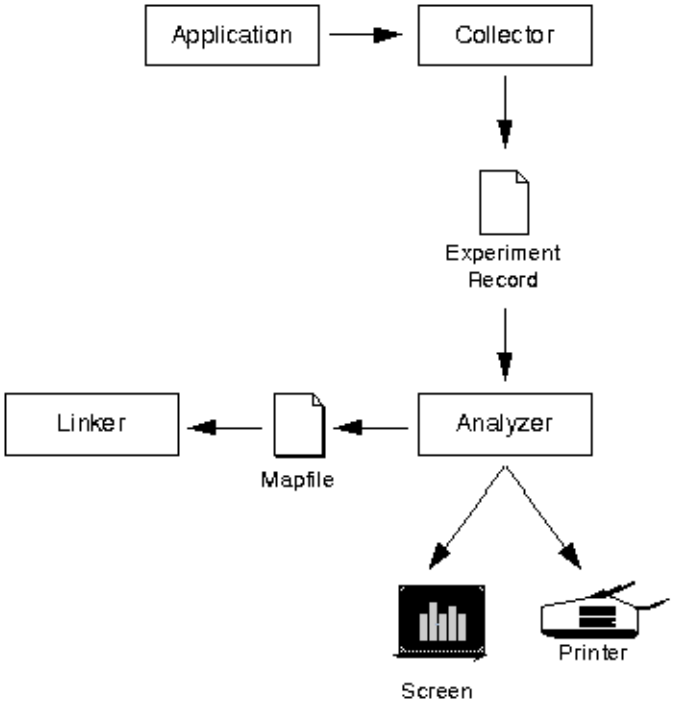


Figure 6-1 Performance-Tuning Architecture

The Collector gathers performance data from the application and writes it to an experiment record. The Analyzer displays that experiment data online and can print the data to either a file or a printer. Additionally, the Analyzer can create a mapfile for the linker, and you can then recompile the application.

The preceeding table shows the basic steps involved in collecting and analyzing performance data. For more detailed information on any of the steps, see “Analyzing Perofrmance Data” and “Collecting Performance Data” in the online help for the WorkShop.

Table 6-1 Task List: Collecting and Analyzing Performance Data

Task	Description
Build the application	1. Compile the application. Though not required for performance analysis, the `-g` compile option gives the Collector more information and yields the most accurate experiment data. 2. Link the application.
Collect performance data	1. From the Debugging window, disable runtime checking and choose Windows ➤ Collector to open the Behavior Data Collector. 2. In the Experiment File text field, type the complete path name for the experiment file to be created. 3. Select whether you want to collect data for one run only or for all runs. 4. Load the program into the Debugging window and run it by clicking either Start or Go.
Analyze performance data	1. Open the Analyzer by choosing Collect ➤ Analyze from the Behavior Data Collector. 2. Load the experiment file just created. 3. View the performance data in different ways by selecting various data types and display options from the Analyzer window. See “Selecting Data Types” and “Selecting Display Options” in the WorkShop online help for descriptions of the data types and display options.
Reorder the application, if needed	1. Compile the application using the `-xF` option. 2. Use the Behavior Data Collector to regather performance data for the application.

Table 6-1 Task List: Collecting and Analyzing Performance Data *(Continued)*

Task	Description
	3. Load the experiment file containing the collected data into the Analyzer. 4. Create a reordered map by choosing Experiment ➤ Create Mapfile and typing the samples to be used, the mapfile directory, and the name of the mapfile in the file chooser. 5. Link the application again using the new mapfile.

Collecting Performance Data

The Behavior Data Collector gathers performance data about a program as it runs through the Debugging window. To view the results of the experiment file you collect, you must use the Analyzer window, which is available separately through the WorkShop main window or can be displayed by choosing Collect ➤ Analyze from the Collector.

When you create an experiment file in the Collector, you also create a hidden directory. The hidden directory name is preceded by a dot (.)—for example, if the experiment file is called `test.1.er`, the hidden directory is called `.test.1.er`. Files in this directory contain information on segments, modules, lines, and functions.

Types of Data You Can Collect

You can collect three types of data:

- **Address Space —** Process address space represented as a series of segments, each of which contains a number of pages
- **Execution Profile (excluding called function times) —** Time consumed by functions (but not called functions), modules, and segments during execution
- **Execution Profile (including called function times) —** Time consumed by functions (including called functions), modules, and segments during execution

For more information on choosing the type of data to collect, see “Choosing the Type of Data to Collect” in the Workshop online help.

Frequency of Sample Collection

Samples are sets of data collected over specific periods of time while an application is running.

You can specify the frequency of sample collection by moving the Collect Profile data slider to the desired number of samples per second. Both data accuracy and collection overhead increase as the number of samples collected per second increases.

Analyzing Performance Data

After you collect performance data with the Behavior Data Collector, you can view it only through the Analyzer, a separate window available from the WorkShop main window. The three types of data you can collect with the Behavior Data Collector separate into nine types of data available to analyze; each of these nine types of analyzable data has one or two Analyzer display options associated with it.

Types of Data You Can View and Analyze

You can analyze the following types of data by selecting them from the Data list in the Analyzer window:

- **Process Times —** Summary of process state transitions
- **User Time —** Time spent in the user process state from the execution of instructions
- **System Wait Time —** Time the process is sleeping in the kernel but is not in the suspend, idle, lock wait, text fault, or data fault state
- **System Time —** Time the operating system spent executing system calls
- **Text Page Fault Time —** Time spent faulting in text pages
- **Data Page Fault Time —** Time spent faulting in data pages

- **Program Sizes —** Sizes in bytes of the functions, modules, and segments of your application; in conjunction with Address Space data, allows you to examine the size of your application and helps you establish specific memory requirements
- **Address Space —** Reference behavior of both text pages and data pages; in conjunction with Program Sizes data, allows you to examine the size of your application and helps you establish specific memory requirements
- **Execution Statistics —** Overall statistics on the execution of the application

For more information on selecting a data type, see “Selecting Data Types” in the WorkShop online help.

Display Options

The Analyzer offers five ways to view collected performance data:

- **Overview —** The default display provides a high-level overview of the performance behavior of the application, including:
  - Number of samples that were taken during the collection process
  - Breakdown of the process activity for each sample during the collection process
  - Breakdown of the process activity averaged over selected samples
  - Percentage of the entire experiment that you’re viewing
- **Histogram —** Summary of the amount of time spent executing functions, files, and load objects
- **Cumulative —** Cumulative amount of time spent by a function, file, or load object, including the time spent in called functions, files, or segments
- **Address Space —** Information about memory usage
- **Statistics —** Aggregate data about performance and system resource usage

For more information on selecting a display option, see “Selecting Display Options” in the WorkShop online help.

Comparing Samples

By choosing View ➤ New Window in an Analyzer window, you can open a new Analyzer window to compare two samples or view the same set of samples in two different ways.

Reordering Program Functions

If text page faults are consuming a lot of your application's time, the Analyzer can reorder your program, generating a mapfile containing an improved ordering of functions. With the most-used functions grouped together on the same set of pages, a function is more likely to already be in memory when called. See the WorkShop online help topic "Reordering an Application" for instructions for reordering an application.

Printing

You can print a plain text version of the current display or a text summary of the experiment (including average sample times for each data type and the frequency of use of functions, modules, and segments) by choosing Experiment ➤ Print or Experiment ➤ Print Summary in the Analyzer window and typing the appropriate information in the dialog box.

Exporting Experiment Data

You can export collected experiment data to files for use by other programs (such as spreadsheets or custom-written applications) by choosing Experiment ➤ Export in the Analyzer window and typing the name and directory of the experiment file to be exported in the Export dialog box.

6

Merging Source Files 7

Merging lets you compare two text files, merge two files into a single new file, and compare two edited versions of a file against the original to create a new file which contains all new edits.

This chapter explains how to start Merging, load it with files, and save the output file. The chapter is organized into the following sections:

Understanding Merging

Merging loads and displays two text files for side-by-side comparison, each in a read-only text pane. Any differences between the two files are marked, and a merged version of the two files which you can edit to produce a final merged version.

When you load the two files to be merged, you can also specify a third file from which the two files were created for comparison. When you have specified this *ancestor* file, Merging marks lines in the *descendants* that are different from the ancestor and produces a merged file based on all three files.

7

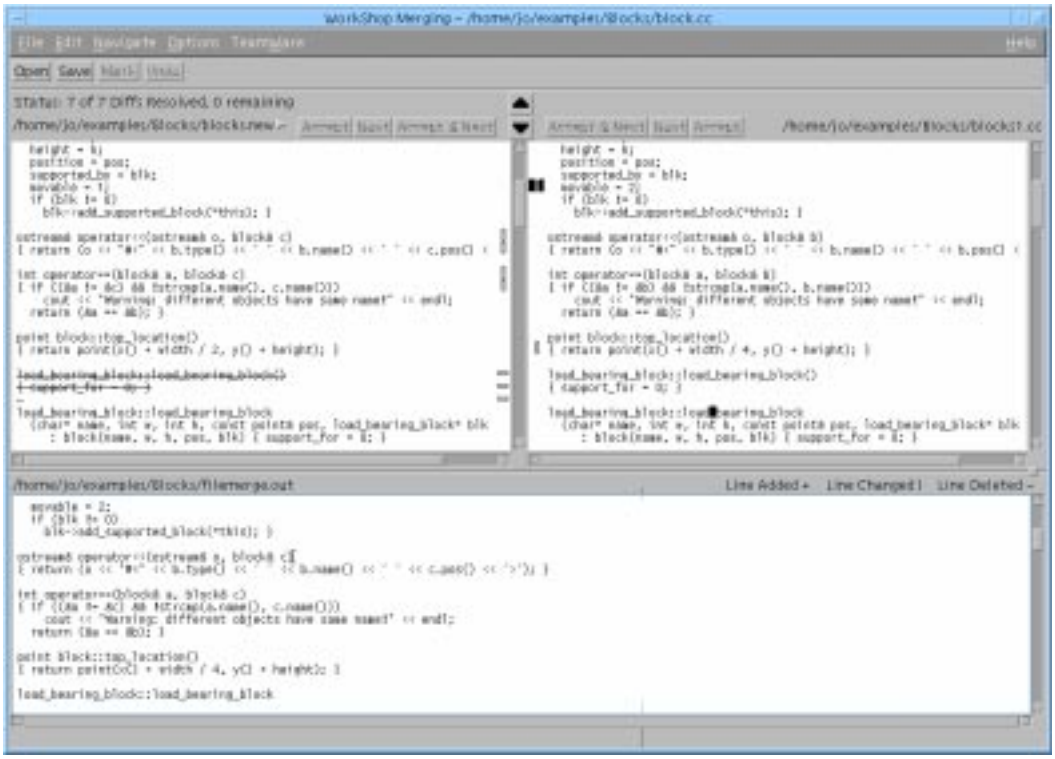


Figure 7-1 Merging Window with Loaded Files

The merged version contains two types of lines:

- Lines common to both input files
- Resolved differences

Starting Merging

Use one of the following methods to start Merging from the WorkShop main window:

- **Click the Merging icon**
- **Choose Tools ➤ Merging**

Loading Files

To load files into Merging:

1. **Choose File ➤ Open.**
2. **In the Directory field, select a working directory.**
   This is the default directory used to select and save files.
3. **In the Left File and Right File fields, select the two files you want to compare.**
4. **If you are comparing the files against a common ancestor, enter the earlier version of the two files in the Ancestor File field.**
   An ancestor file is required to use Auto Merge.
5. **If you want to specify the name of the output file, enter it in the Output File field.**
   The name `filemerge.out` is the default, and is stored in the working directory.
6. **Click Open to load the files.**
   Figure 7-1 illustrates a loaded Merging window. The names of the left file, right file, and output file are displayed above each text pane. In a three-way comparison, the name of the ancestor file is displayed in the window header.

Working with Differences

Merging operates on *differences* between files. When Merging discovers a line that differs between the two files to be merged (or between either of the two files and an ancestor), it marks the lines in the two files with glyphs corresponding to how the lines differ. Together, these marked lines are called a *difference*. As you move through the files from one difference to the next, the lines that differ and their glyphs are highlighted.

Current, Next, and Previous Difference

The highlighted difference is called the *current* difference. The differences immediately before and immediately after are called the *previous* difference and the *next* difference.

Resolved and Remaining Difference

A difference is *resolved* if the changes to a line are accepted. A *remaining* difference is one that has not yet been resolved.

If the Auto Merge feature is on, Merging resolves differences automatically.

Understanding Glyphs

Glyphs help you understand the differences between files. There are three types of glyph:

Table 7-1 Filemerge Glyphs

Glyph type	Designates
Plus sign	New line
Minus sign	Deleted line
Vertical bar	Change in line
No glyph	No changes in line

Three Input Files

When an common ancestor file is designates, glyphs next to the lines in each file indicate when they differ from the corresponding lines in the ancestor:

- If a line is identical in all three files, there is no glyph.
- If a line was added to one or both of the descendants, then a plus sign (+) shows where the line was added, and the line is highlighted.
- If a line is present in the ancestor but was removed from one or both of the descendants, then a minus sign (-) shows where the line was removed, and the line is highlighted and in strikethrough font.
- If a line is in the ancestor but has been edited in one or both of the descendants, then a vertical bar (|) shows where the line was changed, and the characters that differ are highlighted.

When differences are resolved, the glyphs change to an outline font.

Moving Between Differences

You can move between differences using the buttons above the two panes, or the Navigate menu. Use the Previous and Next buttons to scroll through the differences without accepting them.

Resolving Differences

To resolve a difference, you accept the change in either the left or right pane.

To accept a difference:

- **Click the Accept button, or Click the Accept & Next button to accept the difference and move to the next difference.**

Merging Automatically

Merging can resolve differences automatically, based on the following rules:

- If a line has not changed in all three files, it is placed in the output file.
- If a line has changed in one of the descendants, the changed line is placed in the output file. Changes could be the addition or removal of an entire line, or an edit to a line.
- If identical changes have been made to a line in both descendants, the changed line is placed in the output file.
- If a line has been edited in both descendant files so that it is different in all three files, no line is placed in the output file. You must decide how to resolve the difference—either by choosing a line from a descendant, or by editing the merged file by hand.

When Merging automatically resolves a difference, it changes the glyphs to outline font. Merging lets you examine automatically resolved differences to be sure that it has made the correct choices.

You can disable Auto Merge by choosing Options ➤ Auto Merge. When automatic merging is disabled, the output file contains only the lines that are identical in all three files and relies on you to resolve the differences.

If you do not specify an ancestor file, Merging has no reference with which to compare a difference between the two input files. Consequently, Merging cannot determine which line in a difference is likely to represent the desired change. The result of an auto merge with no ancestor is the same as disabling automatic merging: Merging constructs a merged file using only lines that are identical in both input files and relies on you to resolve differences.

Saving the Output File

Save the output file by clicking on the Save button or choosing File ➤ Save. The name of the output file is the name you specify in the Output File field.

To change the name of the output file while saving, choose Save As and fill in the new file and directory names in the resulting pop-up window, as shown in the following figure.

Part 3 — Appendixes

Setting WorkShop Resources A

If you decide to change some of the resource settings in WorkShop, you should read this appendix for information you need before actually changing the resources.

This appendix organized into the following setcions:

Which Resource File?

WorkShop provides two variations, CDE (Common Desktop Environment) and non-CDE, of each of its two resource files, `WORKSHOP` and `ESERVE`. The WorkShop program loads the correct resource file automatically based on whether the CDE window manager is running or not. The only difference between the CDE and non-CDE variations is that the latter does not define generalized color and font resource for the moftif widgets; it allows the CDE Style Manager control to these widgets.

A

Resources Available for Editing

Both the ESERVE and WORKSHOP files contain comments that indicate what the following group of resources pertain to, for example:

```
! Resources for highlight colors used by WORKSHOP in the editors

WORKSHOP.curPCColor:   #8BD98B
WORKSHOP.visitPCColor:  #EDC9FF
WORKSHOP.breakptColor:   #FF9696
```

The types of WorkShop resources you can edit include:

- Highlight colors used by WORKSHOP in the editors
- Colors used in the Data Grapher
- Colors for non-Common Desktop Environments
- Colors for hyperlink windows
- Automatic text wrapping
- Fonts for HTML and certain monospaced fonts and proportional fonts

You should be aware that resources that affect components in the core WorkShop product do not affect TeamWare components or any components started from the Tools menu. Be sure to read the comments that precede each set of resource definitions.

Changing a Resource

Create a `WORKSHOP` file or an `ESERVE` file in your `HOME` directory (or some other directory as specified in your `XFILESEARCHPATH` or `XAPPLRESDIR`). Next, copy those resource definitions from either the `CDE/WORKSHOP` or `non-CDE/WORKSHOP`, depending on whether you run under the CDE desktop or not. Consult with your system administrator for assistance if you have difficulty locating the files.

Here are the steps involved in changing a resource definition from the `CDE/WORKSHOP` file (the process is the same for non-CDE and for `ESERVE` resource definitions):

1. Create a file called `$HOME/WORKSHOP`.

2. **Go to the** `CDE/WORKSHOP` **file:**
   `cd /opt/SUNWspro/WS4.0/lib/locale/<`*lang*`>/app-defaults/CDE`
3. **Open the** `CDE/WORKSHOP` **file and copy only the resource definitions you want to change.**
4. **Paste the resource definitions into** `$HOME/WORKSHOP`**.**
5. **Change the resource settings and start WorkShop.**
   If WorkShop is already running, exit and restart it.

Note – If you modify the default colors inWorkShop to use a non-specified color, you can cause WorkShop to fill up the color map.

Changing Wide Character Fonts in Hyperlink Windows

The HTML widget has a number of wide character (WC) font resources. The names of these resources, as they would appear if set in the WorkShop resource file, are:

```
WORKSHOP*HTML*WCfont:
WORKSHOP*HTML*boldWCFont:
WORKSHOP*HTML*italicWCFont:
WORKSHOP*HTML*fixedWCFont:
WORKSHOP*HTML*fixedboldWCFont:
WORKSHOP*HTML*fixeditalicWCFont:
WORKSHOP*HTML*header1WCFont:
WORKSHOP*HTML*header2WCFont:
WORKSHOP*HTML*header3WCFont:
WORKSHOP*HTML*header4WCFont:
WORKSHOP*HTML*header4WCFont:
WORKSHOP*HTML*header5WCFont:
WORKSHOP*HTML*header6WCFont:
WORKSHOP*HTML*addressWCFont:
WORKSHOP*HTML*plainWCFont:
WORKSHOP*HTML*plainboldWCFont:
WORKSHOP*HTML*plainitalicWCFont:
WORKSHOP*HTML*listingWCFont:
```

These resources serve as flags indicating that non-ascii characters written to a hyperlink display are to be interpreted as multi-byte characters. The multi-byte characters are displayed in the font indicated by the resource.

The resources should be set *only* in locales in which there is to be multibyte interpretation of non-ascii characters.

Each WC font resource corresponds to a non-WC font resource. If the WC font resource is set, WC font dimensions determine the line spacing and baseline of text elements written in both the WC font and corresponding non-WC font. The purpose isto produce consistent spacing of a line where ascii and multi-byte characters are mixed. The WC font dimensions are also used for formatting a line written only in the non-WC fonts.

Note – Where WC font resources are set for hyperlink displays of multi-byte characters and you change a WC font resource, the size and spacing of WC fonts should be proportional to the size and spacing of non-WC fonts. To get proportional formatting you might need to modify the resources for non-WC fonts.

WorkShop Resources

The following tables lists the WorkShop color and font resources that you can change.

Table A-1 Highlight Colors in WorkShop Editors

Resource Name	Description	Default Value
WORKSHOP.curPCColor:	Current function	#8BD98B
WORKSHOP.visitPCColor:	Visited function	#EDC9FF
WORKSHOP.breakptColor:	Breakpoint	#FF9696
WORKSHOP.disabledBreakptColor	Disabled breakpoint	#BDBDBD
WORKSHOP.matchColor:	Pattern or Symbol match	#99CFFF
WORKSHOP.errorColor:	Current build error	#FFCC40

The following resources apply to the graph types in the Data Grapher window.

Table A-2 Data Grapher Colors

Resource Name	Description	Default Value
WORKSHOP.dgLineColor:	Color for Line graph type	#OOOOFF
WORKSHOP.dgFillColor:	Color for Fill graph	#FDF5E6
WORKSHOP.dgMeshColor:	Color for Mesh graph type	#OOOOFF

The following resources control the colors of the nodes, the lines (lines or arrows) connecting the nodes, and background color of the graph pane.

Table A-3 Resources for Class Grapher and Call Grapher

Resource Name	Description	Default Value
WORKSHOP*labelNodeBackground:	Background color of each node	#EFEFEF
WORKSHOP*viewBackground:	Graph pane background (Default uses X's Old Lace	#FDF5E6
Node properties when unhighlighted		
WORKSHOP*arcForeground:	Arrow between nodes	#000000
WORKSHOP*nodeForegroundColor:	Node border	#000000
WORKSHOP*labelNodeForeground:	Node text	#000000
Node properties when highlighted		
WORKSHOP*arcHighlightColor:	Arrow between nodes	#FF0000
WORKSHOP*nodeHighlightColor:	Node border	#FF0000

The next two tables list the resource settings for the standard Help window and the smaller Quick Help window.

Table A-4 Colors for General Help Viewer

Resource Name	Description	Default Value
Colors for Help viewer		
WORKSHOP*XmDialogShell.DtHelpDialog*DisplayArea.background:	Background color	White
WORKSHOP*XmDialogShell.DtHelpDialog*DisplayArea.foreground:	Background color	Black

Table A-4 Colors for General Help Viewer

Resource Name	Description	Default Value
Colors for Quick Help viewer		
WORKSHOP*XmDialogShell.DtHelpQuickDialog*DisplayArea.background:	Background color	White
WORKSHOP*XmDialogShell.DtHelpQuickDialog*DisplayArea.foreground:	Background color	Black

Hperlink Resources

The following resources set the font type, weight, and angle used in the windows and dialog boxes of the English version of WorkShop.

Table A-5 Hyperlink fonts for 'C' locale (English)

Resource Name	Default Value
WORKSHOP*HTML*TitleFont:	-*-lucida-bold-r-normal-*-24-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*Font:	-*-lucida-medium-r-normal-*-12-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*ItalicFont:	-*-lucida-medium-i-normal-*-12-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*BoldFont:	-*-lucida-bold-r-normal-*-12-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*FixedFont:	-*-lucidatypewriter-medium-r-normal-*-15-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*FixeditalicFont:	-*-lucidatypewriter-medium-o-normal-*-14-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*FixedboldFont:	-*-lucidatypewriter-bold-r-normal-*-15-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*Header1Font:	-*-lucida-bold-r-normal-*-24-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*Header2Font:	-*-lucida-bold-r-normal-*-18-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*Header3Font:	-*-lucida-bold-r-normal-*-18-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*Header4Font:	-*-lucida-bold-r-normal-*-14-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*Header5Font:	-*-lucida-bold-r-normal-*-12-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*Header6Font:	-*-lucida-bold-r-normal-*-10-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*AddressFont:	-*-lucida-medium-i-normal-*-10-*-*-*-*-*-*
WORKSHOP*HTML*PlainFont:	-*-lucidatypewriter-medium-r-normal-*-12-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*PlainboldFont:	-*-lucidatypewriter-bold-r-normal-*-12-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*PlainitalicFont:	-*-lucidatypewriter-medium-i-normal-*-12-*-*-*-*-*-iso8859-1
WORKSHOP*HTML*ListingFont:	-*-lucidatypewriter-medium-r-normal-*-12-*-*-*-*-*-iso8859-1

Hperlink WC font resources for locales with multi-byte characers. If set, non-ascii characters written to HTML displays are interpretted as multi-byte characters and displayed with font indicated by the resource.

Table A-6 Hyperlink fonts for Japanese locale

Resource Name	Default Value
WORKSHOP*HTML*WCfont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*italicWCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*boldWCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*fixedWCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*fixedboldWCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*fixeditalicWCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*header1WCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*header2WCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*header3WCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*header4WCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*header4WCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*header5WCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*header6WCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*addressWCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*plainWCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*plainboldWCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*plainitalicWCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0
WORKSHOP*HTML*listingWCFont:	-jis-fixed-medium-r-normal--16-150-75-75-c-160-*-0

The following resource lets you set text to automatically wrap or start a new line in a WorkShop window.

Table A-7 Automatic Wrapping of Text

Resource Name	Default Value
WORKSHOP*HTML*wrapPreformatText:	True

This resource enables you to turn vertical scrollbars off or on.

Table A-8 Availability of Vertical Scrollbars

Resource Name	Default Value
WORKSHOP*HTML*verticalScrollbarAlways:	True

Motif-specific Resources

The following tables list resources that are specific to Motif environments only and are not used by CDE.

Table A-9 Fonts for Motif (non-CDE) Windowing Systems

Resource Name	Description	Default Value
WORKSHOP.labelFontList:	Font types for labels	-*-lucida-medium-r-normal-*-12-*-*-*-*-*-*-*
WORKSHOP.buttonFontList:	Font types on buttons	-*-lucida-medium-r-normal-*-12-*-*-*-*-*-*-*
WORKSHOP.textFontList:	Font types in lists	-*-lucidatypewriter-medium-r-normal-*-12-*-*-*-*-*-*-*

In your resource file, uncomment the following resources to change the fonts in a specific WorkShop window.

Table A-10 Fonts for Specific Windows

Resource Name	Default Value
WORKSHOP*ipeDbxCommandWindow*userFont:	-*-lucidatypewriter-medium-r-normal-*-12-*-*-*-*-*-iso8859-1:
WORKSHOP*ipeProgramIOShell*userFont:	-*-lucidatypewriter-medium-r-normal-*-12-*-*-*-*-*-iso8859-1:
WORKSHOP*threadsList*fontList:	-*-lucidatypewriter-medium-r-normal-*-12-*-*-*-*-*-iso8859-1
WORKSHOP*handlerList*fontList:	-*-lucidatypewriter-medium-r-normal-*-12-*-*-*-*-*-iso8859-1
WORKSHOP*processList*fontList:	-*-lucidatypewriter-medium-r-normal-*-12-*-*-*-*-*-iso8859-1

This resource is applicable to text in a tabular format, such as tables.

Table A-11 Font Used in Tabular Windows

Resource Name	Default Value
WORKSHOP.DataMonospacedFont:	-*-lucidatypewriter-medium-r-normal-*-12-*-*-*-*-*-*-*

The next table lists all the foreground and background colors used in most WorkShop windows.

Table A-12 Colors for WorkShop Windows, Dialog Boxes, Menus, and Buttons

Resource Name	Default Value
WORKSHOP*foreground:	#000000
WORKSHOP*XmTextField*background:	#FFFFFF
WORKSHOP*XmText*background:	#FFFFFF
WORKSHOP*threadsList.background:	#FFFFFF
WORKSHOP*ipeDbxCommandWindow*dtTerm.background:	#FFFFFF
WORKSHOP*ipeProgramIOShell*dtTerm.background:	#FFFFFF
WORKSHOP*XmDrawingArea.background:	#FFFFFF
WORKSHOP*background:	#DEDEDE
WORKSHOP*XmPushButton*background:	#DEDEDE
WORKSHOP*XmMenuShell*background:	#DEDEDE
WORKSHOP*XmList*background:	#DEDEDE
WORKSHOP*topShadowColor:	#FFFFFF

Colors for scrollbar background and toggle buttons to indicate toggle on or off.

Table A-13 Colors for Trough and Toggle Buttons Used in WorkShop

Resource Name	Default Value
WORKSHOP*HTML*troughColor:	#DEDEDE
WORKSHOP*XmToggleButton.selectColor:	#FF9696
WORKSHOP*XmToggleButton.fillOnSelect:	true
WORKSHOP*XmToggleButtonGadget.selectColor:	#FF9696
WORKSHOP*XmToggleButtonGadget.fillOnSelect:	true

ESERVE Resources

The following tables lists the ESERVE resources that you can change.

These resources are used by the edit server to invoke the GNU Emacs and XEmacs text editors. If a fully qualified path is specified, it is executed.

Table A-14 Default Paths for Emacs Editors

Resource Name	Default Value
ESERVE*defaultGnuEmacsPath:	emacs
ESERVE*defaultXEmacsPath:	xemacs

The values for these resources can either be fully qualified paths or the basenameof the command (for instance, `myfavoriteemacs`).

If a basename is used then it is invoked from the path environment variable. In either case, there is a check to see if the binary to be invoked is a valid executable.

Resource to change cursor to a blinking cursor. Default setting is for non-blinking cursor.

Table A-15 Blinking Cursor Resource

Resource Name	Default Value
ESERVE*DtTerm.blinkRate:	0

The following tables list resources that are specific to Motif environments only and are not used by CDE.

Table A-16 Fonts for Motif (non-CDE) windowing systems

Resource Name	Default Value
ESERVE.labelFontList:	-*-lucida-medium-r-normal-*-12-*-*-*-*-*-*-*
ESERVE.buttonFontList:	-*-lucida-medium-r-normal-*-12-*-*-*-*-*-*-*
ESERVE.textFontList:	-*-lucidatypewriter-medium-r-normal-*-12-*-*-*-*-*-*-*
ESERVE*dtTerm*userFont:	-*-lucidatypewriter-medium-r-normal-*-12-*-*-*-*-*-*-*:

ESERVE Colors

Colors for text editor windows.

Table A-17 Colors for WorkShop Windows, Dialog Boxes, Menus, and Buttons

Resource Name	Default Value
ESERVE*foreground:	black
ESERVE*background:	#dedededededede
ESERVE*XmPushButton*background:	#dedededededede
ESERVE*XmMenuShell*background:	#dedededededede

Background color for scrolling lists available from a text editor.

Table A-18 Readonly Text Areas (lists)

Resource Name	Default Value
ESERVE*XmList*background:	#dedededededede

Colors for areas in text editors containg text other than menus and buttons.

Table A-19 Writable Text Areas

Resource Name	Default Value
ESERVE*XmTextField*background:	white
ESERVE*XmText*background:	white
ESERVE*dtTerm*background:	white
ESERVE*readwriteBackground:	white

A

Index

Symbols

A

B

C

X