

# **C and C++ SoftBench User's Guide**



**Manufacturing Part Number: B6454-97413**

**June 2000**

© Copyright 2000 Hewlett-Packard Company.

---

## Legal Notices

The information contained in this document is subject to change without notice.

*Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.* Hewlett-Packard shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Copyright © 2000 Hewlett-Packard Company.

This document contains information which is protected by copyright. All rights are reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

### Corporate Offices:

*Hewlett-Packard Co.  
3000 Hanover St.  
Palo Alto, CA 94304*

Use, duplication or disclosure by the U.S. Government Department of Defense is subject to restrictions as set forth in paragraph (b)(3)(ii) of the Rights in Technical Data and Software clause in FAR 52.227-7013.

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Use of this manual and flexible disc(s), compact disc(s), or tape cartridge(s) supplied for this pack is restricted to this product only. Additional copies of the programs may be made for security and back-up purposes only. Resale of the programs in their present form or with alterations, is expressly prohibited.

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

Copyright © 1980, 1984, 1986 AT&T Technologies, Inc. UNIX and System V are registered trademarks of AT&T in the USA and other countries.

Copyright © 1994 X/Open Company Limited.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Copyright © 1980, 1984, 1986 Novell, Inc.

Copyright © 1979, 1980, 1983, 1985-1990 Regents of the University of California. This software is based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California.

Copyright © 1990 Motorola, Inc. All Rights Reserved.

PostScript is a trademark of Adobe Systems, Inc.

Copyright © 1985-1986, 1988 Massachusetts Institute of Technology.

X Window System is a trademark of the Massachusetts Institute of Technology.

Portions of this software and documentation are based in part on software and documentation for the X Window System, Version 11, developed and distributed by Massachusetts Institute of Technology.

Copyright © 1989, 1990, 1993 Open Software Foundation.

Portions of this software and documentation are based in part on Motif software and documentation developed and distributed by the Open Software Foundation.

OSF/Motif is a trademark of the Open Software Foundation, Inc. in the U.S. and other countries.

RCS, the Revision Control System, manages multiple revisions of files. Copyright © 1982, 1988, 1989 Walter Tichy. Copyright 1990, 1991 by Paul Eggert. Distributed under license by the Free Software Foundation, Inc.

Copyright © 1986 Digital Equipment Corp.

---

## Printing History

New editions of this manual incorporate all material updated since the previous edition.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. Minor corrections and updates incorporated at reprint do not cause this date to change. The manual part number changes when extensive technical changes are incorporated.

August 1992	Edition 1 (B2600-90010)
August 1994	Edition 1 (B2600-90031)
January 1996	Edition 1 (B5072-90002)
August 1996	Edition 1 (B5072-90017)
February 1998	Edition 1 (B6454-90002)
September 1999	Edition 1 (B6454-93003)

---

## Typeface Conventions

Table 1

Convention	Description
<i>italic font</i>	Information you supply, either in syntax examples or in text descriptions. For example, if told to type: <i>filename</i> , you supply an actual file name like <i>sample</i> . Italics are also used for <i>emphasis</i> , and for <i>Titles of Books</i> .
typewriter font	Computer commands or other information that must be typed exactly as shown. For example, if told to type: <code>sample</code> , you type exactly the word in typewriter font, <code>sample</code> . Menu selections are in typewriter font separated by colons. See "Menu Conventions" in this chapter.
<b>boldface font</b>	A term that may need further clarification or definition, especially a familiar word (such as <b>menu</b> ) used with a computer-specific meaning. These terms are clarified in the glossary.
[...]	Optional parameters in syntax examples are enclosed in brackets.
<b>KeyCap</b>	Represents a key on your keyboard that you must press, or an on-screen button that you must select, as part of the operation. For example, <b>Return</b> is the "Carriage Return" key, which completes a command input. This key may be labelled "RETURN", "Return", or "Enter".
<b>Key1-Key2</b>	A hyphen between keys indicates that two or more keys must be pressed at the same time. For example, "Control-C" means to press and hold the Control key while pressing and releasing the <b>C</b> key. The Control key may be labelled "CTRL", "Ctrl", or "Control".



**1. Maximizing Your Results with SoftBench**

SoftBench Task Flow .....	20
Understanding Projects .....	21
Building Targets .....	22
Understanding Build Configurations and Packages .....	23
Choosing Between Project Build and External Build Models .....	25
Sharing Projects with a Team .....	27
One Project, One Author .....	27
One Project, Many Authors .....	27
One Project with Subprojects, Many Authors .....	29
Planning Your Project .....	31
Using the SoftBench Integrated Environment .....	32
Using SoftBench Tools Together .....	34
Using SoftBench Tools on Multiple Projects .....	35
Using SoftBench Tools as Standalone Tools .....	36
Reusing Tool Windows .....	37
Copying Data between SoftBench Tools .....	37
An Example SoftBench Session .....	38
Learning SoftBench .....	40

**2. Using SoftBench**

Prerequisites to Using SoftBench .....	42
Starting SoftBench .....	43
Understanding SoftBench Window Areas .....	44
Understanding the Builder Page .....	47
Understanding the SoftBench CodeAdvisor Page .....	48
Setting Up a Project .....	50
Creating a Project Using Project Build .....	50
Creating a Project Using External Build .....	52
Cloning a Project from an Existing Project .....	53
Repartitioning an Existing Project .....	54
Creating a Subproject .....	55
Modifying a Project Definition .....	56
Creating Files within a Project .....	57
Adding Existing Files to a Project .....	58
Defining Targets for Project Build .....	60
Creating a Target .....	60
Specifying Dependency Relationships .....	61

---

# Contents

Customizing Build Configurations .....	62
Using Build Packages .....	63
Defining Targets for External Build .....	64
Using the Target Graph .....	66
Understanding the Graph .....	67
Displaying Dependencies .....	68
Controlling Graph Complexity .....	69
Building a Selected Target .....	69
Starting Your Configured Editor .....	69
Building Projects and Targets .....	70
Setting the Compile Mode .....	70
Using the "External Build Command" Dialog Box .....	72
Previewing the Build .....	72
Compiling Instead of Building .....	73
Building Subprojects .....	73
Handling Errors .....	73
Running the Build on a Remote System .....	75
Checking Your Code Using SoftBench CodeAdvisor .....	76
Running Other SoftBench Tools .....	77
Managing Your SoftBench Environment .....	78
Adding and Removing Tool Icons .....	79
Choosing Tool Preferences .....	80
Registering New Tools with SoftBench .....	80
Customizing SoftBench by Setting Resources .....	81
Accessing Distributed Data and Tools .....	81
Running SoftBench on a Remote System .....	82
Integrating with CDE .....	82
Stopping SoftBench .....	82
Restoring Your Previous SoftBench Session .....	83
Getting Help .....	84
Using the Help Menu .....	85
Accessing On Item Help .....	85
If Something Goes Wrong .....	86
For More Information .....	87

## 3. Using SoftBench Configuration Manager



Understanding SoftBench CM .....	90
Getting Started - A Brief Overview .....	92
Managing the Archive System .....	95
Browsing Local Network Servers and Archives .....	95
Creating an Archive Directory .....	97
Creating a Mapping between an Archive Directory and Local Directory .....	98
Modifying Mappings between Local and Archive Directories .....	99
Managing Archive Files and Directories .....	102
Creating Initial Archive Files .....	102
Checking Out Archive Files .....	102
Cancelling Archive File Check Out .....	103
Updating a Local Directory .....	103
Viewing Contents of Archive Files .....	104
Deleting Archive Files and Directories .....	106
Locking an Archive File .....	106
Breaking a Lock on an Archive File .....	107
Viewing the Revision History of Archive Files .....	107
Setting Archive Display Filters .....	109
Managing Local Files .....	110
Modifying Local Source Files .....	110
Checking In Modified Files .....	110
Creating a Default Symbolic Name for Archive Files .....	112
Understanding Symbolic Names .....	112
Defining Symbolic Names .....	113
Symbolic Name Example .....	113
Using the SoftBench CM Command Line Interface .....	115

## **4. Using SoftBench Editors**

Using Editors with Projects .....	118
Configuring an Editor .....	119
Configuring SoftBench vi Editor .....	119
Configuring SoftBench Program Editor .....	119
Starting the Configured SoftBench Editor .....	121
Editing with SoftBench XEmacs Editor .....	122
Using SoftBench XEmacs .....	123
Accessing Help .....	124
Editing Multiple Files .....	124
Editing with SoftBench vi Editor .....	126

---

# Contents

Using the Mouse Pointer Versus the Text Cursor .....	127
Editing Multiple Files .....	127
Reusing the Edit Window .....	128
Selecting, Copying, and Pasting Text .....	129
Calling Other SoftBench Tools from the Editors .....	130
Compiling a Program File .....	130
Building a Project .....	130
Accessing SoftBench Static Analyzer from the Editor .....	130
Setting Breakpoints in a Program File .....	131
Using Configuration Management .....	131
For More Information .....	132

## 5. Using SoftBench Class Graph/Editor

Editing C++ Structures with SoftBench Class Graph/Editor .....	134
Understanding the SoftBench Class Graph/Editor Use Model .....	134
Using SoftBench Class Graph/Editor with Other SoftBench Editors .....	136
Synchronizing Editor Views and the Static Database .....	136
Using SoftBench Class Graph/Editor in Your Work .....	138
Viewing the Existing Class Hierarchy .....	138
Creating New Components .....	139
Modifying Existing Components .....	139
Deleting Existing Components .....	140
Sample Use Models .....	141
Creating a New Program .....	141
Modifying an Existing Program .....	141
Working with Class Templates .....	141
Using Configuration Management .....	142
If Something Goes Wrong .....	143

## 6. Using SoftBench CodeAdvisor

Comparing SoftBench CodeAdvisor to Debuggers or Dynamic Analyzers .....	148
Performing the "Check Code" Operation .....	149
Preparing Your Program with Project Build .....	149
Preparing Your Program with External Build .....	149
Accessing SoftBench CodeAdvisor .....	150
Selecting Rule Groups .....	150

Checking Your Program .....	150
Viewing Violations .....	151
Terminating SoftBench CodeAdvisor .....	152
Filtering Rule Violations .....	152
For More Information .....	154

## **7. Using SoftBench Debugger**

Understanding SoftBench Debugger .....	156
Preparing Your Program for Debugging .....	159
Using SoftBench Debugger Window Areas .....	160
Tearing Apart the Main Toolface .....	163
Loading or Rerunning an Executable Program .....	165
Specifying the Runtime Environment .....	165
Specifying Source Locations .....	168
Debugging Executables in a Project .....	169
Stepping through Your Program .....	171
Interrupting a Running Program .....	172
Interrupting in System or Non-debuggable Routines .....	172
Interacting with Your Program .....	174
Interacting with a Standard I/O Program .....	174
Interacting with a Terminal-Smart Program .....	175
Interacting with a Window-Smart Program .....	175
Specifying Identifier Locations .....	177
Specifying Program Location .....	177
Specifying Variables .....	178
Examining and Changing Data in Your Program .....	181
Examining Data in Your Program .....	182
Printing Hex or String Values .....	183
Changing Data in Your Program .....	183
Using Debugger Variables .....	183
Using Expressions .....	184
Using Constants .....	185
Calling Functions .....	186
Viewing the Call Stack .....	186
Viewing Thread Stacks .....	187
Understanding the Operation of Your Program .....	188
Setting and Using Breakpoints .....	189
Debugging a Program Using Breakpoints .....	189

---

# Contents

Setting a Breakpoint .....	190
Viewing and Modifying Breakpoints .....	193
Clearing a Breakpoint .....	194
Executing DDE Commands at a Breakpoint .....	195
Setting C++ Breakpoints .....	197
Setting Group Breakpoints .....	197
Viewing and Modifying Group Breakpoints .....	197
Setting and Using Watchpoints .....	199
Creating Watchpoints .....	200
Viewing and Modifying Watchpoints .....	201
Clearing a Watchpoint .....	202
Tracing Program Flow .....	204
Creating Traces .....	204
Viewing Traces .....	205
Clearing Traces .....	206
Correcting Errors in Your Program .....	207
Editing Source Code .....	207
Synchronizing Files .....	208
Debugging Dynamic Libraries .....	209
Viewing Assembly Language and CPU Registers .....	211
Tracing Assembly Language .....	211
Tracing Registers .....	212
Handling Signals and Events .....	214
Viewing and Editing Intercepts .....	214
Handling Signals .....	216
Debugging After a Program Fails (Core Dump) .....	217
Debugging with a Core File .....	217
Debugging Forked Processes .....	219
Debugging Threaded Applications .....	220
Viewing and Manipulating Threads .....	220
Setting Breakpoints on Threads .....	221
Attaching the Debugger to a Running Program .....	222
Debugging C++ Programs .....	223
Using Breakpoints for Exception Handling .....	225
Accessing Inherited C++ Values .....	225
Debugging Static Constructors .....	225

Debugging Optimized Code .....	227
Customizing SoftBench Debugger .....	228
Specifying Debugger Options .....	228
Customizing User Buttons .....	228
If Something Goes Wrong .....	231
For More Information .....	234

## **8. Using SoftBench Debugger Data Graph Window**

Starting and Stopping the Data Graph Window .....	236
Beginning a Browsing Session .....	236
Stopping a Browsing Session .....	236
Understanding Data Graph Window Areas .....	237
Understanding the Layout Control Area .....	237
Understanding the Display Control Area .....	238
Understanding the Graph Area .....	238
Understanding the Window Control Area .....	240
Using the Graph Area .....	241
Displaying New Nodes .....	241
Using the "Node Values" Dialog Box .....	241
Suspending Graph Updates .....	244
Deactivating the Graph .....	244
Stopping a Graph Process .....	244
Sample Use Models .....	245
Verifying Correct Data Structures .....	245
Viewing Values of Data Members .....	247
For More Information .....	250

## **9. Using SoftBench Static Analyzer**

Starting SoftBench Static Analyzer .....	252
Preparing to Make Queries .....	254
Generating Static Data .....	254
Updating Static Data without Building .....	255
Specifying Static Data to Analyze .....	255
Using SoftBench Static Analyzer Window Areas .....	258
Making Textual Static Queries .....	260
Making General Queries .....	260
Making Queries Based on a Program Identifier .....	260
Using Query Results .....	264

---

# Contents

Browsing the Query Result .....	264
Editing the Source File .....	264
Updating the Database .....	264
Performing a Query .....	265
Simplifying Query Results .....	266
Filtering Results Using the File Set .....	266
Filtering C++ Query Results .....	268
Using Scoping .....	268
Redisplaying Past Queries .....	271
Redisplaying Query Results .....	271
Deleting a Query Result .....	271
Saving and Printing a Query Result .....	271
Using SoftBench Static Analyzer in Standalone Mode .....	273
Generating Static Data from the Command Line .....	273
Searching Subdirectories .....	273
Using the Staticfileset File .....	274
Customizing SoftBench Static Analyzer .....	275
If Something Goes Wrong .....	276
For More Information .....	281

## 10. Using Static Graphs

Starting SoftBench Static Analyzer Graphs .....	285
Making Graphical Static Queries .....	286
General Static Graph Features .....	286
Finding Graph Nodes .....	287
Operating on Static Graph Nodes .....	287
Switching between Static Graphs .....	288
Displaying Nodes on Another Graph .....	288
Using Description Boxes .....	289
Setting Breakpoints for SoftBench Debugger .....	290
Saving Static Graph Images to Files .....	290
Simplifying Graph Displays .....	292
Reducing Graph Complexity .....	292
Filtering Sourceless Nodes .....	293
Customizing Static Graphs .....	294
Removing the Graph Legend .....	294

Viewing Multiple Graphs .....	294
<b>11. Using SoftBench File Compare</b>	
Understanding the SoftBench File Compare Window .....	296
Understanding the Menu Bar .....	296
Understanding the "Working Directory" Input Box .....	296
Using the "Left File" and "Right File" Input Boxes .....	297
Using the "Merge File" Input Box .....	297
Understanding the Text Areas .....	297
Reading the Gutter Column .....	297
Selecting Lines for Merging .....	298
Highlighting Differences .....	299
Traversing by Single Line .....	299
Comparing Two Files .....	300
Merging Compared Files .....	301
If Something Goes Wrong .....	302
For More Information .....	303
<b>12. Using SoftBench Message Monitor</b>	
Starting SoftBench Message Monitor .....	306
Understanding the SoftBench Message Monitor Window Area .....	307
Understanding the Menu Bar .....	307
Understanding Broadcast Messages .....	307
Clearing the Broadcast Message Area .....	308
Composing and Sending a Message .....	309
Logging Messages .....	311
Specifying a Log File .....	311
Starting and Stopping Message Logging .....	311
<b>13. Using SoftBench with SQL</b>	
Determining Supported Environments .....	314
Configuring SoftBench with SQL .....	315
Using Default SQL File Types .....	315
Using SQL with Project Build .....	315
Using SQL with External Build .....	317
Updating RDBMS Versions .....	317
Using SQL with SoftBench Tools .....	319
Using SQL Preprocessor Wrappers .....	319

---

# Contents

Debugging with SQL .....	319
Editing and Rebuilding with SQL for SoftBench Debugger .....	321
Using SoftBench CodeAdvisor and SoftBench Static Analyzer with SQL .....	321
For More Information .....	322

## A. Using SoftBench Graph Windows

Accessing SoftBench Graph Windows .....	324
Using Graph Window Areas .....	326
Using Popup Menus .....	326
Using Save Options for the Graph Image .....	326
Using Vertical and Horizontal Scrolling .....	330
Zooming In or Zooming Out .....	331
Clearing the Graph Area .....	331
Understanding Nodes and Arcs .....	332
Reading Graph Area Nodes .....	332
Selecting Nodes and Arcs .....	332
Moving Nodes .....	334
Customizing SoftBench Graphs .....	336
Controlling Graph Layout .....	336
Controlling Graph Display .....	336
Understanding Window Status Information .....	337
For More Information .....	338

## B. Customizing SoftBench CM Configuration

Modifying the Configuration Files .....	340
Configuring Where Archive Files are Stored .....	341
Defining User Access to the Server .....	342
Recommended Format for Permissions File .....	345
Setting Logging and Debug Options .....	346
Controlling Client Machine Access to the SoftBench CM Server on HP-UX .....	346
Performing SoftBench CM Administrator Tasks .....	348
Migrating Archive Files From RCS .....	348
Migrating Archive Files From SCCS .....	348
Modifying the Lockinfo File .....	349
Creating Archive Backups .....	349
Moving Archive Storage Locations .....	349



Troubleshooting .....	351
<b>C. Using Regular Expressions</b>	
Pattern Matching .....	354
<b>D. Customizing SoftBench for Native Language Support (NLS)</b>	
Preparing to Use NLS in SoftBench .....	362
Setting the LANG Environment Variable .....	364
Converting from One Encoding Method to Another .....	365
Rebinding Alt .....	366
SoftBench Mnemonics and Non-USASCII Character Inputs .....	367
Changing or Removing Menu Mnemonics .....	367
SoftBench Keyboard Accelerators and Non-USASCII Character Inputs .....	369
Customizing Keyboard Accelerators .....	369
Starting Your Localized SoftBench .....	371
Remote Execution Hosts and NLS .....	372
Editing in SoftBench .....	373
Character Input Example .....	373

**Glossary**

---

# Contents

# 1 **Maximizing Your Results with SoftBench**

SoftBench facilitates the development of reliable software, an especially difficult and time-consuming process when you work on large and complicated projects. SoftBench provides an integrated, consistent windowed interface to the tools you need the most.

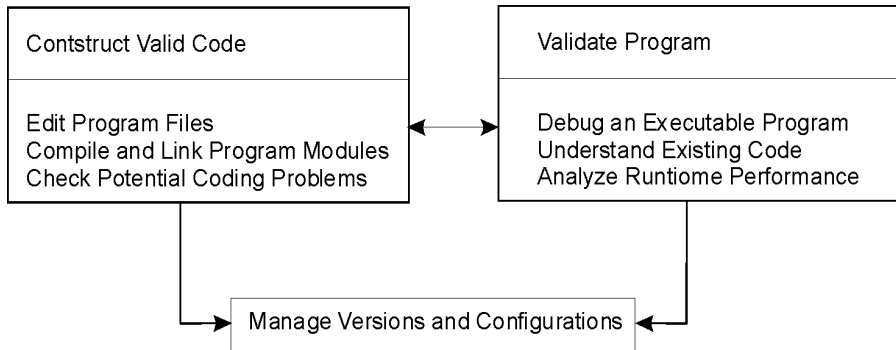
---

## SoftBench Task Flow

The tools pass information and control from one tool to the next, helping you move smoothly through the software development and maintenance tasks shown in Figure 1-1.

**Figure 1-1**

### Major Software Development Tasks Supported by SoftBench



SoftBench tools and the data on which they operate can be distributed across a network and accessed transparently by individuals or teams. SoftBench provides an open, customizable environment, allowing users to add their own tools or purchase products integrated by third party software vendors. Additionally, SoftBench provides online help and an online tutorial to simplify learning.

SoftBench provides a dynamic build environment. As you develop your project, SoftBench learns about your source files and how to transform them into your build **targets**. Based on build information contained in **build configurations**, SoftBench can maintain the Makefile instructions for you. SoftBench automatically shares this knowledge about files and targets in your project with other SoftBench tools.

SoftBench project management allows you:

- ease of navigation through project code and files that may be spread across file system directories
- automatic maintenance of simple to complex target dependencies
- definition of the project once, so that all tools benefit from the knowledge rather than maintaining multiple lists of files throughout your toolset

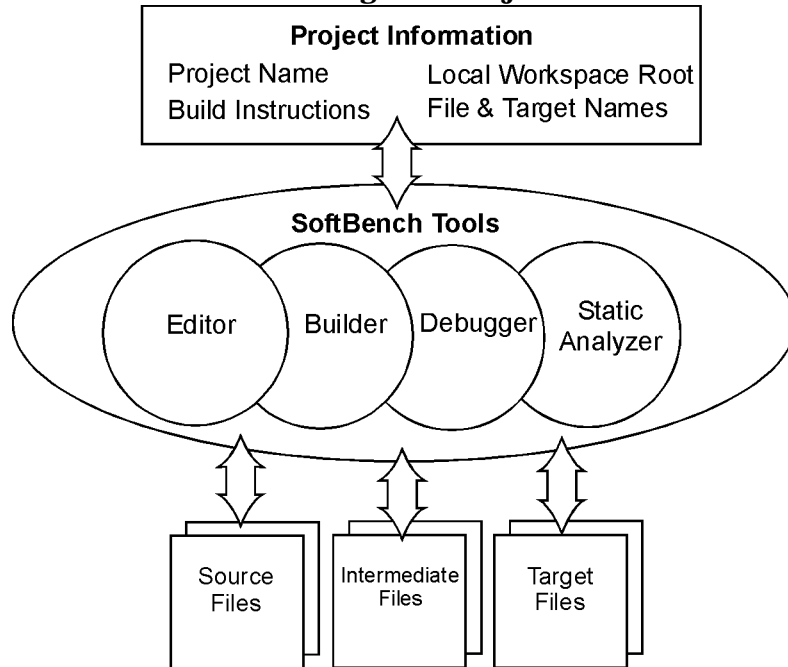
---

## Understanding Projects

**Projects** are the fundamental work unit in SoftBench. Projects consist of a named set of files that produce a set of related targets using one or more sets of build instructions and dependencies.

**Figure 1-2**

**SoftBench Tools Interacting with Project and Files**



The critical parts of a project definition are:

**Project Name** The logical name for the project.

**Local Workspace Root** The root directory of the file system hierarchy under which all your source files are found, and where editing and building take place. The project does not have to include all the files under the hierarchy, and files in the hierarchy can be used in different projects if desired.

**Project file set** The source, documentation and test files you wish to associate with the project.

**Project targets** The executables, libraries, message catalogs, and other files that are the product of one or more builds.

**Build Configurations** Instructions for how to transform your source files into your target files.

Project names contain regular characters. Control characters, spaces, and punctuation characters special to the shell are not allowed in project names. The files and targets that comprise your project are located under your local workspace root. SoftBench's project description data contains lists of these files and their relationships.

When you change file and target information in SoftBench, you change only SoftBench internal descriptions of their relationships. For example, deleting a file from a project only changes your project description data. It has no impact on your files in the file system.

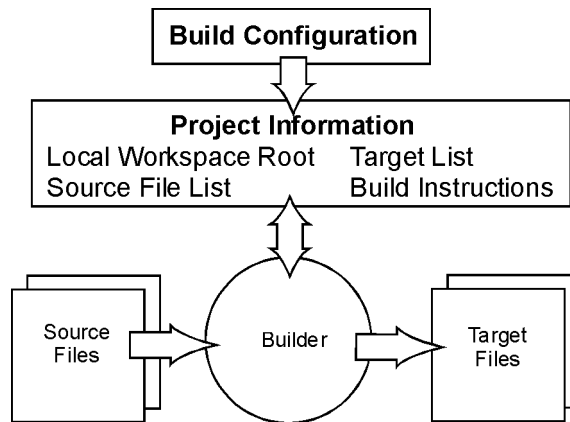
**Subprojects** are projects with a defined dependent relationship to a parent project. A subproject relationship implies a build order dependency. A project is always an entity to itself; that is, it does not dynamically inherit options or build information from any other project. Examples of subprojects include:

- a project to build a library that is linked into a parent project target
- a project to build an executable that is used to generate files or targets in the parent project

## Building Targets

Source files are files you create and edit directly. A **target** is a project file that is the end product of a build. Examples of targets include linked executables, shared libraries, archive libraries, and message catalogs.

**Figure 1-3 Building Sources into Targets Using Project's Build Instructions**



Targets that SoftBench builds:

- are project-specific. They are not shared between projects, though projects may have targets of the same name
- are user defined
- are created from a build within a project
- are built in no specified order unless a specific dependency has been created between them
- each have a physical file system location specified when you define the target
- are distinguished from intermediate files (.o's or .C's built from lex or SQL files)

Not every target is an executable file and not every executable file in a project is a target. Targets are created from a build within the project and are only those things a user has defined as such.

## Understanding Build Configurations and Packages

A **build configuration** provides the complete set of build instructions to produce a target of a particular type, such as a C++ executable, a C shared library, or an Oracle C executable. Build configurations specify libraries, include directories, compiler options, defines, and other information required to transform source files into a target. When you use project build, you can use one of the many default build

configurations provided by SoftBench.

Build configurations are independent of projects and targets. SoftBench ships with a basic set of build configurations. You can create new user build configurations based on the shipped build configurations, adding build instructions that are unique to your environment. To have your administrator save these as shared customizations available to other users, see "Sharing Build Configurations with Your Team" in SoftBench Online Help.

Once you decide on build configurations (system or customized) that work for your environment, you can use the same build configuration for many targets, and you can further customize a build configuration for a specific target.

A build **package** is a collection of build instructions that makes it easy to use third-party libraries, utilities, or compiler directives in many build configurations. Packages also make it easy to specify the compiler to use for the target. Packages are similar to build configurations because they include library and include directories, compiler options, defines, and other information needed for using the library or utility. First, choose a system or customized package; then include the package in all appropriate build configurations. If the package's build instructions need to change, you modify the package, and all build configurations which use the package update automatically.

For example, SoftBench provides packages for Motif, X11R6, SoftBench Encapsulator, RogueWave, and Oracle. If the requirements for building the library change, you can modify customized packages. All build configurations that include the package automatically update to use the modified package.

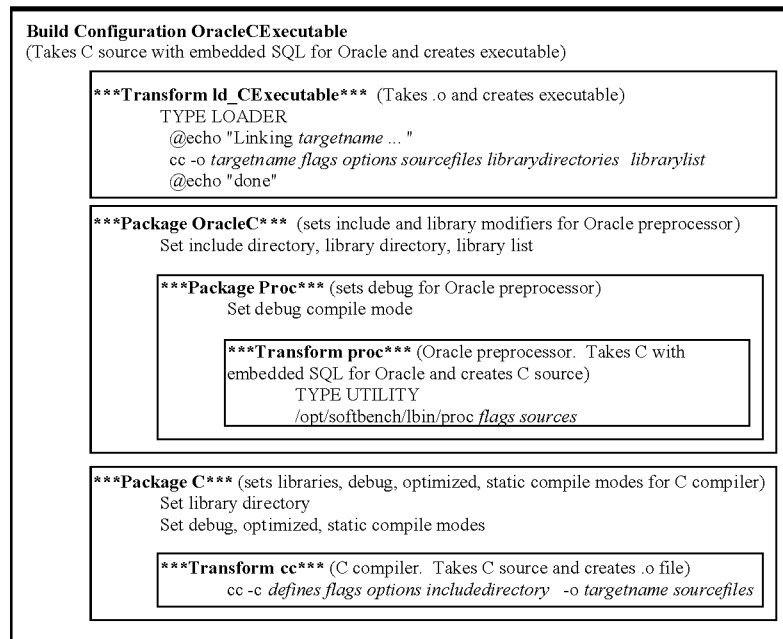
A **transform** is a set of operations that run a program or shell utility that converts a file from one form to another. For example, the C compiler transforms a source file into an object file. SQL preprocessors transform a source file with embedded SQL into a generated source file with all the SQL statements expanded. The `gencat` utility transforms a file of message strings into a message catalog. For more information about these concepts, refer to the "SoftBench Key Concepts" topics on "Build Configurations", "Build Packages", and "Transforms" available through "`Help: Overview`".

Transforms are the building blocks of build packages and build configurations. A build package contains zero or more transforms plus modifiers that influence how transforms operate. An example of a



package with no transform, just modifiers, would be the X11 library to be linked into an executable. A build configuration contains one or more transforms and zero or more packages (which may include transforms and/or other packages). A very simplified version of the actual build configuration to create an executable target from C source containing embedded SQL is shown in Figure 1-4.

**Figure 1-4 Simplified Build Configuration, Package, and Transform Series**



## Choosing Between Project Build and External Build Models

SoftBench supports two build models, allowing you two ways to specify your build instructions:

- With **project build**, you supply the necessary information to SoftBench, and SoftBench converts the build configuration and dependency information into build commands.

Project build lets SoftBench manage your build instructions. You can specify source-to-target dependencies and build order dependencies, choose convenient system or customized build configurations and build packages, and automatically generate Makefiles or do without Makefiles altogether if you prefer.

- SoftBench provides an **external build** model as an alternative to project build for people who already have a highly-tuned build environment. With external build, you edit and maintain the Makefile or build script. SoftBench provides an "External Build Command" dialog box for you to initiate your builds.

External build lets you use your own make utilities, such as `imake`, `make`, or `nmake` files or scripts. External build is the opposite of project build, where SoftBench handles building without the need for a Makefile. SoftBench does not read or parse your Makefile. By using external build you lose some of the conveniences that project build provides, such as access to the target graph and automatic generation of a Makefile. You can also add secondary source locations to SoftBench's search lists through **alternate source roots**.

The "External Build Command" dialog box can be utilized during project build to build software which is not in a project, or to execute a shell command and view the output in the output browser.

*Recommendation:* Use **project build** for your projects whenever possible. You may choose to use **external build** and maintain the make process yourself when:

- You have one source that becomes many types of objects.
- Your process cannot use file suffixes to tell what file types are in use.
- Your build process uses the `VPATH` environment variable for `make(1)`.
- You have a working build process and are perfectly happy with it.

## Sharing Projects with a Team

SoftBench projects provide the flexibility to have your development environment reflect your team interactions.

Sharing a **project description** has all the benefits and difficulties associated with sharing any source file. A centralized project description means that everyone is working from the same project files, but there can be collisions when more than one person wants to make changes. You may "break up" your file into several **subprojects** so that people can work independently and let SoftBench handle the complexities and relationships of the build.

There are several scenarios for partitioning a project:

- One Project, One Author
- One Project, Many Authors
- One Project with Subprojects, Many Authors

### One Project, One Author

When your project has only one author, you may want to set up a single project definition.

Alternatively, you can choose to use subprojects to organize your work hierarchically, if that is more convenient. For example, if the subproject is a library, you can modify the **build configuration** of targets in the parent project to use the library. If you work on both the library and its parent project, using the subproject relationship provides the flexibility to either build the subproject or use it in its current state. If you never want to build the subproject alone, setting up the subproject relationship may not be the best way to structure the project. You can just leave all your targets in one project and use build-order dependencies.

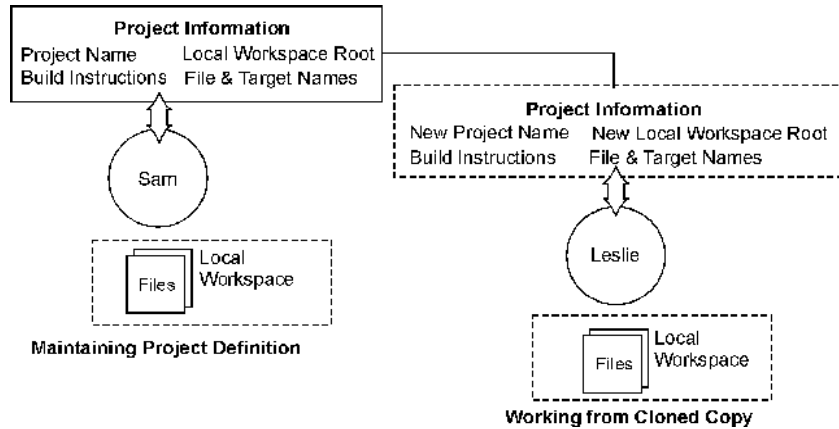
### One Project, Many Authors

As your project grows past what one author can accomplish, you face the issues of sharing the work across the team.

The preferred way to work at this level is to designate one person as the owner of the project definition file and have the rest of the team clone

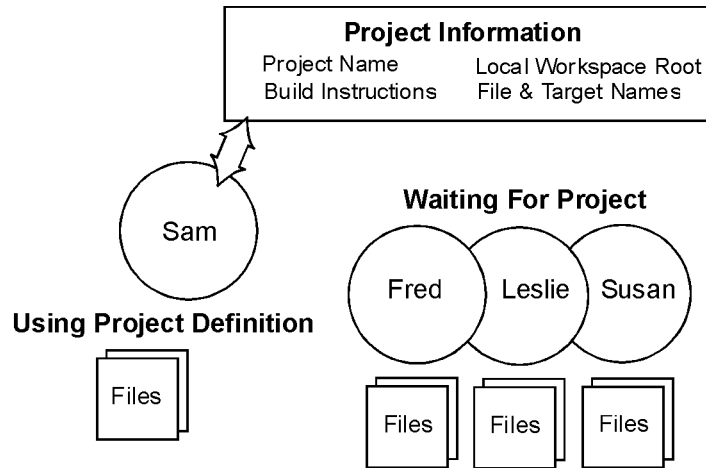
from that definition for their local working copies. Anyone who discovers the need for a change in the project definition communicates that to the owner, who makes the update, and then all of the team members re-clone their project definitions.

**Figure 1-5 Team Members Cloning the Project Definition**



The least usable method is to share the project definition files. Choose "Options: Set Default Project Root..." to specify the location of the shared project definition. Sharing a project definition has several limitations, primarily that only one person can open the project at a time.

**Figure 1-6 Team Members Sharing the Project Definition**



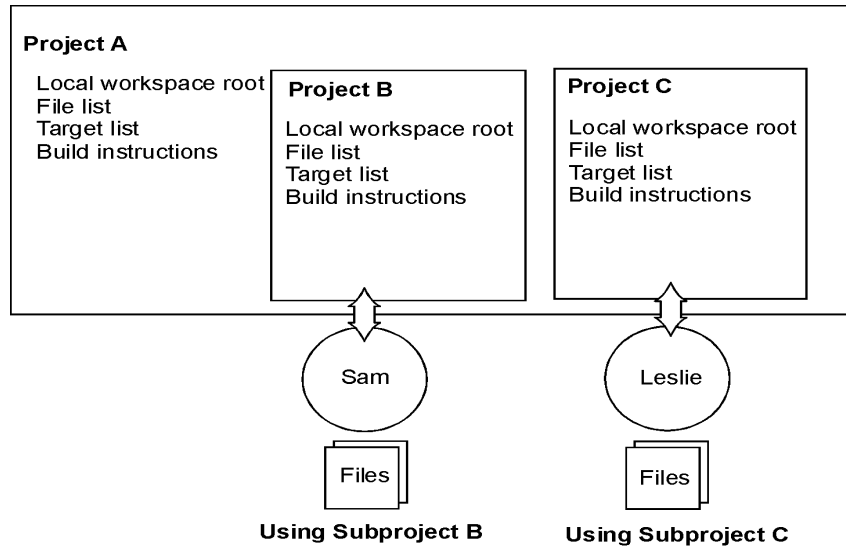
### **One Project with Subprojects, Many Authors**

Finally, your team can choose to divide up the project along the same lines as they divide up the work, especially if the project is large enough that multiple teams are cooperating on a single system release.

**Figure 1-7 Team Members Using Subprojects**

Each person's module of work may be designated as a subproject of the team's project. Team members then open the parent project for short periods of time, as necessary, but work primarily in the subprojects. Each team member's project is a subproject of the larger project, then the whole team's project is actually a subproject of the overall system delivery. Using SoftBench to manage this level of hierarchical complexity saves tremendous time and confusion when system builds are needed.

Maximizing Your Results with SoftBench  
Sharing Projects with a Team



## Planning Your Project

You can use SoftBench as your project planning environment by specifying the files of the project and their relationships even before the files exist. Unique icons in the Files view remind you which files have yet to be created in the file system. You can always change file lists to add, remove, or rename files as your project actually takes shape.

First determine how you will share pieces of your project among your team members. From this you can lay out your parent project and subproject definitions. See “Setting Up a Project” on page 50 for more information.

Next populate your projects with your files. You can add existing files into your project or create new files in your editor and have them automatically become members of the project. See “Adding Existing Files to a Project” on page 58 and “Using Editors with Projects” on page 118 for more information.

Define your targets and how to build them. Then set up dependencies between your source files and your targets, or between two targets that must be build in a specific order. Refer to “Specifying Dependency Relationships” on page 61 for details.

Now continue your development process in your SoftBench environment where all of your tools, from editor to debugger, understand your project files and build dependencies.

For in-depth information about using the project model in your development process, choose "Help: SoftBench How To".

## Using the SoftBench Integrated Environment

The SoftBench project environment provides smooth transitions between all of your SoftBench tools. See Figure 1-6 on page 29 and “An Example SoftBench Session” on page 38 for how the tools flow together. The tools shipped with SoftBench include:

### Main SoftBench Window with Builder and SoftBench CodeAdvisor

Define your projects, customize the SoftBench interface, and launch other SoftBench tools through the main toolface. You use Builder to transform your source files into your targets. CodeAdvisor inspects your source files for coding problems beyond what your compiler can detect.

### Editors

Making changes in your project code is easy with editors that understand your project structure. Configure your choice of SoftBench XEmacs Editor or SoftBench vi Editor. SoftBench Program Editor is also provided as contributed software for your convenience, but it neither understands projects nor automatically adds files to projects. SoftBench Class Graph/Editor allows you to see and modify your C++ program visually.

### Configuration Management

Keep your software changes under version control with your integrated configuration management system. SoftBench ships with SoftBench CM. Third party vendors also offer integrated configuration management systems integrated with SoftBench.

### SoftBench Static Analyzer

Evaluate the structure of your code without executing it using SoftBench Static Analyzer. For example, you can determine where variables are modified, classes are declared, and functions are called. Static Graphs



show you pictures of your code structure.

### SoftBench Debugger

Monitor your program's execution with SoftBench Debugger. You can set breakpoints, trace function calls, and watch variable values to isolate defects in your code. Data Graph Window gives you a visual image of your data structures as they change.

### SoftBench File Compare

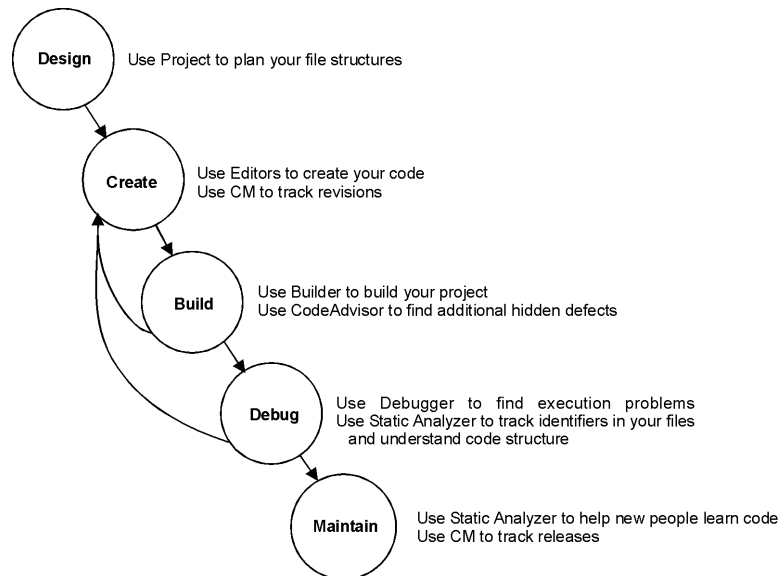
Compare two versions of a file and select lines from each to create a merged file using SoftBench File Compare.

### SoftBench Message Monitor

Watch SoftBench internal messages between tools to assist in isolating problems with SoftBench and to debug the integration of new tools into the SoftBench environment.

**Figure 1-8**

### Using SoftBench Tools Throughout Your Development Cycle



SoftBench supports three ways to use tools:

- using SoftBench tools together on one project
- using SoftBench tools together on more than one project
- using individual SoftBench tools in a standalone mode outside of projects

## Using SoftBench Tools Together

The recommended way to use SoftBench is as a set of tools that communicate with each other. To accomplish this, start SoftBench and launch other tools as you need them from the menus or toolbar in the main SoftBench window.

From the SoftBench window, you can perform many actions on the project data (source files, targets, and build information). Select the data on which you want to work, then choose the task you need to accomplish. Some tasks, such as building your application and checking your code for SoftBench CodeAdvisor violations, take place within the main SoftBench window. Other tasks automatically start other SoftBench tools, such as your configured SoftBench editor or SoftBench Debugger.

SoftBench's integrated environment allows you to move easily from one software development task to another. You can use the main SoftBench window to initiate tasks, and you can move from one task to another seamlessly using menu commands in other SoftBench tools.

**Figure 1-9 Tools That Can Directly Invoke Other Tools**

From:	You Can Start:							
	Main Window	Builder	CodeAdvisor	Editor	CM	Debugger	Static Analyzer	Class/Graph Editor
Main Window		X	X	X	X	X	X	
Builder				X				
CodeAdvisor				X				
Editor		X			X	*	X	
CM								
Debugger		X		X	X		X	
Static Analyzer		X		X		*		X
Class/Graph Editor				X	X	*		

\* Ability to set a breakpoint, not actually start Debugger

## Using SoftBench Tools on Multiple Projects

If you want to work on multiple projects at the same time, you can either start multiple sessions of SoftBench or you can open multiple projects within a single session of SoftBench. When you run a single session of SoftBench, you can access only one project, the current project, at a time in the main SoftBench window. With either alternative your assisting tools, such as the editor, stay open and available for use.

To open multiple projects within a single SoftBench session, select the project you want to open, then select **Open**. When you open a project, SoftBench:

- locks the project for write access

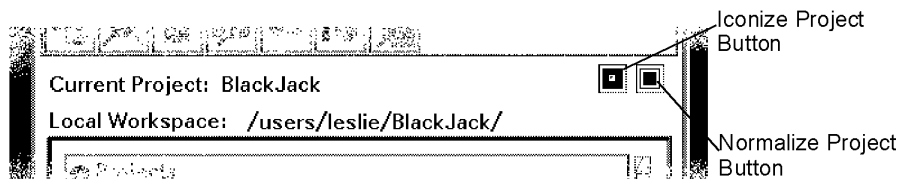
- makes the opened project the current project
- switches the project browser from the Projects view to the Files view, displaying the project's files

Only the current project's files display in the Files view. If you had another project open, SoftBench by default leaves it open and all its tools running. To close the previous project and stop its associated tools whenever a new project opens, set the "Options:  Close Tools when Switching Projects" toggle button. When you set it, the previous project closes, stopping associated tools. When you turn it off, the previous project remains open and any associated SoftBench tools remain running.

Each tool instance shows its associated project in its title bar. A build initiated from one of the tools builds the project named in its title bar. When you open a tool with one current project, then switch projects in the main SoftBench window, the tool remains attached to the project for which it was started. The editors save files in the project with which they are associated.

To help simplify window management when working in multiple projects, SoftBench provides "Iconize Project" and "Normalize Project" buttons (see Figure 1-10). From the Projects view, select the project, then select the button to either iconize or normalize the associated tool windows. Alternatively, you can use CDE workspaces to simplify window management for multiple projects.

**Figure 1-10** Iconize and Normalize Project Buttons



## Using SoftBench Tools as Standalone Tools

When you start SoftBench tools directly from the command line, the tools have no knowledge of project data associated with the files, and you cannot access build functionality. Other types of tool interactions, such as accessing the editor from SoftBench Static Analyzer, still work. However,

without access to project data, tool interactions may behave unpredictably.

*Recommendation:* Even if you want to use only one or two SoftBench tools, launch the tools from the SoftBench main window.

## Reusing Tool Windows

Within a single project, you can start only one instance of some tools and multiple instances of other tools. For example, within a project, you can open only one instance of SoftBench Static Analyzer, but you can open a different instance of SoftBench Debugger for each executable target in the project.

If you prefer to see only one instance of a tool, change the data on which the tool operates from within the tool itself. For example, in SoftBench Debugger choose "File: Unload Executable" to unload the current target, then choose "File: Load Executable..." to debug another program.

## Copying Data between SoftBench Tools

SoftBench shares internal project definition data across all the tools for you. At times you may want to copy some information from one tool's screen to another tool's input box. You can do this using the clipboard.

To copy information to the clipboard:

1. Select the text using the left mouse button and dragging across the text.
2. Press Control-Insert to copy the text to the clipboard.

To paste information from the clipboard to an input field:

1. Position the mouse cursor within the input field and click the left mouse button.
2. Press Shift-Insert to paste the text from the clipboard.

## An Example SoftBench Session

The following example shows how you can use SoftBench as you develop an application. To use the **project build** model where SoftBench takes care of the build instructions and Makefile:

1. Create a **project** by choosing "Project: New → Create...".
2. Set up a configuration management mapping between your local workspace root and the CM archive.
3. Write the code for your project. If the files already exist, choose "Project: Add File(s) to Project..." to make the files part of the project. To edit the file, double-click on the file name. To create new files, choose "File: New...".
4. Throughout the development cycle, remember to check files into your configuration management system. Select the files, then choose the desired command from the "File: Configuration Management" submenu.
5. Define the **targets** in your project and their dependencies:
  - a. Choose "Target: New..." to specify the target name and its **build configuration**. The build configuration provides instructions on how to build the target.
  - b. Create a dependency relationship between the files and targets. Select each source file that makes up the target, then choose "File: Link Source to Target...".
  - c. If the target needs special build instructions, select the target, then choose "Target: Modify Properties...". Within the "Modify Target Properties" dialog box, select **Customize Build Configuration....** In the "Customize Build Configuration" dialog box, make the changes that you need.
6. To **build** your project, select the project, then select the **Build** button. SoftBench displays the build results in the Builder's output browser. From the output browser, you can browse syntax errors in the editor and correct them before rebuilding.

If your project has **subprojects**, setting the "■ Build Subprojects" toggle button on the main toolface causes SoftBench to update all subprojects before building the current project.

7. When all syntax errors are fixed, use SoftBench CodeAdvisor (available with C++ SoftBench only, although rules exist for both C and C++) to find hidden coding problems not found by the compiler. Select the project or files of interest, then select the **Check Code** button.
8. To validate how well your application works, select the desired target, then choose either "Target: Debug..." or "Target: Run...". If you use "Target: Debug..." and find problems, you can start your editor from SoftBench Debugger, edit the source code, then choose "File: Build" to recompile the application and restart the debugging process.
9. To better understand your code, select the "Static Analyzer" icon in the toolbar. Alternatively, you can access SoftBench Static Analyzer from your configured SoftBench editor or SoftBench Debugger by selecting a program identifier, then choosing a command from the "Static" menu.
10. Cycle through editing, building, and debugging as many times as necessary to develop a correct executable.

If you are using the **external build** model where you write and maintain the build instructions, replace step 3 above with:

3. Define the targets in your project and their dependencies
  - a. Edit the Makefile or build script. With external build, you can save the build instructions for your Makefile targets when your build command supports entering a target name as part of the command:
  - b. Choose "Target: New..." to display the "External Build Command" dialog box.
  - c. Specify the build instructions for the external build target. For example, enter `make` in the "Command" input box, and enter the target name in the "Target" input box.
  - d. Select **Save as Target** to save the instructions as a target node that you can select in the project browser and target graph.

## Learning SoftBench

The SoftBench Online Tutorial teaches you the fundamentals of getting started with SoftBench. You can learn SoftBench using either a C or C++ example. The tutorial walks you through code development and maintenance tasks, introducing you to key SoftBench tools. Through this hands-on experience, you can learn the basic functions of the SoftBench tools and understand how to use these tools together to help you develop software.

To access SoftBench Online Tutorial, choose "Help: Tutorial" from the main SoftBench window. If you are interested in more in-depth training, contact your Hewlett-Packard sales representative.



## 2 **Using SoftBench**

This chapter provides the basic information that you need to start using SoftBench. From setting up your project to making sure that your software does what you intend, SoftBench provides an easy-to-use, integrated environment to support your software development tasks.

---

## Prerequisites to Using SoftBench

SoftBench requires that your `PATH` environment variable be set correctly. Your system administrator may have done this for you when SoftBench was installed. If you have a problem with your `PATH`, SoftBench notifies you.

To see the current value of your `PATH`, type

```
echo $PATH
```

Verify that `/opt/softbench/bin` precedes `/bin`, `/usr/bin`, `/usr/ccs/bin`, and `/opt/aCC/bin` and that `/usr/bin/X11` is included in your `PATH`.

If your `PATH` is incorrect, then you need to modify the `PATH` environment variable:

1. Edit the appropriate file for your environment.  
For Bourne, Korn, or Posix shell, edit `$HOME/.profile`. For C shell, edit `$HOME/.cshrc` or `$HOME/.login`.
2. Add the `PATH` statements shown below to the file. Use the format already in your file to guide your use of these examples.

**Table 2-1**

Shell	Syntax
For Bourne, Korn, or Posix shell	<pre>PATH=/opt/softbench/bin:/usr/bin/X11:\$PATH export PATH</pre>
For C shell	<pre>set path = ( /opt/softbench/bin /usr/bin/X11 \$path )</pre>

3. If you are running CDE, you may also need to modify your `$HOME/.dtprofile` file.  
Uncomment the line near the bottom of the file that instructs CDE to read your `$HOME/.profile` or `$HOME/.login` file (depending on which shell you are using). By uncommenting this line, the changes you made in Step 2 are used by your windowing environment.
4. In order for these changes to take effect, log out and log in again.

## Starting SoftBench

You can start SoftBench in two ways:

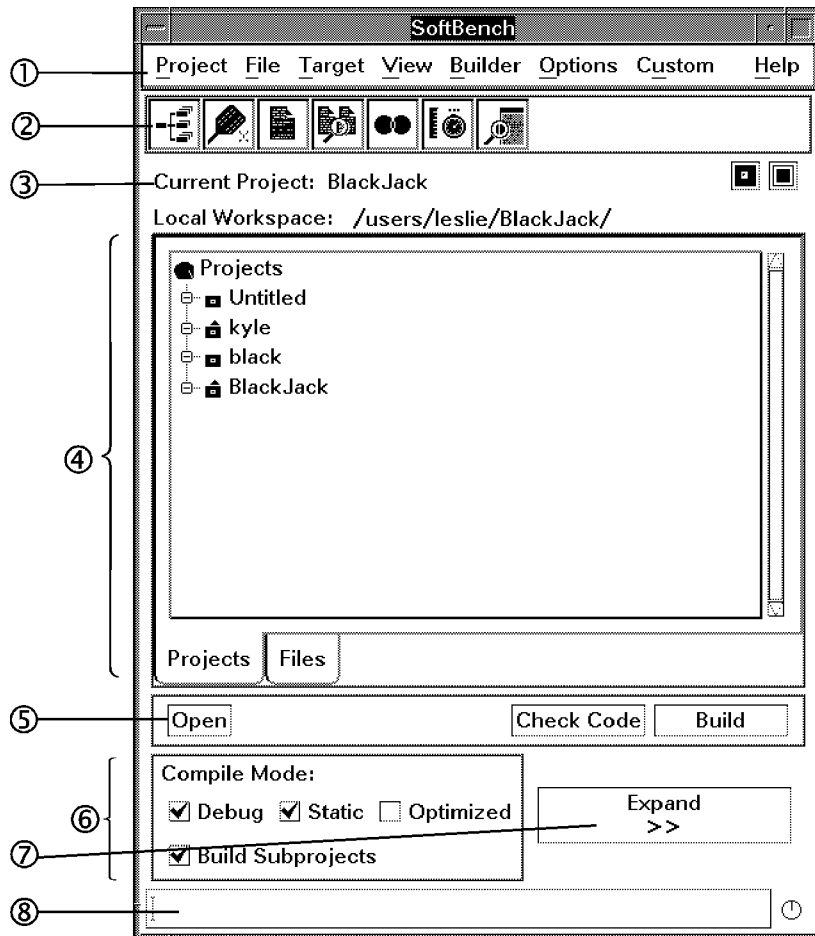
- From the command line, type  
`softbench`
- From CDE,
  1. Select the CDE Application Manager icon on the CDE front panel.
  2. In the CDE Application Manager, open the SoftBench folder and double-click on the SoftBench icon.

Either action displays the main SoftBench window and opens a scratch project named `Untitled`. If you have existing projects, SoftBench opens the last project in which you worked.

## Understanding SoftBench Window Areas

The main SoftBench window is shown in Figure 2-1. Table 2-2 contains descriptions of each window area.

**Figure 2-1**      **Unexpanded SoftBench Window**



**Table 2-2 Description of SoftBench Window Areas**

<b>Window Area</b>	<b>Description</b>
1 Menu Bar	Provides access to many SoftBench functions. When using most commands on the "Project", "File" and "Target" menus, select the <b>project</b> , file, or <b>target</b> of interest in the project browser or the target graph, then choose the menu command. Refer to SoftBench Online Help for detailed descriptions of any menu command.
2 Toolbar	Starts associated SoftBench tools using the data selected in the project browser or target graph. SoftBench displays the name of the tool when you place the mouse pointer over an icon.
3 Current Project Area	Displays the name of the current project and <b>local workspace root</b> , and provides buttons to iconize and normalize tool windows.
4 Project Browser	Presents project data in tree outline form. The project browser offers two views of your project data—the Projects view and the Files view, accessible by selecting the appropriate tab: <ul style="list-style-type: none"><li>• Use the Projects view to see available projects and understand dependencies and relationships between various levels of the application being developed.</li><li>• Use the Files view to access and use specific files and targets within the current project.</li></ul>

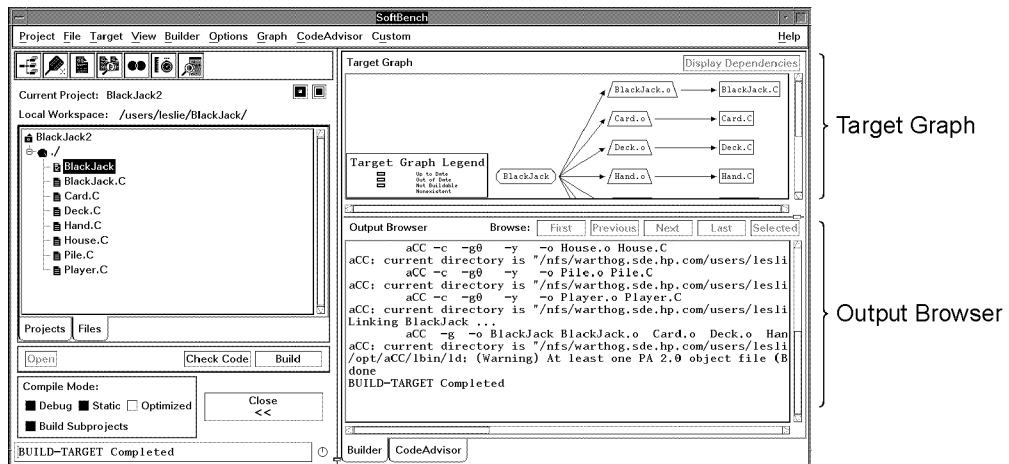
**Table 2-2 Description of SoftBench Window Areas**

<b>Window Area</b>	<b>Description</b>
5 Button Area	<p>Provides quick access to important actions:</p> <ul style="list-style-type: none"><li>• <b>Open</b> to open a project or make an already open project current</li><li>• <b>Check Code</b> (available with C++ SoftBench only) to find coding violations beyond errors already caught by the compiler</li><li>• <b>Build</b> to build the entire project or selected target(s)</li></ul> <p>Select one or more list items in the project browser or target graph, then select the desired button to initiate the action.</p>
6 Build Control Area	<p>Contains controls for the build process:</p> <ul style="list-style-type: none"><li>• "Compile Mode" toggle buttons set the "■ Debug", "■ Static", and "■ Optimized" compiler options</li><li>• " Build Subprojects" toggle button, controls whether <b>subprojects</b> of the current project are also built or used in their current state</li></ul>
7 Expand/Close Button	<p>The <b>Expand &gt;&gt;</b> button expands the SoftBench window to display the Builder or CodeAdvisor pages. When the window expands, the button changes to <b>Close &lt;&lt;</b>. The <b>Close &lt;&lt;</b> button closes the Builder and CodeAdvisor pages and returns the SoftBench window to its compact state.</p>
8 Status Line	<p>Displays messages about the status (startup, completion, success, failure, problem encountered) of actions that you take. The spinning clock icon indicates when a "Build" or "Check Code" operation is in progress.</p>

## Understanding the Builder Page

The Builder page, shown in Figure 2-2, consists of two main parts: the target graph and the output browser. For information on using the Builder page, see “Building Projects and Targets” on page 70.

**Figure 2-2** Expanded SoftBench Window Displaying Builder Page



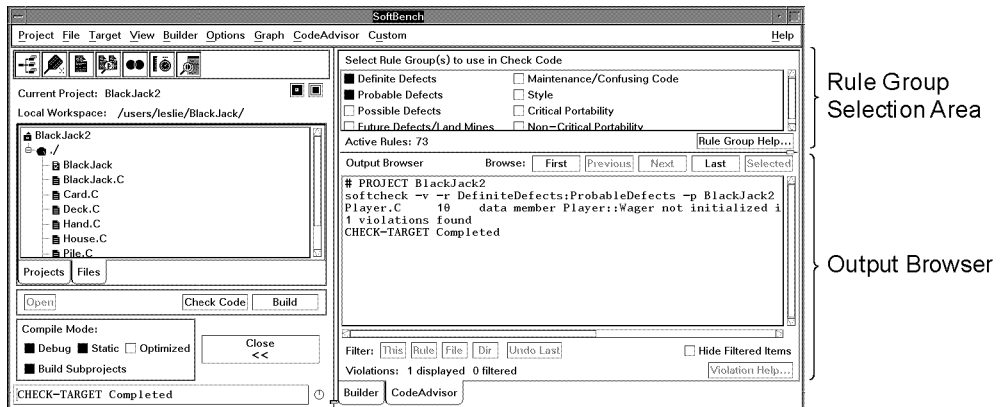
**Target Graph** Provides a graphical view of target dependencies in a project using the **project build** model. For **external build** projects, the target graph shows each target as a separate, selectable node, but shows no dependencies. The only operation available from target graph for external projects is Build. See “Using the Target Graph” on page 66.

**Output Browser** Displays build output. When compile errors occur, the error browsing buttons (**First**, **Previous**, **Next**, **Last**, **Selected**) provide easy access to the source code that triggered the compile error. See “Browsing and Correcting Build Errors” on page 74.

## Understanding the SoftBench CodeAdvisor Page

The CodeAdvisor page, shown in Figure 2-3, consists of two main parts: the rule group selection area and the output browser. This page is available only when you have C++ SoftBench.

**Figure 2-3** Expanded SoftBench Window Displaying SoftBench CodeAdvisor Page



**Rule Group Selection Area** Provides toggle buttons to select the type of violations you want to find. You can see the number of active rules that CodeAdvisor checks, based on the rule groups you select. The **Rule Group Help...** button provides help on the rules in each rule group. See “Selecting Rule Groups” on page 150 for more information.

**Output Browser** Displays SoftBench CodeAdvisor output. Controls include:

- Error browsing buttons (**First**, **Previous**, **Next**, **Last**, **Selected**) which provide easy access to the source code that triggered the rule violation.
- Filtering buttons (**This**, **Rule**, **File**, **Dir**, and **Undo Last**) which allow you to suppress rule violations you do not plan to fix. The **Undo Last** button allows you to undo the last filter operation.
- **Violation Help...** button which provides help on the



specific rule violation selected in the output browser. See “Viewing Violations” on page 151 for more information.

- “ Hide Filtered Items” toggle button which allows you to control whether you view the filtered violations, in the output browser.

For more information about SoftBench CodeAdvisor see “Checking Your Code Using SoftBench CodeAdvisor” on page 76 and Chapter 6, “Using SoftBench CodeAdvisor,” on page 145.

## Setting Up a Project

You can create a project several ways:

- Create a project from scratch. (See “Creating a Project Using Project Build” on page 50 and “Creating a Project Using External Build” on page 52.)
- Create a project by cloning another project. (See “Cloning a Project from an Existing Project” on page 53.)
- Repartition a project into two projects. (See “Repartitioning an Existing Project” on page 54.)

When you create a project, you must specify:

- The project name. Project names contain regular characters. Control characters, spaces, and punctuation characters special to the shell are not allowed in project names.
- The **local workspace root**, the top-level directory of the local workspace. For example, assume you have a project named `MyBigApp` with executables in `bin`, source code in `src`, and libraries in `lib`. These subdirectories are grouped under a directory named `$HOME/MyBigApp`. In this example, the local workspace root would be `$HOME/MyBigApp`. SoftBench resolves all relative file paths from the local workspace root.
- The type of build model. You can choose either **project build**, where SoftBench takes care of the build instructions and Makefile, or **external build**, where you write and maintain the build instructions (probably in your own Makefiles).

You can also share projects with other team members. (See “Sharing Projects with a Team” on page 27.) In addition to creating projects, you can set up **subprojects**, which create build order dependency relationships between projects. (See “Creating a Subproject” on page 55.)

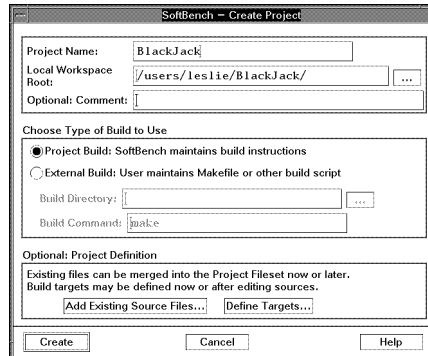
## Creating a Project Using Project Build

With **project build**, SoftBench manages the build instructions for the project. Project build frees you from writing and maintaining Makefiles, while allowing you to create complex projects spanning multiple directories and involving multiple targets.

To create a project build project:

1. Choose "Project: New → Create..." which displays the dialog box shown in Figure 2-4.

**Figure 2-4** Create Project" Dialog Box



2. In the "Create Project" dialog box, provide the project name, **local workspace root**, and a descriptive comment.
3. Select the "○ Project Build" radio button.
4. Optionally, select **Add Existing Source Files...** to specify the source files during project creation. Alternatively, you can choose "Project: Add File(s) to Project..." after you create the project. The button and the menu command perform the same function. (For more information, see "Adding Existing Files to a Project" on page 58.)
5. Optionally, select **Define Targets...** to specify the build commands for targets in the project. Alternatively, you can add targets after the project is created by choosing the command "Target: New...". The button and the menu command perform the same function. (For more information, see "Defining Targets for Project Build" on page 60.)
6. Select **Create** to close the "Create Project" dialog box and create the project.
7. Once you have created the project, added source files, and defined targets, specify source file dependencies by selecting the files for a target, then choosing "File: Link Source to Target...".
8. If necessary, modify the build configuration for the target by selecting the target, then choosing "Target: Modify Properties...". In the "Modify Target Properties" dialog box, select **Customize Build**

**Configuration...** and make the needed changes. (For more information, see "Customizing Build Configurations" on page 62.)

If you want to have SoftBench generate a Makefile for use external to SoftBench, for example in a nightly build script, choose "Builder: Generate Makefile...".

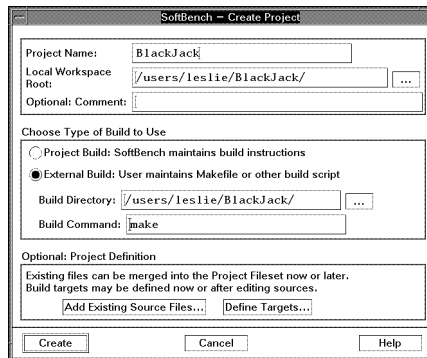
## Creating a Project Using External Build

**External build** provides build support when you write and maintain your own Makefiles or build scripts. With external build, you also need to specify the files that belong to the project, and you may also want to specify the targets (see "Defining Targets for External Build" on page 64). Doing so enables SoftBench to show your targets on the target graph for easy selection and building. SoftBench does not read or parse your Makefile.

To create an external build project:

1. Choose "Project: New → Create...".

**Figure 2-5** "Create Project" Dialog Box for External Build



2. In the "Create Project" dialog box, provide the project name, **local workspace root**, and descriptive comment.
3. Select the "○ External Build" radio button.
4. Your build directory should be the same as your local workspace root. If for some reason it is not, enter the build directory.
5. Enter the build command for building the project. For example, enter make to build the default target in a *make(1)* Makefile. Your Makefile

must properly handle paths relative to your build directory.

6. Optionally, select **Add Existing Source Files...** to specify the source files during project creation. Alternatively, you can choose "Project: Add File(s) to Project..." after you create the project.
7. Optionally, select **Define Targets...** to specify the build commands for targets in the project. Alternatively, you can choose "Target: New..." after you create the project.
8. Select **Create** to close the "Create Project" dialog box and create the project.

## Cloning a Project from an Existing Project

SoftBench allows you to clone existing projects. Cloning uses an existing project definition as the basis for a new project. The cloning process copies only the SoftBench description of the file set, target list, and build configurations. It does not copy the physical project source and target files.

See "One Project, Many Authors" on page 27 for a diagram of using cloning.

To clone a project:

1. Choose "Project: New → Clone...".
2. In the "Clone Project" dialog box, provide the project name, your **local workspace root**, and descriptive comment. choose a different local workspace root for the cloned project; otherwise you might overwrite files or targets unintentionally.
3. Select the project from the "Project to Clone" list.

If the project you are cloning is not in the list, then change the project search path:

- a. Enter the new "Project Search Path".
- b. Select **Search for Projects** to update the "Project to Clone" list.
- c. Select the project to clone.

For example, if you are cloning a project created by a coworker, you need to tell SoftBench where the coworker's project definitions are located. By default, SoftBench stores project definitions under `$HOME/.softbench`.

4. Select **Create**.
5. Once the project is cloned, make any needed changes to the project. For example, you may need to change the build instructions, add or remove files, or remove or define additional targets.
6. Then, make sure you have the project files under your local workspace root. For example, you may need to copy the files or check out the files from configuration management.

## Repartitioning an Existing Project

SoftBench provides the ability to restructure a project, creating a second project from parts of the original project. You may want to do this as your project outgrows a simple model or as you add developers to your team. During this process, you choose whether the new project is a **subproject** of the current project. You also choose whether the selected targets and files remain in the current project.

To restructure a project:

1. Select the files and/or targets to be restructured. Selecting targets automatically includes the source files required in the project description data.
2. Choose "**F**ile: Convert to Project...".
3. In the "Convert Files to Project" dialog box, provide the project name, local workspace root, and descriptive comment.

When you change your local workspace root to point somewhere further up or down its current hierarchy, SoftBench does not recalculate the relative path names. You can select the files in the project and choose "**M**odify Properties..." to change the relative path names.

4. Decide whether to make the new project a subproject of the current project.
5. Decide whether to remove the selected files and targets from the current project or leave them shared by both projects.
6. Select **Create**.
7. Once the project has been restructured, make any needed changes to both projects. You may need to change build instructions, add or remove files, or remove or define additional targets in the new project.

For external build projects, you may need to change build instructions in the current project if files have been removed.

8. Finally, make sure you copy the physical project files under your new local workspace root.

## Creating a Subproject

When you create a subproject relationship, you also need to create the build instructions that implement the relationship. For example, if the subproject is a library, you need to add the library to the target's build configuration in the parent project. (For more information, see “Customizing Build Configurations” on page 62.)

When you create a subproject, you can treat the subproject as code that you do not control and do not want to build, or you can build the subproject before building the project. The “ Build Subprojects” toggle button in the Build Control Area controls whether the subproject is built prior to building the parent project.

You can create a subproject in two ways:

- When the project already exists, select the project in the Projects view, then choose “Project: Make Subproject Of...”. In the dialog box, specify the parent project.
- When you are restructuring the current project and wish to convert part of it into a subproject, select the targets and/or files that you want in the subproject, then choose “File: Convert to Project...”. See “Repartitioning an Existing Project” on page 54 for more information.

## Modifying a Project Definition

Under certain circumstances you may wish to change information about your project definition. SoftBench allows you to change the name, location, and type of project by choosing "Project: Modify Properties...".

- To change the name of your project enter a new project name. Project names contain regular characters. Control characters, spaces, and punctuation characters special to the shell are not allowed in project names.
- To change the location of your project enter a new **local workspace root**. You can use the ... button to select from a list.

When you change the local workspace root, SoftBench preserves the exact relative path names for files. This allows you to maintain identical subtrees under two different roots for building different product releases. You can just change your local workspace root to point to one, then the other. However, if you try to change your local workspace root to point somewhere further up or down its current hierarchy, SoftBench does not recalculate the relative path names. You can select the files in the project and choose "Modify Properties..." to change the relative path names.

- To change the build type of your project select the " Project Build" or the " External Build" radio button. For external build, enter the build directory and build command. See "Creating a Project Using External Build" on page 52 for complete information.

Changing the build model for a project significantly affects SoftBench's internal representations of your project. All targets are deleted and must be redefined.

- To change the Static database location, enter a directory in the "Static Database Location" input box. You can use the ... button to browse the directory you want.
- To return to the default Static database location, clear the "Static Database Location" input box.



## Creating Files within a Project

New files that you create within SoftBench automatically become members of your project. Choosing "File: New" starts your configured editor with an untitled file. Alternatively, choose the "Editor" icon when you have no files selected. SoftBench vi Editor and SoftBench XEmacs's "File: Save" and "File: Save As..." menu commands automatically add the file into your project.

SoftBench also allows you to pre-define files before you create them. You can use this when you plan your project. Choose "Project: Add File(s) to Project". Select the "■ Choose Files Individually" toggle button and enter the name of the file in the text field. The path to the file must exist, but the file itself does not have to. You can later use your editor to open and edit this file.

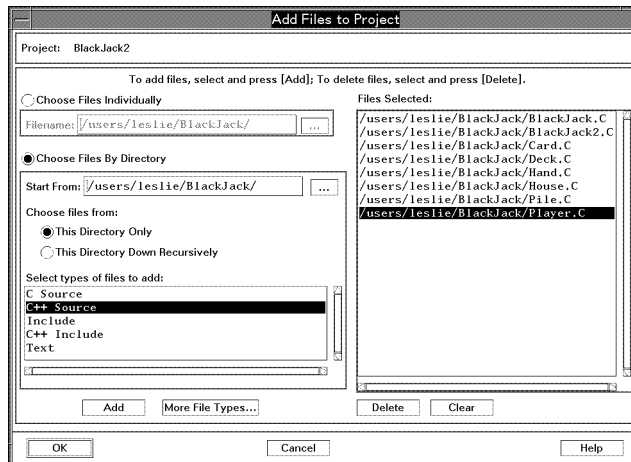
## Adding Existing Files to a Project

Once a project is created, you can add files to the project in a number of ways. SoftBench vi Editor and SoftBench XEmacs's "File: save" and "File: Save As..." menu commands automatically add a file into the project listed in the editor's title bar.

From the main SoftBench window, you can choose "Project: Add File(s) to Project..." (see Figure 2-6). This command allows you to add files individually, or you can add files in groups based on directory and file type. As you select files, SoftBench displays the file names in the "Files Selected" list.

You do not need to add intermediate files, such as .o files and generated C files, explicitly. Add files that you want to build as target files using the "Target: New" menu command.

**Figure 2-6** "Add Files to Project" Dialog Box



To add files one at a time:

1. Select the "Choose Files Individually" radio button.
2. Enter the file name, or select ... to browse to the desired file.
3. Select **Add** to add the file to the "Files Selected" list.

To add files in groups:

1. Select the "○ Choose Files By Directory" radio button.
2. Enter the directory name, or select ... to browse to the desired directory.
3. Optionally, select the "m This Directory Down Recursively" radio button. Note: The recursive option on a deep hierarchy can take a long time, because this process looks at every file in the hierarchy. The recursive option works best on fairly shallow or restricted hierarchies.
4. Select the desired file types. For example, to add C++ source files and header files, select both "C++ Source" and "C++ Include" in the "Select types of files to add" list. If the file types you need are not in the list, select **More File Types** to edit the list.
5. Select **Add** to add the qualified files to the "Files Selected" list. Qualified files are files in the specified directory (and possibly its subdirectories) with the specified file types.
6. To selectively delete files from the "Files Selected" list, select the files and select the **Delete** button. (This does not delete files from your file system.)
7. Select **OK**.

After you close the dialog box, the Files view displays the files that you added to the project. The Files view is a logical representation of a file set. It shows you all the files in the project, regardless of whether the files exist on the file system.

The icon beside each file in the Files view shows whether the file exists. When you create a file in the file system, or check a file out of configuration management, the file's icon appearance changes. A page with lines indicates that the file exists in the local workspace and a blank page indicates that it does not exist in the local workspace. A page with an arrow indicates a target, and lines indicate whether it exists in the local workspace or not. To update the Files view with the current file status, choose "View: Refresh Files View".

---

## Defining Targets for Project Build

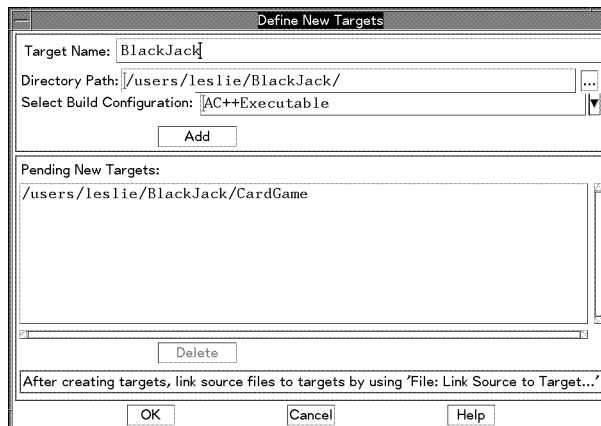
SoftBench attaches build instructions to targets. Each target in your project must be defined and have its build instructions described.

### Creating a Target

To define a new target:

1. Choose "Target: New...", which displays the "Define New Targets" dialog box shown in Figure 2-7.

**Figure 2-7** "Define New Targets" Dialog Box



2. Enter a target name in the "Target Name" input box. Target names must be unique within a project. Target names contain regular characters. Control characters, spaces, and punctuation characters special to the shell are not allowed in target names.
3. If desired, modify the directory name for storing the target in the "Directory Path" input box. The ... button lets you select a valid directory name. Otherwise, the directory name defaults to the **local workspace root**.
4. From the "Select Build Configuration" pulldown list, select the build configuration that most closely matches the target you want to create.

5. Select **Add** to add the new target to the "Pending New Targets" list.
6. Define as many targets as you need while the dialog box is posted.  
When you are done, select **OK**.

If the SoftBench-provided build configurations do not meet your needs, you can either define your own new build configuration or add target-specific customizations to an existing build configuration. See "Customizing Build Configurations" on page 62 for more information.

## Specifying Dependency Relationships

In the main SoftBench window, the Files view shows the newly created targets. Targets can have two types of dependency relationships:

- source-to-target dependencies
- build order dependencies between two targets

Dependency relationships can only be created for **project build** targets.

To create a source-to-target dependency:

1. Select the files to link into the target, then choose "**File: Link Source to Target...**".
2. In the dialog box, select the target.

To see the dependency relationship:

1. Select **Expand >>**. This action expands the main SoftBench window to display the Builder page.
2. If the CodeAdvisor page is displayed, select the Builder tab.
3. In the target graph, select the target.
4. Select "Source File Dependencies" from the **Display Dependencies** menu button.

To create a build order dependency, you need at least two targets:

1. Select the target that must be built first, then choose "**Target: Add Build Order Dependency...**".
2. In the dialog box, select the target that must be built last.

You can also use **subprojects** to create build order dependencies. See "Creating a Subproject" on page 55 for more information.

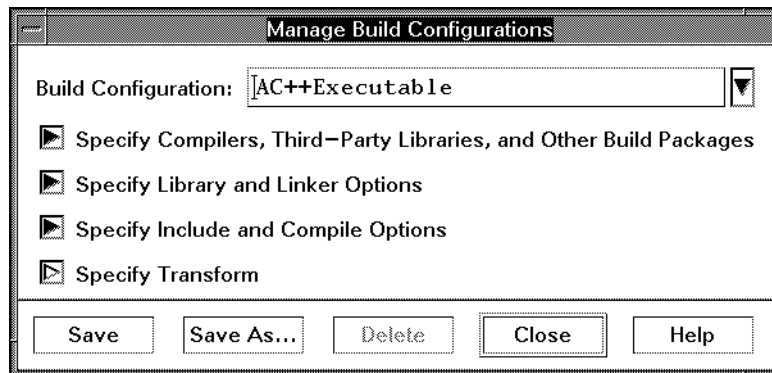
## Customizing Build Configurations

A **build configuration** specifies how to convert your source files into your targets. Build configurations may include **packages** as a shorthand notation for a common **transform** and its associated options or for common libraries.

To create or modify build configurations:

1. Choose "Builder: Manage Build Configurations..." which displays the dialog box shown in Figure 2-8.

**Figure 2-8** "Manage Build Configuration" Dialog Box



2. Select the build configuration from the "Build Configuration" drop-down list.
3. Modify the build configuration by expanding the sections of the dialog box and making the changes.

For example, if you need to add the math library to the build configuration:

- a. Expand the "Specify Library and Linker Options" section.
- b. Select the **Add...** button for adding libraries.
- c. Enter "m" in the input box, and select **OK** to close the "Add -l Libraries" dialog box.

In the "Manage Build Configuration" dialog box, the "Libraries (-l) to Use" list now contains "-lm" for the math library.

Depending on the changes you need, you can add packages (see “Using Build Packages” on page 63), libraries, library directories, include directories, compiler options, and other build instructions. To modify compile modes, defines, undefines, and compiler flags, you must select your compiler first.

4. Select **Save** to save changes under the current name. Select **Save As...** to save changes under a new name. If you selected a SoftBench build configuration, you must use **Save As...** to provide a new build configuration name. See “Understanding Build Configurations and Packages” on page 23 for information about SoftBench build configurations.

To customize a build configuration for a specific target:

1. Select the target in the project browser or target graph.
2. Choose "Target: **Modify Properties...**"
3. From the "Modify Target Properties" dialog box, select **Customize Build Configuration...**
4. Modify the build configuration by expanding the sections of the dialog box and making the changes.
5. Select **OK** to save the changes as a build configuration customization to the build configuration associated with the selected target. The "Modify Target Properties" dialog box reflects the addition of the customizations in the "Current Build Configuration" field.

Once you make a specific build configuration, SoftBench does not maintain a connection with the base build configuration from which it was derived. Changes you later make to the base build configuration are not inherited by the customized build configuration. You must repeat the changes for each customized build configuration individually.

For more information on customizing build configurations, refer to SoftBench Online Help for the "Customize Build Configuration" dialog box.

## Using Build Packages

To create or modify **build packages**, choose "Builder: **Manage Packages...**". Once a package is defined, you can include it in build configurations as a quick and reliable way to specify all build instructions for the compiler, library, or utility.

For more information on customizing build packages, refer to SoftBench Online Help for the "Manage Package Information" dialog box.

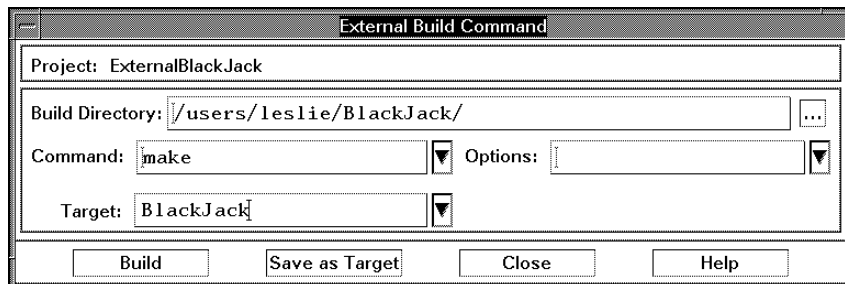
---

## Defining Targets for External Build

As part of SoftBench's support for the external build model, you can create targets and save the command to initiate your build:

1. Choose "Target: New..." to display the "External Build Command" dialog box shown in Figure 2-9.

**Figure 2-9** "External Build Command" Dialog Box



2. Enter the build directory, build command, target name, and options for a target.
3. Select **Save as Target**. This adds the target to the project file set, saves its build instructions, and allows you to bypass this dialog box in the future.
4. Repeat as necessary to define all of the project's targets, then select **Close** to close the dialog box.

Now you can build this project or individual targets within it as described in "Building Projects and Targets" on page 70.

When you use external build, your Makefile or build script contains the knowledge of how to build your targets. You must manually edit the appropriate file whenever the build instructions change.

To edit your Makefile or build script:

1. Add the Makefile to the project file set by choosing "Project: Add File(s) to Project...".

You can add Makefiles to projects using the file types mechanism if



the file is named "Makefile", "makefile", or ends in the ".mk" suffix.

- a. In the "Add Files to Project" dialog box, select **More File Types...**
- b. Add the "Makefile" file type to the "File Types to Use" list.
- c. Select the file type in the "Add Files to Project" dialog box.
- d. Select **Add**.

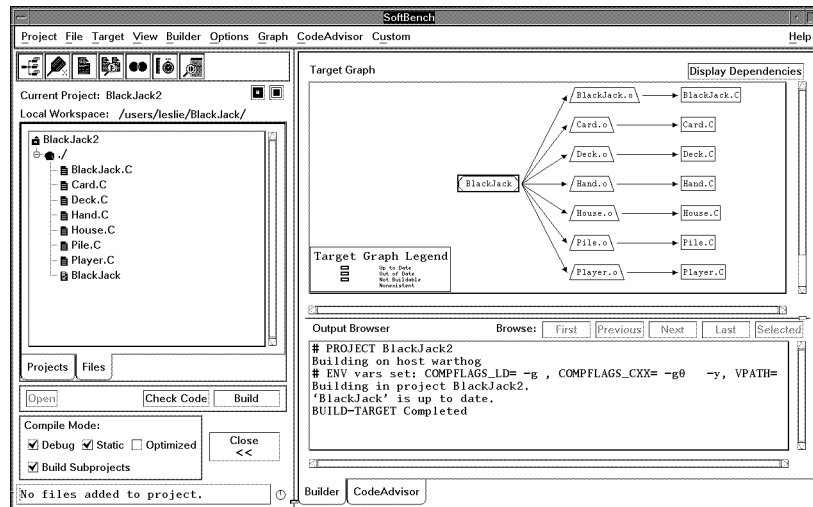
Alternatively, you can add a Makefile to the project by selecting the " Choose Files Individually" radio button and entering the Makefile name in the "Filename" input box.

2. Double-click on the Makefile in the Files view of the project browser, or choose "File: Open..." to load the Makefile into the editor. Modify the Makefile as necessary.

## Using the Target Graph

To conserve screen real estate, the target graph is not always visible. To display the target graph, select **Expand >>**, then select the **Builder** tab. The target graph is in the top half of the Builder page (see Figure 2-10). To hide the target graph entirely, turn off "view: Display Target Graph".

**Figure 2-10** Target Graph in Builder Page



Alternatively, you can display the target graph by selecting a target in the project browser, then choosing "Target: Display on Graph". Use the sash to expand or reduce size of the graph display area.

The target graph fully supports the **project build** model and partially supports the **external build** model.

**Table 2-3** Build Model Operations in the Target Graph

Operation	Project Build	External Build
Display Target Nodes	Yes	Yes

**Table 2-3 Build Model Operations in the Target Graph**

<b>Operation</b>	<b>Project Build</b>	<b>External Build</b>
Display Dependencies	Yes	No
Select Target Nodes	Yes	Yes
Operations Available	All	Build only
Graph Operations (save, find, zoom, hide, clear)	Yes	Yes

The target graph has selection-sensitive popup menus. Depending on what is selected, an appropriate popup menu is available. You can edit the dependencies shown on the graph by choosing the appropriate menu commands from the "File", "Target", or "Graph" menu or from the popup menu. Commands that interact with **intermediate files** are available only from the popup menu.

## Understanding the Graph

The target graph contains four types of nodes:

- **targets**—the end product of a build, such as a linked executable, library, or message catalog
- **intermediate files**—a file derived from a source file, but not the end product of the build, for example, an object file or a generated source file. These files are not part of the project file set.
- **source files**—files in the project file set from which targets are derived
- **include files**—files included in source files, and required, through the include relationship, to successfully derive a target

SoftBench uses shape and color to show the type and state of the node. Each node type has a different shape. Color describes the possible possible states for target graph nodes. See Table on page 68.

**Table 2-4**      **Graph Node States**

<b>Color</b>	<b>Meaning</b>
Blue	Up-to-date—the target or intermediate file does not need to be built.
Red	Out-of-date—the files from which the target or intermediate file is derived have changed and the node needs to be rebuilt.
White	Not buildable—the file is not an appropriate build target; source files and include files are not buildable.
Grey	Non-existent—the file does not currently exist on the file system.

Within SoftBench, actions which change a node's status cause the graph to update automatically. To update the display of graph nodes if you make changes to the files in the project from outside of SoftBench, choose "Graph: Update Status of Nodes".

## **Displaying Dependencies**

When you display targets on the target graph, choose one of the commands on the **Display Dependencies** menu button. The **Display Dependencies** menu commands include:

- Target Only (default value)
- Intermediate File Dependencies
- Source File Dependencies
- Include File Dependencies

To display dependencies:

1. Select the nodes for which you want to see dependencies.
2. Select the desired command on the **Display Dependencies** menu button.

## Controlling Graph Complexity

Controlling how many dependencies the target graph displays with the **Display Dependencies** menu button, discussed in the previous section, helps control graph complexity. Other mechanisms for controlling the complexity and appearance of the target graph include:

- Choose one of the "Hide" commands from the "Graph" menu. These commands allow you to hide selected nodes, children of selected nodes, or unselected nodes from the graph.
- Use the "Zoom" commands on the "Graph" menu to make the graph nodes larger or smaller.

## Building a Selected Target

To build a target from the target graph:

1. Select the target on the target graph.
2. Select the **Build** button, use the popup menu to access the "Build" command or choose "Target: Build" from the menu.

## Starting Your Configured Editor

When you have source or include file dependencies displayed on your target graph, you can start your editor preloaded with a file by either

- double-clicking on the source or include file
- selecting the file and choosing "Open File" from the popup menu or "File: Open" from the main menu
- selecting the file, then selecting the "Editor" icon from the toolbar

## Building Projects and Targets

Whether you use the **project build** model or the **external build** model, SoftBench provides the flexibility to build either the entire project or individual targets. When nothing is selected in the project browser, SoftBench acts as if the current project is selected. Selecting the **Build** button in this state builds the entire current project. Alternatively, you can select targets from the Files view or the target graph and select **Build** or choose "Target: Build".

The build action completes all steps necessary to create the target. For example, out-of-date files are compiled and object files are linked into an executable file or a library. When a build is not successful, SoftBench displays compile errors in the Builder output browser. (See "Handling Errors" on page 73 for more information.)

After you request a build, the **Build** button changes to **Terminate**. If you do not want the build to complete, select **Terminate** to end the build process. In most cases, this stops the build process. Occasionally, the **Terminate** button changes to **TERMINATE!**. If the build process does not stop, selecting **TERMINATE!** sends a `SIGKILL` signal which cannot be ignored by the build process. The build stops and displays an appropriate message in the output browser.

## Setting the Compile Mode

Setting the compile mode before you build is important, because using the correct compile mode is a prerequisite to debugging and static analysis. Regardless of the build model your project uses, you can set the compile mode using the "Compile Mode" toggles below the project browser. The compiler flags used for the compile modes are shown in Table 2-5. Initially, the toggle buttons for "■ Debug" and "■ Static" are set.

**Table 2-5**      **Compile Mode Settings**

Compile Mode	Default Compiler Flag
■ Debug	Select to prepare an object file for debugging; "-g" is the default for all compilers except aCC; "-g0" is the default for the aCC compiler.
■ Static	Select to prepare for static analysis; "-y" is the default.
■ Optimized	Select to tell the compiler to optimize the executable (to maximize the execution speed and minimize space usage); "-O" is the default.

For **project build** projects, you can override the default compiler flags sent to the compiler for a compile mode by changing the build configuration:

1. To change the compile mode settings for *all* targets using a build configuration, choose "Builder: Manage Build Configurations...".  
To change the compile mode settings for a *single* target, select the target, then choose "Target: Modify Properties...". In the "Modify Target Properties" dialog box, select **Customize Build Configuration...**
2. Expand the "Specify Include and Compile Options" section of the dialog box.
3. Select the compiler in the "Set Information for Compiler" drop-down list. This setting controls what compiler is affected when you enter information such as additional compiler flags, defines, undefines, and compile modes.
4. Select **Set Compile Modes...** to display the "Set Compile Modes" dialog box.
5. In the "Set Compile Modes" dialog box, enter the compiler flags. For example, to override the "-g0" compile mode setting for debugging aC++ code, enter another value, such as "-g" in the "DEBUG Compiler Flags" input box.
6. Save your changes and close the dialog boxes.

**External build** projects do not use the "Set Compile Modes" dialog box to change the compiler flags sent via the "Compile Modes" toggle buttons. To override the compiler flags for an external build:

1. Turn off the toggle buttons for the compile mode that you want to override.
2. Choose "Builder: Use External Build Command..." to display the "External Build Command" dialog box.
3. In the "Options" input box, specify the options you want make to send to the compiler. Which macro you use depends on how your Makefile is structured. The default rules for make accept CFLAGS for C and CXXFLAGS for C++. For example, you might enter CFLAGS=-y into the options field to have make pass the whole string to your compiler.

## Using the "External Build Command" Dialog Box

External build projects can use the **Build** button to repeat a build or "Builder: Use External Build Command..." to customize a build. This allows you to control the target to be built, the build command used, and the options you want make to send to the compiler. Once you enter this information, you can save it by selecting **Save as Target**. Then when you select the target in the browser, its information automatically appears in the dialog box. The dialog box also remembers a history of entries made in each field, accessible via the arrow button next to the field.

## Previewing the Build

When you preview a build, SoftBench simulates the build without performing the actions. SoftBench displays the commands that the build would run in the output browser. This information helps you determine which source code files create the targets, if any of your files are out-of-date or missing, or if there are problems in the build instructions.

To preview a build, choose "Target: More Build Actions → Preview Build".

As an alternative to previewing the build, you can choose "Graph: Update Status of Nodes". This command updates the target graph, changing the status of the nodes when appropriate. SoftBench displays out-of-date nodes in red and up-to-date nodes in blue.

The "Preview Build" command works only with **project build**. The command is equivalent to the make -n command. To see the same result



for **external build** projects using `make(1)` as the build command, enter the `-n` option in the "Options" input box of the "External Build Command" dialog box.

## Compiling Instead of Building

SoftBench supports both compiling and building the elements of a project. With a "Build" command, you select a target and SoftBench transforms the dependencies to create the target. During a build, SoftBench typically compiles files into object files, then links object files into a target.

In contrast, the "Compile" command transforms the selected source file into the next transformation state, for example, transforming the selected source file into an object file, but not linking the object file into a target.

To compile a file:

1. Select the file in either the project browser or target graph.
2. Choose "File: Compile".

The "Build" command is more reliable. SoftBench can predictably build your projects and targets, using either **project build** or **external build**. SoftBench can only make intelligent guesses about how to compile files in an **external build** project, because the build instructions are contained within the Makefile or build script and are not available to SoftBench's internal "Compile" command.

## Building Subprojects

When you specify a subproject relationship, SoftBench by default builds the subproject before building the current project. As with any build, SoftBench builds only out-of-date files (files that have changed since the last build). If you do not want SoftBench to build the subprojects, turn off the "■ Build Subprojects" toggle button in the Build Control Area.

## Handling Errors

When a build detects errors in your source files, SoftBench displays error messages in the output browser. You can go directly from the errors to the supporting source code that caused the error by double-clicking on the message.

## Interpreting Error Messages

The error messages displayed in the output browser depend on the compiler you are using. The messages always contain a line number, indicating the point in your source code where the compiler detected the error. The file name may be on the same line or on a previous line in the error output. An error message indicates the problem the compiler found. For more information on compiler errors, refer to "Help: Language Reference" and your appropriate language.

Error messages may not point to the actual error. For example, if you forget to declare a variable in a C program, the C compiler indicates the first *use* of the variable with the message:

```
cc: 'filename', line number: error errnum: name undefined.
```

To fix this problem, you may need to declare the variable earlier in the file, or perhaps in another file. You get the same message if *name* is misspelled; in that case, the compiler finds the actual error location, and the error is easy to fix.

After the first error in a location, the compiler sometimes produces misleading errors. Subsequent errors may be a ripple effect from the first error. You may want to fix the earlier errors, then build again.

## Browsing and Correcting Build Errors

You can double-click on the errors in the output browser to view and edit the related source code. The editor opens with the source file loaded and the cursor at the beginning of the line where the error was detected.

Alternatively, you can browse the compile errors using the error browsing buttons. Select **First** to select the first error in the output browser and go to the related source file in the editor. If you have write access to the file, you can edit the source, correcting problems as you browse.

Select **Next**, **Previous**, or **Last**, or double-click on other errors in the output browser, to display and correct other compile errors. If you have scrolled the window to where the currently highlighted error is no longer visible, select **Selected** to see it again. When all your errors have been corrected, save your file from the editor, and rebuild your project to make sure it builds successfully.

The output browser has a popup menu and pulldown menu available from "Builder: Browser" submenu with additional features to help you handle your errors:

- Choose "Open Editor on Selected Item..." to edit your source file.
- Choose "Find String..." or type Control-S to find a specific string in the text displayed in the output browser.
- Choose "Print/Save Browser Output..." to print or save the output browser's display.
- Choose "Load Browser from File..." to reload a saved copy of the browser output that allows you to address the compile errors later without recreating the error list through another build.

## Running the Build on a Remote System

You can specify a remote computer on which the "Build" command runs, including both the compile and link processes. When you take this action, the build process runs on another computer. Before you can run a remote SoftBench build, you need to configure both systems. Refer to *Installing SoftBench* for the configuration steps required.

To run the build on a remote computer:

1. Choose "Options: Build Settings...".
2. In the "Build Settings" dialog box, select the "Build Behavior" tab.
3. Enter the hostname in the "Compile Host" input box. When you want the build to run on the same computer as SoftBench, clear the input box or if the setting is "Local Host", allow that value to remain.
4. Select **OK** to save the remote compile host and close the dialog box.

## Checking Your Code Using SoftBench CodeAdvisor

If you purchase C++ SoftBench, you can use SoftBench CodeAdvisor to check your code for critical coding violations beyond compiler errors. SoftBench CodeAdvisor allows you to check the entire project, selected targets, or selected files.

To check your code:

1. Select **Expand >>** to expand the main SoftBench window.
2. Select the "CodeAdvisor" tab to display the CodeAdvisor page.
3. In the rule group selection area, select the rule groups that you want. For information on rule groups, select **Rule Group Help...**
4. Select the project, targets, or files that you want to check. When nothing is selected in the project browser or on the target graph, SoftBench assumes the entire current project.
5. Select **Check Code** to begin the code checking process.

You can see SoftBench CodeAdvisor results in CodeAdvisor's output browser. The output browser for SoftBench CodeAdvisor works similarly to the output browser for Builder errors (see "Handling Errors" on page 73). You can browse the source of rule violations as described in "Browsing and Correcting Build Errors" on page 74.

In addition, SoftBench CodeAdvisor offers several unique features as well:

- The line format is different:  

```
[ filename line_number violation_description [rule_name]]
```
- SoftBench CodeAdvisor provides filtering mechanisms (see Chapter 6, "Using SoftBench CodeAdvisor,") to suppress output that you do not want to see.
- SoftBench CodeAdvisor provides help on rule violations. Select the violation in the output browser, then select **Violation Help...**

For more information on using SoftBench CodeAdvisor, see Chapter 6, "Using SoftBench CodeAdvisor," on page 145.

## Running Other SoftBench Tools

SoftBench provides menu commands and a toolbar for starting other tools in the SoftBench environment. When you position the mouse over a tool icon, SoftBench tells you which tool the icon represents.

You can start tools in several ways:

- Select files or targets in the project browser or on the target graph, then select the tool icon in the toolbar. When you start tools with data selected, the tool loads that data.
- Use the tool icon on the toolbar when nothing is selected in the project browser. This starts the tool without any associated project data. Once the tool starts, you may need to specify a file. For example, if you select the "Editor" icon, choose "File: Open..." in the editor to specify the file to edit.
- Select files or targets in the project browser or on the target graph, then choose a menu command. For example, select a target, then choose "Target: Debug..." to start SoftBench Debugger with the executable file loaded.
- Drag and drop files from the CDE File Manager onto the tool icon. For example, drag an executable file onto the "Debug" icon to start SoftBench Debugger with the executable file loaded. However, dragging and dropping files does not make them project files. To do so, you must add them to the project with the "Project: Add File(s) to Project..." command or, for source files, save them in your project through the editor.

If the tool you want to start is not present on the toolbar or menus, SoftBench provides several options for accessing the tool:

- Choose "Options: Toolbar Setup..." and see if the tool you want is available in SoftBench. (See "Adding and Removing Tool Icons" on page 79.)
- Choose "Custom: Edit Menus..." to define a menu command that starts the tool you need. (See SoftBench Online Help on the "Custom" menu for information on adding custom menu commands.)
- Advanced users can integrate tools using SoftBench Encapsulator, then register the tools with SoftBench, following the instructions in "Registering New Tools with SoftBench" on page 80.

## Managing Your SoftBench Environment

SoftBench provides several ways to customize your environment. In general, customizations are located on the "View" and "Options" menus. The "Options" menu lets you tailor SoftBench's behavior to meet your needs. For example, you can:

- change the tool icons displayed on the toolbar
- set tool preferences
- change where SoftBench looks for projects
- customize build behavior
- set behavior of current project when opening a new project
- set language preferences (requires restarting SoftBench to take effect)

The "View" menu lets you tailor SoftBench's appearance. You can:

- refresh the Files view, which updates the status of the file icons
- show or hide the toolbar
- show or hide the target graph
- show or hide the local workspace root

The Custom menu allows you to create your own commands on the SoftBench toolface. You can add entries under the Custom menu, and have those entries available on all tool menu bars ("System" commands), or on all instances of a specific tool class ("User" commands). Any command that could be entered at a shell prompt can be launched from the Custom menu. (See SoftBench Online Help on the "Custom" menu for information on customizing SoftBench menus.)

This feature is especially useful if you need to run a command within the SoftBench environment. Commands launched from the Custom menu inherit the full SoftBench environment, including environment variables. See "SoftBench Environment Variables" in SoftBench Online Help for a listing of useful environment variables.

Choose "Custom: Edit Menus..." to add your commands. New commands are added to the "Custom" menu under the "User Commands" label. These commands are visible only to you.

You can make your custom menu entries visible to all tool classes in your SoftBench session. These entries appear under the "System Commands" label. To create these menu entries:

1. Create the desired menu commands using "Custom: Edit Menus...". SoftBench saves the menu information in `$HOME/.softbench/menus/custom/toolname`.
2. Rename the *toolname* file to `$HOME/.softbench/menus/custom/shared`.

Your SoftBench administrator can create system-wide commands that are visible to all users. These commands appear under the "System Commands" label as well. System commands may be scoped to apply to an individual tool class, or may be made available to every instance of every SoftBench tool. For information on creating "Custom" menus that are available to all users on the system, refer to SoftBench Online Help.

You must use the "Custom: Edit Menus..." command to create the menu files. Do not edit the menu files directly.

For more information on adding menu commands, see SoftBench Online Help.

## Adding and Removing Tool Icons

The toolbar contains a set of frequently used tools, but this set may not be right for you. You can easily add new tool icons to the toolbar:

1. Choose "Options: Toolbar Setup...".
2. From the "Available Tools" list, select the tool you want to add.
3. Select **Add to Toolbar**.

Alternatively, you can double-click on a tool in the "Available Tools" list to select and add the tool in a single step.

4. Select **OK**.

If your preferred tool does not appear on the "Available Tools" list, see "Registering New Tools with SoftBench" on page 80.

To remove tool icons using the "Toolbar Setup" dialog box, select the tool in the "Tools on the Toolbar" list, then select **Remove from Toolbar**.

To reorder the tool icons, remove all the icons, then add them back in the order you want.

## Choosing Tool Preferences

Some classes of SoftBench tools offer more than one choice of tool. For example, SoftBench supports two editors, SoftBench XEmacs Editor and SoftBench vi Editor. Other editors are available as SoftBench contributed tools and as third-party encapsulations.

To see your tool choices, choose "Options: Tool Preferences...". If your preferred tool does not appear on the "Available Tools" list, see "Registering New Tools with SoftBench" on page 80 for information about making it available. To change which tools are used:

1. From the "Select a Tool" list on the left, select the tool class.
2. From the "Available Tools" list on the right, select the specific tool you want.

If you have created a `$HOME/.softbench/bmsinit` file, setting tool preferences through SoftBench overrides your `bmsinit` entries. To use an entry in your `bmsinit` file, set the tool preference to `Default` for the tool class.

3. Select **OK**.

To make your change in tool preferences take effect, you need to stop all processes related to the tool class you changed. For example, to start a new editor, close all text editing windows and the editor index window (if applicable).

## Registering New Tools with SoftBench

SoftBench provides an open, extensible environment in which advanced users can develop tool encapsulations and run those encapsulations as SoftBench tools.

To use a third-party encapsulated tool or create a new encapsulation and allow all users on the system to access it from the SoftBench toolbar:

1. Install the third-party encapsulated tool or write the encapsulation using SoftBench Encapsulator. For more information, see the Encapsulator SDK Integration documentation on the SoftBench online support page found at:

<http://devresource.hp.com/softbench>

2. Register the tool with SoftBench Broadcast Message Server by adding



- a file to the `/opt/softbench/config/bmsinit` directory. See the *bmsinit(5)* man page for more information.
3. Register the tool with SoftBench toolbar by editing the `/opt/softbench/config/toolbar/config` file. Changing this file requires superuser access. The file documents the format for new entries.
  4. Add the new tool to the toolbar. See "Adding and Removing Tool Icons" on page 79. You may need to use the "Default" selection for your tool preference for the tool class.

## Customizing SoftBench by Setting Resources

You control most customizations through the SoftBench user interface. However, a few customizations must be made by setting X resources in the `$HOME/.softbench/softbenchrc` file or by using an X mechanism such as `xrdb`.

Set X resources only when the customization is not available from the user interface. Refer to the tool's man page for detailed information on X resources. SoftBench Online Help provides a high-level summary of the resources as well. For SoftBench resources, see the *softbench(1)* and *softbench(5)* man pages and the "Customizing SoftBench" entry in SoftBench Online Help (accessible by choosing "Help: Welcome"). Most resource files under `$HOME/.softbench` should not be edited since they are overwritten by the tools.

## Accessing Distributed Data and Tools

Before you use SoftBench in a distributed environment, your network must be properly configured. Work with your system administrator, using the 'Setting Up Network-Distributed Operation' chapter in *Installing SoftBench*.

SoftBench takes advantage of networking in several ways:

- You can direct SoftBench tools to access data anywhere on your network that you can reach through NFS. Supply the path name to the directory or file that you want.
- You can execute build and debug processes on any system where SoftBench is installed. Remote execution allows you to use dedicated servers for particular jobs, such as compilation.

To specify remote execution for builds, choose "Options: Build Settings...", then select the "Build Behavior" tab. Enter the remote system name in the "Compile Host" input box. See "Running the Build on a Remote System" on page 75 for more information.

To specify remote execution for SoftBench Debugger, choose "Options: Debug Host..." from the SoftBench Debugger menu bar and enter the remote system name.

- You can execute SoftBench on one system and display it on another. See "Running SoftBench on a Remote System" on page 82 for more information.

## Running SoftBench on a Remote System

To run SoftBench remotely on another computer that has SoftBench C.06.0 or newer installed and display SoftBench on your local system, do the following:

1. On your local system, start the X Window System and execute the following command in a terminal window:

```
xhost remotehost
```

Where *remotehost* is the name of the system on which you want to run SoftBench.

2. Log in to *remotehost* and set your DISPLAY environment variable to the name of your local display.
3. Start SoftBench on *remotehost* from a command line:

```
softbench
```

## Integrating with CDE

When you install SoftBench, you automatically get a CDE integration. With the integration, you can start SoftBench from CDE Application Manager, and you can drag and drop files from CDE File Manager onto the SoftBench toolbar. However, dragging and dropping files does not make them project files.

## Stopping SoftBench

You can stop individual SoftBench tools, or you can stop SoftBench and all tools associated with the SoftBench session:

- To stop a SoftBench tool, choose "File: Exit" on the tool's menu bar.
- To stop all tools associated with a project, close the project. Select the project in the project browser, then choose "Project: Close".
- To stop SoftBench and all tools associated with a SoftBench session, choose "Project: Exit SoftBench" in the main SoftBench window. If you are working in an Untitled project, SoftBench asks you to name your project or delete it before you exit.

## Restoring Your Previous SoftBench Session

When you stop SoftBench, session information is saved. When you restart SoftBench, SoftBench reopens the current project, but does not restart any tools.

To start SoftBench with a different current project, type

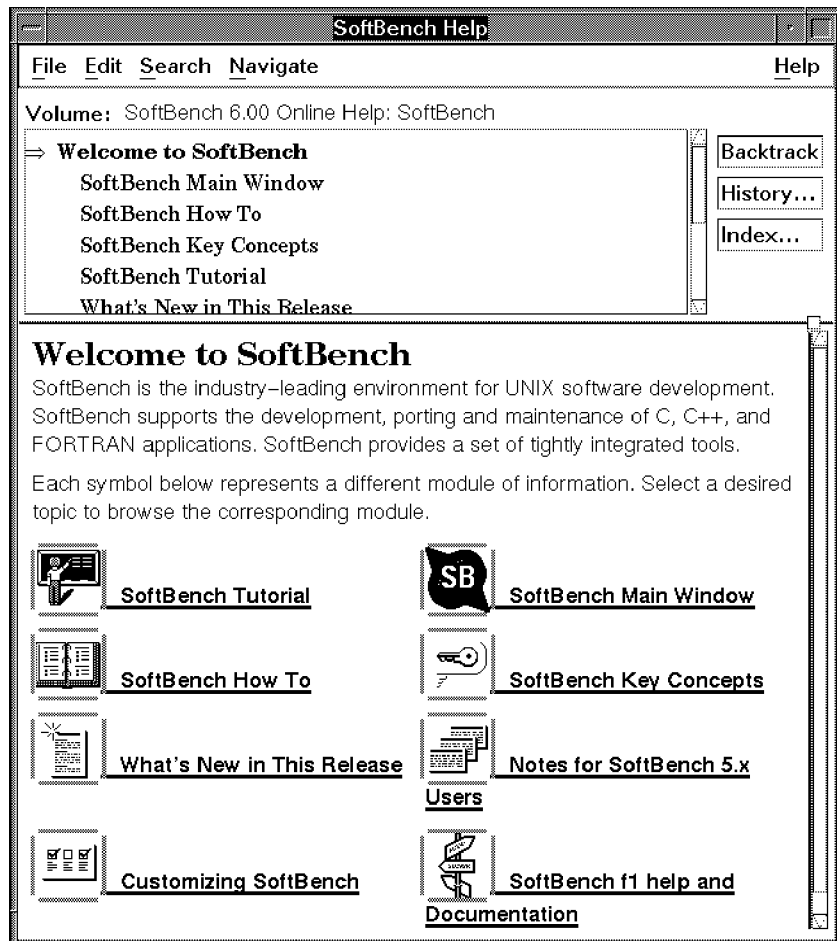
```
softbench -project project_name
```

---

## Getting Help

SoftBench provides online help, accessible from the "Help" menu. You can access general help on any SoftBench tool, task-oriented help, or specific help on a selected window item. Figure 2-11 shows the Welcome window from SoftBench Online Help.

**Figure 2-11**      **SoftBench Online Help Window**



## Using the Help Menu

All SoftBench tools display "Help" as the last item in the menu bar. Common online help menu commands include:

**Tool Overview** Gives you an overview of the tool and a list of subtopics. In the main SoftBench window, this menu command is "Help: Overview".

**Show Man Page** Displays the man page for the current tool.

**On Item** Displays a question mark cursor; move the cursor over the portion of the screen for which you want help and click the left mouse button.

**Using Help...** Provides assistance for using SoftBench Online Help.

Other help options are available on a tool by tool basis. For example, you can access the SoftBench Welcome window, SoftBench Online Tutorial, SoftBench copyright information, and language specific help from the main SoftBench window. You can access help for the underlying DDE debugger in SoftBench Debugger.

## Accessing On Item Help

To view information about screen areas such as menu commands, input boxes, or dialog boxes, move the mouse pointer over the item, then press the Help key (F1).

If the system cannot find any help information for the screen area under the pointer, a dialog box appears with the message:

```
A request to the help server failed.  
The desired help is not available.
```

You can move the mouse pointer to a slightly different screen element and press F1 again.

---

## If Something Goes Wrong

Table 2-6

Condition or Message	Explanation
Toolbar is missing	Check the setting of the "View: <input type="checkbox"/> Display Toolbar" toggle button. Alternatively, check "Options: Toolbar Setup" to ensure that you have tools selected or to configure your tool preferences.
Target graph missing from Builder page	Check the setting of the "View: <input type="checkbox"/> Display Target Graph" toggle button. Set the toggle button to display the target graph.
You want to rebuild a target or project, but SoftBench reports that the target is up-to-date.	Choose "Target: More Build Actions → Force Build" to force SoftBench to rebuild a <b>project build</b> project. Alternatively use "Target: More Build Actions → Remove Intermediate Files (clean)" or "Target: More Build Actions → Remove All Derived Files (clobber)". For an <b>external build</b> project, touch or remove the "*.o" files.
You set a new tool preference, but it does not take effect.	Some tools have background processes that do not shut down when you close the tool window. For example, SoftBench CM has both a user interface process and a message server process. Only the user interface process stops when you close SoftBench CM. You can stop all running processes by stopping SoftBench.
You cannot open a project because the project is locked.	Close it in another SoftBench session you may be running, or ask your teammate to do so.

## For More Information

- On getting started with SoftBench, choose "Help: Tutorial".
- On understanding particular features of a SoftBench tool, build configurations, build packages, and transforms, see SoftBench Online Help.
- On compiler-generated error messages, see the language reference manual available under "Help: Language Reference".
- On SoftBench CodeAdvisor, see Chapter 6, "Using SoftBench CodeAdvisor," on page 145 and the *softcheck(1)* man page, available under "Help: Show Man Page".
- On installing and configuring your SoftBench environment, see *Installing SoftBench*.
- On customizing SoftBench, see the "Customizing SoftBench" entry in SoftBench Online Help (accessible by choosing "Help: Welcome").

Using SoftBench  
**For More Information**



# 3 Using SoftBench Configuration Manager

SoftBench CM is a configuration management tool that helps manage code between software team members, software teams, and even corporate sites. SoftBench CM is fully integrated with the SoftBench environment, allowing access to configuration management functionality.

SoftBench CM provides many advantages:

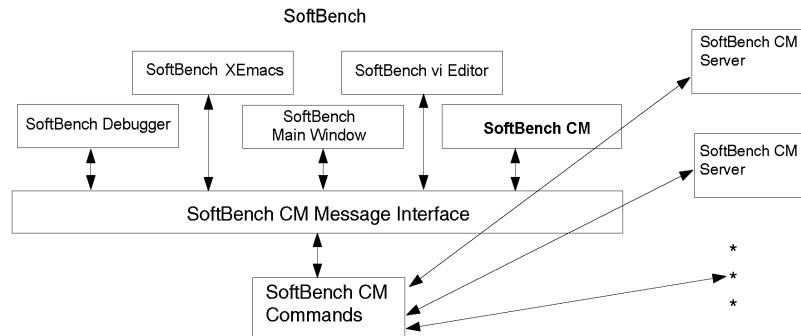
- network access to archive files
- consistent access to local and remote archive files
- menu interface for common commands
- browsing of archive directories
- delete and rename archive file operations
- lock, unlock, and break-lock archive file operations
- recursive display of archive directories
- display, cut, and paste from archive files without the overhead of check out
- tight integration with SoftBench

## Understanding SoftBench CM

SoftBench CM helps you manage software development projects that consist of many versions and configurations. Using SoftBench CM, you can retrieve and build any version of an application in a consistent, repeatable manner.

SoftBench CM is fully integrated with the SoftBench environment. You can access the tool through the SoftBench main window or from the SoftBench vi Editor, SoftBench XEmacs Editor, SoftBench Debugger, and SoftBench Class Graph/Editor (see Figure 3-1). The SoftBench main window and related tools include menu selections for checking files into and out of an archive, creating initial file versions, cancelling file check outs, showing the revision history of files, and comparing file revisions.

**Figure 3-1** Integration of SoftBench CM with SoftBench



Each SoftBench CM server is configured to manage one or more **archives** — a directory hierarchy consisting of **versioned files**. You can access archives on multiple local or remote servers, support an unlimited number of licensed users per server, and store as many files as archive disk space allows. SoftBench CM uses GNU RCS™ as its versioning system, so each versioned file is an RCS file that contains file revision information, multiple revisions of content, descriptive text, and control attributes.

To access files contained within a given archive, you establish a **mapping** between the archive file system and your local file system. This lets you create local copies of the files you need to access or browse

within the archive. Using the SoftBench CM interface and the SoftBench main window together provides a complete view of your configuration management files. The SoftBench main window lets you browse the local files associated with a **project**, and SoftBench CM lets you browse archive files and directories.

## Getting Started - A Brief Overview

SoftBench CM is an archive browser that lets you view, traverse, and modify the SoftBench archive system. SoftBench CM and SoftBench are integrated so that you use SoftBench CM to browse archive files and the SoftBench main window to manage projects containing your local files.

To begin using SoftBench CM for your basic configuration management needs, follow the steps in this section:

1. Verify that SoftBench CM is your preferred configuration management tool.
  - a. Choose "Options: Tool Preferences..." in the SoftBench main window to ensure your configuration management tool is set to SoftCM. (See "Choosing Tool Preferences" on page 80 for more information.)
  - a. Start SoftBench CM by clicking on the "Configuration Manager" icon from SoftBench main window.

SoftBench CM starts and displays the setup instructions if this is the first time you have started SoftBench CM. Once you close this window, you can view the instructions again by selecting "Actions: Show Set Up Instructions..." or print the file located in `/opt/softbench/share/welcome.txt`.

2. Select an archive.

Your system administrator should have configured one or more servers for managing your archive(s).

- a. Choose "Actions: Show Local Server Information..." to see a list of local servers and archives.

Machine configuration automatically sets up a test archive (/TestArchive) on each server. If the dialog box displays with no server/archive names, ask your system administrator to recheck the installation process.

- b. Select the archive you want to view.
- c. Select **Browse Archive**.
- d. Select **Done** to close the dialog box.

The SoftBench CM window displays the files and directories contained within the selected archive.

3. Create an archive directory.
  - a. Navigate through the current server to the area where you want to create an archive directory.
  - b. Choose "Directory: Create...".
  - c. Enter the name of the directory you want to create in the "Create Directory" input box.
  - d. Select **OK**.

4. Create a mapping.
  - a. Choose "Actions: Create New Mapping...".
  - b. In the "Mapped to Local Directory" input area, enter the path and name of the local directory that corresponds to the selected archive directory.
  - c. Select **OK**.

The SoftBench CM window displays the empty archive directory that is mapped to your local directory.

5. Create an initial archive file.
  - a. Open the archive directory in which you want to create an initial archive file.
  - b. Choose "File: Create...".
  - c. Enter the name of your new file in the "Create File" input area.
  - d. Select **OK**.

The name of the file you created displays in the current archive directory of SoftBench CM. If a mapped file of the same name exists on your local system, the newly created archive file is identical to the local file. Otherwise, the archive file contains no data.

6. Check out archive files to a local directory.

Once you have created a mapping, you can check out files and directories from the archive to your local system.

- a. Select the archive file(s) you want to check out.

- b. Choose **"Actions: Check Out to Local Directory"**.
  - c. Select the appropriate submenu:

If you want to edit an archive file, check it out as locked so that you have read-write permissions and others know you are modifying the file.

Checking a file out unlocked gives you a read-only copy of that file. If a file with the same name exists in your local directory, SoftBench CM overwrites it with the current archive version. If this file does not yet exist in your local directory, SoftBench CM creates a copy.
7. Check in files to an archive.

Once you finish modifying a file, check the file in so that the changes are reflected in the archive.

  - a. Select the archive file(s) you want to check in.
  - b. Choose **"Actions: Check In from Local Directory..."**.

You can include comments, revision number, and/or state.
  - c. Select **OK**.
8. Include new files or directories in the related project.

When you update your local directory from SoftBench CM, new files or directories are not included in a SoftBench project. The files and directories are updated only into the mapped local file system.

  - a. From the SoftBench main window, choose **"Project: Add File(s) to Project..."** to include new files or directories in a project.

## Managing the Archive System

An **archive** is a file system located on a separate archive server. You use archives to store **revisions** of your local files and related information, including file revision history, descriptive text, and control attributes. You can establish a relationship between archive files and directories and your local system by creating **mappings**. (See “Creating a Mapping between an Archive Directory and Local Directory” on page 98 for more information.)

### Browsing Local Network Servers and Archives

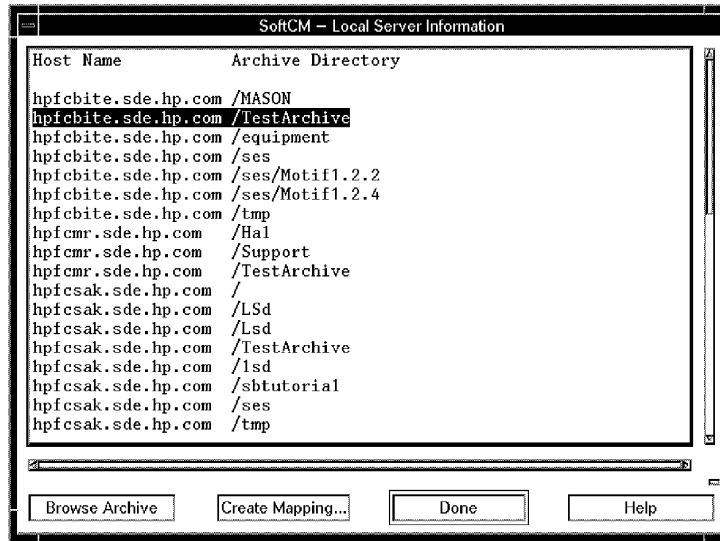
SoftBench CM lets you access archives on multiple servers from your local network.

To browse a server and its associated archive:

1. Choose "Actions: Show Local Server Information...".

The "Local Server Information" dialog box displays the Host Name for each archive directory that is set up on your local network (see Figure 3-2).

**Figure 3-2** Viewing Local Servers and SoftBench CM Archives



2. Select the server and archive directory you want to open and the archive you want to browse.
3. Select **Browse Archive** to view the contents of this archive.

The SoftBench CM main window displays the contents of the current archive directory (see Figure 3-3). To traverse directories, double-click on the desired directory. To move up a directory, double-click on the "Parent" directory. If the archive is not mapped, SoftBench CM displays a "Not mapped" message at the top of the directory display area. (See "Creating a Mapping between an Archive Directory and Local Directory" for more information.)

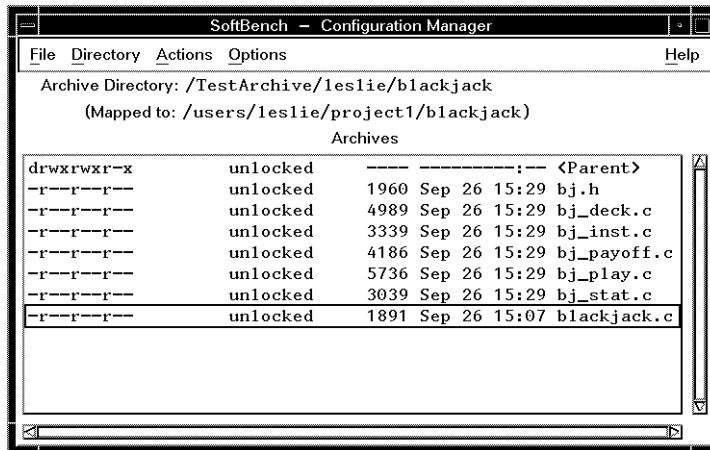
The default archive listing for the SoftBench CM main window includes:

- UNIX file permissions
- File type
- Lock status (If locked, SoftBench CM displays the name of the lock holder)
- Archive file size
- Date of last archive modification



- File or directory name

**Figure 3-3**      **SoftBench CM Main Window**



## Creating an Archive Directory

You use an **archive** to store revisions of your local files and make them available to other development team members. To help better organize projects, structure files in directories.

To create an archive directory:

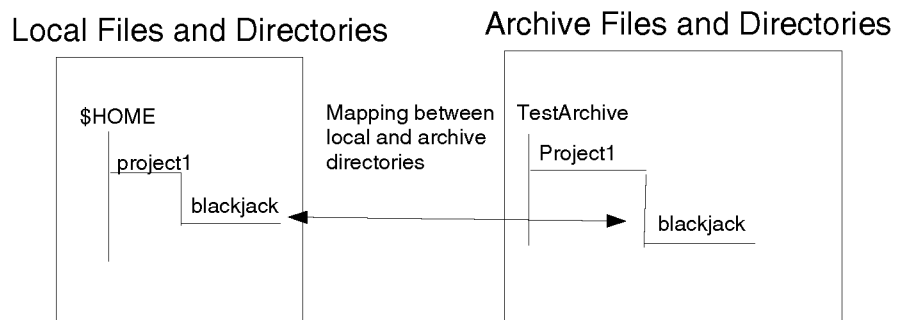
1. Navigate through the current server to the area where you want to create an archive directory.
2. Choose "Directory: Create...".
3. Enter the name of the directory you want to create in the "Create Directory" input box.
4. Select **OK**.

SoftBench CM displays the name of the archive directory you created in the main window.

## Creating a Mapping between an Archive Directory and Local Directory

SoftBench CM uses **mappings** to establish relationships between your local files and directories and the corresponding SoftBench CM server archive files and directories (see Figure 3-4). Before you can modify files in an archive directory, you need to create a mapping between that directory and a directory on your local system. The directory display area lists the mappings that exist between the current archive directory and your local system.

**Figure 3-4** SoftBench CM File Mapping



To create a mapping:

1. Choose "Actions: Create New Mapping...".  
SoftBench CM displays the "SoftCM Create Mapping" dialog box.  
The current archive directory automatically appears in the "Archive Directory" input area.
2. Enter the path of the local directory in the "Mapped to Local Directory" input area.
3. Select **OK**.  
The SoftBench CM window lists the local directory to which the archive directory is mapped and displays the contents of the current archive directory. Once a mapping exists, you can create copies of archive files and directories on your local file system, modify the local files, or create new files and directories.

## Modifying Mappings between Local and Archive Directories

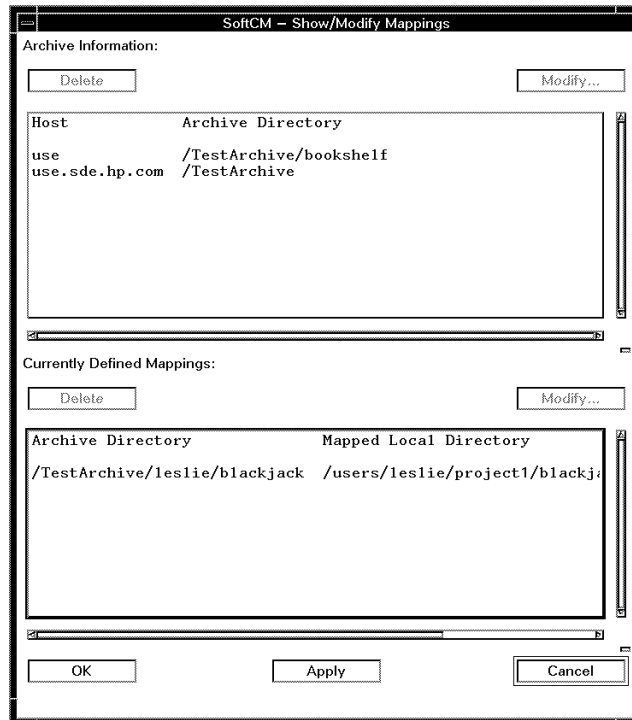
SoftBench CM lets you view and modify existing mappings between local and archive files and directories.

To view existing mappings, choose "Actions: Show/Modify Mappings...". SoftBench CM displays the "Show/Modify Mappings" dialog box (see Figure 3-5).

The upper area displays the Host and Archive Directory information for the archives that are currently defined in your local mapping file. The lower area displays the current mappings between local and archive directories and the associated **symbolic names** (see "Creating a Default Symbolic Name for Archive Files" on page 112). You can add, modify, and delete all entries to your local mapping file from this dialog box.

Changes are written to your local mapping file only if you select **OK** or **Apply**. Selecting **OK** writes the changes and exits the dialog box. Selecting **Apply** writes the changes, but leaves the dialog box open for further modifications. Selecting **Cancel** discards any changes and exits the dialog box.

**Figure 3-5** "Show / Modify Mappings" Dialog Box



To modify the server and archive locations:

1. Select an entry in the upper area of the "Show/Modify Mappings" dialog box.
2. Select **Modify...**
3. Enter the correct server and archive information in the "Modify Mapping" dialog box.
4. Select **OK**.

To modify a mapping between an archive and local directory:

1. Select the entry from the bottom window of the "Show/Modify Mappings" dialog box.
2. Select **Modify...**
3. Enter the correct "Archive Directory", "Local Directory", and

optional "Symbolic Name List" entry (see "Creating a Default Symbolic Name for Archive Files" on page 112 for more information).

4. Select **OK**.

To delete a server/archive directory location or a local mapping:

1. Select the entry.
2. Select the **Delete** associated with its window.

## Managing Archive Files and Directories

SoftBench CM offers complete configuration management functionality that lets you manage your archive files and directories from menu selections. For example, you can create initial files, check files out, delete directories, cancel file check outs, show the revision history of files, and compare file revisions.

### Creating Initial Archive Files

To use SoftBench CM, you need to create an initial archive file before checking a new file into an archive for the first time.

To create an initial archive file:

1. In the current archive list area, open the archive directory in which you want to create an initial archive file.
2. Choose "File: Create..".
3. Enter the name of your new file in the "Create file" input area. You can choose from three file types (text, binary and RAW).

*Text (the default):* Stores the file using RCS(1) with default RCS keywords.

*Binary:* Stores the file using RCS(1) without default RCS keywords.

*RAW:* Stores the file as a binary file with no revision history.

4. Select **OK**.

The name of the file you created displays in the current archive directory of SoftBench CM. If a mapped file of the same name exists on your local system, the newly created archive file is identical to the local file. Otherwise, the archive file contains no data.

### Checking Out Archive Files

When you check files out of the archive, SoftBench CM creates a copy of the archive file on the local file system. You can check out multiple files and directories or a specific **file revision**.

To check out a file from an archive:

1. Select the file(s) you want to check out.
2. Choose "Actions: Check Out to Local Directory".
3. Choose the appropriate submenu:

*Locked* checks out the most current revision of the file, gives you read-write permissions, and lets others know you are modifying the file. This helps avoid the problem of two developers working on the same file simultaneously.

*Unlocked* checks out a read-only copy of that file. If a file with the same name exists in your local directory, it is overwritten with the current archive version. If this file does not yet exist in your local directory, SoftBench CM creates a copy.

*Check Out...* lets you specify the revision, date, and/or state of a file to check out locked or unlocked.

## Cancelling Archive File Check Out

If you check out a file locked and later want to discard changes to your local copy, you can cancel the file check out. This reverts the local copy of the file back to the latest archive file revision.

To cancel archive file check out:

1. Select the local file(s).
2. Choose "Actions: Cancel Check Out (Discard Changes)".

SoftBench CM removes the lock and reverts the local file back to the latest archive file revision.

## Updating a Local Directory

SoftBench CM lets you copy archive files to your local system without checking them out. Using this option, you can obtain the latest changes from other developers before doing a local build. Updated files are read-only on your local system.

To update your local file system:

1. Select the archive files or directories you want to update.

A mapping must already exist between the archive and your local system to perform an update.

2. Choose "Actions: Update to Local Directory".
3. Choose the appropriate submenu:

*Current Directory Only* copies the latest file revisions in the selected directory to the mapped local system. SoftBench CM also copies the subdirectories, but not their content.

*Recursive (Files and Directories)* copies the latest file revisions in the selected directories and the latest file revisions from any subdirectory to the mapped local system.

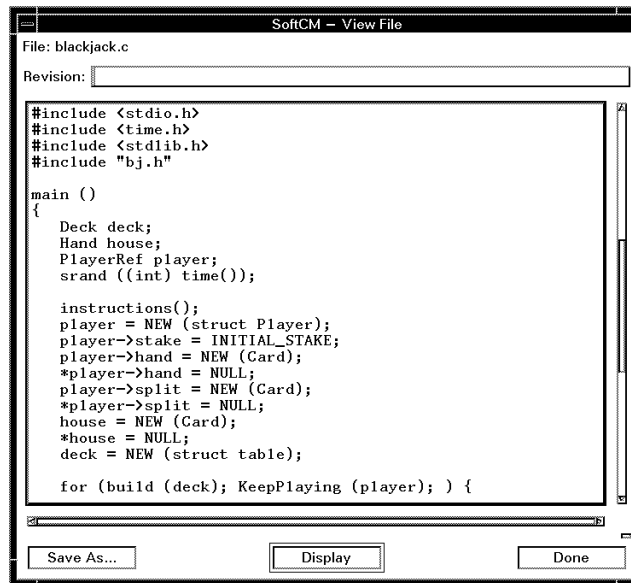
*Directories Only* creates an empty directory structure on the local system that reflects the structure of the selected archive directories.

## Viewing Contents of Archive Files

SoftBench CM lets you view the contents of archive files without checking them out to a local directory (see Figure 3-6). You may also perform cut and paste operations from archive files to local files. SoftBench CM displays archive files; use the SoftBench main window to view local files.



**Figure 3-6** Viewing an Archive File



To view a file, double-click on the desired file. SoftBench CM displays the default revision of the file in the "View File" dialog box.

To save the contents of the display to a local file:

1. Select **Save As....**
2. Enter the local file name and location where you want to save the file.

No link exists between the archive file displayed in the "View File" dialog box and the new local file.

To view a particular **file revision**:

1. Double-click on the desired file.
2. Enter the revision number in the "View File" dialog box.

If you are not sure which revision number to use:

- a. Select **Done** to close the "View File" dialog box.
- b. Choose **File: Show Revision History...**.
- c. Locate the file revision number that you want.

- d. Select **Done** to close the "Show Revision History" dialog box.
  - e. Double-click on the same file to display the "View File" dialog box.
  - f. Enter the revision number you located previously in the "Revision" input area.
3. Select **Display**.  
SoftBench CM displays the specified file revision.
  4. Select **Done**.

## Deleting Archive Files and Directories

SoftBench CM lets you delete unlocked archive files and empty directories. However, deletions are permanent. Check with your SoftBench CM administrator to set permissions for deleting archive files and directories. (For more information, see "Defining User Access to the Server" on page 342).

To delete an archive file:

1. Select the file(s) you want to delete.
2. Choose "File: Delete...".
3. Select **OK** from the "Delete" dialog box.

To delete an archive directory:

1. Select the directory or directories to delete.
2. Choose "Directory: Delete...".
3. Select **OK** from the "Delete" dialog box.

## Locking an Archive File

SoftBench CM lets you lock an archive file without checking out the file to your local directory. This prevents others from making changes to an archive file that you are editing. If the lock is broken, the system sends you an e-mail message indicating who broke the lock.

To lock an archive file:

1. Select the file(s) you want to lock.
2. Choose "File: Lock File".  
The file status changes from unlocked to locked and shows your login name.

## Breaking a Lock on an Archive File

You may need to break the lock on an archive file under certain circumstances, such as when a developer forgets to check in a source file before leaving town. Using SoftBench CM, you can determine who holds the lock and then break the lock so that another developer can work on the source file.

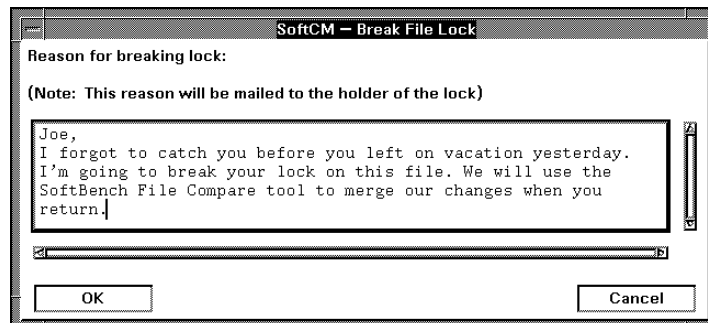
To break the lock on an archive file:

1. Select the file.
2. Choose "File: Break File Lock...".
3. Enter the reason for breaking the lock in the "Break File Lock" input box (see Figure 3-7). SoftBench CM uses this as the contents for the associated e-mail message.
4. Select **OK**.

SoftBench CM breaks the lock, makes the file available for check out, and e-mails a message to the person who locked the file.

You may or may not have permission to break file locks in certain archives. Check with your SoftBench CM administrator about permission to break locks on archive files.

**Figure 3-7** "Break File Lock" Dialog Box



## Viewing the Revision History of Archive Files

SoftBench CM lets you view the **revision history** for each archive file. This is useful when you want to determine when a particular change was made to a file.

To view the revision history of an archive file:

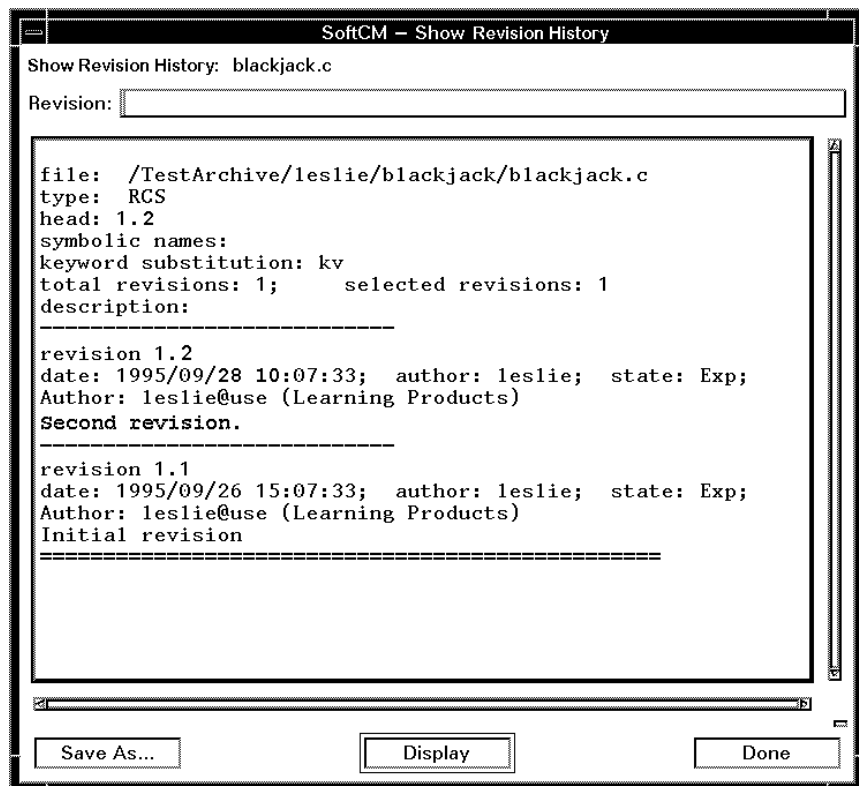
1. Select the file.
2. Choose "File: Show Revision History...".

The "Show Revision History" dialog box displays the revision history for the selected file (see Figure 3-8).

To view the revision history of a particular file revision:

1. Enter the revision number in the "Revision" input area.
2. Select **Display**.

**Figure 3-8** "Show Revision History" Dialog Box for an Archive File



## Setting Archive Display Filters

The archive display filter lets you view only those **archive** files and directories in which you are interested. From the "Display Filter" dialog box, you can customize the filter to include any combination of options:

- *Show a concise listing:* Displays only the names of the files in the current archive directory.
- *Show only locked files:* Displays only locked files in the current archive directory.
- *Files locked by an individual:* Requires the user's login name in the text entry field.
- *Include subdirectories:* Lists the files in the current archive directory and the files in the associated subdirectories.
- *Reverse the display order:* Reverses the order in which the files and directories are sorted and displayed.
- *Sort by time:* Displays files by the last date and time they were modified instead of alphabetically (default).
- *Show numeric mode and time:* Displays the time and permissions information in numeric format.

To set the display filter:

1. Choose "Options: Display Filter...".
2. In the "Display Filter" dialog box, select the desired option(s).
3. Select **OK**.

When you change settings, the new settings are in effect for the current session. To save settings for other sessions, select the menu item "Options: Save All Settings".

## Managing Local Files

Your local file system contains one revision of each file. Changes to **archive** files, such as editing, are done in the local file system through the SoftBench main window. Working on your local system lets you make and test changes to local files before introducing those changes back into the archive file system.

### Modifying Local Source Files

To modify a local file from the SoftBench main window:

1. Select the file you want to modify.
2. Choose "File: Configuration Management: → Check Out Locked".

This lets others know that you are modifying the file and ensures you have the most recent revision of a file with read-write permissions.

3. Choose "File: Open..." or click on the editor icon.

The configured SoftBench editor starts.

4. Make your modifications and save the results to the local file system.
5. Check in the local file to update the archive version.

### Checking In Modified Files

Checking a file in changes your permissions on the local file to read-only, updates the archive file, and makes the file available to others to modify. You can operate on multiple files and directories and add comments during file check in. File comments are kept with the **revision history** of a file. You can check a file in and retain the lock to allow others access to your recent changes while you continue modifying the file.

To check in a file to the archive:

1. Select the file(s) you want to check in.
2. Choose "Actions: Check In from Local Directory...".
3. In the "Comment" section of the "Check In" dialog box, enter comments.

4. Include optional attributes:

**Revision:** Defines an instance of a file in a series of changes. You can leave this field blank because SoftBench CM assigns the file a default number that denotes the latest version of a file.

**State:** Gives a revision of a file a value such as "draft," "prototype," "final," or "release." You can leave this field blank because SoftBench CM uses "Exp" (experimental) as the default value.

5. Select **OK**.

## Creating a Default Symbolic Name for Archive Files

The **mapping** between a local directory and an **archive** specifies which **revision** of files you use. The default setting retrieves the latest revision of each file from the archive during file check out. The default also sets each file as the new, latest revision during check in. However, you can edit these settings to work with other file revisions by specifying a symbolic name list for a mapping.

A **symbolic name** is a special user-defined string (containing no white space characters) assigned to a particular file revision. A **symbolic name list** is a comma separated list of symbolic names with a special default revision specifier at the end. The special default revision specifier can be a symbolic name or it can be either a dash (-) or an asterisk (\*), optionally followed by a number. Below is an example of valid symbolic name list:

```
Revision1_1,*DemoVersion,Revision1_1,-
```

### Understanding Symbolic Names

After defining the symbolic name list, the active revision within the archive files is determined by the revision associated with that symbolic name. The symbolic name list also determines how files are created and checked back into the archive.

#### *During Check Out*

- If the first entry in the symbolic name list exists for a file, SoftBench CM returns that revision of the file.
- If the first entry does not exist, and the next entry exists for the file, SoftBench CM returns that revision of the file.
- If SoftBench CM reaches the end of the symbolic name list before it checks out the file and you did not include a special default entry specifier, an error occurs. If you included an (\*) as the special default entry specifier, SoftBench CM checks out the latest file revision (default). If you included a (-) as the special default entry specifier, SoftBench CM does not check out the file, and no error occurs.



### *During Check In*

- SoftBench CM tags the checked in file with the first symbolic name on the list. Two revisions of the same file cannot have the same symbolic name. If the system already includes a revision tagged with the specified symbolic name, the tag no longer applies to that revision.
- SoftBench CM creates a branch, if necessary, for the new tagged revision.

### *During File Creation*

- SoftBench CM creates the file and tags the initial version with the first symbolic name on the list.
- If the special default entry contains a number, the initial version includes that revision number.

## **Defining Symbolic Names**

You can define a symbolic name list when creating a mapping or editing an existing mapping.

To define a symbolic name list for an existing mapping:

1. Choose **Actions: Show/Modify Mappings...**.
2. In the "Currently Defined Mappings" section of the dialog box, select the mapped archive for which you want to define a symbolic name.
3. Select **Modify...**
4. In the "Modify Mapping" dialog box, enter the a "Symbolic Name List" entry.
5. Select **OK**.

## **Symbolic Name Example**

The following example shows the revision history of two files (see Table 3-1) and the results of the configuration management actions after setting the symbolic name list to "DemoVersion,Revision1\_1,\*3" (see Table 3-2).

**Table 3-1 Symbolic Names and Revision Numbers**

File X		File Y	
Revision Number	Symbolic Name	Revision number	Symbolic Name
1.5		1.5	— Revision1_1
1.4		1.4	
1.3	— Revision1_1	1.3	
1.2	— Demo Version	1.2	
1.1		1.1	

**Table 3-2 Actions and Results After Setting Symbolic Name List**

Action	— >	Result
Check out File X	— >	Checks out revision 1.2.
Check out File Y	— >	Checks out revision 1.5.
Check in File X	— >	Creates a branch and a new revision numbered 1.2.1.1 and DemoVersion now is set to 1.2.1.1.
Check in File Y	— >	Creates revision 1.6 and assigns DemoVersion to that revision.
Create File Z	— >	Creates a revision numbered 3.0 and labeled with DemoVersion.

---

## Using the SoftBench CM Command Line Interface

SoftBench CM lets you execute commands from the command line. For more information about the command line, review the following man pages located in `/opt/softbench/man` directory.

**Table 3-3**      **SoftBench CM Command Line Man Pages**

<b>Man Page</b>	<b>Task Description</b>
<code>cmdate(1)</code>	Sets dates across a network of machines using SoftBench CM.
<code>fci(1)</code>	Check in local files to archive, create new archive files.
<code>fco(1)</code>	Check out archive files or directories.
<code>fdiff(1)</code>	Print differences between two versions of archive files.
<code>fhist(1)</code>	Display the revision history of an archive file.
<code>fls(1)</code>	List the contents of an archive directory.
<code>fmerge(1)</code>	Merge revisions of an archive file into a local file.
<code>fupdate(1)</code>	Update local files from SoftBench CM archives.
<code>futil(1)</code>	Perform miscellaneous SoftBench CM tasks.
<code>softCM(1)</code>	Describe the SoftBench CM GUI.

Using SoftBench Configuration Manager  
Using the SoftBench CM Command Line Interface

# 4 Using SoftBench Editors

SoftBench provides SoftBench XEmacs and SoftBench vi Editor for text editing. SoftBench comes preconfigured with SoftBench XEmacs as the default editor, with SoftBench vi Editor as an alternative. SoftBench Program Editor is now contributed software that you must custom configure if you choose to use it. Once your editor preference is set, actions which invoke an editor use the configured SoftBench editor. SoftBench also provides SoftBench Class Graph/Editor for graphical editing. See Chapter 5, “Using SoftBench Class Graph/Editor,” on page 133 for information.

## Using Editors with Projects

When the editor is started from SoftBench, it understands your project information. SoftBench XEmacs and SoftBench vi Editor can save a file into the current project. The editor displays the current project on the title bar. You can also save a file outside of the current project.

- **Save**—Save the file as it currently appears in the window. If the file is "Untitled", you are prompted for a file name. If the editor is project aware (editor was opened from the SoftBench main window or from a SoftBench tool), the editor saves the file into the current project.
- **Save As...**—Save the file as it currently appears in the window using a name you provide. If the editor is project aware, the editor saves the newly named file into the current project.
- **Save Out Of Project**—Save the file as it currently appears in the window. If the file is named "Untitled", you are prompted for a file name. SoftBench does not automatically add the file to the project.
- **Save Out Of Project As...**—Save the file as it currently appears in the window under the name you provide. SoftBench does not automatically add the file to the project.

Note that if your editor is project aware, any file you "Save" or "Save As" will be added to your project file set, even if you just happen to make a change to your personal files. Use "Save Out Of Project" for files you do not want added to the project named in the editor's titlebar.

If you rename a file to a name that indicates a different file type, SoftBench Program Editor and SoftBench XEmacs switch mode to the new file type.

---

## Configuring an Editor

Select your editor by choosing "Options: Tool Preferences...." Once your editor preference is set, actions which invoke an editor use the configured SoftBench editor. Refer to SoftBench Online Help for detailed information on each editor.

SoftBench XEmacs is the default editor. If you are a new SoftBench XEmacs user, SoftBench automatically sets up a default XEmacs configuration for you. If you are already using GNU XEmacs and you want to maintain your user options, SoftBench automatically loads the preferences specified in your `$HOME/.xemacs` file. You may discover some preference differences you want to change by editing the `$HOME/.softxemacs-options` file.

### Configuring SoftBench vi Editor

To change your default editor to SoftBench vi:

1. From the SoftBench main window, choose "Options: Tool Preferences..."
2. From the "Tool Preferences" dialog, choose "Editor". The available editors are displayed.
3. Select "Softvi" from the scrollbar list, then select **OK**.

### Configuring SoftBench Program Editor

SoftBench Program Editor is only available for backward compatibility and will not be available in future releases. It has not been enhanced to work with the SoftBench project model. Files that you save are not automatically added to your project. SoftBench Program Editor is not preconfigured and requires special steps to activate it:

1. With root permissions, edit the file  
`/opt/softbench/config/toolbar/prefsConfig.`
2. Search the file for the line containing:

```
`${SOFTBENCH:-/opt/softbench}/contrib/bin/softeditsrv -scope net
```

3. Remove the "#" symbol from the front of the line.

## Using SoftBench Editors

### Configuring an Editor

4. Save the changes to the file.
5. From the SoftBench main window, choose "Options: Tool Preferences".
6. From the dialog box, choose "Editor". The available editors are displayed.
7. Choose "SoftEdit" from the "Available Tools" list, then select **OK**.
8. Select **OK**.

If you do not have root permissions, copy the line found in `/opt/softbench/config/toolbar/prefsConfig` into `$HOME/.softbench/bmsinit`. Be sure it is all on one line. For more information, refer to the *bmsinit(5)* man page. When selecting your tool preferences, use the "Default" selection to enable your local `bmsinit` file to override the system settings.



## Starting the Configured SoftBench Editor

To edit a project file:

- Double-click on it in the Files view
- Select the file in the Files view and select the "Editor" icon.
- Select the file and choose "File: Open...".
- Alternatively, you can double click on the output browsers from other SoftBench tools to start your editor preloaded with the relevant file.

To create a new file:

1. Choose "File: New...",

*or*

Make sure that nothing is selected in the project browser.

2. Select the "Editor" icon.

## Editing with SoftBench XEmacs Editor

SoftBench XEmacs is an advanced GUI editor with features that go beyond a typical text editor or `vi`. SoftBench XEmacs is built on GNU XEmacs. In addition to having pulldown menus and a toolbar, SoftBench XEmacs is self-documenting, customizable, and extensible.

Self-documenting means that any time you type `Control-H` to find out what your options are, you can also type a special character string to find out what a command does. In addition, you can find commands relevant to a particular topic. Customizable means you can change the definitions of SoftBench XEmacs commands to suit your individual needs.

Extensible means you can go beyond simple customization and write entirely new commands using the Lisp programming language.

Before starting SoftBench XEmacs, you should be familiar with how SoftBench XEmacs uses buffers, files, windows, and frames.

- **Buffers**—A buffer holds characters in a region of memory. As the basic editing unit, one buffer corresponds to one piece of text being edited. You can have multiple buffers but you can edit only one buffer at any one time.
- **Files**—SoftBench XEmacs edits a file by reading it into a buffer, editing that buffer, and writing the buffer contents back to the file. To save your work permanently you must write it to a file.
- **Windows**—You can open multiple windows with multiple buffers and edit them by selecting the corresponding buffer. When you start SoftBench XEmacs, it automatically opens a window for you. From there you can open multiple files in separate buffers. Each buffer can be displayed in a separate window, or displayed on the main window by using the buffer menu.
- **Frames**—A frame, in the terminology of GNU XEmacs is an X-window complete with menus, toolbars, a message area, and one or more windows.

SoftBench XEmacs documentation consists primarily of online help. You can obtain additional GNU XEmacs user and reference material on the Web at:

<http://www.xemacs.org>

## Using SoftBench XEmacs

SoftBench XEmacs offers several robust user interface features (See Figure 4-1):

- A menu bar provides access to pulldown menu functions.
- A tool bar provides quick and easy access to selected pulldown menu features.
- An edit area provides a multi-line editing area that responds to keyboard commands, as well as the pulldown menu items.
- An input status area at the bottom of the window allows you to input file and command information.
- The copy and paste feature allows you to copy and paste between windows.
- You can access other SoftBench tools from SoftBench XEmacs, and other tools can access SoftBench XEmacs. See “Calling Other SoftBench Tools from the Editors” on page 130 for further information.

**Figure 4-1**      **SoftBench XEmacs Main Window**



## Accessing Help

Once you have started SoftBench XEmacs, you can access online help from the "Help" menu. The "Help" menu also provides access to online manuals via the "Help: XEmacs Help → UNIX Manual..." command.

## Editing Multiple Files

SoftBench XEmacs allows you to sequentially edit multiple files in the same window or in multiple windows. For example, suppose you are

editing a file and wish to make a quick change to another file and then return to editing the first file.

1. Within SoftBench XEmacs you can choose "File: Open..." and specify another file. If you use "Save" or "Save As..." the file will be added to the current project file set.
2. SoftBench XEmacs loads the new file, where you can edit the file's contents as desired.
3. You can switch back to editing the previous file by choosing "Windows: List All Windows".
4. Double click on the name of the file you want to edit, and SoftBench XEmacs reloads the file into the editing area.
5. To open a new window, select "Window: New Frame".

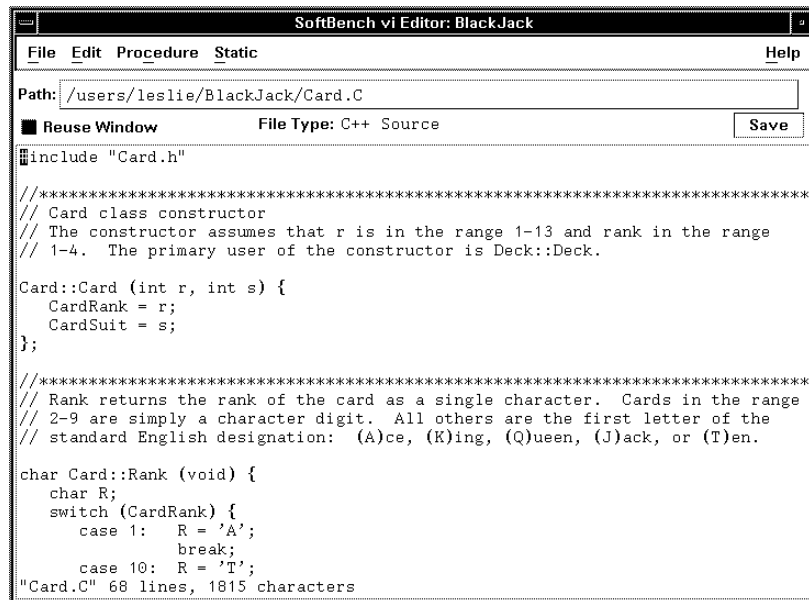
## Editing with SoftBench vi Editor

SoftBench vi Editor is an easy-to-use encapsulation of the UNIX vi editor. This tool gives you all of the features of vi plus the ability to execute SoftBench commands and tools from the pulldown menus. See “Calling Other SoftBench Tools from the Editors” on page 130 for integration information.

Consult your system's vi documentation to learn about vi. For additional information about SoftBench vi Editor, see "Help: Show Man Page" from within the tool. There are also several user guides on the Web. Simply search for "vi editor".

Figure 4-2 displays the SoftBench vi Editor Window.

**Figure 4-2** SoftBench vi Editor Window



```
SoftBench vi Editor: BlackJack
File Edit Procedure Static Help
Path: /users/leslie/BlackJack/Card.C
Reuse Window File Type: C++ Source Save
#include "Card.h"
//*****
// Card class constructor
// The constructor assumes that r is in the range 1-13 and rank in the range
// 1-4. The primary user of the constructor is Deck::Deck.
Card::Card (int r, int s) {
    CardRank = r;
    CardSuit = s;
};
//*****
// Rank returns the rank of the card as a single character. Cards in the range
// 2-9 are simply a character digit. All others are the first letter of the
// standard English designation: (A)ce, (K)ing, (Q)ueen, (J)ack, or (T)en.
char Card::Rank (void) {
    char R;
    switch (CardRank) {
        case 1: R = 'A';
                break;
        case 10: R = 'T';
    }
}
"Card.C" 68 lines, 1815 characters
```

The vi command syntax operates only while you are in SoftBench vi Editor. Other SoftBench Edit Areas such as dialog boxes implement SoftBench XEmacs behavior, even when you have configured your system to use SoftBench vi Editor.

With SoftBench vi Editor, you have all of the standard vi capabilities of

using alphanumeric and cursor-control keys for input. In addition, you can move the cursor using the mouse, and you have the ability to access SoftBench operations defined in the window by using the mouse or the Alt key in conjunction with other keys.

## Using the Mouse Pointer Versus the Text Cursor

Note the differences between "mouse pointer" and "text cursor":

Mouse pointer	moves as you move the mouse. SoftBench vi Editor uses an arrow for the mouse pointer in the text area (the same as when the pointer is over the menu bar).
Text cursor	is a solid box that shows vi's current location within a text area. When you insert text, the new characters appear at the cursor location and subsequent characters are moved to the right. SoftBench vi Editor positions the cursor in the upper left corner of a text area when you first edit a file. You can move this text cursor by moving the mouse pointer to the desired new location and then clicking the left mouse button.

## Editing Multiple Files

When using SoftBench vi Editor, you can sequentially edit several files in one window. For example, suppose you are editing a file and you want to edit another file.

1. Choose "File: Open..." or "File: New...", and then specify another file. If you use "Save" or "Save As..." the file will be added to the current project definition.
2. If the current file has not been saved (it has been modified since last being saved), then there are two possible SoftBench vi Editor

behaviors:

**Table 4-1**

<b>If the vi autowrite option</b>	<b>SoftBench vi Editor's behavior</b>
is set	automatically saves the file before loading the new file.
is not set	you must manually save the file before loading the new file.

To set the autowrite option, type `":set autowrite"` from within SoftBench vi Editor or add it to your `$HOME/.exrc` file.

3. SoftBench vi Editor loads the new file and you can modify the file's contents as desired.
4. If you want to "switch" back to the previous file, you can view the Editor Index to see a list of files that you have edited. Choose "File: Editor Index" to display this index. Double-click the mouse on the name of the file which you want to edit, and SoftBench vi Editor loads it into its editing area. (You can also use vi's `":edit #"` command to edit the previously edited file.)

## Reusing the Edit Window

SoftBench vi Editor's **■ Reuse Window** toggle button allows you to control whether or not SoftBench vi Editor generates a new window for an edit request:

**Table 4-2**

<b>Status of "■ Reuse Window" Toggle Button</b>	<b>SoftBench vi Editor Behavior</b>
Set on at least one SoftBench vi Editor window	SoftBench vi Editor reuses a window whose toggle button is selected for subsequent edit requests
Not set on any window	SoftBench vi Editor generates a new window.



For example, suppose that you are editing a file called `FirstFile` in the only existing SoftBench vi Editor window, and this window's "■ Reuse Window" toggle button is not selected. Editing `SecondFile` by choosing "File: Open" generates a new SoftBench vi Editor window with `SecondFile` loaded into it (because the first window is not to be reused).

On the other hand, if you set the "■ Reuse Window" toggle button in the SoftBench vi Editor window editing `FirstFile`, then `SecondFile` loads into the same window.

If you have a mixture of SoftBench vi Editor windows with and without the "■ Reuse Window" toggle button selected, then requesting that another file be edited causes that file to be loaded into a window where the "■ Reuse Window" toggle button is selected. If you select "■ Reuse Window" toggle buttons in several windows, then SoftBench vi Editor cycles through the windows as you make additional edit requests.

## Selecting, Copying, and Pasting Text

You can take advantage of the copy and paste operations between windows. For example:

1. In the SoftBench vi Editor window, select some text by dragging the mouse pointer.
2. Press `Control-Insert`, which copies the selected text into the Clipboard.
3. Move the cursor to the point where you want to insert text. Enter SoftBench vi Editor insert mode by typing `i`, then paste the text from the Clipboard into the file by pressing `Shift-Insert`. If the destination window is not in `vi`'s insert mode, SoftBench vi Editor interprets the characters as commands until one of them puts `vi` into insert mode.

## Calling Other SoftBench Tools from the Editors

One of the most useful benefits of having the `vi` and `XEmacs` editors encapsulated and integrated into SoftBench is that you can call other tools from within the editors, and you can call the editors from other tools.

The default editing mode is "C" code, and changes based on the filename's suffix. The suffixes `.C`, `.cxx`, `.cpp`, `.Cxx`, `.cc` and `.H` cause the editors to switch to C++ mode.

### Compiling a Program File

Once you finish editing and saving the changes to a program file (save the file by choosing **File: Save**), you can compile the file while in an editor by choosing **File: Compile File**.

See "Compiling Instead of Building" on page 73 for information comparing compiling to building.

### Building a Project

Once you finish editing and saving the changes to your program file, (save the file by choosing **File: Save**), you can build the current project by choosing **File: Build Project**.

See "Compiling Instead of Building" on page 73 for information comparing compiling to building.

### Accessing SoftBench Static Analyzer from the Editor

You can initiate static analysis directly from the editor. For example, you can generate a list of all calls to a particular variable by selecting a token (dragging with the left mouse button or double-clicking) and choosing **Static: References**. If no token is currently selected, then SoftBench Static Analyzer uses the first token following the text cursor in the **Static: References** operation.

## Setting Breakpoints in a Program File

You can also set SoftBench Debugger breakpoints in a program by choosing "File: Set Breakpoint". SoftBench Debugger sets the breakpoint on the line where the cursor is located.

## Using Configuration Management

SoftBench provides access to frequent Configuration Management commands for operations such as checking in a file or checking out the latest copy of a file. The "File: Configuration Management" submenu contains the following commands:

- Check Out Locked...
- Check In...
- Check Out Unlocked
- Create Initial Version
- Cancel Check Out
- show Revision History...
- Compare Revisions...
- Start Configuration Management Tool...

The exact behavior of these commands depends on which configuration management tool you use. SoftBench ships with SoftBench CM. Other vendors also provide configuration management solutions that integrate with SoftBench.

## For More Information

- On specifying an editor other than SoftBench XEmacs and SoftBench vi Editor, see “Registering New Tools with SoftBench” on page 80 and the *bmsinit(5)* man page.
- On SoftBench vi Editor and its resources, see the *vi(1)* manual page.
- On SoftBench XEmacs and its resources, see the *xemacs(1)* manual page.

# 5

## Using SoftBench Class Graph/Editor

SoftBench Class Graph/Editor allows you to edit the class constructs in your C++ program using a visual, graphical interface. You can create and modify class hierarchies and edit class components with a few clicks of the mouse.

SoftBench Class Graph/Editor is incorporated into the SoftBench Static Analyzer graphical interface. You need some familiarity with SoftBench Static Analyzer to use the SoftBench Class Graph/Editor features. See Chapter 10, “Using Static Graphs,” on page 283 for more information on Static Graphs.

## Editing C++ Structures with SoftBench Class Graph/Editor

The class hierarchies and other data structures in C++ can be understood much more easily when they are presented in a graphical format. Static Graphs provide an excellent tool for viewing these structures.

The SoftBench Class Graph/Editor in SoftBench Static Analyzer also allows you to edit C++ structures in the graphical display. While viewing the graphical representation of C++ structures, you can add, modify, or delete classes, class components, and component definitions.

SoftBench Class Graph/Editor helps you to become much more productive in your C++ programming. You can understand, create, and modify C++ programs much more quickly using the high-level conceptual displays. You can concentrate on the semantics of your program, instead of worrying about the syntax and details of C++ class definition. SoftBench Class Graph/Editor handles the details for you.

SoftBench Class Graph/Editor requires the Static Analysis database used by SoftBench Static Analyzer. If you try to use SoftBench Class Graph/Editor with a directory or project that does not contain a Static Analysis database, SoftBench Static Analyzer may create one for you using your new and existing C++ code. Building your code with the ■ `Static` option produces a more reliable database.

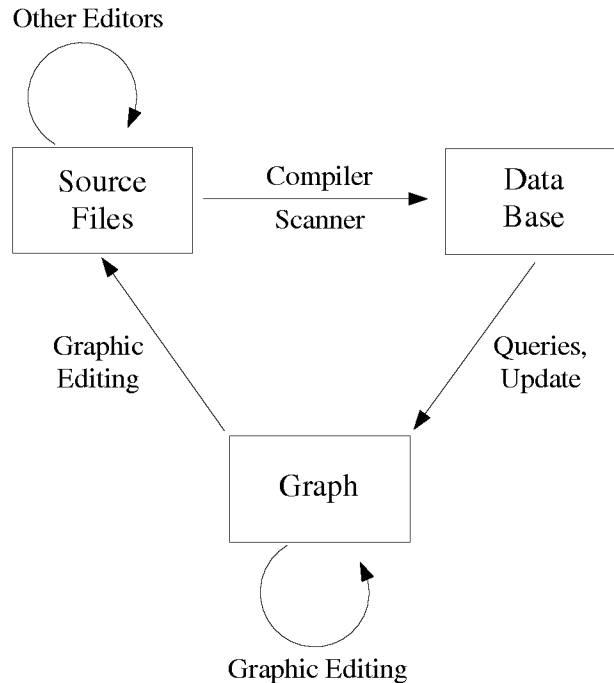
### Understanding the SoftBench Class Graph/Editor Use Model

When you edit using SoftBench Class Graph/Editor, several different files and programs are involved including your source files, the static analysis database, the SoftBench Class Graph/Editor program, and your compiler. See Figure 5-1. In this diagram, the rectangles represent items on your system. The arcs represent actions by you or by various programs that modify those files.

In order to use SoftBench Class Graph/Editor, the graph and the Static Analysis database must accurately reflect the source files. However, SoftBench Class Graph/Editor and other editors may change the source files, and any changes to the sources must be reflected in the database.

Any change to the sources or database must be reflected in the graph.

**Figure 5-1** **SoftBench Class Graph/Editor Use Model**



Each time you make an edit, SoftBench Class Graph/Editor updates the graph, changes the source files, and invokes the source scanner to update the database. The scanning process takes time, and it's possible for the graph to become unsynchronized if you make edits before the scan completes.

Because SoftBench Class Graph/Editor does not have access to all the information that the compiler has, it's also possible for the graph to become unsynchronized with the source files. Compiler errors and source changes in other editors can cause the database and the graph to become out-of-date.

If you notice that the graph appears to be unsynchronized with your sources, you may need to re-analyze the database (choose "File: Analyze File Set" in the main SoftBench Static Analyzer window) and select the **Update Graph** button to display the current state of the

database.

## Using SoftBench Class Graph/Editor with Other SoftBench Editors

As you interact with SoftBench Class Graph/Editor, it immediately writes each change you make to the appropriate source files. SoftBench Class Graph/Editor maintains an "undo" list so that you can reverse changes even though they have been written to disk.

If you view the sources in any SoftBench tool, your source view is updated each time you make a change in SoftBench Class Graph/Editor. If you change the file in one of the SoftBench text editors, SoftBench Class Graph/Editor detects the change and discards its "undo" list. This prevents SoftBench Class Graph/Editor from damaging the source changes you made in another editor. Select the **Update Graph** button to display the new changes in SoftBench Class Graph/Editor.

Because of this close interaction with the SoftBench editors, it can be very productive to use SoftBench Class Graph/Editor in conjunction with an editor. You are not forced to use only SoftBench Class Graph/Editor to make C++ class edits; you can use your editor to make arbitrary text changes (such as adding comments or entering non-class code) in the natural flow of your work.

## Synchronizing Editor Views and the Static Database

SoftBench Class Graph/Editor uses the Static database for its view onto your C++ source. If no Static database exists for the current project (or in the current directory for standalone mode), SoftBench Class Graph/Editor requests one to be generated after you make your first edit.

When you edit using SoftBench Class Graph/Editor, you make changes to the C++ source files. SoftBench Class Graph/Editor then triggers an update of the Static database to reflect the new source code.

If you make changes using other editors, you may want to force an update of the Static database to ensure SoftBench Class Graph/Editor displays the current state of your code. To do this, choose **File: Analyze File Set** from the main SoftBench Static Analyzer window, and select the **Update Graph** button in SoftBench Class Graph/Editor when the file set is updated.

If you do not force an update, SoftBench Class Graph/Editor still checks



all files before making an edit. If any files have been changed since the last database update, SoftBench Class Graph/Editor updates the database before proceeding with the edit.

## Using SoftBench Class Graph/Editor in Your Work

SoftBench Class Graph/Editor has several basic operations that you can apply to different use models. You can:

- View and modify your existing C++ class hierarchy. The SoftBench Class Graph/Editor graph in SoftBench Static Analyzer gives you many ways to examine the structure of your C++ program.
- Create new C++ components (classes, instances, member functions, data members, and inheritance relationships).
- Delete existing C++ components. You can delete any C++ component displayed in the SoftBench Class Graph/Editor, even if you did not create it using SoftBench Class Graph/Editor.
- Modify existing C++ components. You can change the name, type, parameters, accessibility (public, private, protected) and other declarations (virtual, static, const) for any component. You can change the definition of any component that supports definitions. If you change a class or member name, SoftBench Class Graph/Editor changes all references to that identifier.

### Viewing the Existing Class Hierarchy

SoftBench Class Graph/Editor is implemented as one of the Static Graphs. You can view your complete class hierarchy by entering SoftBench Class Graph/Editor in SoftBench Static Analyzer and choosing "Graph: Class Graph/Editor → All Classes".

You can view any subset of your class hierarchy by entering a class of interest in the "Symbol ()" input box and selecting **Display**. SoftBench Class Graph/Editor displays the requested class. By default, it also displays the immediate base and derived classes of the class. (You can change these defaults by choosing "Options: Queries On New Nodes...".) You can add other classes as needed, or select classes and press the right mouse button to display other information.

Select a class and hold down the right mouse button to display a popup menu of choices. Choose "Show →" to display various information about the class, including its data members and member functions.

## Creating New Components

You can create new classes and class members by choosing one of the selections under the "Edit" menu. The actions you take depend on what you have chosen. Any files created in the process of adding components become part of your current project.

"Edit: Create Class..." prompts you for the name of the new class. By default, SoftBench Class Graph/Editor places the new class declaration in a new header file named *classname.H*. You may specify a different header file. SoftBench Class Graph/Editor displays the new class near the middle of the graph. You may move it to a more convenient location by dragging it with the middle mouse button.

All other choices under "Edit" act on a specified class. You must select the class you want to edit before choosing any of these choices. (The popup menu also provides these choices when a class is selected.) "Edit: Create Member Function..." and "Edit: Create Data Member..." add members to the class. SoftBench Class Graph/Editor prompts you for information about the new member.

"Edit: Create Member Function..." allows you to specify all properties of the new function. You can specify the signature (return type, name, and parameter list) and other properties (public, private, virtual, static, const). You can also specify a definition for the function, or only a declaration. See SoftBench Online Help for a more complete explanation.

You can add an inheritance relationship to the selected class by choosing "Edit: Add Base Class". The mouse pointer changes to a small "class" icon with an arrow pointing *away* from it. Select the desired base class.

SoftBench Class Graph/Editor also provides convenient shortcuts for creating classes and inheritance relationships:

- To create a class, hold down the Ctrl key and click the middle mouse button where you want the class created.
- To create an inheritance relationship, hold down the Ctrl key and the middle mouse button, and drag the mouse pointer from the base class to the derived class.

## Modifying Existing Components

Select the component you want to modify. For member functions and data members you must first display the contents of the desired class. Select the class, press the right mouse button, and select "Show →

Member Functions" or "Show → Data Members" from the popup menu.

Once you have selected the desired component, choose "Edit: Modify...", or press the right mouse button and select "Modify...". A dialog box similar to the corresponding "create" dialog box appears. Modify the desired information and select **OK**.

You can also edit the source for the object by double-clicking on the object, or by pressing the right mouse button and selecting "Edit Source". Remember that you must save any changes you make in the editor, and they clear the SoftBench Class Graph/Editor "undo" history.

You can modify the properties of an inheritance relationship (such as public or private) by selecting the inheritance arc, but you cannot change the base or derived class. To do this, you must delete the arc and create a new inheritance relationship.

You can change the name of a class or member. SoftBench Class Graph/Editor updates all references to the class or member to use the new name, even if the name is referred to in many files. SoftBench Class Graph/Editor changes only the proper references to the name, not references to other variables with the same identifier.

"Edit: Undo", if active, "undoes" the most recent edit from the edit history stack. You can use this repeatedly to "walk" back through your history of editing changes.

## Deleting Existing Components

Select the component you want to delete. For member functions and data members you must first display the contents of the desired class. Select the base class, press the right mouse button, and select "Show → Member Functions" or "Show → Data Members" from the popup menu.

Once you have selected the object to delete, press the right mouse button and select "Delete" from the popup menu. You can select and delete multiple objects of the same type simultaneously.

## Sample Use Models

Using SoftBench Class Graph/Editor is a simple matter of applying the basic operations as needed in your specific C++ programming situation. Here are some examples of common scenarios.

### Creating a New Program

SoftBench Class Graph/Editor is an excellent tool to help you "rough out" the structure of your program. You can create classes to "sketch out" the class hierarchy to implement your program, and return to fill in implementation details when you are ready. You can move between SoftBench Class Graph/Editor and the SoftBench editor of your choice to fill in other (non-class) code as you go along.

In order to use this:

1. Create your project.
2. Define your source files, include files, and target files.
3. Link your source files to your target files and define your build model.
4. Create your C++ source file with the `include` statement.
5. Use SoftBench Class Graph/Editor to create your classes and save them into your include file.
6. Use Builder to keep your Static database synchronized as you work.

Alternatively, if you started SoftBench Static Analyzer in standalone mode, you can update the Static database immediately.

### Modifying an Existing Program

You can use SoftBench Class Graph/Editor to add classes or class members to an existing program, or to change or remove existing classes or class members. You can also use SoftBench Class Graph/Editor to restructure the class hierarchy of a program. For example, you could add new functionality to an existing class by adding a new base class.

### Working with Class Templates

You cannot use SoftBench Class Graph/Editor to *create* a class template

or function template member (parameterized function). (You may want to use SoftBench Class Graph/Editor to create a class with the desired components, and then convert that class to a template using another editor.) However, you can perform any other operation with class templates, such as adding or removing member functions and data members, and deriving classes from the template.

When you derive a class from a class template, SoftBench Class Graph/Editor prompts you for the template parameters. You cannot derive a class directly from a template; you must derive from an *instance* of the template. When you enter the parameters, SoftBench Class Graph/Editor references a template instance of the appropriate type (creating one if necessary), and derives the new class from the template instance.

For example, suppose you have a template A that accepts one parameter. If you specify an inheritance relationship between A and a class B, SoftBench Class Graph/Editor prompts you to enter the template parameter. If you enter "int", SoftBench Class Graph/Editor creates a class (template instance) A<int>, and derives B from A<int>. If you want to derive other classes from an int instance of A, specify A<int> as the base class.

## Using Configuration Management

SoftBench Class Graph/Editor attempts to edit all files containing updated class information. (A single change may affect many files. For example, if you change the name of a class, every reference to that class must be updated.) SoftBench Class Graph/Editor assumes any read-only files are locked by your configuration management system. SoftBench Class Graph/Editor displays a dialog box to give you the option of checking out the files. The dialog box also allows you to simply `chmod` the files to force them to be writeable. You should not do this if the files are actually locked by the CM system, since the changes you make may be lost when you check out the file. Only force the files to be writeable if no CM system manages them.

---

## If Something Goes Wrong

Table 5-1

Condition	Explanation
Unable to update the data base. No files modified.	SoftBench Class Graph/Editor was unable to update the Static database to reflect your changes. Make sure the data base is writeable. You may want to regenerate the database by choosing "File: Analyze File Set" or by rebuilding your program using Project.
A recent (external?) edit is causing compile errors. Graphic editing may be impaired until this is fixed.	A recent edit, either from SoftBench Class Graph/Editor or from another editor, generated a compile error. Fix the compile errors and regenerate the Static data base.
Database Synchronization Error	This error can be caused by the database being updated in the middle of an edit. It may also be generated if the graph is out-of-date with respect to the database, or by certain unusual programming styles. Select the <b>Update Graph</b> button and try the edit again.

Using SoftBench Class Graph/Editor  
**If Something Goes Wrong**



# 6 Using SoftBench CodeAdvisor

SoftBench CodeAdvisor provides advanced code checking for C and C++. SoftBench CodeAdvisor (available in C++ SoftBench) can help you find and fix a variety of subtle and dangerous errors that most C and C++ compilers can not detect. SoftBench CodeAdvisor does not duplicate the error-checking functions of the compilers. The compilers check for many syntactic and some logic problems, but are limited to checking for fairly simple, localized problems.

SoftBench CodeAdvisor can check for much more complex and far-reaching problems, including problems that cross compilation units (two separately compiled programs linked together). Using SoftBench CodeAdvisor you can find problems such as potential heap corruption, dangling pointers, ambiguous initializations, and dependencies on system-specific compiler/linker behavior.

SoftBench CodeAdvisor also helps you to make your programs faster, more reliable, and more portable by alerting you to actual and potential code problems. SoftBench CodeAdvisor uses specific rules to identify potential problems. Each rule is a set of instructions that queries the SoftBench Static database for the information of interest, and then performs a test for the presence of an error condition. When SoftBench CodeAdvisor detects an error (rule violation), it displays the violation's location in an output browser. The browser helps you navigate to the problem and use a preconfigured editor to correct the error.

SoftBench CodeAdvisor helps especially with C++ programs since C++ performs many activities "behind the scenes." For example, it automatically and invisibly calls class constructors when you create a new instance of a class. C++ calls destructors when an instance goes out of scope and is no longer valid. This transparent functionality allows C++ to perform many of the operations that make it so useful.

However, because C++ does so much for you, you may not be completely aware of some of the things the software does. As a result, you may unintentionally write your program in a way that could cause problems.

For example, C++ allows you to define an assignment operator, `operator=`, on your classes. Your `operator=` should free any memory allocated by the class before copying the new class value into it. However, if the assignment actually assigns the class to itself (`classA = classA`),

freeing the memory would be an error.

SoftBench CodeAdvisor can detect this sort of logic error. It scans your program, looking for `operator=` definitions, and makes sure you check for the `classA = classA` case. If not, it warns you to fix your code.

SoftBench CodeAdvisor rules focus on finding actual or potential defects in your code. Other rules check for maintenance, performance, and future problems or porting issues. These example rules show how to check for *potential* problems:

*If any member of a class is virtual, the destructor should be virtual.*

When a class contains a non-virtual destructor, there is a danger that a class instance of a derived class may be deleted by one of its base class destructors instead of by its own destructor. This can cause a memory leak, or worse problems. For example, some static or global data should have been modified by the derived class destructor.

The problem doesn't actually occur unless a derived class instance with a non-empty destructor is deleted through a *pointer* to one of its base classes. However, non-virtual destructors can result in maintenance problems, since new derived classes can be added at any time. Nothing prevents the new derived object from being deleted through a base class pointer.

*Provide an `operator=` for classes that dynamically allocate memory or declare a copy constructor.*

The default `operator=` simply copies the fields of one instance to another. If the instance contains pointers to memory allocated by `new`, the default `operator=` makes two copies of the pointers. When one of the instances is deleted, it deallocates the memory using `delete`, and the other instance contains a dangling pointer.

Refer to SoftBench Online Help for a complete list of rules shipped with SoftBench CodeAdvisor.

SoftBench CodeAdvisor continues to increase the number of rules that it checks. Consequently, rules are grouped into categories which allows you to choose which rule groups you want to check. The rules are divided into the following groups:

- Definite Defects (default)
- Probable Defects (default)
- Possible Defects
- Maintenance/Confusing Code
- Critical Portability
- Non-critical Portability
- Style
- Future Defects/Land Mines

You or your local programming staff and site administrators can extend SoftBench CodeAdvisor to add even more rules and rule groups.

User-defined rules allow you to check for specific problems that concern your organization. You can also change the contents of rule groups, including breaking out platform specific portability dependencies. Refer to the *SoftBench SDK: CodeAdvisor and Static Programmer's Guide* for details on creating rule groups.

SoftBench CodeAdvisor cannot detect every logic error in your code. However, by running SoftBench CodeAdvisor on your code, you can be confident that a variety of subtle problems have been detected.

## **Comparing SoftBench CodeAdvisor to Debuggers or Dynamic Analyzers**

SoftBench CodeAdvisor detects rule violations by performing static analysis of the code using the Static database. Static analysis differs from dynamic or run-time analysis, in that it examines all of the available code. Dynamic or run-time analysis examines only code that actually executes and does not find defects in branches that are not executed. Also, dynamic analysis requires that the code is developed to the point where it can be executed; whereas static analysis can run code checking as soon as the code compiles, even if the code cannot successfully execute.

## Performing the "Check Code" Operation

Perform the following steps to use SoftBench CodeAdvisor:

1. Build your project with the static option on (default). This gives the compiler the `-y` option and generates a Static database.
2. Run SoftBench CodeAdvisor on your compiled program.
3. View any rule violations in the SoftBench CodeAdvisor error browser.
4. Filter out any rule violations that you want to ignore (optional).
5. Change your program to correct the errors.

## Preparing Your Program with Project Build

You must build your program at least once with Static database creation enabled and add the set of files to the project that you want to run SoftBench CodeAdvisor on. To make sure that your **project build** creates a Static database, examine the Compile Mode area on the main tool face and verify that the "■ Static" toggle button is selected.

By default the Builder remembers information about your program needed for code checking, including the compile options it needs to rebuild your program after you make changes to your code.

## Preparing Your Program with External Build

If you already have an existing build script/makefile and do not want to spend the time to teach SoftBench to build it, you can create an **external build**. (See SoftBench Online Help for details on setting up an external build project.)

To compile from the command line and enable code checking for your program, use the `-y` compiler option.

Now bring up SoftBench, create an external build project, and perform a complete rebuild as you normally do. Select the project, and run Check Code from the SoftBench main window. Alternatively, you can check code from the command line using "softcheck" (see *softcheck(1)*); however, you can access filtering and rule specific help only via the SoftBench main window.

## Using SoftBench CodeAdvisor

### Performing the "Check Code" Operation

If some of your compiles take place in directories other than your build directory you need to set the SB\_DBNAME environment variable prior to building. Set SB\_DBNAME to the path of the static database file expected by SoftBench CodeAdvisor (i.e.,  
`SB_DBNAME=builddir/Static.sadb;export SB_DBNAME` where "*builddir*" is the "Build Directory" specified in the "Create Project" dialog box).

Refer to the SoftBench Online Help node for "SoftBench How To" for details on preparing your programs for SoftBench CodeAdvisor.

### Accessing SoftBench CodeAdvisor

1. From the main SoftBench window select the **Expand >>** button. The Builder and SoftBench CodeAdvisor roll-out area appears.
2. Select the CodeAdvisor tab at the bottom of the roll-out area. See Figure 6-1.

### Selecting Rule Groups

You can either use the default rules "Definite Defects" and "Probable Defects", or you can choose the rules you want to check your code against.

1. If not already selected, select the **Expand >>** button on the SoftBench main window to open the roll-out area.
2. Select the CodeAdvisor tab. The configured rule groups appear at the top of the SoftBench CodeAdvisor area.
3. Activate the desired rule groups by selecting the appropriate radio buttons.

If necessary, use the **Rule Group Help...** button to access rule group descriptions.

### Checking Your Program

If you have successfully built your program, select the **Check Code** button on the SoftBench main window to update the database and initiate rule checking. SoftBench displays a spinning clock while SoftBench CodeAdvisor completes your code check.

SoftBench CodeAdvisor only checks files and/or targets selected (or all

files if a project is selected). For example, if a violation depends on code in two files and you have only selected one of them for checking, SoftBench CodeAdvisor cannot detect the violation. Even if the file selected contains the actual violation, the other file was not selected, so SoftBench CodeAdvisor cannot detect the error. To safeguard against missing violations, select all files and filter out those files whose violations you don't want to see.

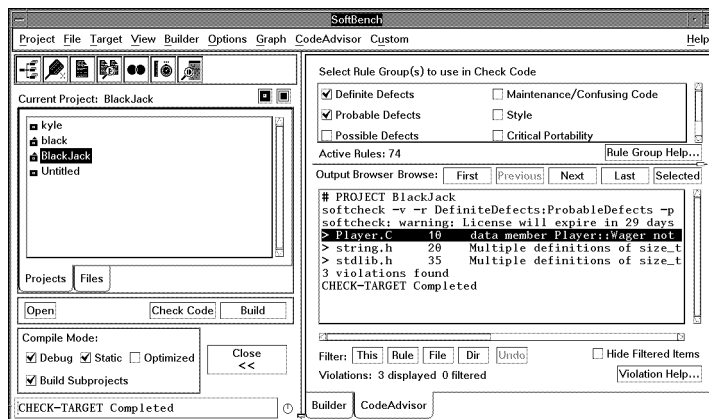
## Viewing Violations

SoftBench CodeAdvisor scans your program and lists rule violations as it encounters them. SoftBench CodeAdvisor displays violations in the SoftBench CodeAdvisor Browser, where you can browse them just like build errors.

1. Double-click violations with the mouse, or select the **First**, **Next**, **Previous**, **Last**, or **Selected** buttons, to view or edit the source code that triggered the violation. See Figure 6-1.
2. Using your pre-configured editor, fix the error and save the file. For information on configuring editors, see "Configuring an Editor" on page 119.
3. If you do not understand the violation, you can request help on the violation by selecting it and then selecting **Violation Help**. The online help explains the rule and describes the type of code that triggers the violation.

Figure 6-1

## Violations Display



## Terminating SoftBench CodeAdvisor

After **Check Code** has been pressed, the button changes to read **Terminate**. At any point you can cancel the Check Code operation by selecting the **Terminate** button, or you can allow the check to attempt to finish. If SoftBench CodeAdvisor for some reason cannot parse a file, SoftBench CodeAdvisor displays a message and SoftBench CodeAdvisor does not check this file. If the error resulted from a missing include file and/or a define, the check continues, even though some data is invalid. At any time, you may terminate the check, fix the problem and re-initiate another Check Code operation.

## Filtering Rule Violations

Filtering gives you the ability to hide violations in the browser. Once you have examined the violations, you may decide that you don't want to display some of them. You may choose not to change the code that triggers certain violations. The suggested fix may violate a local programming convention, or it may not be possible to fix the problem at this time. You can filter out any violations that you wish to ignore. Filters allow you to ignore the same set of violations each time you code-check your program. You can filter on a single violation, a Rule, a File and a Directory basis.

To filter a specific instance of a rule violation:

1. Select a violation in the output browser.
2. Select the **This** button or choose "Filter Selected Item" from the "Browser Actions" popup menu.

A single instance of a violation is filtered out. The filter is based on the current file and line number. If you subsequently edit the file and change the line number, this filter will no longer filter the unwanted violation.

To filter on a rule name so that no violations of this type are displayed:

1. Select a violation containing the type of rule you want to filter out.
2. Select the **Rule** button. SoftBench CodeAdvisor filters out all instances of violations for that rule.

Sometimes you may not control the contents of every file in a project. In these cases, you can filter all violations from a specific file:

1. Select a violation generated from the file that you want to filter.
2. Select the **File** button.



This action filters all violations from the file that generated the selected violation.

To filter all violations from a specific directory and its subdirectories:

1. Select a violation generated from the directory that you want to filter.
2. Select the **Dir** button.

This action filters all violations from the directory that generated the selected violation. It also filters violations from subdirectories below that directory. For example, if you filter the directory `/usr`, you automatically filter violations from `/usr/include` as well.

You can also write custom rules for your project or site. See SoftBench Online Help for more details on filtering rule violations.

## **For More Information**

For additional task information, refer to SoftBench Online Help by choosing "Help: SoftBench How To" from the SoftBench main window.

## 7

## Using SoftBench Debugger

SoftBench Debugger serves as a window interface to HP's debugger *DDE(1)*, enhancing and extending the DDE functionality. You can use it to examine and control the execution of your programs. The following chapters discuss the SoftBench interface to DDE and common tasks you can perform with SoftBench Debugger.

For complete descriptions of the SoftBench Debugger menus, use SoftBench Online Help. See "Getting Help" on page 84 for information on accessing and using online help. For detailed information on DDE, refer to the *Distributed Debugging Environment Reference* by choosing "Help: DDE Reference".

## Understanding SoftBench Debugger

SoftBench Debugger provides interactive source and assembly-level debugging for software programs. It provides an area for viewing source code, an area for entering debugger commands, and areas for debugger output and program I/O. SoftBench Debugger also provides an interface for editing and rebuilding programs. When you work with SoftBench Debugger, you use the same language constructs as in the program you are debugging.

SoftBench Debugger operates in two modes. When you start it from the SoftBench window, it operates with the knowledge of your **project** data. If you start SoftBench Debugger directly from the command line, it operates with limited knowledge of your application and you may need to provide information such as where the source directories are located.

SoftBench Debugger lets you:

### Display and modify variables

You can view the value of any type of data item in the program and display it in the most appropriate format. When necessary, you can change the value of a data item.

### Trace program flow

You can execute one or more statements at a time, allowing you close examination of program flow and data areas. If you have a large program, you might prefer to set breakpoints at certain statements in the program. When the breakpoints occur, you can examine data areas and alter them if necessary. If your program contains several procedure calls, you might want to display the program stack to trace those calls.

### Monitor variable values

You can set a **watchpoint** on a variable, causing it to be monitored after each instruction, statement, or function entry point, or when the program returns control to the debugger. SoftBench Debugger displays the current variable values in the Data Watch Window.

### View machine instructions

You can view disassembled machine code with symbolic addresses at any address in your program. You can also view and access register values. SoftBench Debugger shows associated source line numbers where possible, and displays source code.

### Visually navigate complicated data structures

You can visualize what is happening to your data structures by using the Data Graph Window to de-reference pointers and look at data structure values.

### Debug C++ programs

SoftBench Debugger fully supports C++ data and control structures. You can set breakpoints on specific classes, member functions, templates, instances, and overloaded functions. You can specify how inherited data and functions should be treated.

### Edit source files

SoftBench Debugger has a "Source File Area" that allows you to edit source files using a text editor that supports emacs-style key bindings. You can also edit programs with your configured editor from the File menu.

### **Build Your Code**

SoftBench Debugger allows you to build without returning to the main SoftBench window. This function unloads, requests the builder to build, and reloads the executable file automatically.

### **Customize the user interface**

You can create buttons on the SoftBench Debugger window to streamline frequently repeated operations.

## Preparing Your Program for Debugging

Use Builder to compile (build) your projects for debugging. By default, Builder automatically generates the debug information required by SoftBench Debugger if you set the the "■ Debug" compile mode toggle button on the SoftBench main tool face. It may be necessary to recompile your application after enabling the "■ Debug" toggle.

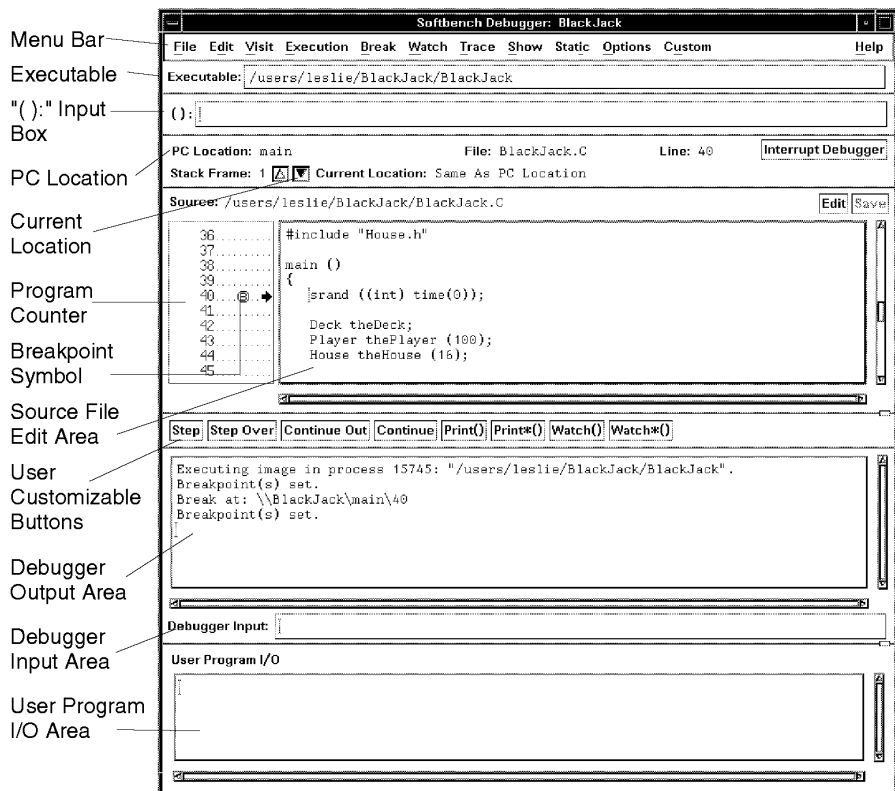
SoftBench Debugger uses information placed in the executable file by the "debug" option of your language compiler. Most compilers use the `-g` option. The aCC compiler also accepts the `-g0` flag, forcing the compiler to generate full debug information. This flag is recommended for use with SoftBench Debugger

You do not need to make changes to your source code to use SoftBench Debugger. SoftBench Debugger works with both window-oriented and standard I/O programs. (See "Interacting with Your Program" on page 174.)

## Using SoftBench Debugger Window Areas

SoftBench Debugger provides an easy-to-use windowed interface to an underlying debugger. (See Figure 7-1.) SoftBench Debugger allows you to debug your programs without in-depth knowledge of the underlying debugger. The window shows the current program being debugged, the source code being debugged, debugger and program I/O, and other useful information.

**Figure 7-1** SoftBench Debugger Window





Like SoftBench Static Analyzer, SoftBench Debugger provides a "( )" input box. This input box provides input to a number of command buttons (such as **Print ( )**) and many pull-down menu commands (such as "**Break: Set At Hex Address ( )**" or "**Visit: Procedure ( )**"). You can enter information into the "( )" input box for use with these commands, either by typing or by selecting text in the source or I/O areas. Text can be selected either by dragging the mouse, or by double-clicking. SoftBench Debugger supports a "language sensitive" double-click selection mode to select multiple-symbol identifiers. See SoftBench Online Help for details about "Options: Text Selection Behavior".

If you pause the program, the current **PC** (Program Counter) Location is displayed below the "( )" input box. If the program is running or if other actions (such as loading) are in progress, the PC Location is replaced with a highlighted status indicator and an animated "clock" icon appears. While the user interface still responds to commands (such as changing options settings) while the "clock" is running, the underlying DDE debugger is busy and does not accept commands. SoftBench Debugger queues any operations that result in sending commands to DDE while the "clock" is running.

When the program is "Running...", you can interrupt it by selecting the **Interrupt Program** button. This button changes to **Interrupt Debugger** when the program is not running. The **Interrupt Debugger** button allows you to stop the background debugger process. Use this button if you want to stop the debugger, to make changes to your program and restart the debugger.

SoftBench Debugger also shows the **Current Location** of the source code being displayed in the Source File Area. The Current Location is the environment in which variables are evaluated when you use **Print( )** or similar commands. The Current Location follows the PC Location as you execute through your program. In this case, the Current Location indicator is "Same As PC Location". You can change the Current Location to any function in the current call stack by using the ▲ and ▼ buttons next to the "Stack Frame" label. You can change the Current Location to any function in your program (to print or change local static variables, or just to view the source of a function not currently in the stack) by entering the function name in the "( )" input box and choosing "**Visit: Procedure ( )**". This sets the "Stack Frame" to "none".

The Source File Area displays the source code for the program being debugged. Clicking the right mouse button in this area displays a popup menu with several useful operations. To the left of the Source File Area

SoftBench Debugger shows the Annotation Margin, which indicates the Program Counter location with a PC Arrow. The Annotation Margin also shows program **breakpoint** locations. See “Setting and Using Breakpoints” on page 189. Breakpoints can be set and cleared by clicking in the Annotation Margin at the desired breakpoint location. SoftBench Debugger displays other symbols when debugging optimized code. See “Debugging Optimized Code” on page 227.

The user customizable buttons provide easy access to common debugger operations, such as **Step** and **Print()**. You can customize these buttons (and the popup menu over the Source File Area) to provide the exact environment desired for a particular application. See “Customizing User Buttons” on page 228.

Below the user buttons are the DDE output area and the "Debugger Input" input box. SoftBench Debugger uses these areas for communicating directly with DDE, receiving output from DDE and sending raw DDE commands to DDE.

At the bottom of the screen SoftBench Debugger displays the User Program I/O area. Programs being debugged send their output to this area and take their input from this area. You must move the mouse pointer into this area to enter text into the `stdio` of your program.

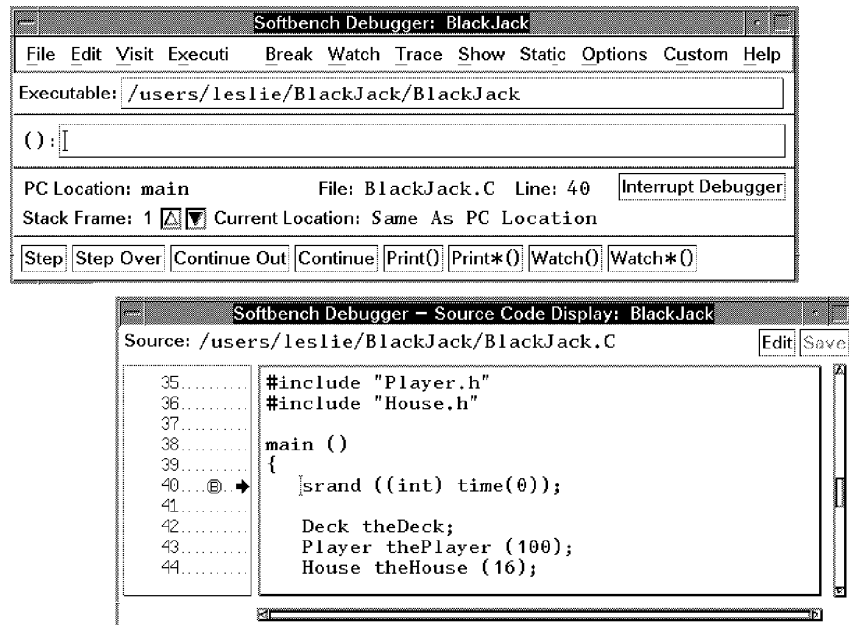
## Tearing Apart the Main Toolface

The main SoftBench Debugger window can become quite large for some screens. Consequently, you can separate the main window into as many as four separate top level windows containing:

- the Menu Bar, Executable, "()" input box, PC Location, and Current Location, and uses customizable buttons.
- the Source File Area
- the Debugger I/O Area, Debugger Input Area
- the User Program I/O Area

Figure 7-2

### Tear Apart Main Toolface



Each of these windows has its own unique icon. Use "Options: Window Configuration ..." to choose which areas appear in the main toolface, and which have their own top level window.

## Using SoftBench Debugger

### Tearing Apart the Main Toolface

To iconify all of the windows at once choose "**F**ile: **I**conify Windows".  
To normalize all windows at once choose "**F**ile: **N**ormalize Windows".  
These commands apply to all top level windows; those listed above as well as these windows:

- stack trace
- disassembly
- register windows
- data graph
- watchpoint values.

## Loading or Rerunning an Executable Program

When you start SoftBench Debugger by selecting a target in the main SoftBench window, then selecting the "Debug" icon, SoftBench Debugger automatically loads the correct executable file.

To load an executable program into SoftBench Debugger when you are already in SoftBench Debugger, choose "File: Load Executable...". (If SoftBench Debugger already has an executable loaded, you must first unload it by choosing "File: Unload Executable".) A dialog box lets you specify the executable to load and the environment in which you want to run the program. See "Specifying the Runtime Environment" on page 165.

To restart an already-loaded executable program using the same runtime environment, choose "File: Rerun". To restart the program and specify a new runtime environment, choose "File: Rerun...".

See "Debugging After a Program Fails (Core Dump)" on page 217 for information on loading and debugging a core file.

## Specifying the Runtime Environment

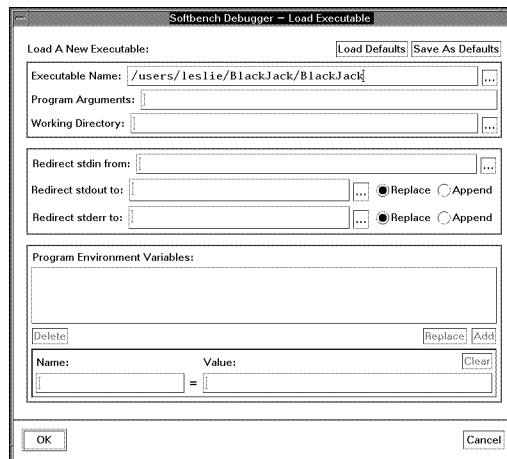
Ordinarily, the program runs using the default runtime environment. You can specify the runtime environment in the main SoftBench window by choosing "Target: Modify Properties...". The "Execution/Debug Properties" tab of the resulting dialog box lets you set program arguments, I/O redirection, and the working directory. The "Runtime Variables" tab lets you set the program's environment variables. In SoftBench Debugger, choose "Options: Default Load/Rerun Settings...". Both the SoftBench main window and SoftBench Debugger store the runtime environment information for your project targets in the same place.

When you load or rerun an executable, SoftBench Debugger starts the program and pauses at the automatically-created breakpoint on the first line of your `main()` procedure. It's important to realize that your program is *already running* at this point, so it's too late to specify program arguments, I/O redirection, and environment variables. SoftBench Debugger provides several ways to run your program that let you control settings before SoftBench Debugger starts your program.

Choosing the menu option	Allows you to
"File: Load Executable..." or "File: Rerun..."	set the arguments, I/O redirection, and environment variables for your program. You can specify the <i>default</i> values of these settings by choosing "Options: Default Load/Rerun Settings...".
"File: Load Executable..."	initialize all values to the defaults specified in "Options: Default Load/Rerun Settings...".
"File: Rerun"	initialize the dialog box with the settings specified for the previous run. If you change some values and want to revert to the default values, select <b>Load Defaults</b> .

All three of these commands display almost identical dialog boxes. Figure Figure 7-5 shows the "File: Load Executable..." dialog box.

**Figure 7-3** "Load Executable" Dialog Box



Select **Save As Defaults** to save the current values as the new default values. This is equivalent to specifying the current values in "Options: Default Load/Rerun Settings...", or in the "Target: Modify Properties..." dialog box in the main SoftBench window.

Select **OK** to start the program with the current settings.

"Execution: Get Current Program Info..." displays a dialog box containing useful information about the current program, including the values set in the above dialog boxes.

### Specifying the Working Directory and Program Arguments

By default, the loaded executable runs in the project's local workspace root. When you start SoftBench Debugger from the command line, the executable runs in the current working directory.

If you want the program to run in another directory, choose "File: Change Working Directory..." and specify the desired directory in the "Working Directory" input box. Setting the working directory does not affect SoftBench Debugger's or DDE's working directory.

To send program arguments, choose "Options: Default Load/Rerun Settings...". Enter your desired arguments in the "Program Arguments" input box. You can also specify these arguments from the SoftBench main window using "Target: Modify Properties..." and selecting the "Execution/Debug Properties" tab.

### Specifying Standard I/O

By default, your program's `stdin`, `stdout`, and `stderr` are intercepted by the User Program I/O area on the SoftBench Debugger window. To attach the standard I/O to a different file:

1. Choose "Options: Default Load/Rerun Settings...".
2. Enter the desired files in the "Redirect" input boxes. Select the ... button to display a file selection dialog box.
3. Select the appropriate  Replace" or  Append" radio button.

Selecting the  Replace" button on the `stdout` line is equivalent to "`program > filename`". Selecting the  Append" button is equivalent to "`program >> filename`".

When you redirect `stdin`, `stdout`, and `stderr`, you have no access to your program's I/O from SoftBench Debugger. The User Program I/O area is removed to indicate this. You can also specify this information from the SoftBench main window using "Target: Modify Properties..." and selecting the "Execution/Debug Properties" tab.

### Setting Environment Variables

The "Program Environment Variables" list on the "Set Default

Program Load Values" dialog box shows the environment variables SoftBench Debugger passes to your program.

To add new environment variables:

1. Choose "Options: Default Load/Rerun Settings..".
2. Enter the variable name and desired value in the "Name" and "Value" input boxes.
3. Select **Add**.

To remove an environment variable:

1. Choose "Options: Default Load/Rerun Settings..".
2. Highlight a listed *name=value* pair.
3. Select **Delete**.

To edit an existing environment variable:

1. Choose "Options: Default Load/Rerun Settings..".
2. Select it to copy its values into the edit fields.
3. Make any desired changes to the name or value.
4. Select **Replace** to replace the old values with the new values.

You can also specify this information from the SoftBench main window using "Target: Modify Properties.." and selecting the "Runtime Variables" tab.

## Specifying Source Locations

SoftBench Debugger uses three sources of information for locating source code for the loaded executable:

- The compiler stores the location of files in the executable using the *same string*, either absolute or relative path, used to specify the file during compilation. As long as you don't move the source files and run SoftBench Debugger from the same directory as you ran the build process, SoftBench Debugger can find the source files.
- SoftBench stores information about your projects, including source code locations. If for some reason, the compiler's knowledge of source code locations is insufficient, then SoftBench Debugger uses the local and **alternate source roots** of the source code files in your project.



- If you start SoftBench Debugger from the command line (which means it has no access to the project data), or if you load an executable that is not part of the current project, you may need to list the source directories explicitly by choosing "File: Add Source Directories..".

## Debugging Executables in a Project

When you start SoftBench Debugger from the main SoftBench window, the debugger benefits from the knowledge SoftBench has about the executable. SoftBench may provide information like the local workspace root, the source directories of the files in the project, and the execution properties for the target. The type of information SoftBench provides to SoftBench Debugger depends on how the executable was built. When you use **project build**, SoftBench knows where to find the source code. When you use **external build**, you can successfully build your project using your Makefile or build script, without telling SoftBench where to find all the source files.

### Adding Source Directories for External Builds

If you run into source location problems with a target from a project using external build, you can choose: "File: Add Source Directories.." in SoftBench Debugger. When you need to add source directories you can specify the source directories in two ways:

- Explicitly list the directories where source can be found.
- List *mappings* that map the old file location into a new location. For example, assume the executable file was created under the home directory of another user, fred. All source and include files existed in a directory tree under /users/fred. Now you have copied fred's directory tree under your home directory, /users/myname. You could specify each directory in the directory tree where files can be found, but it is simpler to define a **mapping**: /users/fred=/users/myname. Now each reference to /users/fred/*dir* maps to /users/myname/*dir*.

To specify alternate directories:

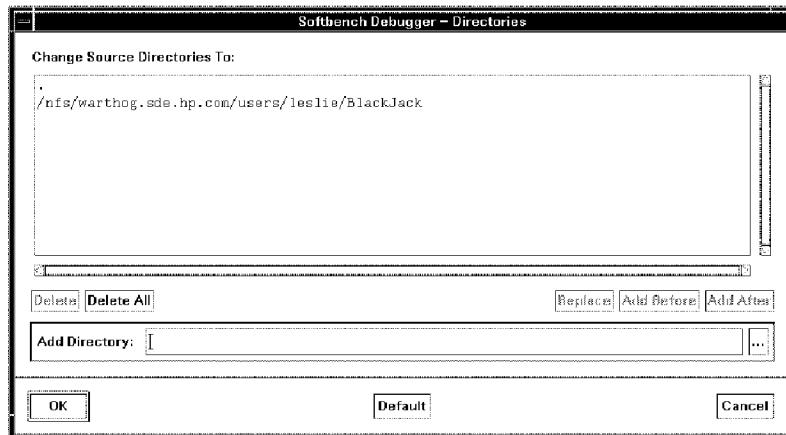
1. Choose "File: Add Source Directories..".
2. Type the directory names or mappings in the input box of the "Add Source Directory" dialog box. See Figure 7-4. Or use the ... button to browse the directories.

Using SoftBench Debugger  
Loading or Rerunning an Executable Program

3. Select **Add After**, **Add Before**, or **Replace** to add a directory to the list of directories.
4. Select **OK**.

SoftBench Debugger uses the new directory settings for the current debugging session.

**Figure 7-4** "Add Source Directories" Dialog Box



## Stepping through Your Program

When SoftBench Debugger loads a program, it begins executing the program and pauses at the first line with a breakpoint. You can then use SoftBench Debugger to execute your program one or more statements at a time.

In the default configuration, SoftBench Debugger displays the following buttons above the Debugger Output Area:

<b>Step</b>	Execute one statement, then stop. This is called <b>single step</b> execution.
<b>Step Over</b>	Execute a statement, treating any procedure call as a single statement. SoftBench Debugger calls the procedure, but control does not return to the debugger until the procedure returns. When the <b>PC</b> is just before a procedure call, this has the effect of "stepping over" the call.
<b>Continue Out</b>	Finish executing the current procedure. Run without stopping until the current procedure completes and returns to its caller (or until SoftBench Debugger encounters another breakpoint or similar event), then stop. Use this when you accidentally step into a procedure that you do not want to step through, or when you interrupt your program in the middle of non-debuggable code. Each <b>Continue Out</b> causes your program to "pop out" one procedure level.

When you select one of these buttons, the PC arrow moves to the next statement to be executed.

SoftBench Debugger steps over undebuggable routines, such as system library routines and routines that were not compiled with the `debug` option, even when using **Step**.

To pause at a specific point in your program, see "Setting and Using Breakpoints" on page 189.

## Interrupting a Running Program

When the PC Location is "Running...", your program has control, and you cannot interact with the debugger. You may want to interrupt your program so you can regain control of the debugger.

Instances when you might want to interrupt a running program include:

- Your program hangs in some internal loop.
- Your program did not arrive at breakpoints as planned.
- Your program is waiting in an input statement (such as `scanf`), and you want to set a breakpoint, or **Continue Out** after you enter the data.
- It appears that something is going very wrong with some internal state.

In these cases and many more, the PC Location shows "Running..." and displays a small "clock" animation. You can interact with the program only under the program's control. SoftBench Debugger queues any commands (except commands that restart, kill, or unload your program) until your program returns control to the debugger. Selecting **Interrupt Program** discards any queued commands and returns control to SoftBench Debugger.

If you interrupt your program while it executes code that was compiled with the debug options on, you can continue working just as if you had encountered a breakpoint at that location. A PC arrow appears in the Annotation Margin and the source for the code is displayed. At this point the PC Location shows a valid location and you can enter debugger commands.

## Interrupting in System or Non-debuggable Routines

If you interrupt the program while it executes some system-supplied routine (kernel code), or any other routine that was compiled without debug options on, the PC Location may consist of a "virtual" address. SoftBench Debugger clears the Editable Source Area to indicate that no source is available. You cannot examine local variables or step through statements. You can only step by assembly instructions, and examine other procedures on the call stack.

You can run the nondebuggable routine until it reaches the point where

it returns to its calling procedure by selecting **Continue Out**. You can continue doing this until your program returns to debuggable code. You could also set a breakpoint at some later point in debuggable code.

If the nondebuggable code is in an infinite loop, or does not return for some other reason, you must kill or rerun the program. There is no way to return the program to debuggable code.

## Interacting with Your Program

Many UNIX programs function quite well when their standard input and output are redirected. These programs are easy to debug using SoftBench Debugger, since SoftBench Debugger redirecting their I/O to the User Program I/O area does not affect them.

Some programs are designed to be invoked from a terminal environment. They may use the terminal for program input and output, perhaps even using cursor-movement commands. (The screen editor `vi` is an example of this.) When you run a program in the SoftBench Debugger environment, it does not have access to a terminal environment. SoftBench Debugger intercepts the I/O streams to the program. You must interact with your program differently when it is running under SoftBench Debugger.

Finally, X-based programs may or may not use standard I/O, but may conflict with SoftBench Debugger's use of the display.

All these types of programs can be debugged using SoftBench Debugger.

## Interacting with a Standard I/O Program

The User Program I/O Area provides access to the UNIX `stdin`, `stdout`, and `stderr` file descriptors. If the program uses standard UNIX I/O, you can see your program's output and provide input in this area. If the program performs input or output using other file descriptors, or if you redirect input or output, it happens at the specified place.

The PC Location shows "Running..." while your program is in control, whether executing or waiting for input. The PC Location *stays* "Running..." after you have started execution of your program until control passes back to SoftBench Debugger.

If the program requires input, enter it in the User Program I/O Area, just as you would in a terminal window. (Remember, however, that it is not a full terminal emulator.) SoftBench Debugger sends the input line to your program when you press **Return**. Your program has control while it executes under SoftBench Debugger. When your program encounters a breakpoint, control passes back to SoftBench Debugger, and your program no longer accepts keyboard events. SoftBench Debugger buffers any program input entered when the program is not "Running..." until SoftBench Debugger passes control back to the program.

Use the scroll bar at the right of the User Program I/O Area to see previous output or input. You can edit lines in this area (see Chapter 4, “Using SoftBench Editors,” on page 117). When you press **Return**, SoftBench Debugger sends the line where the text cursor is located as input to the program. For example, to repeat a previous input, move the cursor to that line and press **Return**.

To redirect standard input, output, or error (for example to read `stdin` from a file), choose “Options: Default Load/Rerun Settings...”. See “Specifying the Runtime Environment” on page 165.

## Interacting with a Terminal-Smart Program

The User Program I/O Area is not a full-function terminal emulator. Some programs use *ioctl(2)* or *curses(3x)* to manage a terminal screen. These programs do not run as expected in the User Program I/O Area.

As one way to debug such a program, you can run it in a terminal window and then adopt it. See “Attaching the Debugger to a Running Program” on page 222 for more information. You can also debug programs that use lower-level I/O with I/O redirection. For example, follow these steps:

1. Create a terminal emulator window (`hpterm&` or `xterm&`) with a running shell.
2. Run the *tty(1)* command in the terminal window to get the device name of the terminal (for example, `/dev/pty/ttyp5`).
3. Run the *sleep(1)* command in the terminal window to keep the shell from intercepting any input (for example, `sleep 100000`).
4. Redirect standard I/O to the terminal window. Choose “File: Rerun...”, and specify the device name of the terminal window (which is `/dev/pty/ttyp5` in this example) as the `stdin`, `stdout`, and `stderr`.
5. Select **OK**. Your program runs with the new I/O redirections. SoftBench Debugger removes the User I/O area, since all standard I/O has been redirected.

## Interacting with a Window-Smart Program

Your program should operate in **synchronous mode** so that window events (for example, mouse operations) are not queued. This ensures

that interactions between SoftBench Debugger and your program do not lead to a deadlock. Events are processed when they happen, and are directed to the appropriate window. Debugging X programs not running in synchronous mode can lead to deadlock situations in which the **keyboard focus** is on an unresponsive window.

The `XtAppInitialize` function called to initialize the X Toolkit can set your application to synchronous mode. Do one of the following:

- Pass the `-synchronize` argument to your program on the command line. To always pass this argument to your X program, add `-synchronize` to the default argument list using "Options: Default Load/Rerun Settings...".
- Use the `synchronize` resource. You can specify it using `xrdb`, or in your `$HOME/.Xdefaults` file:

```
[Application*synchronize: on]
```

where *Application* is the class name of your program.

Your program has full control while it executes under SoftBench Debugger. Mouse functions and keyboard events pass to your program as usual, with the **keyboard focus** controlled by the X Window System and your window manager. When your program reaches a breakpoint, SoftBench Debugger regains control.

Your program's input and output work as they normally do. If you have not redirected `stdin`, `stdout`, or `stderr`, they refer to the User Program I/O Area.

In some situations it may be necessary to display the program being debugged on a different display terminal. You can do this by specifying the `DISPLAY` environment variable in the "File: Load Executable" dialog before you execute the program.



---

## Specifying Identifier Locations

When you enter an expression into the "(" input box or in the "Debugger Input" input box, you must enter the data in a way that can be understood by DDE. See "Help: DDE Reference" for detailed information on the syntax required.

### Specifying Program Location

Although SoftBench Debugger is designed as a "point and click" environment, in some cases you might want to specify a **location** that cannot conveniently be selected with the mouse. A location is a line in a file and the corresponding address in your program (if there exists executable code for that line). You can specify a location when you set a breakpoint or when you choose "Visit: Procedure ( )".

The syntax for locations is:

```
line  
\proc [ \line ]  
\module [ \proc ] [ \line ]
```

*line*            The line number of the statement you want to locate, counting from the start of the file. If you do not specify a line number, SoftBench Debugger assumes the first line in a file or the first executable line in a procedure. SoftBench Debugger displays the line number in the Annotation Margin.

*module*        The *module* containing the statement you want to locate. For most languages, *module* is the basename of the source file name. For example, you would use `process` to refer to locations in a file named `process.c`. If you have several files with the same basename, such as `prog.c` and `prog.C`, you can use the full quoted filename for *module* (`\\ "prog.C" \20`).

If you do not specify *module*, SoftBench Debugger uses the file being displayed in the Editable Source Area.

*proc*           The procedure containing the instruction you want to

locate. If you do not specify *proc*, SoftBench Debugger uses the Current Location procedure.

For example, "23" would correspond to line 23 of the file displayed in the Editable Source Area. "\\xmotion\14" would correspond to line 14 of file *xmotion.c*.

DDE prints locations in the Debugger Output Area using this format. You can highlight these locations using the mouse to copy them to the "()" input box for use in "visit: Procedure ( )" and similar commands.

## Specifying Variables

You specify a variable using the same general syntax you use to specify it in the current procedure of your program. If you do not specify a complete location for a variable, SoftBench Debugger evaluates it in the scope of the Current Location.

SoftBench Debugger provides other forms for you to choose variables outside the current procedure:

Form	Description
<i>var</i>	To search the stack for the most recent instance of <i>var</i> in the current procedure. If <i>var</i> is not a parameter or local variable in the current procedure, SoftBench Debugger searches for a global variable named <i>var</i> .
<i>\proc\var</i>	To search for <i>var</i> in the most recent instance of procedure <i>proc</i> .
<i>\\var</i>	To search for the global variable <i>var</i> .
<i>`run(depth)\var</i>	To search for <i>var</i> at relative stack depth <i>depth</i> , instead of the more recent instances. A depth of 2 is one below the currently executing

procedure (that is, the procedure that called the currently executing procedure), a depth of 3 is two below, and so on. You can use this for debugging recursive procedures where the stack contains multiple instances.

When you enable stack frame numbering in the "Options: Stack Settings..." dialog box, and select "■ Stack Top is 1", SoftBench Debugger uses the leftmost numbers in the list of stack frames to show the stack depth relative to the currently executing procedure. This number can be used for *depth* in ``run( )`.

```
`main(depth)\var
```

To search for *var* at a fixed stack depth *depth*, counting from the bottom of the stack. ``main(1)` (or ``main`) corresponds to the first procedure on the stack, ``main(2)` is the first routine called from the first procedure, and so on.

When you enable stack frame numbering in the "Options: Stack Settings..." dialog box, and do *not* select "■ Stack Top is 1", SoftBench Debugger uses the leftmost numbers in the list of stack frames to show the stack depth relative to the bottom of the stack. You can use this number as the *depth* in ``main( )`.

```
`env(offset)\var
```

To search for *var* at a *relative* offset from the current environment. *offset* must include a + or - to indicate the direction of the offset.

*+offset* indicates an offset toward the top of the stack (towards ``run`), and *-offset* indicates an offset toward the base of the stack (towards ``main`).

You can also create DDE variables using the DDE `declare` command. These are useful if you need temporary variables to store intermediate results, pointers, or other values you will need later. See “Using Debugger Variables” on page 183.

## Examining and Changing Data in Your Program

You can use SoftBench Debugger to view or change the values of variables in your program. This feature directly accesses the underlying debugger, and as such is very dependent on DDE features.

DDE accepts commands only when SoftBench Debugger has control (the "Debugger Input" prompt is not greyed out) and your program is not running. If you enter DDE commands when DDE is not accepting them, DDE buffers the commands until it is ready to accept them. You can stop your program with a breakpoint (see "Setting and Using Breakpoints" on page 189) or by selecting **Interrupt Program**. You can examine variables between steps if you single-step, or dynamically by setting a watchpoint on the variables.

DDE evaluates variables in the scope of the **Current Location**, as indicated in the "Current Location" line above the Editable Source Area. Usually the Current Location follows the PC Location, so DDE evaluates variables in the environment where your program executes. For example, when you single-step through your source, the Current Location is in the procedure you are stepping through.

If you want to evaluate a variable in the scope of another function on the current call stack, use the ▲ and ▼ buttons next to the "Stack Frame" label in the Current Location line (or choose "Show: Stack..." and choose a stack frame in the "Stack View" window). This sets the Current Location to the specified function in the call stack.

To evaluate in the scope of a function *outside* the current call stack (for example, to evaluate a static variable local to a function), enter the function name in the "()" input box and choose "Visit: Procedure ( )" to set the Current Location to that function.

Finally, you can always specify the variable fully with the appropriate DDE syntax (see "Specifying Variables" on page 178). This syntax overrides the Current Location.

The PC arrow points to the line that will be executed *next*. If the PC points to an assignment statement, the assignment has not yet been executed. To see the result of an assignment statement, **Step** past it (or **Step Over** it if it calls a function).

For example, suppose you debug the following code fragment:

```
x=0; y=9;
while (y<1000) {
    x=sqrt(y);
    x++;
    y=x*x;
}
```

Set a breakpoint on the "`x=sqrt(y)`" line, and run the program. When the program stops at the breakpoint, the PC arrow is at that line. Enter "x" in the "(" input box (or simply double-click on "x", and SoftBench Debugger automatically enters it into the "(" input box) and select **Print ()**. SoftBench Debugger displays the value "0", because the assignment has not been executed. Select **Step Over**, and the PC arrow moves to the next line. Select **Print ()** to display the result of the assignment ("3").

## Examining Data in Your Program

To print the value of a simple variable or expression:

1. Enter a variable name in the "(" input box. You can double-click or highlight the variable name in the program source to copy it to the "(" input box. If the Current Location points to the procedure containing the variable, you can use the name of the variable without any qualifiers. If not, specify it according to the rules defined in "Specifying Variables" on page 178.
2. Select **Print ()**. SoftBench Debugger displays the value of the variable in the Debugger Output Area.

You can also evaluate expressions. For example, if `n` is a numeric variable in the current querying scope, `n/12.0` prints one-twelfth of `n`.

If you print an expression that contains a function call, the function executes. Be aware that this invokes any side effects in the function.

To print the value of a variable referenced by a pointer:

1. Enter the pointer name in the "(" input box. If the pointer is defined in the procedure referred to by the Current Location, you can use the name of the pointer directly. If not, specify it according to the rules defined in "Specifying Variables" on page 178.
2. Select **Print\* ()**. SoftBench Debugger displays the value of the variable in the Debugger Output Area.

For example, if you declare a pointer as `int *numptr`, then selecting **Print ( )** with `numptr` in the "( )" input box prints the integer pointed to by `numptr`. You can also **Print ( )** the expression `*numptr`.

## Printing Hex or String Values

SoftBench Debugger knows the type of the variables you print, and usually prints them in a usable form. However, it cannot tell if an integer should be printed in decimal or hexadecimal, nor can it tell if you want a `char *` value to be printed as a character or a string.

You can specify the print format by choosing one of the selections under "Show: Data Value → *Print Format*". Refer to online help for information on printing strings and character arrays.

If you frequently need to print hex or string values, you can define a button to do it on the front panel. See "Customizing User Buttons" on page 228.

You can also use C-style "casting" to view variables in a different format. If you wanted to examine the bottom byte of `mask` as a character, you could enter `(char) mask` in the "( )" input box and select **Print ( )**.

## Changing Data in Your Program

1. Type an assignment statement in the "( )" input box.
2. Select **Print ( )**.

For example, the assignment `n = 12` sets the value of `n` to 12. The assignment statement must be legal in the current scope. That is, `n` must be a declared numeric data type.

You can also call functions that modify their arguments.

For more information on the syntax of variables and expressions, see "Using Debugger Variables" on page 183 or "Specifying Variables" on page 178.

## Using Debugger Variables

DDE supports user-created special variables for use as global temporaries during debugging. For example, to create a pointer to an integer, enter

```
declare int *tptr
```

in the "Debugger Input" input box. The `tptr` variable can then be used in any expressions in the "()" input box. For example, if `counter` is an integer in your program, you could store a pointer to it by entering

```
tptr = &counter
```

in the "()" input box, and selecting **Print()**. If you then enter "tptr" in the "()" input box, you can print its value by selecting **Print()**, or print the value of `counter` by selecting **Print\*()**.

See "Help: DDE Reference" for detailed information on the `declare` statement.

You can use these special variables as local memory. For example, suppose your program defines a tree structure. To examine several nodes under a node of the tree:

1. Create two DDE variables by entering "`declare node-type *top, *here`" in the "Debugger Input" input box, using whatever node type is appropriate for your tree.
2. Set `top` to the root of your tree, or wherever you want to start.
3. Traverse down to the node of interest by setting `here` to `top->left` or whatever is appropriate for your example.
4. View or modify the `here` node of the tree, using **Print()**, **Print\*()**, and any other appropriate commands.
5. When you finish accessing the current node, set `here` to the next node you want to work with (such as `top->right` or `here->left`).

Enter "list declarations" into the "Debugger Input" input box to list all DDE variables you have defined in the Debugger Output Area. You can scroll this area to examine them if you have more than fit in the screen area.

## Using Expressions

Expressions are composed of any combination of variables, constants, and operators. SoftBench Debugger keeps track of the language the program was written in, and recognizes language-specific operators used by that language.

You can change the language used by DDE to evaluate expressions using the DDE "`prop lang`" command. "`list prop lang`" shows the current language. See "Help: DDE Reference" for detailed information.



If your program uses several different source languages, you can use "prop lang" to re-evaluate a complicated expression without re-entering it in the new language. For example, suppose you have a main program written in C that calls FORTRAN subprograms for some of its functions. Suppose you evaluate a complex C expression to check the value of a certain variable. After you **Step** into one of the FORTRAN subprograms, you can evaluate the expression from the main program, still using C syntax, even though you are now in a FORTRAN subprogram.

1. Enter "prop lang C" in the "Debugger Input" input box and press Enter.
2. If the "(" input box does not still show the desired expression, enter the expression you want to evaluate into the "(" input box.
3. Select **Print ()** to evaluate the expression.
4. Enter "prop lang -default" into the "Debugger Input" input box and press Enter. This returns DDE to the default language, which in this case is FORTRAN.

You can also use the `-lang` option on the DDE print command:

```
print -lang C chapter="string"
```

## Using Constants

Expressions can contain constants as well as variables. When DDE is in C mode, integer constants can begin with "0" for octal, or "0x" or "0X" for hexadecimal. Floating point constants are of the same form as those in the source language, except that, unlike FORTRAN, no spaces are allowed within a floating point constant. For example, 1.0, 3.14e8, and 26.62D-31 are valid FORTRAN floating point constants. See the Language Manager appendices in *HP/DDE Debugger User's Guide* for other restrictions on floating point constants.

Character and string constants must be entered in the syntax appropriate for the current language. In C mode, character constants must be entered in single quotes (*'char'*) and are treated as integers. C string constants must be entered in double quotes (*'string'*) and are treated like "char \*" (pointer to char). FORTRAN strings can be enclosed in either single or double quotes.

Some operators are not supported or have different semantics from the standard language definition. See "Help: DDE Reference" for other specifics.

## Calling Functions

You can call a procedure by putting the procedure call expression in the "()" input box and selecting **Print ()**.

For example, printing:

```
myproc(1, 2, "string")
```

calls the function "myproc( )". Any side-effects of the function occur normally.

## Viewing the Call Stack

When enabled, the "Stack View" window presents you with a listing of the current execution call stack. (See Figure 7-5.) The call stack and the source code associated with each entry may be browsed by selecting a stack entry or using the ▲ and ▼ buttons.

To view the call stack:

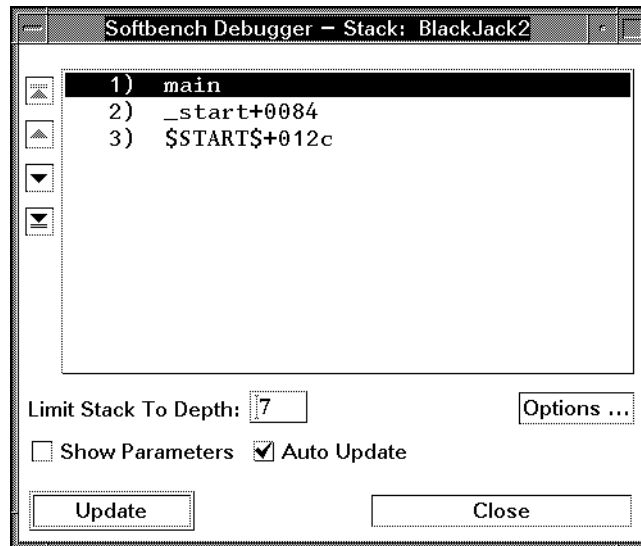
1. Choose "Show: Stack...".
2. Select an entry in "Current Stack," or use the buttons on the left side of the dialog box to move to the top of the stack, up a level, down a level, or to the bottom of the stack. The associated source code appears in the Editable Source Area.
3. Leave the "Stack View" window visible while selecting **Step** or **Continue** and watch the Current Stack change.

Select  Show Parameters" to display the parameters passed to each procedure on the stack.

Clear  Auto Update" if you do not want the stack display to update each time the debugger gains control. This can speed up your debugging if you **Step** or **Continue** frequently. Select **Update** to force an update while you have  Auto Update" cleared.

By default, the stack display numbers the stack levels, with the top of the stack being level 1. Stack numbering can be turned off, or can be reversed so that the bottom of the stack is 1, using the "Options: Stack Settings..." dialog box. Select the **Options...** button to display this dialog box directly from the Stack View.

**Figure 7-5** "Stack View" Window



## Viewing Thread Stacks

In threaded programs, SoftBench Debugger can display multiple stacks at once. Choose "Execution: Threads..." and select **Stack...** for the desired thread.

Each "Stack View" window indicates its thread identity. An additional "Stack View" window, labeled "Current Thread", displays the stack for the currently executing thread.

## Understanding the Operation of Your Program

SoftBench Debugger supports four different **monitors** to help you understand the operation of your program.

Breakpoints	Cause your program to stop executing at a specified location, and return control to SoftBench Debugger. Breakpoints can be used to examine the status of the program when a certain line executes one or more times. See “Setting and Using Breakpoints” on page 189.
Watchpoints	Monitor and display the values of specified variables as the program executes. Use watchpoints when you need to determine when a variable changes value. See “Setting and Using Watchpoints” on page 199.
Traces	Display notices when the program reaches certain locations, such as procedure entry and exit. See “Tracing Program Flow” on page 204.
Signals/ Intercepts	Detect events such as UNIX signals. See “Handling Signals and Events” on page 214.

Each form of monitor is useful in different situations, and monitors are often useful when used together. The following sections describe the use of each monitor.

## Setting and Using Breakpoints

A **breakpoint** is a "hook" placed in your executable program by SoftBench Debugger to halt execution at a specific line. SoftBench Debugger indicates a breakpoint by a breakpoint annotation in the Annotation Margin, to the left of the Editable Source Area. The breakpoint annotation appears on the same line as the source statement. See Figure 7-6.

When your program runs or continues, and the condition specified in a breakpoint occurs, control returns to the debugger and any DDE command attached to the breakpoint executes. If the attached command does not execute a "go", then the program pauses and you can query variables and perform other SoftBench Debugger operations. For information on attaching commands to breakpoints, see "Executing DDE Commands at a Breakpoint" on page 195.

Your program pauses *before* executing the source statement associated with the breakpoint. For example, suppose you set a breakpoint on an assignment statement. When your program pauses at the breakpoint, the assignment has *not* been executed. If you **Print ( )** the value of the variable, you see its *previous* value. If you **Step**, the assignment takes place and you can **Print ( )** the *result* of the assignment.

While the program stops, you can examine the values of variables and look at anything else that might help you find defects in your program. You can even change the PC location within the current function, to repeat a section of code or to skip over unwanted code.

You can have any number of active breakpoints in your program. Execution stops when SoftBench Debugger encounters *any* breakpoint.

SoftBench Debugger automatically creates breakpoints on the first and last lines of your `main()` procedure.

To get a complete list of breakpoints, use the "Break: Show..." menu selection.

## Debugging a Program Using Breakpoints

"Setting a Breakpoint" on page 190 describes how to pause your program during execution. This can be useful any time you want to execute part of your program before pausing to debug a section of code. For example,

when you are responsible for only a part of a large program, you need only debug your particular module. Set breakpoints at the entry points of your module, and run the program to execute the code down to your module. When you have reached your module, you can **Step** through it to see it in more detail.

1. Set a breakpoint where you want to pause the program. See “Setting a Breakpoint” on page 190.
2. **Continue** your program from its current location, or choose "File: Rerun" to restart your program. See “Loading or Rerunning an Executable Program” on page 165.

While the program being debugged runs to the next breakpoint, the PC Location indicator displays a highlighted "Running..." and a "clock" animation. SoftBench Debugger buffers further commands until SoftBench Debugger becomes ready to execute them.

3. When SoftBench Debugger reaches the breakpoint, the PC Location changes to the current program location. (Notice that the PC arrow displays on the same line as the breakpoint annotation.) You are now ready to continue the debugging process.

If the PC Location remains "Running...", the program being debugged could be waiting for input. See “Interacting with Your Program” on page 174.

To run from the current PC to the next breakpoint, select **Continue**. You can use this to execute sections of the program that do not need to be debugged.

## Setting a Breakpoint

SoftBench Debugger provides many ways to set breakpoints, depending on your needs. See “Setting Breakpoints in a Program File” on page 131 for information on setting breakpoints from within your editor.

### Using the Editable Source Area

You may find this method convenient if you can see or easily retrieve the desired source line.

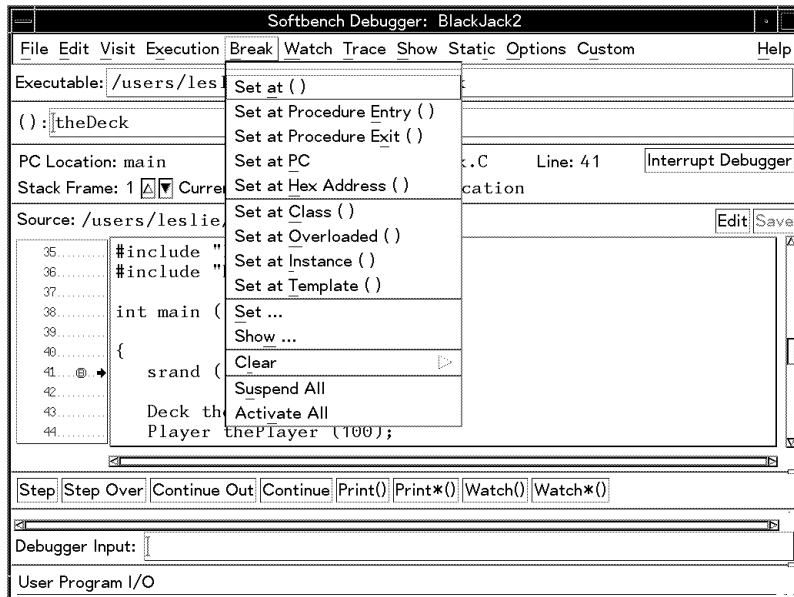
1. Locate the source statement in the Source File Area:
  - If the currently viewed source file contains the statement, scroll until you can see the line.

- If another source file contains the statement , put the name of its file or procedure in the "()" input box and choose "Visit: Procedure ()" or "Visit: File ()".
2. Click in the Annotation Margin next to the line on which you want a breakpoint. A breakpoint annotation appears in the Annotation Margin, indicating the location of the new breakpoint.

If your source code has several executable statements on the line you select, SoftBench Debugger sets the breakpoint before the first statement on the line. You can set a breakpoint on later statements by entering the statement location (as specified in "Specifying Program Location" on page 177) in the "()" input box, and adding "+1" to specify the second statement, "+2" to specify the third, and so on. Choose "Break: Set at ()" to set the breakpoint.

If you attempt to set a breakpoint on a specific line and the breakpoint annotation appears several statements further down in your program, this indicates that you selected a non-executable statement. SoftBench Debugger sets the breakpoint on the first executable statement following the one you selected.

**Figure 7-6** "Break" Menu



### Using the Break Menu

The "Break" menu contains several entries that allow you to set a breakpoint at the current PC location, or associated with an identifier entered in the "()" input box.

For example, if you want to set a breakpoint on the entry of a procedure named `compute`, you could enter `compute` in the "()" input box and choose "Break: Set at Procedure Entry ( )".

Refer to Figure 7-6 to see the available choices. See SoftBench Online Help for details on each choice.

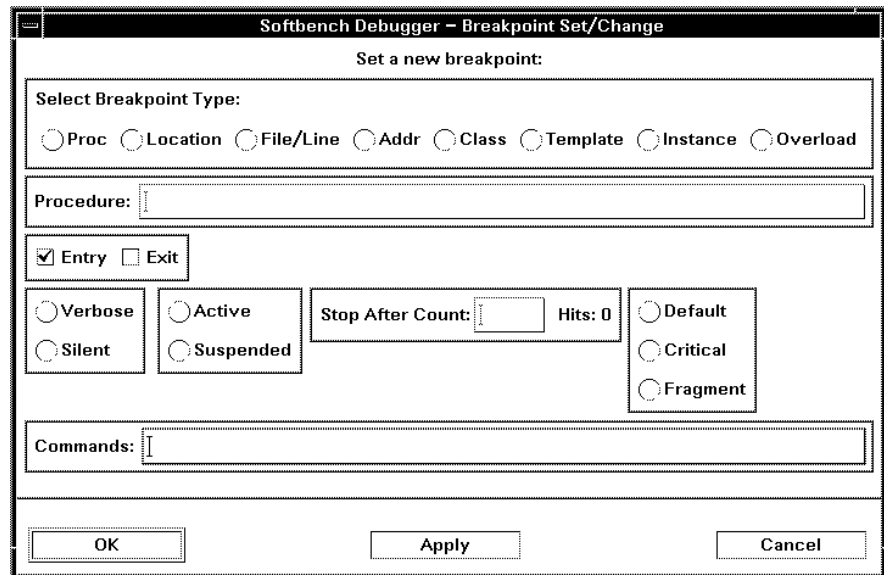
### Using the Breakpoint Set/Change Dialog Box

This method is very general, and can be used to set breakpoints in any location with any allowable attributes. See Figure 7-7.

1. Choose "Break: Set...".
2. Choose the type of breakpoint you want, such as Proc, Location, Class, and so on.
3. The area under the "Select Breakpoint Type" radio buttons changes to reflect the information needed by that breakpoint type. Enter the appropriate information.
4. Modify any other information as required. For example, you may want to specify a DDE command to be executed when the breakpoint occurs, or you may want the breakpoint to occur only after the specified event happens a certain number of times.
5. When you have specified the information you want, select **OK** to create the breakpoint.



Figure 7-7 "Breakpoint Set/Change" Dialog Box



If the program is multi-threaded, the "Breakpoint Set/Change" dialog box also includes a scrolled list of thread ID's. You can specify that a breakpoint is to affect any specific thread or set of threads.

## Viewing and Modifying Breakpoints

To see a list of breakpoints, from the SoftBench Debugger menu bar, choose "Break: Show...".

The "Breakpoint Listing" dialog box appears, showing all existing breakpoint groups and their status information. Double-click on a single breakpoint to see its associated source code, if available.

Select a breakpoint entry to modify it. Once selected, you can change the **Active/Suspend** status or **Delete** the breakpoint. To modify the breakpoint, select **Change...** SoftBench Debugger displays the "Breakpoint Set/Change" dialog box, allowing you to modify all editable attributes of the breakpoint.

Breakpoints can also be modified by holding down the Shift button while clicking on the breakpoint symbol in the Annotation Area of the Source File Area. This displays the "Breakpoint Set/Change" dialog box for the

specified breakpoint.

### Changing Active / Suspend Status

By default, a new breakpoint is *Active*. If you want to deactivate a breakpoint without deleting it, you can *Suspend* the breakpoint. This may be useful if, for example, you want a breakpoint to occur only in certain circumstances. You could *Suspend* the breakpoint, proceed with debugging your program, and then re-*Activate* the breakpoint as desired.

SoftBench Debugger provides several ways to change the status of a breakpoint:

- Click the middle mouse button on the Breakpoint Annotation in the Annotation Margin. This toggles the breakpoint status. The Breakpoint Annotation changes to reflect the new status. A suspended breakpoint has a slash through its Breakpoint Annotation.
- Modify a breakpoint (by Shift-clicking on its breakpoint symbol in the Annotation Margin, or by selecting the breakpoint in the "Breakpoint Listing" dialog box and selecting **Change...**), select the  *Active* or  *Suspended* radio button, and select **OK**.
- Select a breakpoint in the "**Break: Show...**" dialog box and select the **Active** or **Suspend** button.
- Choose "**Break: Suspend All**" or "**Break: Activate All**" to activate or suspend ALL breakpoints in your program.

### Changing Verbose / Silent Status

By default, SoftBench Debugger reports the address of a breakpoint in the Debugger Output Area when it occurs. You control this reporting ability by the *Verbose* status. If you do not want the breakpoint address reported, modify a breakpoint (by Shift-clicking on its breakpoint symbol in the Annotation Margin, or by selecting the breakpoint in the "Breakpoint Listing" dialog box and selecting **Change...**), select the  *Silent* radio button, and select **OK**. Select the  *Verbose* radio button and select **OK** to enable address reporting.

### Clearing a Breakpoint

When you no longer want your program to stop at a breakpoint, you can clear the breakpoint. If you can see the breakpoint you want to clear in

the Source File Area, you can simply click on the breakpoint symbol to clear it. If the breakpoint is not currently visible:

1. Choose "**Break: Show...**" to display all current breakpoints.
2. Click on the breakpoint you want to clear. You can double-click on a breakpoint to display its source in the Source File Area, so you can be sure you delete the correct breakpoint.
3. Select **Delete** to remove the breakpoint.

The breakpoint description disappears, indicating that the breakpoint has been cleared.

You can clear all breakpoints by choosing "**Break: Clear → All**".

You can also "deactivate" (Suspend) breakpoints without permanently deleting them. See "Viewing and Modifying Breakpoints" on page 193.

## Executing DDE Commands at a Breakpoint

You can have SoftBench Debugger execute a debugger command whenever it stops at a breakpoint. When you create or modify a breakpoint using the "Breakpoint Set/Change" dialog box:

1. Enter the desired debugger command in the "Commands" input box. See "**Help: DDE Reference**" for more information on these commands.
2. Enter a number in the "Stop After Count" input box to specify the number of times you want your program to ignore the breakpoint before it stops and executes the commands. The "Hits" field tells how many times the breakpoint has already been encountered. "Count" defaults to 0, meaning SoftBench Debugger stops on the first encounter.
3. Select **OK**. If you select **Cancel**, SoftBench Debugger discards the command entered in the input box, and does not modify the breakpoint.

For example, if you are sure an error happens on the 53rd time through a loop, you could set a breakpoint in the loop and enter 52 in the "Stop After Count" input box.

If the error happens only when the variable `x` is 14, you could enter "`print x; if x != 14 -then [go]`" in the "Commands" input box. This prints the value of `x` each time SoftBench Debugger encounters the

Using SoftBench Debugger  
**Setting and Using Breakpoints**

breakpoint, but does not stop until  $x$  reaches the critical value. You may also want to make the breakpoint `Silent` so the breakpoint notification message does not print each time the breakpoint is encountered.

## Setting C++ Breakpoints

When you debug a C++ program, SoftBench Debugger provides some additional breakpoint settings.

- "Break: Set at `Class ( )`" SoftBench Debugger sets breakpoints on all *member functions* of all instances of the specified class.
- "Break: Set at `Overloaded ( )`" SoftBench Debugger sets breakpoints on all functions with the specified name. Use this when your program defines multiple functions with the same name.
- "Break: Set at `Template ( )`" SoftBench Debugger sets breakpoints on all member functions created by the specified template.
- "Break: Set at `Instance ( )`" SoftBench Debugger sets breakpoints on all member functions of the specified instance.

When you set a breakpoint on a function which has been declared inline, it may or may not be accepted, depending on whether the compiler has made all calls to that function inline. If any calls to this function were inlined, the breakpoint does not occur at that particular call. All calls which were not inlined break as expected. To avoid this problem, recompile your code with the `+d` option which prevents the expansion of all inline functions.

## Setting Group Breakpoints

Setting breakpoints on C++ components from SoftBench Debugger either from the "Break: Set at..." menu, or using the "Breakpoint Set" dialog, may result in the creation of several DDE breakpoints. These breakpoints comprise a "group" and may be manipulated individually or as a group.

## Viewing and Modifying Group Breakpoints

Groups may be "expanded" and "collapsed" by double-clicking on them. A "+" in the first column indicates that SoftBench Debugger has currently expanded the group, and a "-" indicates that it has collapsed the group. When SoftBench Debugger has expanded a group, individual breakpoints within the group can be selected then deleted, activated, or

## Using SoftBench Debugger

### Setting C++ Breakpoints

suspended. You can change all breakpoints in the group at the same time by selecting the group title line. Double-click on a group to display all breakpoints in the group.

## Setting and Using Watchpoints

Watchpoints allow you to monitor the value of an expression, or the contents of a range of memory, while your program executes. SoftBench Debugger checks the value (as often as specified in the watchpoint definition), notifies you when it changes, and pauses your program.

This can be extremely useful when searching for "mystery" errors. For example, suppose at some unknown time your program corrupts an important area in memory. Without watchpoints, it would be very difficult to determine exactly when the corruption occurred. Using watchpoints, you can quickly find when the corruption happens, which gives you a good idea of what caused it.

When you create a watchpoint, you specify the expression or address range to monitor, and an option to specify "granularity". Granularity specifies how often the value of interest should be monitored:

- At every procedure entry
- At every procedure exit
- At the entry *and* exit of every procedure
- Every statement
- Every assembly instruction
- whenever the program stops and returns control to the debugger

The Watchpoint dialog setting defaults to "Stop". This default allows you to monitor the current values of some data items when debugging a program.

Be certain you specify an appropriate granularity. If you want SoftBench Debugger to watch a variable while your program runs, you should not have the "○ Stop" radio button active in the "Set At" area. This setting only checks your watchpoints when your program reaches a breakpoint or is interrupted for another reason. You should specify a procedure- or statement-level granularity so SoftBench Debugger checks the variable while your program runs. You can also use the "○ Modification" radio button to identify when an address or variable is unexpectedly being changed. This button issues a default DDE watchpoint command on the contents of the "()" input box. This same feature can also be accessed from the menu under "Watch: Set at Modification".

For performance purposes, you should set the granularity as coarsely as possible. For example, if you only need to know the value of the monitored expression each time you enter a procedure, there is no sense in monitoring it after every assembly instruction. Typically you would locate a problem by using a granularity of "Proc Entry+Exit" to narrow the source of the problem down to one procedure. Once the problem is localized, use a finer granularity (such as "Statement" or "Instruction") but limit it to the offending procedure by entering the procedure name in the "When In" input box on the "Data Watchpoint Set/Change" dialog box.

In many other ways, watchpoints are similar to breakpoints. In addition to stopping the program, as a breakpoint does, a watchpoint also displays the new value of a changed variable. Nearly all other breakpoint concepts apply: watchpoints can be Active or Suspended, Verbose or Silent, and can have debugger commands associated with them. If you want the watchpoint to display changed values without stopping, you can enter the DDE command `go` in the "Commands" input box of the "Data Watchpoints" dialog box. Unlike breakpoints, watchpoints have no "Stop after Count" associated with them. See "Setting and Using Breakpoints" on page 189 for a description of these concepts.

## Creating Watchpoints

As with breakpoints, SoftBench Debugger provides multiple ways to create watchpoints.

### Using the Watch Menu

The "Watch" menu contains several entries that allow you to set a watchpoint associated with a token entered in the "()" input box.

For example, if you want to create a watchpoint associated with the variable `counter`, to be checked each time a procedure was entered, you could enter `counter` in the "()" input box and choose "Watch: Set at Entry ( )". See SoftBench Online Help for a description of these choices.

### Using the "Data Watchpoint Set/Change" Dialog Box

This method is very general, and can be used to set watchpoints on any expression *or* any memory location with any allowable attributes. This method is the *only* way to set a watchpoint on a range of memory.

1. Choose "Watch: Set...".



2. Select the type of watchpoint you want: " Expression" or " Address".
3. The area under the "Select Watchpoint Type" radio buttons changes to reflect the information needed by that watchpoint type. Enter the expression or address range desired. You can `Print ()` the expression `&var` to find the address of the variable `var`.
4. Select the granularity (how often SoftBench Debugger checks the value).
5. Modify any other information as required. For example, you may want to specify a debugger command to be executed when the watchpoint occurs.
6. When you have specified the information you want, select **OK** to create the watchpoint.

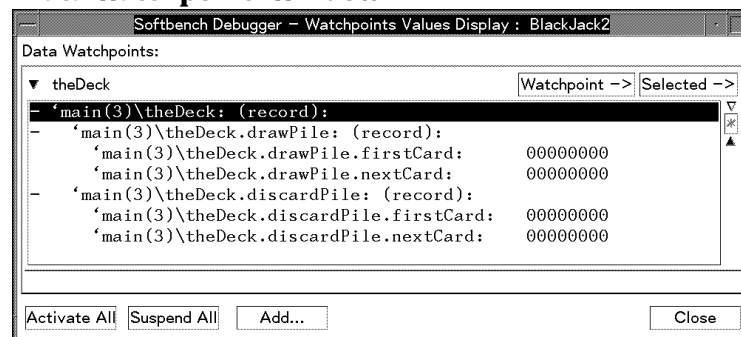
## Viewing and Modifying Watchpoints

Choose **watch: Show...** to list all current watchpoints. SoftBench Debugger displays the "Watchpoint Listing" dialog box, showing all existing watchpoints and their status information. You have the same abilities to activate, deactivate, create, delete, or edit as in the "Breakpoint Listing" dialog box. See "Viewing and Modifying Breakpoints" on page 193 for a description of these operations.

Choose **watch: Values Display...** to display the "Data Watchpoints" window if it is not already displayed. See Figure 7-8.

**Figure 7-8**

### "Data Watchpoint" Window



Each watchpoint displays the variables or memory area to which it is attached. The data display highlights when your program modifies the

data being watched. Complete watchpoints can be hidden by selecting the ▼ button next to the watchpoint. Select the button again (which is now >) to redisplay the watchpoint.

Individual elements in a watchpoint (such as some elements in an array) can be hidden by selecting them and selecting "Selected -> Hide". Select "Watchpoint -> Show All Hidden Values" to redisplay them.

Compound objects (such as arrays or structures) can be "collapsed" to simplify the display, either by double-clicking the beginning of the compound object, or by selecting it and choosing "Selected -> Collapse". Double-click the collapsed object, or choose "Selected -> Expand", to display the entire object.

### Changing Active / Suspend Status

To modify the watchpoint, choose "Watchpoint -> Change...". SoftBench Debugger displays the "Data Watchpoints Set/Change" window, allowing you to modify all editable attributes of the watchpoint.

To change the Active / Suspend status of a watchpoint, use the "Data Watchpoints Set/Change" window. You can also activate or suspend *all* watchpoints by selecting the **Activate All** or **Suspend All** buttons in the "Data Watchpoints Set/Change" window, or by choosing the "Watch: Suspend All" or "Watch: Activate All" menu selections.

### Changing Verbose / Silent Status

By default, SoftBench Debugger reports the information about a watchpoint in the Debugger Output Area when it occurs. You control this reporting ability by the Verbose status. If you do not want the watchpoint information reported, in the "Data Watchpoints Set/Change" window, select the  Silent radio button, and select **OK**. Select the  Verbose radio button and select **OK** to re-enable the reporting.

### Clearing a Watchpoint

When you no longer want SoftBench Debugger to monitor your variables, you can clear the watchpoint. Select "Watchpoint -> Delete" on the "Watchpoints Value Display" window. You can also use the "Watchpoint Listing" dialog box:

1. Choose "Watch: Show..." to display all current watchpoints.

2. Highlight the watchpoint(s) you want to delete in the "Watchpoint Listing" dialog box.

3. Select **Delete**.

You can clear all watchpoints by choosing "Watch: Clear **All**".

You can also "deactivate" (Suspend) watchpoints without permanently deleting them. See "Viewing and Modifying Watchpoints" on page 201.

## Tracing Program Flow

SoftBench Debugger's trace functions help you monitor the flow of your program. This can be useful in many situations: perhaps you want to see when your program calls a particular function, or executes certain statements, but you don't want the debugger to stop your program. Tracing simply displays the current location when a trace is hit, and keeps executing your program. Creating a basic trace is similar to adding a `printf` statement to your program. Traces can also stop your program or can execute DDE commands when they are encountered, similar to breakpoints.

### Creating Traces

SoftBench Debugger provides several levels of trace granularity:

- At every procedure entry
- At every procedure exit
- At the entry *and* exit of every procedure
- Every statement
- Every assembly instruction

Creating a new trace on procedure entry or exit can take a significant amount of time as DDE finds all the appropriate procedures. The label on the "Debugger Input" input box is not sensitive while DDE is working.

### Using the Trace Menu

SoftBench Debugger has several menu selections to simplify trace creation. Choose the appropriate option under "Trace: Trace **E**very" submenu to enable tracing.

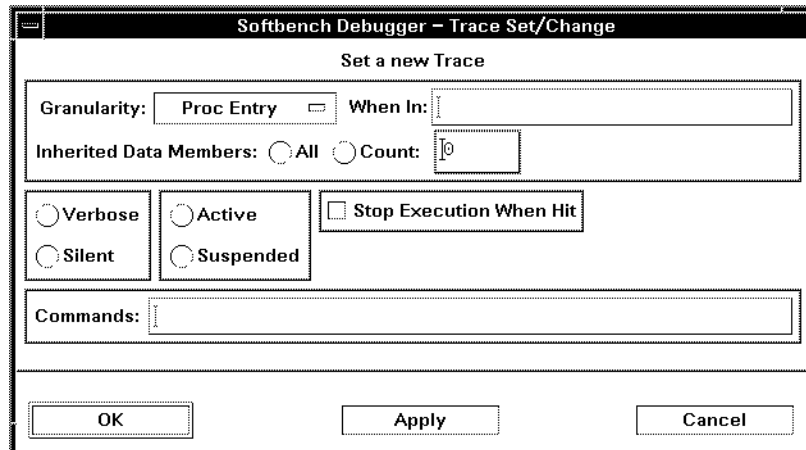
Traces can also be enabled only in certain "blocks". DDE allows you to narrow traces to a particular file, C function, or C++ object. See "Help: DDE Reference" for a full explanation of blocks.

Enter the desired block in the "()" input box, and choose the appropriate option under "Trace: Trace **O**nly" submenu to enable tracing.

## Using the Trace Set/Change Dialog Box

Choosing "Trace: Set..." allows you to specify all aspects of a trace. The "Trace Set/Change" dialog box is nearly identical to the "Data Watchpoint Set/Change" dialog box. See Figure 7-9.

**Figure 7-9** "Trace Set/Change" Dialog Box



1. Choose "Trace: Set...".
2. Select the granularity (how often SoftBench Debugger triggers the trace). Specify any "block," if desired, in the "When In" input box.
3. Modify any other information as required. For example, you may want to specify a DDE command to be executed when the watchpoint occurs.
4. Select the "■ Stop Execution When Hit" toggle button if you want your program to pause when it reaches the trace.
5. When you have specified the information you want, select **OK** to create the trace.

## Viewing Traces

Choose "Trace: Show" to display the current list of active traces. The "Trace Listing" dialog box is nearly identical to the "Breakpoint Listing" dialog box. You have the same abilities to activate, deactivate, create, delete, or edit as in the "Breakpoint Listing" dialog box. See "Viewing and Modifying Breakpoints" on page 193 for a description of

these operations.

### **Clearing Traces**

Choose "Trace: Clear All" to remove all traces from DDE's memory.

## Correcting Errors in Your Program

Developers frequently spend a lot of time in the debugger, finding and fixing defects. SoftBench Debugger provides several commands to help you when you find an error:

- You can alter variable values (see “Specifying Variables” on page 178)
- You can change the location of the PC within the current procedure (see SoftBench Online Help for details), skipping the problem code and continuing debugging.
- You can edit the source code, fixing the problem, then rebuild and restart the debugging process.

### Editing Source Code

SoftBench Debugger provides an editable source code area. Without ever leaving SoftBench Debugger you can fix the error and rebuild.

- Edit the source file by selecting the **Edit** button to make the source code area editable. If you need to check the file out of Configuration Management, do so before making changes. Choose the "File: Configuration Management" submenu.
- Save the changes and use the "Build" facility to rebuild the target by choosing "File: Build" from the SoftBench Debugger menu bar. If the file is not writable, you may need to check it out using the "File: Configuration Management" submenu.

SoftBench rebuilds the current debugged program. If SoftBench encounters any compile errors, it displays the offending source in the Debugger's Source File Area or your configured editor, depending on the setting of the "Options:  Use External Editor for Compile Errors" toggle button. Any errors after the first error are displayed in your configured editor.

If your edits introduce compile errors, browse and fix the compile errors from the main SoftBench Builder output browser, then rebuild again from SoftBench Debugger.

- When your build is successful, the new executable loads automatically in SoftBench Debugger, and you can continue to debug the program.

Alternatively, edit your source code in your configured editor by choosing "File: Edit..." to specify the file you want to edit. Make your changes, then start the build from the editor, main SoftBench window, or SoftBench Debugger, whichever is easiest for you. The editable source code area in SoftBench Debugger displays the "Out of Date" message to inform you that the source has been changed. Choose "Edit: Update Buffer" or "Edit: Update All Buffers" to synchronize your view of the source.

## Synchronizing Files

When you debug a program, you debug the object code generated from the compilation of one or more source files. When the source is out-of-date, the PC Location indicated in the Source File Area may not accurately reflect the location of the program counter.

This unsynchronized behavior occurs any time the source has been modified without recompiling the program and reloading it into SoftBench Debugger.



## Debugging Dynamic Libraries

Dynamically-loaded libraries present special challenges to a debugger, due to the way the operating system implements them. SoftBench Debugger handles most of these issues for you. You can specify which libraries you want to debug.

Programs can contain several kinds of library images:

- **Static:** the library code is physically linked in with your program.
- **Shared:** the linker knows about the library, but it is not linked in with your program. All programs that use the library share a common copy of the library.
- **Dynamically loaded:** the library is not known to the linker, and is loaded when your program requests it by calling a system routine.

When you choose "Execution: Enable Images/Libraries...", the "Images/Libraries" dialog box presents you with a list of all currently known shared or dynamically loaded libraries used by the program being debugged. (See Figure 7-10.) Because maintaining debug information for each library increases the overhead on your debugging session, SoftBench Debugger allows you to enable only the libraries of interest.

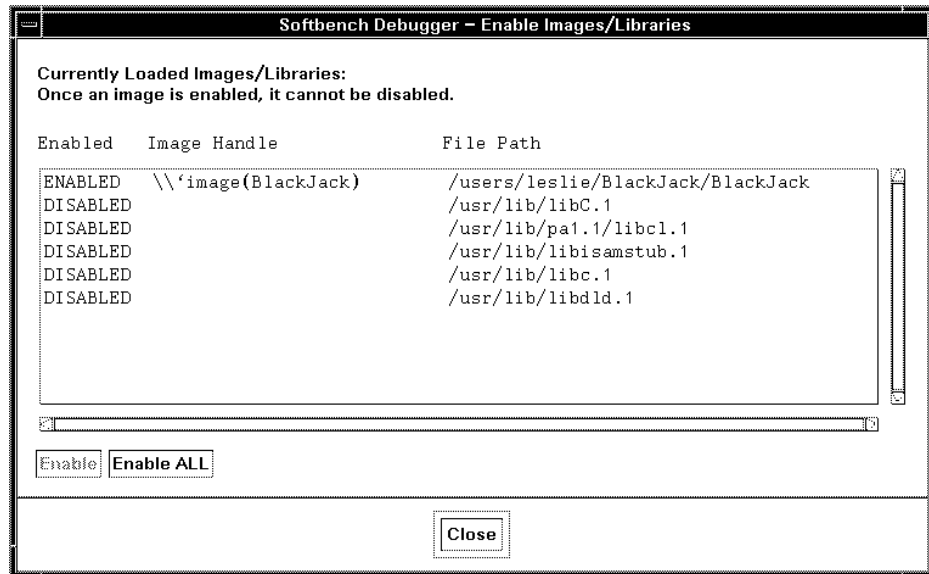
To enable libraries for debugging:

1. Choose "Execution: Enable Images/Libraries..." to display the known libraries.
2. Select one or more libraries from the "Images/Libraries" dialog box and select **Enable**. You may also double-click on a single item to enable it.

Selecting the **Enable All** button enables all libraries. Once you enable a library for debugging, it cannot be disabled without exiting SoftBench Debugger.

Figure 7-10

"Dynamic Images/Libraries" Dialog Box



## Viewing Assembly Language and CPU Registers

SoftBench Debugger is intended as a high-level (source language) debugging aid. However, sometimes it is useful to examine your program's behavior at the assembly language level. This can help you determine when you have misunderstood something about the way the compiler works.

You can examine the machine code produced by the compiler for your program and step your program at the assembly code level. You can examine processor and floating point registers and see how these change.

### Tracing Assembly Language

The "Show: **A**ssembly Instructions..." menu choice brings up the "Assembly Instructions" window. (See Figure 7-11.) This dialog box contains a scrollable view of the assembly language code for the current procedure of your program. The format of the code depends on your system. The window normally shows the following:

- Source line number
- Memory address
- Disassembly listing of the actual machine code

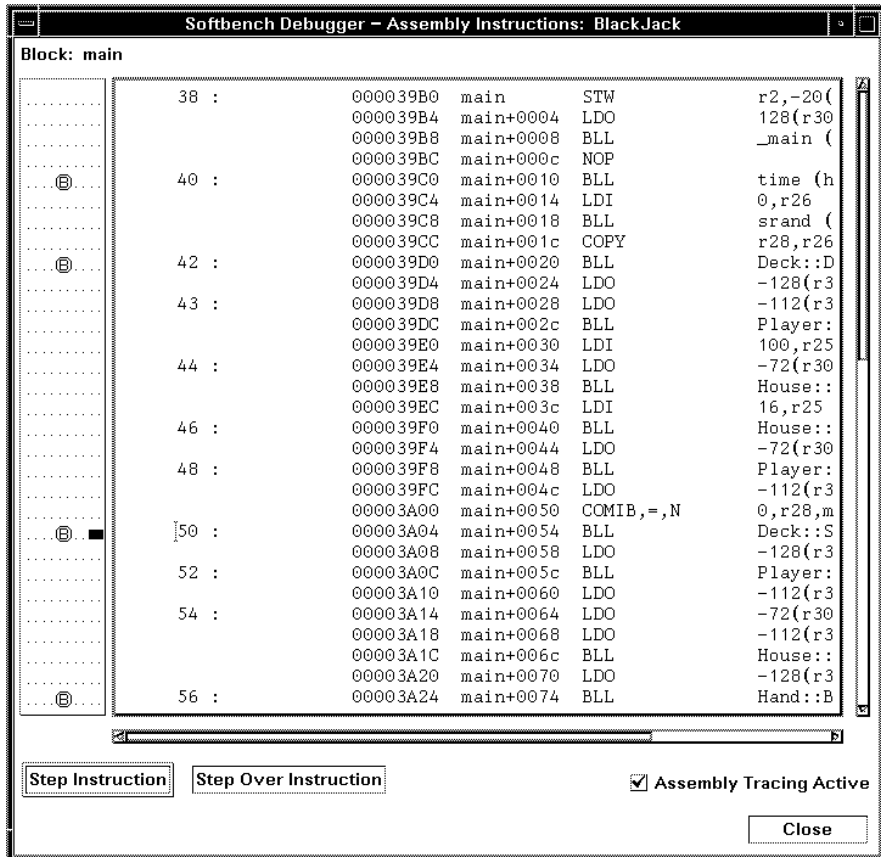
To set a breakpoint at the assembly level:

1. Select the address (in the "Assembly Instructions" window). This copies the address into the "()" input box.
2. Choose "Break: Set At Hex Address ( )".

You can also click the left mouse button in the Annotation Margin of the "Assembly Instructions" window, or press the right mouse button on the desired assembly line, to set or clear breakpoints.

Figure 7-11

"Assembly Instructions" Window



Select the **Step Instruction** button to step through your program one assembly instruction at a time. Select the **Step Over Instruction** button to step over a procedure call instruction.

The **Assembly Tracing Active** button enables updating of the Assembly Instructions window. Leaving assembly tracing disabled until you need it allows your program to run faster.

## Tracing Registers

Selecting one of the "show: Registers" submenu items brings up a window that shows the contents of a group of hardware registers. SoftBench Debugger highlights register values that change when you **Step** or **Continue** your program.

Choose the format used to display the register values with the "Display Format" button. The "Single" and "Double" buttons on the HP-PA Floating Point Registers display allow you to specify whether SoftBench Debugger displays the registers as two 4-byte values or one 8-byte value.

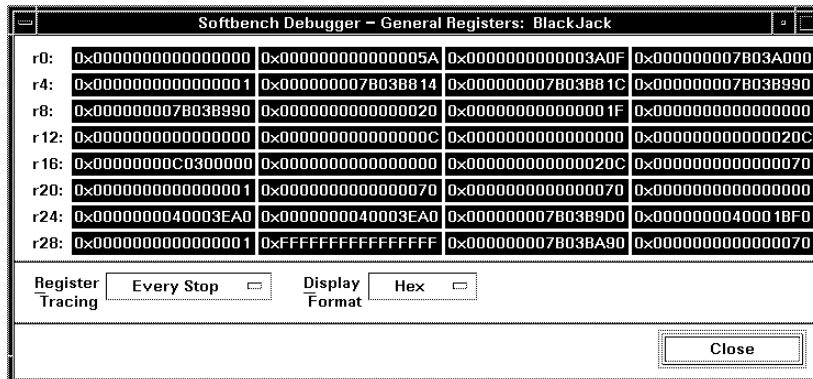
The "Register Tracing" option menu in the registers window specifies how often SoftBench Debugger updates the register values, just like a watchpoint's granularity. You can turn register updating off, or enable it as often as you need it (such as whenever the program stops or after every source statement). Leaving register updating disabled until you need it allows your program to run faster.

When you activate register tracing, SoftBench Debugger creates an implicit watchpoint to check the register values at the appropriate procedure entry, statement, and so on.

Figure 7-12 shows one of the "Show: Registers" windows.

**Figure 7-12**

**"Show Registers" Window (PA-RISC)**



## Handling Signals and Events

The UNIX operating system notifies a program of asynchronous events via signals. HP-UX also supports other "events" such as `exec` (program load) and `throw` (used in C++ exception handling). SoftBench Debugger handles these events using **intercepts**.

Intercepts are distinct from any signal handlers in your program. They intercept signals *before* they are sent to your program, and may either pass signals through to your program (to be handled by your signal handlers) or discard them.

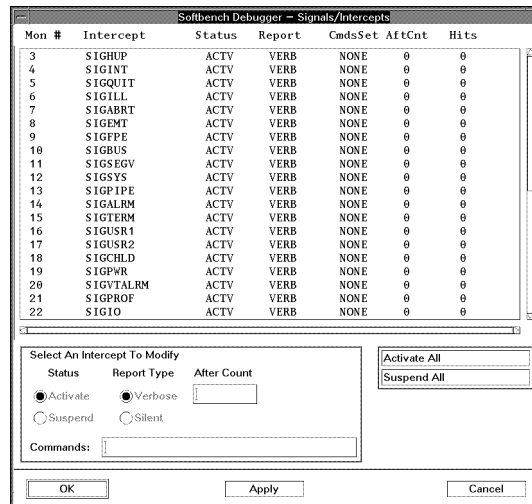
### Viewing and Editing Intercepts

"Execution: Signals/Intercepts..." brings up a dialog box with the current state of debugger signal and event handling, and allows you to control what SoftBench Debugger does with any intercept.

You can modify intercept handling only when SoftBench Debugger is in control. When your program runs, you must interrupt it (using **Interrupt Program**) or wait for it to hit a breakpoint before you can specify intercept handlers.

The scrollable list at the top of the "Intercepts" dialog box (see Figure 7-13) shows the current status of all available intercepts. This list shows all the attributes of each intercept, and also shows the "monitor number" (useful if you want to issue certain commands to DDE) and the number of times the intercept has been received. Note that the "monitor number" does *not* correspond to the UNIX signal number.

**Figure 7-13** "Intercepts" Dialog Box



SoftBench Debugger automatically creates intercepts for all signals and events. Signal intercepts are all active, and other intercepts are suspended. Highlight an intercept and select "Activate" to use it.

The buttons below the list allow you to modify the intercepts' attributes. You can control four attributes for each intercept:

**Status** Controls whether the intercept is currently Active. Select Suspend to deactivate an active intercept.

**Report Type** Controls whether SoftBench Debugger reports the intercept in the Debugger I/O Area (Verbose) or handles it Silently.

**Count** Specifies how many times the intercept should be ignored before SoftBench Debugger handles it.

**Commands** Specifies the DDE command string (if any) to execute when the intercept occurs. See "Help: DDE Reference" for a description of these commands.

When you highlight an intercept, you can change any of the above attributes and select **OK** or **Apply** to save the changes.

You can also Activate or Suspend *all* intercepts by choosing the buttons

on the right side of the dialog box.

SoftBench Debugger retains information (such as Count and Commands) for suspended intercepts. Selecting **Activate** reactivates the intercept with the same information it had before.

## Handling Signals

When your program receives a signal, you have the option of ignoring it or accepting it. If you ignore the signal, you can continue debugging your program as if no signal was received. If you accept the signal, the signal triggers the appropriate signal handler, if you have one defined. If you have no signal handler defined, the signal may cause your program to abort.

When the signal arrives, your program pauses. A message appears in the Debugger Output Area to tell you what signal has been received. If you **Step** or **Continue**, SoftBench Debugger delivers the signal to your program.

To ignore the signal, choose either "Execution: Step Operations → Step and Ignore Signal" or "Execution: Continue Operations → Continue and Ignore Signal".

If your program does its own signal handling for a specific signal (using *signal(2)*), you can set the status of that signal to:

- Status: Suspend
- Report type: Silent

This causes the signal to be passed through to your program.



---

## Debugging After a Program Fails (Core Dump)

SoftBench Debugger can diagnose some run-time errors in a program after the program has failed.

Some signals, if not caught by the program, cause programs to write a core file and to terminate. These signals are:

**Table 7-1**

**SoftBench CM Command Line Man Pages**

<b>Signal Name</b>	<b>Error Message Text</b>
SIGQUIT	quit
SIGILL	illegal instruction
SIGTRAP	trace trap
SIGIOT	abort
SIGEMT	emulation trap
SIGFPE	floating point exception
SIGBUS	bus error
SIGSEGV	segmentation violation
SIGSYS	bad argument to system call

When HP-UX detects one of these errors, it writes a `core` file in the directory where the process was executing (if it has write permission). This directory does not necessarily contain the executable file; it is the current working directory of the executing process. This `core` file contains information about the state of the process when it terminated.

### Debugging with a Core File

Choose "File: Load Corefile..." to debug a core file. Enter the core file name (usually `core`) and the name of the executable file that generated the core file.

When SoftBench Debugger first loads your program, SoftBench

Debugger sets the PC to the line that caused the core dump. The stack and all variables appear as they were when the program was about to execute that line. Here are some useful SoftBench Debugger actions that help find problems:

- Examine the line in the Source File Area that was responsible for the core dump to look for an obvious problem.
- Enter a local or global variable in the "( )" input box, then select **Print ( )** to examine its value. Check for a value out of range.
- Choose "show: Stack..." to look at the procedure call stack. Check values of passed parameters to see if the problem originated earlier in your program.
- Choose "show: Assembly Instructions...", and examine CPU registers or assembly instructions to get a low-level view of your program (see "Viewing Assembly Language and CPU Registers" on page 211).

You can often determine your program's problem using this information without actually executing the program again. If that's not possible, you can use these hints to set up breakpoints and tracing, and execute your program under SoftBench Debugger control.

## Debugging Forked Processes

Some applications use the `fork(2)` command to create a child process. When a program calls `fork`, a second process is created that is an exact "clone" of the first process. The only difference between the two processes is the return value of the `fork` call; this tells each process whether it is the "parent" or "child" process.

Since the `fork` call results in two processes, you must decide which process or processes you want to debug:

- |              |   |
|--------------|---|
| Debug Parent | Your existing debugger session stays attached to the parent process. The child process continues from the <code>fork</code> call. |
| Debug Child  | Your existing debugger session attaches to the child process. The parent process continues from the <code>fork</code> call.       |
| Debug Both   | <i>Both</i> processes are debugged. SoftBench spawns a second debugger to debug the new process.                                  |

Choose one of the above options under "Options: Fork Behavior ...". This selection affects all *future* `fork` calls. See "Attaching the Debugger to a Running Program" on page 222 if your program has already forked and the process you want to debug is not loaded in the debugger.

`vfork(2)` operates slightly differently. `vfork` is a "light-weight" fork. Most programs `exec` a new executable in the child process immediately after returning from the `fork`. When `fork` "clones" the parent process, it copies the parent's data space, and the `exec` immediately discards it. This can be very inefficient if the parent has a large data space.

Instead of creating an exact copy of the parent process, `vfork` creates a child process that *shares* the parent's data space. The parent is effectively suspended, and the child is treated as a thread of the parent, until the child `execs` another program or `exits`. If the child calls `exec`, it is then treated as a new process, and the debug behavior above takes effect.

## Debugging Threaded Applications

Threads are "mini-processes"—multiple "threads" of execution within a single process. Each thread has its own execution stack, but shares global variables with other threads in the process. There may be an arbitrary number of threads in a process, numbered 1 through  $N$ . (DDE assigns this thread number arbitrarily.)

### Viewing and Manipulating Threads

Choose "Execution: Threads..." to display a list of the threads in your program. The "Threads" dialog box displays status information about each thread and allows you to perform certain actions on a thread.

Highlight one or more threads and select a button to request the following actions:

<b>Disable</b>	Remove the selected thread(s) from the list of threads that can run when the program resumes.
<b>Enable</b>	Add the selected thread(s) to the list of threads that can run when the program resumes.
<b>Kill</b>	Kill the selected thread(s).
<b>Stack...</b>	Display the selected thread's execution stack in a new window.
<b>-&gt;Next To Run</b>	Make the selected thread the current <i>execution</i> thread. This thread runs first when the program resumes.
<b>Examine</b>	Change environment to the selected thread. This allows you to examine the status of a thread without changing the current execution thread.

You can use the "Threads" dialog box and the "Stack View" window to monitor the execution of threaded programs. SoftBench Debugger indicates the currently executing thread, which is analogous to the "PC Location," by a "->" arrow in the "Threads" dialog box. The "Stack View" window indicates the thread being examined, which is analogous to the "Current Location".

You can examine other threads by selecting them in the "Threads" dialog

box. You can also display a new "Stack View" window for a specific thread by selecting it in the "Threads" dialog box and selecting **Stack...**

## Setting Breakpoints on Threads

You can activate breakpoints on all threads, or on any selected set of threads.

Choose "**B**reak: Show..." to display the "Breakpoints Listing" dialog box. If your program has multiple threads, the dialog box contains a thread area.

Select " Break on All Threads" if you want the breakpoint to apply to all threads. Select one or more thread numbers if you want the breakpoint to apply to specific threads.

## Attaching the Debugger to a Running Program

SoftBench Debugger allows you to debug a process that is already running. For example, suppose you have a program that runs continuously in the background. Occasionally it does something wrong. Stopping the process and re-starting it in SoftBench Debugger can lose valuable state information. You can instead attach SoftBench Debugger to the process after it has encountered its error, and examine the data structures to learn the cause of the behavior.

To debug the running program:

1. Start your program. Do whatever you normally do to get it to the desired state.
  2. Choose **File: Debug Running Process...**.
  3. Enter the full path name to your executable file in the "Executable" input box.
  4. Enter the process ID in the "Process ID (pid)" input box, or select **Find Matching Processes** to have the debugger find the process for you. If SoftBench Debugger finds only one matching process, it automatically places the Process ID (pid) in the Process ID field. If SoftBench Debugger finds more than one matching process, the debugger displays all matching processes. Select the desired process from the list.
  5. Select **OK**.
  6. Your program stops, and the PC Location display indicates the value of the Program Counter (PC). The PC arrow might not be visible in the Annotation Margin, depending on whether or not the PC is in a debuggable procedure.
  7. Use SoftBench Debugger to examine the program. All SoftBench Debugger facilities are available for adopted processes. For example, if the process pauses in a system library, you could move down in the stack until you encounter a function in your code. You could then examine the local variables in that function to determine how the library function was called.
- When you have finished debugging the process, you can release it so it continues running normally by choosing **File: Free Running Process**.

## Debugging C++ Programs

If you have installed C++ SoftBench, the following features support debugging of C++ programs (see also “Setting C++ Breakpoints” on page 197):

**C++ name handling:** SoftBench Debugger lets you debug using your actual C++ variable and function names, without worrying about the underlying naming system. This means that you can highlight C++ text in the source view window and act upon it using SoftBench Debugger commands. The highlighted text becomes the "(") entry for use in your debugging sessions.

**Overloaded functions and operators:** SoftBench Debugger displays ambiguous overloaded functions or operators along with their arguments. You resolve the ambiguity by selecting the appropriate function. You can also set breakpoints at all overloaded functions or operators specified with a given name using a single command.

**C++ scope rules:** SoftBench Debugger conforms to C++ scope rules by allowing access to identifiers either directly from within its scope or by means of the C++ scoping operator ("::") from outside its scope.

**C++ data types:** SoftBench Debugger provides full support for C++ constant types, enumeration types, anonymous unions, and type conversions.

**Classes and objects:** You can view simple or extended versions of class information. That is, you have the choice of whether to display inheritance members with the extended version. The function and data members of a class can be accessed with dot ("."), arrow ("->"), and scope ("::") operators.

**Class commands:** SoftBench Debugger provides powerful commands which allow you to access all members of a class. You can choose to list or set breakpoints at all member functions of a class with a single command.

**Member functions:** Breakpoints can be set on specific member functions. You can also call a member function directly from the "( )" input box.

**Object identification:** SoftBench Debugger recognizes whether a specified object pointer is a pointer to a declared class or a specific derived class.

**Instance breakpoints:** You can set a breakpoint at a member function for a particular instance of a class. This reduces the number of member function breakpoints SoftBench Debugger encounters and reduces the time it takes to debug.

**Nested classes:** Provide the ability for command-line references to static members and functions of a nested class to resolve properly.

**Print:** Prints current data member values for C++ classes and class inheritance hierarchies.

**Exception handling** Provides these features:

- Ability to set breakpoints immediately prior to any exception throw.
- Notification of pending throw.
- Ability to set breakpoints in catch clauses.
- Notification of catch.
- Single step from throw into catch clause.

**Parameterized types (Templates):** Provides these features:

- Allow references to a class-template or class-template expansion.
- Set breakpoints in any or all class-template member functions.
- Allow references to a function-template or function-template expansion.
- Set breakpoints at any location in a function-template.
- Set breakpoints at any location in an expansion of a function-template.
- Print the definition of any class template or expansion.



## Using Breakpoints for Exception Handling

HP-UX provides support for debugging C++ exceptions. You can enable intercepts on the "throw" and "catch" events. "Throw" intercepts occur at the time a C++ exception is detected, and "catch" intercepts occur where the exception is handled.

See "Handling Signals and Events" on page 214 for a description of intercepts.

## Accessing Inherited C++ Values

C++ objects can inherit variables and member functions from other classes. SoftBench Debugger allows you to specify whether inherited member functions and variables should be included in certain operations.

Choosing "Options: C++ Settings..." displays the C++ "Options" dialog box. The settings in this dialog box specify how many "levels" of inheritance should be affected by the respective operations.

The breakpoint, trace, and watchpoint settings affect inherited members in classes, instances, and templates. If you select "■ All", the monitor affects all members in an object regardless of the source of inheritance. If you select "■ Count", only the specified number of "levels" are affected.

For example, if you set the count to 0, the monitor affects only the first "level" of inheritance. This means SoftBench Debugger sets the monitor only on local (non-inherited) member functions or variables. Setting the count to 1 means the monitor affects the object's local functions or variables *and* functions or variables inherited from one level of "ancestor" objects are affected.

You can use the "Print" setting when printing an object. If you select "■ All", SoftBench Debugger prints all component variables, regardless of the source of inheritance. If you set the count to 1, SoftBench Debugger prints the object's local variables and any variables inherited from one level of "ancestor".

## Debugging Static Constructors

When you load a program for debugging, SoftBench Debugger begins executing it and pauses at the first line of `main()`. This is ideal for most debugging situations.

However, if your program contains static objects, the constructors for those objects must be invoked *before* `main()`. By the time the debugger pauses in `main()`, the constructors have already completed.

If you need to debug these constructors, you must tell DDE to pause before calling them. This can be accomplished by executing the DDE command `"property system -on"`. When you set `prop sys`, DDE pauses at the very first assembly instruction of the program. You can then step into the constructors. You should either place the `prop sys -on` command in a `.dderc` file, or issue the command through the "Debugger Input" input box and rerun your program.

When you pause at the first instruction, DDE does not yet realize you are executing a C++ program. This can result in any C++ breakpoints being lost. If you want to retain your C++ breakpoints across executions, execute the DDE command `"prop lang c++"` before rerunning your program. This gives DDE the information it needs to retain your breakpoints.

When you finish debugging your static constructors, issue `"prop sys -off"` and `"prop lang default"` commands. This returns DDE to normal debugging mode and uses less CPU resources.

## Debugging Optimized Code

Source-level debugging of unoptimized code is relatively easy because you have a simple correspondence between source code statements and the assembly code instructions into which they are translated. Also, program variables are stored in memory and are therefore easy to access.

Optimization performs a series of transformations on the object code in order to make the program run faster. **In effect, optimization transforms a program into a different program.** The executable program you debug is actually not the same program as the source program. In addition, program variables may be stored in registers instead of memory and are therefore more difficult to access.

Ordinarily, you first compile and debug your program without optimization. All or nearly all of the bugs in your program show up in the unoptimized version. After eliminating all the bugs that you can find, turn on optimization (compile with `-O`). If the program behaves incorrectly, scan the source code for the most common kinds of bugs that appear for the first time in optimized code:

- Uninitialized variables
- Out-of-bounds array references
- Variable references based on the assumption that two variables are adjacent in memory

SoftBench CodeAdvisor may assist you with some of these categories of problems. Others require more extensive code examination.

These kinds of problems, however, are often very difficult to find by examining the source code. If you cannot determine the reason for the program's misbehavior, you need to debug the optimized code. For tutorial and task-oriented information on how to debug optimized code using the debugger, see SoftBench Online Help.

## Customizing SoftBench Debugger

SoftBench Debugger is very flexible, and you can customize it to meet your needs. You can make most user interface customizations in SoftBench Debugger by choosing the menu selections under "Options". Other, less common customizations can be made in the "\$HOME/.softbench/softbenchrc" file. See the *softdebug(1)* reference page for a description of all SoftBench Debugger resources.

### Specifying Debugger Options

The "Options" menu offers a variety of ways to customize SoftBench Debugger behavior. SoftBench Debugger saves customizations made on the "Options" menu and in "File: Add Source Directories..." automatically. See SoftBench Online Help for details about customizing SoftBench Debugger.

Customizations available under the "Options" menu include:

- Causing the underlying debugger to run on another system
- Specifying default program environment (arguments, I/O redirection, and environment variables) used for program loads and reruns
- Modifying breakpoint, watchpoint, and trace behavior for C++ (see also "Debugging C++ Programs" on page 223)
- Changing the behavior of the Stack and Watchpoints displays
- Specifying the debugger's behavior when `fork()` calls are encountered
- Selecting the edit mode used in the "Debugger Input" input box
- Selecting the language-sensitive text selection behavior
- Specifying the node width in Data Graph Window
- Specifying whether the Current Environment follows the edit cursor, or associates with the source location currently displayed
- Changing the buttons displayed on the front panel.  
This option also allows you to change the the popup menus displayed when you click the right mouse button on the Source File Area or the "Assembly Instructions" source area. See "Customizing User Buttons" on page 228.

### Customizing User Buttons

Several command areas can be changed to fit your specific needs. These

areas are:

- The buttons displayed on the SoftBench Debugger front panel.
- The popup menus displayed when you click the right mouse button on the Source File Area and the "Assembly Instructions" source area.

To modify these buttons:

1. Choose "Options: User Configurable Buttons...". SoftBench Debugger displays the "User Configurable Buttons" dialog box, showing the buttons configured for one of the areas.
2. Select the radio button corresponding to the button set you want to change.
3. To delete a button, highlight its line and select **Delete**.
4. To change a button, highlight its line, make the desired changes, and select **Replace**.
5. To add a button, enter its label and definition in the appropriate input boxes, select the button you want it added before or after, and select **Add Before** or **Add After**.
6. Select **OK** when you finish. SoftBench Debugger displays the new buttons the next time it displays the corresponding area. SoftBench Debugger automatically saves the button definitions.

Buttons issue one of three different kinds of commands:

Op	An internal SoftBench Debugger operation. The <b>Operation Information...</b> descriptions cover these.
Raw Command	A command to be sent directly to the underlying debugger. See " <b>Help: DDE Reference</b> " for a description of DDE commands.
Message	A SoftBench <code>Request</code> message. Any legal SoftBench <code>Request</code> message can be sent. For example, you could define a button to send an <code>EDIT</code> message to SoftBench Program Editor.

The **Operation Information...**, **Raw Information...**, and **Message Information** buttons display a description of the different button types. **Macro Information** explains the various macros (such as "``()`" for the contents of the "`()`" input box) that can be used in button definitions.

When SoftBench Debugger changes to a new context, it looks for the button configuration file in two locations, in this order:

## Using SoftBench Debugger

### Customizing SoftBench Debugger

1. `$HOME/.softbench/debugui.buttons`
2. `install_root/config/softdebug/$LANG/debugui.buttons`

---

## If Something Goes Wrong

Table 7-2

Condition	Explanation
A request to the debugger failed. The request message was:	DDE could not process a request from SoftBench Debugger. If the solution is not obvious, see "Help: DDE Reference" to determine what went wrong.
The following operations were flushed:	Operations that had been queued up needed to be flushed for some reason. The error message shows which operations were discarded.
The asterisk ("*") is invalid as a filename. You MUST provide a filename to softdebug in order to run it.	Some SoftBench tools allow a "*" as the context file, but SoftBench Debugger does not. You must provide a valid filename.
No processes were found.	A "File: Debug Running Process ..." operation failed because no process was found with the name given.
No such user operation <i>op</i> defined.	A user-defined button was created with an invalid operation. Find and repair the faulty button definition using "Options: User Configurable Buttons...".
The button label from <i>filename</i> at line <i>linenum</i> is empty.	The button definition at line <i>linenum</i> of file <i>filename</i> has no label. Add a label in the "Button Label" field.
The button kind (or location) " <i>string</i> " is invalid from <i>filename</i> at line <i>linenum</i> .	These messages appear if an invalid button kind (MSG, RAW, OP) or location (Undefined, Frontpanel, Source, Assembly) was found in the button definition file <i>filename</i> .

**Table 7-2**

<b>Condition</b>	<b>Explanation</b>
<code>operation: invalid relative index: <i>index</i>. Must be a valid positive (or negative) integer.</code>	<b>This operation requires a valid integer offset argument. The <code>UserOpEnvGotoBosRelative</code> and <code>UserOpEnvGotoTosRelative</code> operations only accept positive integers. Check the button definition.</b>
<code>Unable to open the button save file.</code>	<b>SoftBench Debugger cannot create the button definition file. Check to make sure you have write permission in the <code>\$HOME/.softbench</code> directory.</b>
<code>Warning: operation performed in most recent stack activation of <i>procedure</i>.</code>	<b>In certain unusual situations involving recursively-called procedures, SoftBench Debugger may switch from a deeper invocation of a recursive procedure to the most recent invocation. This may result in a print operation, for example, printing values from a different invocation than you expected. This message warns you that this has happened so that you can switch back to the earlier invocation if desired.</b>
<code>Unable to open log file:</code>	<b>SoftBench Debugger cannot open the log file you specified. Check that the filename is a valid location, and that you have write permission on the directory or file.</b>
<code>The item <i>name</i> is not the first line of an aggregate (struct/array). Select a line that indicates the start of an aggregate and try again.</code>	<b>The "collapse" operation in the Show Watchpoints dialog box must be applied to the first element of a collapsible aggregate structure.</b>



**Table 7-2**

<b>Condition</b>	<b>Explanation</b>
A token must be provided in the `()` area for the <i>type</i> request.	An operation from the "Static" menu was chosen without providing an argument in the "()" input box.
`.__vptr` cannot be followed.	The Follow and Follow Recursively operations in the Data Graph Window cannot follow the special variable <code>.__vptr</code> used by the C front CC compiler. This is an internal variable and cannot be examined.
`.__vfp` cannot be followed.	The Follow and Follow Recursively operations in the Data Graph Window cannot follow the special variable <code>.__vfp</code> used by the aCC C++ compiler. This is an internal C++ variable and cannot be examined.
<i>variable</i> cannot be followed. It is not a pointer, or it is a pointer to a string or a function.	The Follow operations in the Data Graph Window cannot expand pointers to strings or functions.
Can't perform operation while one is in progress.	You requested a "Show: Data Graph Indirect()" or "Show: Data Graph()" while another such operation was pending. Wait for the first operation to complete and try again.

## For More Information

- On DDE commands, command syntax, and advanced DDE features, see *HP/DDE Debugger Online User's Guide*. (choose "Help: DDE Reference").
- On any button, screen area, or menu item, use SoftBench Online Help. Move the mouse pointer to any menu selection and press F1.
- On starting SoftBench creating projects and compiling and building projects and targets, see Chapter 2, "Using SoftBench," on page 41.
- On SoftBench editing, see Chapter 4, "Using SoftBench Editors," on page 117.
- On make and Makefiles, see *make(1)* in the manual pages.
- On compiler options, see *cc(1)*, *CC(1)*, *aCC(1)*, or *f77(1)*, as appropriate for your language. Also see the HP-UX Language Reference for your compiler.
- On C, C++, or FORTRAN expression syntax, see the appropriate language reference.
- On standard I/O, see *stdio(3S)* and related UNIX manual pages.
- On writing an X11-compatible program with a window interface, see *Programming With the Xt Intrinsics*, and other X11 documentation.
- On core files, see the reference page for *core(4)*.
- On signals, see the manual pages for *signal(2)*, *sigvector(2)*, *bsdproc(2)*, *sigset(2v)*, and *signal(5)*.
- On register usage and Program Status Word letter codes, see a book on Assembly Language Programming for your computer, and the compiler manual for the language you are using (for example, *HP-UX C Language Reference*).
- On the hardware configuration of your computer, see the list on your screen at boot time.

# 8 **Using SoftBench Debugger Data Graph Window**

SoftBench Debugger Data Graph Window helps you visually navigate your complicated data structures. For basic graph operations such as saving the graph or selecting and moving graph objects, see Appendix A, “Using SoftBench Graph Windows,” on page 323.

## Starting and Stopping the Data Graph Window

### Beginning a Browsing Session

You can begin a browsing session with the Data Graph Window from SoftBench Debugger as follows:

1. Enter a variable or an expression in the "( )" input box.
2. Choose "Show: Data Graph ( )" or "Show: Data Graph Indirect ( )".

You can iconify SoftBench Debugger after the Data Graph Window appears.

### Stopping a Browsing Session

To end the browsing session, select the **Close** button. The **Close** button deletes all information gathered during the session. To print the graph see "Using Save Options for the Graph Image" on page 326.

## Understanding Data Graph Window Areas

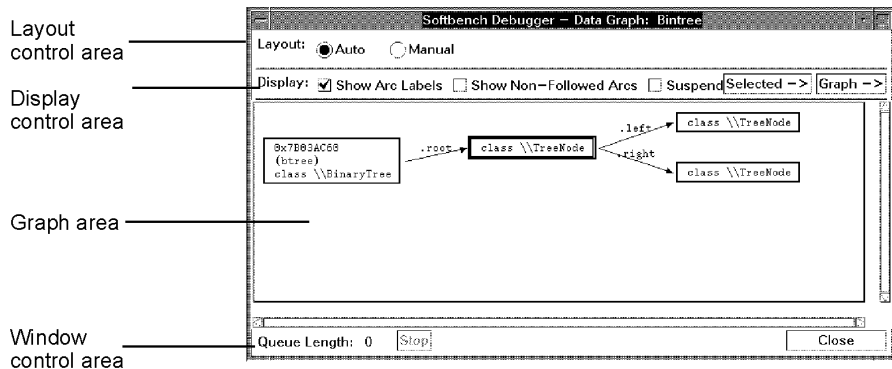
The Data Graph Window is a top-level window that you can iconify when not in use.

The Data Graph Window is divided into four main areas:

- the Layout Control Area
- the Display Control Area
- the Graph Area, which includes:
  - **Nodes** corresponding to data in memory
  - **Arcs** corresponding to pointer references from one structure to another
- the Window Control Area

Figure 8-1 shows an example of Graph Window Areas in the Data Graph Window.

**Figure 8-1** Data Graph Window Areas



## Understanding the Layout Control Area

The Layout Control Area at the top of the browser window includes radio buttons, which control the layout of the graph. For more information on the layout conventions, see Appendix A, "Using SoftBench Graph Windows," on page 323.

## Understanding the Display Control Area

The Display Control Area includes toggle buttons and push buttons which control the display of information on the Graph Area.

Button	Displayed Information
■ Show Arc Labels	When selected, the arc labels indicate the pointer names.
■ Show Non-Followed Arcs	When selected, SoftBench Debugger displays all arcs. When deselected, SoftBench Debugger only displays arcs generated by <code>Follow</code> commands.
■ Suspend	When selected, the Data Graph Window does not update its display. This reduces overhead on the debugging session when graph operations are not needed. When deselected, the Data Graph Window updates its display whenever the user program returns control to SoftBench Debugger.
<b>Selected -&gt;</b>	Pops up the "Node Actions" menu for the selected node or nodes. SoftBench Debugger greys out the button if you have no nodes selected. This menu can also be popped up by clicking the right mouse button on the Graph Area when you have selected one or more nodes.
<b>Graph -&gt;</b>	Pops up the "Graph Actions" menu. This menu can also be popped up by clicking the right mouse button on the Graph Area when you have no nodes selected.

## Understanding the Graph Area

The Graph Area displays nodes and arcs. You can also display the "Node Values" dialog box using the "Node Actions" menu. Select a node, then select **Selected ->** or click the right mouse button on the Graph Area, to

display the popup menu. Select "Show Node Values".

The Graph Area can be manipulated using operations in popup menus. Different popup menus appear depending on whether you have selected a node or not. Table 8-1 shows the popup menus used in the Data Graph Window.

**Table 8-1 Data Graph Window Action Menus**

Object Selected	Actions Allowed
Any Node	"Node Actions" Menu <ul style="list-style-type: none"> <li>• Follow All</li> <li>• Follow All Recursively</li> <li>• Show Node Values...</li> <li>• Watch Node...</li> <li>• Cast Node Type...</li> <li>• Hide Selection</li> </ul>
Nothing	"Graph Actions" Menu <ul style="list-style-type: none"> <li>• Zoom In</li> <li>• Zoom Out</li> <li>• Node Width...</li> <li>• Save Image...</li> <li>• Clear Graph</li> </ul>

For more information on basic operations common to all graph windows, see "Using Graph Window Areas" on page 326.

### Reading Nodes

Nodes represent data in memory. The node label indicates the particular data type. Nodes displayed on the graph as the result of "Show: Data Graph ( )" or "Show: Data Graph Indirect ( )" have two extra lines at the top of the label. one shows the memory address of the expression or variable, and one shows the expression or variable that was in the "( )" input box.

### Reading Arcs

Arcs represent pointers from one structure to another. The arc label is the name of the pointer, if named. You can toggle "■ Show Arc Labels" to display or hide the arc labels.

### Using Dialog Boxes

The popup menu selection "Show Node Values" enables the "Node Values" dialog box, which displays the data members (including arrays) within a particular structure and their values.

### Understanding the Window Control Area

The Window Control Area at the bottom of the Data Graph Window includes "Queue Length" information and push buttons:

- |              |   |
|--------------|---|
| Queue Length | The Queue Length status indicates how many internal graph operations are pending. This gives you feedback about how your request is proceeding. Be aware that this number can increase and decrease as new operations are added to and removed from the queue.  |
| Stop         | <b>Stop</b> is active when the number of operations in the queue is greater than one (otherwise greyed out). As SoftBench Debugger processes each operation, it checks to see if this button has been pressed. When selected, SoftBench Debugger deletes the pending operations, halts the operation in progress, and displays partial results in the Graph Area. |
| Close        | When selected, the Data Graph Window closes. The Escape key is bound to this button. The <b>Close</b> button on the window manager frame also closes the window.<br><br>When you close the Data Graph Window, SoftBench Debugger discards all the graphed data.   |



## Using the Graph Area

The Graph Area helps you understand your program by creating a visual image of the data structures in memory and controlling the information displayed.

### Displaying New Nodes

You add new nodes in the Graph Area by either of the following methods:

- Entering an expression or variable in the "(" input box and choosing "Show: Data Graph ( )" or "Show: Data Graph Indirect ( )".
- Selecting a node in the Graph Area and selecting "Follow All" or "Follow All Recursively". SoftBench Debugger displays the dereferenced pointers as new nodes, and connects them by arcs. You can select these new nodes and follow them.

"Follow All Recursively" recursively follows pointers in each new node. In order to avoid uncontrolled growth of the graph, this operation follows only pointers with the *same name* as the pointers in the node you originally specified. For example, if you "Follow All Recursively" on a structure that contains pointers to several large data structures, the Data Graph Window does not trace down every location pointed to by all pointers within the structure. It displays the immediate children and stops. However, suppose one of the child structures is the root of a binary tree, each node of which contains left and right pointers. Choosing "Follow All Recursively" on any tree node displays the entire tree below that node, since all nodes contain the same pointer names (left and right) as the original node.

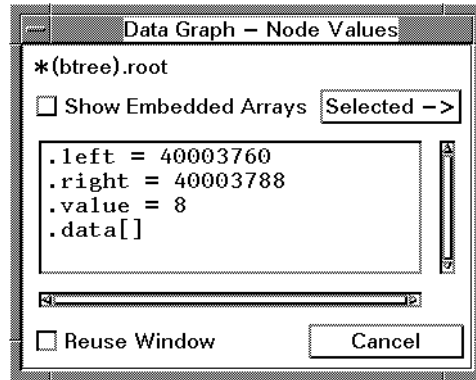
You can access these operations by selecting the **Selected ->** menu button, or by clicking the right mouse button on the Graph Area to bring up a popup menu.

### Using the "Node Values" Dialog Box

You can view the values of the members within a particular structure by double clicking on the node or by selecting the node and selecting "Show Node"

Values". The "Node Values" dialog box displays a list of all data members and their values. (See Figure 8-2.) The list can be scrolled and the text selected and copied as with any other Edit Area list.

**Figure 8-2** "Node Values" Dialog Box



This dialog box includes two toggle buttons and one push button:

- | Button   | Action  |
|--|---|
| <input checked="" type="checkbox"/> Show Embedded Arrays | When selected, displays all the array element values for embedded arrays. Selecting this in one "Node Values" dialog box globally selects it in all other boxes.  |
| <input type="checkbox"/> Reuse Window                    | When selected, the next "Show Node Values" selection displays the new node values in the existing "Node Values" dialog box, overwriting the previous node values. |
| Selected ->  | Displays the "Values Actions" menu.   |

Selecting the **Selected ->** menu button, or clicking the right mouse button on the "Show Node Values" display area, brings up the "Values Actions" menu. This menu allows you to operate on selected values.

Table on page 243 shows the popup menu available in this dialog box.

**Table 8-2 Node Values Dialog Box Action Menu**

Object Selected	Actions Allowed
Data Member	Values Actions Menu <ul style="list-style-type: none"> <li>• Follow</li> <li>• Follow Recursively</li> <li>• Set Value...</li> <li>• Show On Graph</li> <li>• Hide On Graph</li> </ul>

### Displaying Data Members

You can display a particular member value or values on the labels of nodes in the Graph Area by selecting the member in the "Node Values" dialog box and selecting "Show On Graph". "Hide On Graph" removes the selected value or values.

### Setting Data Member Values

Selecting "Set Values..." invokes the "Set Values" dialog box, which enables you to set a data member value. A one-line editable area contains the selected member's address as "*expression*". Enter the desired value to the right of the = and select **OK** to complete the assignment.

### Following Selected Pointers

"Follow" and "Follow Recursively" perform the same operation as on the Graph Area. It is useful to follow the pointers selecting the "Show Node Values" dialog box if the object contains many pointers and you want to follow only a few of them.

SoftBench Debugger displays the special C++ variable `._vptr` (used with `aCC`) or `._vfp` (used with `CC`) in the "Node Values" dialog box if the object has virtual functions. However, this virtual pointer cannot be followed.

## Suspending Graph Updates

Selecting the "■ Suspend" toggle button suspends updating of the Graph Area as you step through your program, which can improve debugging performance. As soon as you select "■ Suspend", popup menu actions become limited to those which manipulate the existing graph, such as "Save Image..." and "Hide Selection". Deselecting "■ Suspend" brings the Graph Area and dialog boxes up-to-date, and activates all menu items.

## Deactivating the Graph

SoftBench Debugger deactivates the Graph Area whenever you unload the debugged program from SoftBench Debugger or the debugged process executes an exit statement. Again, popup menu actions become limited to those which manipulate the existing graph.

## Stopping a Graph Process

To stop an undesired or lengthy graph action, select the **Stop** button. The nodes processed prior to the stop remain displayed.

## Sample Use Models

This section introduces you to several Data Graph Window concepts useful in verifying the correct or expected operation of your program's data structures. By setting strategic breakpoints and selecting the pointer dereferencing commands, you can quickly visualize what happens to your data structures during program execution. Using the Data Graph Window increases your program understanding and decrease your defect isolation time.

The following scenario assumes you have a defect in your program, which contains binary trees implemented with pointers. These use models provide scenarios for:

- Visually picturing the data structures, by graphing the overall connections within a large interconnected data structure.
- Displaying values of data members within the structures represented by nodes on the Data Graph Window.

The source for the example used in this section is found in `/opt/softbench/examples/bintree/bintree.C`.

## Verifying Correct Data Structures

Suppose that your program has a binary tree defined by the variable `btree`. At some breakpoint during execution of the program, assume you want to see the state of the binary tree.

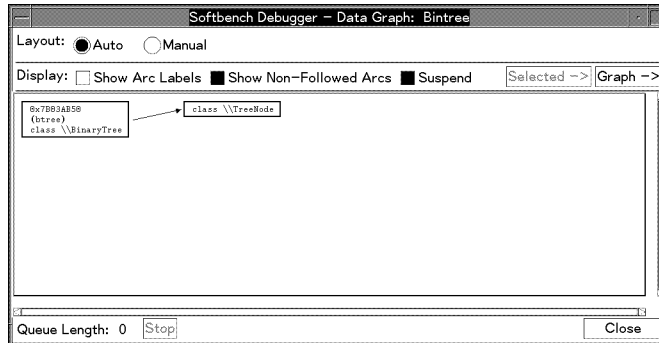
### Viewing `btree`

To see the binary tree in the Data Graph Window, you would enter `btree` in the "(" input box and choose "show: Data Graph ( )". The Data Graph Window opens, displaying a single node representing the `btree` variable. The node is labeled `(btree)` with the type class `\\BinaryTree`.

To follow all the pointers in `btree`, you would select the node and select "Follow All" from the "Selected" menu. SoftBench Debugger dereferences all non-NULL pointers in `btree` and adds new nodes to the Data Graph Window for each pointer. In this model, one node is added of type class `TreeNode`.

If you enabled "Show Arc Labels", you could see the name of the data member pointer, `root`. Figure 8-3 shows what the graph would look like at this point in the execution of this program.

**Figure 8-3** Binary Tree Node

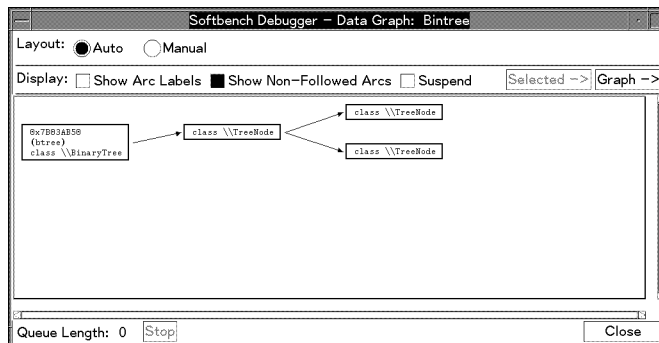


### Continuing Program Execution

Now select the `\\TreeNode` node. Select the **Selected ->** menu button, or click the right mouse button, to display the "Node Actions" menu. Select "Follow All". This causes all pointers in this node to be displayed.

**Continue** execution from the main SoftBench Debugger toolface until the next breakpoint, which is set after the insertion of another tree node. After another **Continue** a third tree node is added. You might see something like Figure 8-4.

**Figure 8-4** Binary Tree Node with Children

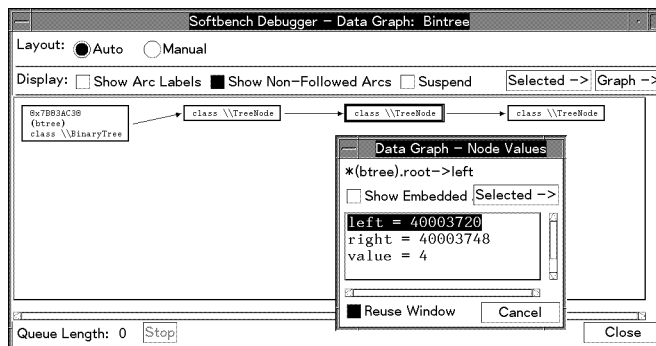


## Showing Nodes

Assume that at this point in your program you expect yet another pointer to be added. But your next **Continue** does not add another node. The reason is because "Follow All" dereferences pointers only one level deep. To follow all pointers to the end of the list, you could select the previous tree node and select "Follow All Recursively" from the "Selected" menu. This displays all nodes from that node forward.

Some structures may have many pointers that you do not want to follow. You could instead choose to dereference only particular pointers. You would first select the node of interest and select "Show Node Values" from the "Selected" menu. The "Node Values" dialog box appears, listing each data member and value. You would then select the pointer data member of interest from the list, click the right mouse button on the "Node Values" dialog box to display the "Values Actions" popup, and select "Follow Recursively". Only pointers with that name would be recursively followed. Figure 8-5 shows a recursive dereferencing of all data member pointers named left.

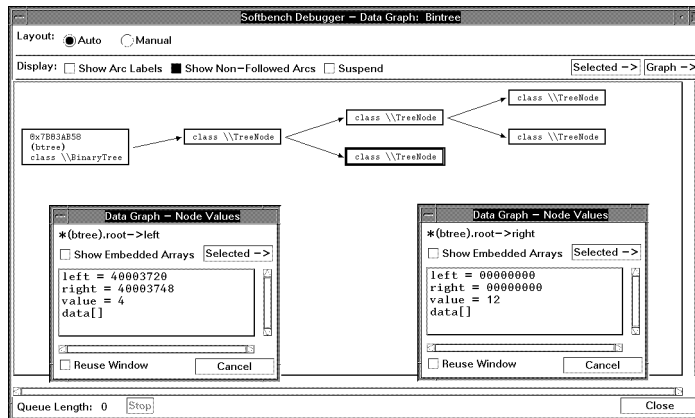
**Figure 8-5** Dereferencing Particular Data Members



## Viewing Values of Data Members

Sometimes you might need to compare the data member values associated with different nodes. You can display multiple "Node Values" dialog boxes by turning off "Reuse Window". The next time you select "Show Node Values" from the "Selected" menu, SoftBench Debugger displays a new dialog box (see Figure 8-6).

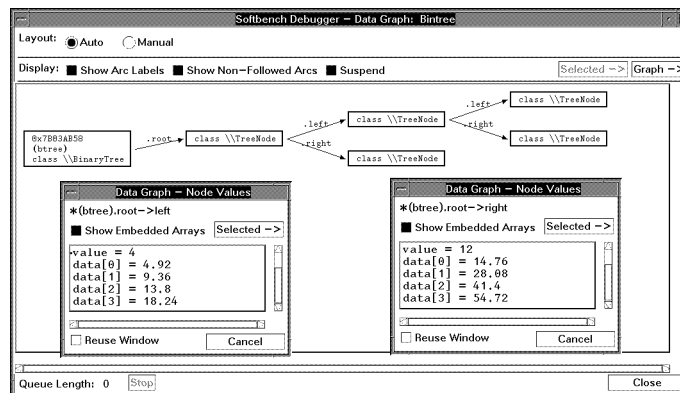
**Figure 8-6** Viewing Values from Multiple Nodes



### Viewing Embedded Arrays

Selecting "■ Show Embedded Arrays" on the "Node Values" dialog box displays the values of elements of arrays embedded in class or struct types. If you are tracking down a problem which might relate to an embedded array, you must turn on "■ Show Embedded Arrays" to view the element values. Figure 8-7 shows two "Node Values" dialog boxes with "■ Show Embedded Arrays" selected.

**Figure 8-7** Viewing Values from Embedded Arrays



### Showing Values On the Graph

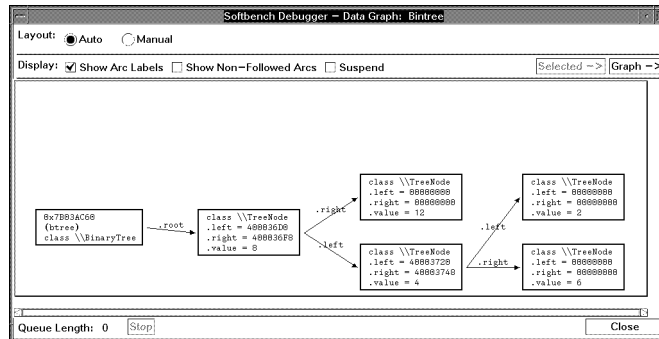
When you want to view the values of many nodes, select the values of



interest and select "Show On Graph" from the **Selected->** menu button on the "Data Graph - Node Values" dialog box. This command labels each selected node with specific data member names and values.

You could transform the prior graph by selecting data members `left`, `right`, and `key`, and selecting "Show On Graph". Any node with a data member matching one of these names adds the matching data member and value to the label, as shown in Figure 8-8.

**Figure 8-8** Viewing Data Members on The Graph Nodes



## **For More Information**

- On any button, screen area, or menu item, use SoftBench Online Help. Move the mouse pointer to any menu selection and press **F1**.
- On basic operations like clearing, saving, and viewing the Graph Area, see Appendix A, “Using SoftBench Graph Windows,” on page 323.

# 9

## Using SoftBench Static Analyzer

SoftBench Static Analyzer aids in the understanding of moderate to large-sized programs during the implementation, test, and maintenance phases of the software lifecycle.

New members of a software team can use SoftBench Static Analyzer to understand the code quickly. Programmers can become more confident about the impact of their software changes using the added information it provides.

SoftBench Static Analyzer helps you better understand your programs by answering questions like:

- What code will be affected if I change this parameter list?
- What functions and classes call this function?
- Where is this identifier's value modified?
- What code accesses any element of this class?
- Where are all the calls to a specific overloaded definition of this C++ member function?

SoftBench Static Analyzer helps you analyze program information such as call trees, class hierarchy, file include relationships, and variable definition and use. SoftBench Static Analyzer presents the information in a text list (on the main window) or with a graphical display of relationships (accessed from the "Graph" menu). You can immediately edit the referenced source file by double-clicking on a text list item or graphical node.

SoftBench Static Analyzer works with code for HP C, C++ and FORTRAN77 language compilers.

## Starting SoftBench Static Analyzer

You can start SoftBench Static Analyzer from the main SoftBench window, from your configured editor, or from SoftBench Debugger. In the main SoftBench window, select the "Static Analyzer" icon in the toolbar, or choose "File: Static Analysis...". SoftBench Static Analyzer starts, displaying all your project's functions. SoftBench Static Analyzer focuses its **queries** on the current project.

Once you start SoftBench Static Analyzer, you can explore the project's source code with queries on the "Show" menu, or with specific "Identifier" queries on the "Symbol" menu. Commands on the "Show" menu display all symbols in the project for a given category. For example, you can show all functions, all classes, and all global variables.

Commands on the "Symbol" menu require an identifier in the "Symbol ()" input box. You can type a symbol name directly in the "Symbol ()" input box, or you can select a list item from a previous query which automatically copies the relevant symbol into the input box. For example, you select a specific global variable, then choose "Symbol: References ()" to display all references to the selected global variable. When you execute a Static query from the configured editor or SoftBench Debugger, select the symbol of interest, then choose a command from the "Static" menu. The "Static" menu in the editor and SoftBench Debugger equates to the "Symbol" menu in SoftBench Static Analyzer.

Within SoftBench Static Analyzer you can choose between textual and graphical queries.

### Textual Queries

The main SoftBench Static Analyzer window provides facilities for making textual query requests, displaying the results in a list, and browsing the results in the editor.

### Graphical Queries

The Static Graphs provide facilities for making query requests, displaying the results graphically, and browsing the results in the editor.

Textual queries via the "Show" and "Symbol" menus are effective while you are learning about the identifiers, such as functions and global variables, used in your code. Graphical queries are effective when you

are learning about relationships between elements in your code. See Chapter 10, “Using Static Graphs,” on page 283 for more information on the Static Graph.

SoftBench Class Graph/Editor provides a graphical way to edit your C++ programs. You can create, delete, and modify C++ classes, inheritance relationships, member functions, and data members. See Chapter 5, “Using SoftBench Class Graph/Editor,” on page 133 for information on SoftBench Class Graph/Editor.

SoftBench Static Analyzer brings up your configured editor to display the queried source code. Using either the textual or graphical view, you can access the associated source file by double-clicking on a list item or graphical node.

## Preparing to Make Queries

To use SoftBench Static Analyzer effectively you must first understand some simple preparation concepts:

- Generating Static Data
- Specifying What Data to Analyze

### Generating Static Data

Before you can use SoftBench Static Analyzer for program understanding, you must first generate Static information about your application by parsing code and creating a **Static database**. The best way to generate the Static information is from the SoftBench main window:

1. Set the "■ Static" compile mode toggle button. This action adds the "-y" compile flag to the build options.
2. Select the project or target of interest in the project browser or target graph.
3. Select **Build**.
4. When the build completes successfully, select the "Static Analyzer" icon from the toolbar. SoftBench Static Analyzer appears with all your program's functions displayed in the query results area.

If your program contains compile errors, the compiler may not be able to generate complete Static information. The compiler analyzes your program as well as it can and generates Static information based on that analysis. This may result in incomplete or missing information. For example, a serious compile error may result in all Static information for a function being discarded. In a less serious case, the compiler may discard the Static information for a specific identifier. You can still use SoftBench Static Analyzer with this incomplete information, but SoftBench Static Analyzer cannot display complete information in these situations. Correct the compile errors and recompile your program to ensure SoftBench Static Analyzer has complete information.

Static databases from previous releases of SoftBench are not compatible with the current Static database format. You must remove and regenerate your static database.

## Updating Static Data without Building

Once the Static database is created, SoftBench has the knowledge to update Static data, whether you build the project from SoftBench or analyze the files from SoftBench Static Analyzer.

To successfully generate Static data without rebuilding the project, choose "File: Analyze File Set".

## Specifying Static Data to Analyze

SoftBench Static Analyzer performs **queries** on source files in your program. It gathers this information from a **Static database** (the `Static.sadb` file), which your compiler created when you set the "■ Static" compile mode toggle button before a build.

SoftBench Static Analyzer allows you to expand the set of files on which queries are based by choosing "File: Customize File Set". When you work with SoftBench projects, you can expand the focus of your queries to include subprojects and parent projects of the current project. When you run SoftBench Static Analyzer directly from the command line, you can expand the focus of your queries by manually adding directories and files to the **analysis file set**.

## Using the Default Analysis File Set

In most cases, SoftBench Static Analyzer properly determines what files to analyze. You do not need to take any special actions. Simply build your project with the "■ Static" compile mode toggle button set, and the build process creates the **Static database**. SoftBench Static Analyzer then reads the database and allows you to make queries on your program.

By default, SoftBench Static Analyzer uses the current project's Static database. The database contains the information on all source files used to create the targets in the project.

This simple model of building a project and analyzing the code in the project is the recommended way to use SoftBench Static Analyzer

## Including Subprojects and Parent Projects

If you need to make queries focused on more than just the current project, you can expand the Static file set to include subprojects and parent projects:

1. Choose "**File: Customize File Set...**" to display the "Customize File Set" dialog box.
2. Set the appropriate toggle buttons to include the current project, its subprojects, and its parent projects in the analysis file set. When you set the toggle buttons, the associated projects are added to the "Projects in File Set" list. When you deselect toggle buttons, associated projects are removed from the "Projects in File Set" list. You can also remove projects from the list by selecting specific projects and pressing the **Delete** button.

Using these toggle buttons, you can even eliminate the current project from the **analysis file set**, focusing your queries entirely on parent projects and/or subprojects.

3. Select **OK** to confirm your changes and close the dialog box.

Once you make these changes, SoftBench Static Analyzer opens the appropriate **Static databases** and future queries use the new set of analysis information. SoftBench Static Analyzer saves the toggle button settings for parent projects and subprojects automatically when you exit SoftBench Static Analyzer.

### Customizing the File Set in Non-Project Mode

When you start SoftBench Static Analyzer directly from the command line, SoftBench Static Analyzer does not have access to project information. Therefore, you must customize the **analysis file set** manually. In this mode, when you choose "**File: Customize File Set...**", SoftBench Static Analyzer posts a different dialog box in which you can add directories and files. SoftBench Static Analyzer then analyzes those files and includes that information in the analysis file set.

### Keeping Analysis Files Current

The files used by SoftBench Static Analyzer must be kept current to provide accurate results. When you modify your source files, rebuild the project. A successful build automatically updates the static information. Subsequent queries use the information in the updated **Static database**.

You can also make sure all static information is current by choosing "**File: Analyze File Set**". This command scans all source files in the current project and updates any out-of-date static information.



## Filtering Queries by File Set

Your programs often include files, such as system header files, that you may not want to include in your **queries**. By default, SoftBench Static Analyzer includes these files in any queries. You can restrict your queries to check only those files in your **analysis file set** by setting the "■ Filter Results Using Fileset" toggle button, located on the dialog box posted by "Options: Behavior Settings...".

## Determining the File Set Status

SoftBench Static Analyzer shows the state of the **analysis file set** in the "File Set Status" field. Possible values are:

None	No analysis file set has been specified, or the default file set is empty. This status usually indicates that static data has not been generated.
Closed	The analysis file set is available but the associated database files could not be found or opened.
Open	The analysis file set is available and the associated database files are current with respect to the source files used to create them.
Open/Out-of-Date	The analysis file set is available but one or more source files are newer than the information in their associated database files. Choose <b>Build</b> in the main SoftBench window to rebuild the current project. Alternatively, choose "File: Analyze File Set" to update the database files without rebuilding.
Updating	An update is in progress because you chose "File: Analyze File Set" and the process has not yet completed.

## Using SoftBench Static Analyzer Window Areas

Figure 9-1 shows the SoftBench Static Analyzer window. The primary window areas include:

**Menu Bar** Provides access to most SoftBench Static Analyzer functions. Refer to SoftBench Online Help for detailed descriptions of any menu command.

**File Set Information Area** Displays the current file set and its status.

The "File Set" field can contain any combination of the following values:

- Project
- Subprojects
- Parent Projects
- Current working directory (in non-project, standalone mode)

The "File Set Status" field shows the current state of the analysis file set. Possible values include:

- None
- Closed
- Open
- Open/Out-of-Date
- Updating

See "Specifying Static Data to Analyze" on page 255 for more information.

**"Symbol ( )" Input Box** Provides a text area to enter the identifier for "Symbol" menu commands. You must enter an identifier before choosing menu items on the "Symbol" menu.

**■ Scoping" Toggle Button** Enables you to select whether scoping is used in a "Symbol" menu command. When SoftBench Static Analyzer uses scoping, it uses the **scope** of the identifier for the next **query**. Otherwise, SoftBench Static Analyzer displays all symbols that match the identifier.

For more information on scoping, see "Using Scoping" on page 268.

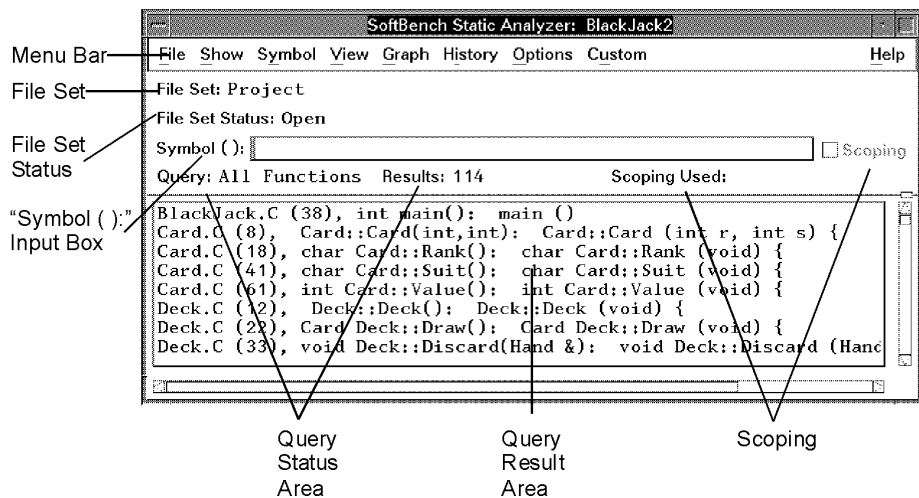
**Query Information Area** Displays what **query** was run last (the "Query" field), how many results are currently displayed (the "Results" field), and what level of scoping was used (the "Scoping Used" field).

Possible values for "Scoping Used" include:

- None
- File &Line
- File Only
- *empty* (the area is blank)

**Query Results Area** Lists the results of the latest **query** executed from the "Show" or "Symbol" menus or the last query chosen from the "History" menu. Query results include file names, line numbers, function names, and associated program text for each query result. Double-clicking on a result causes SoftBench to load the file in your configured editor with the cursor positioned at the location of the query result.

**Figure 9-1** SoftBench Static Analyzer Window and Areas



## Making Textual Static Queries

After you have generated Static information, you can make **queries** on your application. SoftBench Static Analyzer supports both textual and graphic queries on your application. You can execute Static queries in three ways:

- Choose a command from the "Show" menu.
- Enter a program identifier in the "Symbol ()" input box, then choose a related command from the "Symbol" menu.
- Select a line in the Query Results Area, then choose a related command from the "Symbol" menu.

See Chapter 10, "Using Static Graphs," on page 283 for a description of graphical queries.

## Making General Queries

The "Show" menu commands allow you to perform general queries on your entire application. For example, you can find all the source files, global variables, or classes in your application.

- To display all source files in the current **analysis file set**, choose "Show: Source Files".
- To display the global variables in your application, choose "Show: Global Variables".
- To display all C++ classes, choose "Show: Classes".
- To display all functions, choose "Show: Functions".

## Making Queries Based on a Program Identifier

The "Symbol" menu allows textual queries about specific symbols such as a specific class, function, or variable. "Symbol" queries end with " ()" and return information about an identifier. The identifier must be entered in the "Symbol ()" input box. You can either type directly into the input box or select text from the Query Results Area.

You can access the "Symbol" menu commands from your configured editor or SoftBench Debugger as well. In these other tools, select the

symbol of interest in the source code, then choose a command from that tool's "Static" menu.

### Selecting Text

Selecting text from the Query Results Area records not only the text, but the filename, line number, and column number that locates the text. SoftBench Static Analyzer uses this location information in queries when "Scoping" is selected. (See "Using Scoping" on page 268 later in this chapter.) If you type an identifier into the "Symbol ()" input box, SoftBench Static Analyzer has no location information available. The scoping toggle button is turned off and cannot be set.

Select text from the Query Results Area in one of the following ways:

- Select a line in the Query Results Area by single clicking the left mouse button. SoftBench Static Analyzer copies the identifier associated with that query result to the "Symbol ()" input box.
- Drag from the beginning of an identifier to the end. Use the right mouse button in the Query Results Area (since the left mouse button selects the entire line). The identifier highlights as you drag. Press **Control-Insert** to copy the text to the clipboard. Move focus to the "Symbol ()" input box and press **Shift-Insert** to paste the text into the input box.

This method of selection does not work for fully qualified C++ identifier names. The double-click action does not extend past a hyphen or double colon. For example, the C++ name `Picture::Picture` cannot be selected using this method.

The contents of the "Symbol ()" input box must be an identifier name, except when executing "Symbol (): Pattern Match ()". SoftBench Static Analyzer ignores leading and trailing blanks and tabs.

### Understanding the Types of References

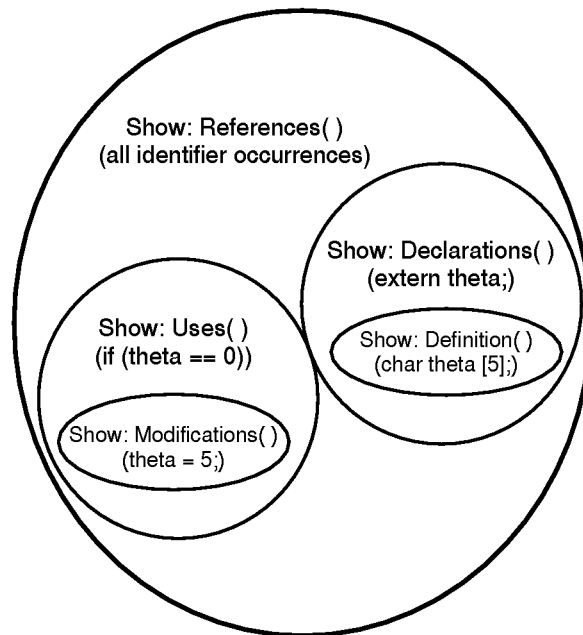
The "Symbol" menu provides several queries about where identifiers occur in your program. The "References" command provides the most complete list. The indentation on the "Symbol" menu indicates the related subsets of queries:

References	Lists all occurrences of the identifier. Encompasses all instances found in any of the remaining categories.
------------	--

Declarations	Lists all occurrences of an identifier that announce properties about it.
Definition	Lists the single declaration of an identifier that causes storage to be allocated.
Uses	Lists occurrences where the program uses, modifies, initializes, or calls an identifier's value.
Modifications	Lists occurrences where the program modifies or initializes an identifier's value.

Figure 9-2 shows examples of the kinds of occurrences that are returned by the respective queries. Notice that several types of occurrences are actually subsets of other types. For example, "Symbol: Uses ()" includes modifications.

**Figure 9-2 Relationships among Reference Queries**



### Determining Identifier Classification

To display the classification (such as variable or function) of an identifier:

1. Enter the identifier into the "Symbol ()" input box.

2. Choose "Symbol: Classification ()".

### Troubleshooting Invalid Identifiers for Queries

SoftBench Static Analyzer returns an "Identifier not found" error for **queries** on identifiers that are not present in its database.

Possible causes for this error include:

1. A source file has been edited to add new identifiers and has not been re-analyzed. Save the changes and either rebuild the project or choose "File: Analyze File Set".
2. A source file contained compile errors, and the compiler was not able to deduce information about the identifier. Correct the compile errors and rebuild or re-analyze the application.

In both cases you could also choose "Symbol: Pattern Match ()" to do a regular expression search for the identifier.

### Using Pattern Matching

You can use pattern matching when you don't know the exact name of the identifier for which you are looking. You can search for a pattern and examine the list of results for an item that meets your needs. You can also search for literal strings or identifiers within comments.

To match a pattern in the source files:

1. Enter the desired pattern in the "Symbol ()" input box. The pattern entered can contain any regular expression or shell expression, depending on which option is set. To set the pattern matching options, choose "Options: Identifier Matching Rules..".
2. Choose "Symbol: Pattern Match ()".

All occurrences of the expression are displayed in the Query Result Area. **Scoping** has no effect.

## Using Query Results

SoftBench Static Analyzer **query** results are displayed in a textual list using the main Static window. There are many ways to use these query results. One common use scenario includes:

1. Browse the SoftBench Static Analyzer query results.
2. Edit the source file at the query result location.
3. Update the **Static database** by rebuilding your application.
4. Perform a SoftBench Static Analyzer query with new Static data.

### Browsing the Query Result

The SoftBench Static Analyzer query result can be browsed by double clicking on a line in the Query Results Area. This action invokes the configured SoftBench editor and positions the cursor at the line where the query result appears in the source file.

### Editing the Source File

After you browse the query result and display the source, you are ready to modify your source file. Make the changes to the source file and save the changes. Notice that the "File Set Status" changes to "Open/Out-of-Date" in SoftBench Static Analyzer.

### Updating the Database

Once you have made the changes to your source files, you are ready to generate updated Static information. The recommended way to update Static information is by building the project:

- From the editor, choose **File: Build Project**.
- Alternatively, from the main SoftBench window, select the project, then select **Build**.

If you do not want to rebuild, choose **File: Analyze File Set** in SoftBench Static Analyzer. (See "Updating Static Data without Building" on page 255.)



After a successful build or analysis of the file set, the "File Set Status" changes to "Open".

## **Performing a Query**

After rebuilding your application you are ready to analyze the changed program with SoftBench Static Analyzer. You can perform the same Static query or initiate new queries. From the main Static window use the "Show" and "Symbol" menus to execute queries.

---

## Simplifying Query Results

SoftBench Static Analyzer provides filtering mechanisms that allow you to focus only on the **query** results in which you are interested. These filters reduce the chance of information overload. SoftBench Static Analyzer also provides specific filters for graphical queries, which are described in “Simplifying Graph Displays” on page 292.

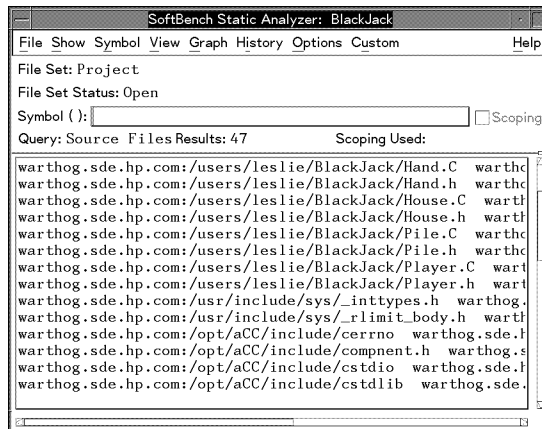
**Table 9-1**      **SoftBench Static Analyzer Display Options**

Command	Description
"View: Display Results" submenu	Controls what data attributes are displayed in the Query Results Area.
"View: Sort Results" submenu	Allows you to sort results by file, result, or attribute.
"View: Filter Results..."	Provides numerous toggle button settings for filtering C++ data. (See “Filtering C++ Query Results” on page 268.)
"Options: Behavior Settings..."	Provides access to the filter that limits queries to the analysis file set.

### Filtering Results Using the File Set

SoftBench Static Analyzer provides a filter to focus queries only on the code in your project. For example, there are many instances when you are not interested in code libraries developed by other teams, or the identifiers declared in the `/usr/include` directory. Choose "show: source Files" to display the source files SoftBench Static Analyzer uses for query results. Without a filter, this query produces a list similar to the one displayed in Figure 9-3. By default, SoftBench Static Analyzer reports results from all files that were compiled, including header files.

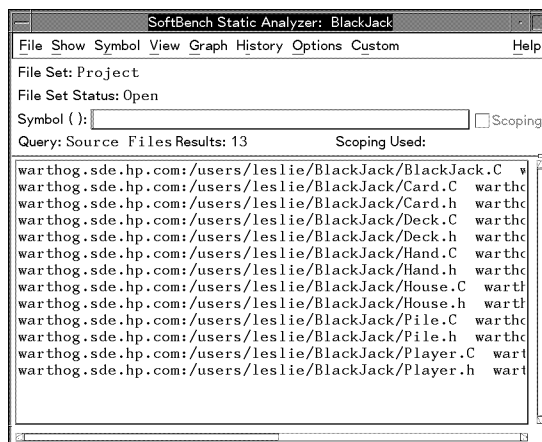
**Figure 9-3 Unfiltered Source File List**



To limit queries to only the files in your **analysis file set**, choose "Options: Behavior Settings...". In the resulting dialog box, set the "■ Filter Results Using Fileset" toggle button.

To display the filtered list of source files, choose "Show: Source Files" again. Figure 9-4 displays the result of this filter.

**Figure 9-4 Filtered Source File List**



Once you enable this filter, SoftBench Static Analyzer reports results based only on the projects specified in the analysis file set. For more information on analysis file sets, see "Specifying Static Data to Analyze" on page 255. The filter applies to all future queries.

## Filtering C++ Query Results

In many cases, the query results of interest are a subset of the default query results that SoftBench Static Analyzer provides. The "view: Filter Results..." command provides a way to generate this subset of query results. These filters are available in C++ SoftBench only.

For example, if you are investigating the interface of a class, you are interested only in public member functions of that class. At this point you would not be interested in viewing the private and protected member functions.

To filter the private and protected member functions:

1. Choose "view: Filter Results...", which displays the "Query Filters" dialog box.
2. Turn off the "■ Protected Class Members" and "■ Private Class Members" toggle buttons. Be sure that "■ Public Class Members" is set.
3. Select **OK**.

All query results are immediately updated so that private and protected members are not displayed. SoftBench Static Analyzer retains filtered results, but does not display them.

## Using Scoping

When you select a program identifier from the Query Results Area, SoftBench Static Analyzer automatically puts it into the "Symbol ()" input box and sets the "■ Scoping" toggle button. After you select a **query** and get results, the "Scoping Used" field indicates that the identifier's location (filename, column, and line number) within the source file was used to display results only on that exact identifier in the program. The label also indicates what type of scoping was used. For example, if you select an identifier `count` defined in function A, only references to `count` in function A are displayed. Variables named `count` in other functions, or in the global scope, are not displayed.

If the location information should not be used, then deselect the "■ Scoping" toggle button. SoftBench Static Analyzer then uses only the name in the "Symbol ()" input box and returns the results for all identifiers with that name.

Scoping is also used with queries made from Static Graphs. Scoping

information is used for queries made on selected nodes or arcs. Queries resulting from the **Display** operation are not scoped. See Chapter 10, "Using Static Graphs," on page 283 for information on Static Graphs.

### Scoping Prerequisites and Constraints

In order to use scoping, you must select the entry in the "Symbol ( )" input box from the Query Results Area, your configured SoftBench editor, or SoftBench Debugger. Entering an identifier from the keyboard precludes the use of scoping information, since SoftBench Static Analyzer has no location information available.

### Troubleshooting Scoping Information

When you select an identifier from a file that has been modified, the location (scoping) information might be incorrect. When the "Symbol" menu queries are executed, SoftBench Static Analyzer automatically ignores first the column and line number and then the filename from the scoping information until it locates a matching identifier. The result may be a partially scoped or non-scoped query.

### Scoping Example

Consider the following example:

```
1  int theta;
2
3
4  main ()
5  {
6      int x;
7      char *theta;
8
9      theta = "Hello World%d\n";
10     printf(theta,2);
11 }
12
13 foo(int theta)
14 {
15     return(theta*2);
16 }
17 }
```

If you select the `theta` identifier on line 9 (not typed into the "Symbol

## Using SoftBench Static Analyzer

### Simplifying Query Results

(`)` input box) and execute a `"Symbol: References ()"` command with scoping. SoftBench Static Analyzer displays the following:

```
file.c (7), main: char *theta
file.c (9), main: theta = "Hello World%d\n";
file.c (10), main: printf(theta,2);
```

The "Scoping Used" value is "File & Line".

If you turn off scoping and perform the same query again, SoftBench Static Analyzer displays the following:

```
file.c (1), Global: int theta;
file.c (7), main: char *theta
file.c (9), main: theta = "Hello World%d\n";
file.c (10), main: printf(theta,2);
file.c (13), foo: foo(int theta);
file.c (16), foo: return(theta*2);
```

SoftBench Static Analyzer displays all occurrences of `theta`, in all scopes. The "Scoping Used" value is None.

## Redisplaying Past Queries

Each time SoftBench Static Analyzer executes a successful "Show" or "Symbol" menu command, SoftBench Static Analyzer places the **query** name in the "History" menu. From the "History" menu, the query result can be reviewed later, printed, saved to disk, or deleted.

Note that SoftBench Static Analyzer saves the query *result*, not the query itself. Any changes that have happened since the original query (such as filters being added or removed, or new source being analyzed) are not reflected in the saved query result.

The "History" menu does not contain the results of unsuccessful queries. For example, if you enter an identifier in the "Symbol ()" input box that is not a function name and choose "Symbol: Function →: Parameters ()", an error message appears and the "Parameters *identifier*" query does not appear in the "History" list.

By default, the "History" menu contains the 10 most recent, successful queries. To change the number of queries saved, use the "Options: History Menu Size..." menu command. Valid values range from 1 to 1000.

If the number of entries in the "History" menu equals the maximum History Menu Size, and you perform a query, SoftBench Static Analyzer automatically deletes the oldest query in the "History" list.

## Redisplaying Query Results

To redisplay the results of a "Show" or "Symbol" command, choose "History: *query name* →: Query". This causes the results to be redisplayed in the Query Results Area. SoftBench Static Analyzer does not re-execute the query.

## Deleting a Query Result

To delete a query result, choose "History: *query name* →: Delete". SoftBench Static Analyzer removes the query from the "History" menu.

## Saving and Printing a Query Result

Choosing "History: *query name* →: Print..." enables you to save a

## Using SoftBench Static Analyzer

### Redisplaying Past Queries

description of the query and the query result in a file, or print to a printer. If you print the results, SoftBench Static Analyzer prompts you for the print command. If you save the results to a file, SoftBench Static Analyzer prompts you for a file name. This action does not remove the command from the "History" menu.



## Using SoftBench Static Analyzer in Standalone Mode

This section describes using SoftBench Static Analyzer from the command line. When you start SoftBench Static Analyzer from the command line, it has no knowledge of your project data. To access project data, start SoftBench Static Analyzer from the main SoftBench window or from the editor or SoftBench Debugger when they are started from the main SoftBench window.

### Generating Static Data from the Command Line

If you build your application outside of SoftBench, you need to use the "-y" compile option. This flag causes your compiler to generate a **Static database**, even when you build outside of SoftBench. To generate Static information without rebuilding the application, use "-y -nocode".

SoftBench Static Analyzer provides the ability to generate **Static data** by parsing your code without first compiling it. To generate Static data without compiling, choose "File: Analyze File Set" in SoftBench Static Analyzer. This data is not as reliable as compiled data, nor is SoftBench Static Analyzer as reliable in knowing how to generate this data.

### Searching Subdirectories

When you use SoftBench Static Analyzer in project-aware mode, SoftBench knows where to find the source files in the project. In contrast, in standalone mode, SoftBench Static Analyzer looks in the current working directory and in any other files and directories specified in the **analysis file set**.

Additionally, in standalone mode, SoftBench Static Analyzer provides a short cut for analyzing a multi-directory application. To recursively search all subdirectories below a specified directory, choose "Options: Behavior Settings..." and select " Recursively Search Subdirectories". This setting causes SoftBench Static Analyzer to recursively search all directories listed in the analysis file set.

## Using the Staticfileset File

The `Staticfileset` file, if it exists in the current directory, specifies the initial file set when SoftBench Static Analyzer starts in standalone mode. All files in `Staticfileset`, and all source files in the directories listed in `Staticfileset`, are loaded as the **analysis file set**.

A different `Staticfileset` file can be used if you choose "File: Restore File Set...".

The format for each entry in the `Staticfileset` file is the data hostname (optional) followed by the path name of the directory or file.

For example, a `Staticfileset` could contain these directories where your application files and `Static.sadb` files reside:

```
/users/static-A  
host2:/users/static-B
```

Including a directory automatically includes all files in the directory. To exclude some files in a directory, you must explicitly list each file to include, and you must not include an entry for the directory.

## Customizing SoftBench Static Analyzer

SoftBench Static Analyzer has many options that allow you to configure the tool for your unique environment and working style:

- Use the "View" menu to make changes to the display of data.
- Use the "Options" menu to change the behavior of SoftBench Static Analyzer.
- Choose "Project: Modify Properties.." in the main SoftBench window to change the location of the Static database.

All SoftBench Static Analyzer queries rely on the information stored in the `Static.sadb` file. If your project encompasses source code from several directories, SoftBench stores all Static information in the project's `Static.sadb` file.

By default, SoftBench stores the **Static database** in the project data for a project that uses **project build**. It stores the Static database in the build directory for a project that uses **external build**.

You should not move the Static database file unless you have unsupported customization needs. To return to the default location for the Static database, clear the "Static database" text field.

- Use the "View" and "Options" menus on the Static Graph to customize the graphs. See Chapter 10, "Using Static Graphs," on page 283.

See SoftBench Online Help for explanations of each menu command. When you customize the appearance and behavior, SoftBench Static Analyzer automatically saves the changes.

When you make changes to the the display of data ("View" menu), and behavior of SoftBench Static Analyzer ("Options" menu), SoftBench Static Analyzer automatically saves the changes.

---

## If Something Goes Wrong

Table 9-2

Condition or Message	Explanation
No static analysis information after compilation.	You did not specify the "■ Static" compile mode during the build. See "Generating Static Data" on page 254 in this chapter. Alternatively, your PATH is incorrect. See "Prerequisites to Using SoftBench" on page 42.
The currently defined file set is empty (contains no source files or database files)...	Either you have specified the wrong <b>Analysis file set</b> (choose "File: Customize File Set..." or "File: Display Files in File Set...") or you did not compile for static analysis. Change compiler settings and recompile.
No analysis database files were found for the current file set...	Either you have specified the wrong <b>file set</b> (choose "File: Customize File Set..." or "File: Display Files in File Set...") or you did not compile for static analysis. Change compiler settings and re-compile.
Analysis file set is not Open.	A query was attempted when no Static Analysis database files were open. Choose "File: Customize File Set..." and select the current project, or for non-project mode, specify a file set.

**Table 9-2**

<b>Condition or Message</b>	<b>Explanation</b>
<p>You have more queries in the "History" menu than fit on your screen.</p>	<ul style="list-style-type: none"> <li>• The "History" menu size is too large. Remove unneeded "History" entries by choosing "History: <i>query name</i> →: Delete" to reduce the size of the list.</li> <li>• Choose "Options: History Menu Size..." and specify a smaller size. Only the last <i>n</i> entries are kept (where <i>n</i> is the new "History Menu Size").</li> </ul>
<p>Some of the files were not found (see Output window for list).</p>	<p>A file in the file set could not be accessed. Check the files specified (do they exist, are the permissions set correctly?), and try again.</p>
<p>Out of Memory.</p>	<p>You ran out of swap space or your process size wasn't allowed to grow any larger. Your system administrator may have to:</p> <ul style="list-style-type: none"> <li>• Add swap space</li> <li>• Reconfigure your kernel</li> </ul>
<p>The identifier that you selected does not appear in the "Symbol ()" input box.</p>	<p>If you were selecting an identifier from the Query Result Area, you may not have used the <i>right</i> mouse button to click or select the identifier.</p>
<p>The identifier that you typed does not appear in the "Symbol ()" input box.</p>	<p>The mouse cursor must be in the "Symbol ()" input box for it to accept keyboard input. Position the cursor, and try again.</p>

**Table 9-2**

<b>Condition or Message</b>	<b>Explanation</b>
source line xxx unavailable.	The results of this command include source file text, but the specified line in the source file is not available, probably due to recent editing. From the SoftBench Static Analyzer window, choose "File: Analyze File Set" or rebuild the program from the Project Window to update the <b>Static database</b> . Then repeat the "Show" command.
Node does not appear on graph or in query results.	The identifier in the "Symbol ()" input box is not a variable, function, class, template type, or file. Compile errors in your source may have prevented the compiler from generating Static information about the identifier. Correct the errors and rebuild.
The "Symbol" command is listing results from the entire file set.	Turn <b>scoping</b> on or, if out-of-date, recompile.
The "Symbol" commands are not listing results from the entire file set.	Turn <b>scoping</b> off.

**Table 9-2**

<b>Condition or Message</b>	<b>Explanation</b>
Source file <i>filename</i> not found.	The source file you requested cannot be found. Choose "File: Customize File Set..." to add the project (Parent or Subproject) containing the source file which couldn't be found. Alternatively, add the file to the project by choosing "Project: Add File(s) to Project..." and rebuild the project. This may also be an NFS access problem. Your system administrator may need to add access to the remote system that contains the file.
Unable to open Static.sadb.	SoftBench Static Analyzer cannot open the Static database. It may be an incompatible version from a previous release of SoftBench. Delete Static.sadb, rebuild the code and try again.
Update of Static Analysis Database failed: Scanner exited (result==1).	Either your Static Analysis Database is from an earlier SoftBench release, in which case you must remove the database and regenerate the analysis database files by rebuilding your project (In standalone mode, choose "File: Analyze File Set"); or, you requested "File: Analyze File Set" when running from SoftBench before creating your Static database through a build process. You must first create the database by building your project.

**Table 9-2**

<b>Condition or Message</b>	<b>Explanation</b>
Duplicate instances of identifier appear in query results.	Your Static database may contain data from source files that have been removed. Remove the database and rebuild the code and repeat the query. Alternatively, use <code>staticrmfile</code> to remove data on specific files. See <i>staticrmfile(1)</i> .
Symbols that you have removed from your code appear in query results.	Your Static database may contain data from source files that have been removed. Remove the database and rebuild the code and repeat the query. Alternatively, use <code>staticrmfile</code> to remove data on specific files. See <i>staticrmfile(1)</i> .



## For More Information

- On Static Graphs, see Chapter 10, “Using Static Graphs,” on page 283.
- On Graph Window features for Static Graph and all SoftBench tools, see Appendix A, “Using SoftBench Graph Windows,” on page 323.
- On `softstatic` command line options and X resources such as `autoload`, see the *softstatic(1)* manual page, available by choosing **Help: Show Man Page**.
- On customizing the SoftBench Static Analyzer user interface, choose SoftBench's **Help: Overview** and select **Customizing SoftBench**.

Using SoftBench Static Analyzer  
**For More Information**

# 10 Using Static Graphs

SoftBench Static Analyzer includes four Static graphs to help you visualize your program structure.

Call Graph	Displays call relationships between functions.
SoftBench Class Graph/Editor	Displays inheritance, friendship, containment, template instantiations, base class, and derived class relationships between classes. You can edit the C++ classes in your program using the SoftBench Class Graph/Editor. This graph is part of C++ SoftBench only. See Chapter 5, “Using SoftBench Class Graph/Editor,” on page 133 for more information.
File Graph	Displays #include relationships between source files.
Query Graph	Provides a general purpose graph that can be used to keep a history of queries used to solve a particular problem. The Query Graph provides the graphical equivalent of the textual queries on the main SoftBench Static Analyzer window. The Query Graph is free format and the results of each query appear to the right of the previous results. SoftBench Static Analyzer implies no relationship between one query's results and the next.

The Call Graph, Class Graph/Editor, and File Graph are specialized graphs associated with functions, classes, and files. These graphs provide visual detail of program structure at different levels and help you perform the following tasks:

Static Graph Tasks	Description
Analyzing Impact of Change	Determining how classes, functions, or files are impacted by a specific change. For example, "What code is affected if I change this parameter list?"
Understanding Legacy Code	Quickly understanding code you did not develop. The graphs provide different perspectives that can accelerate this learning process.
Re-architecting Code	Understanding the current structure and dependencies between files, functions, or classes to determine what re-architecting is needed.
Query Graph	Showing the sequence of queries you used to solve a particular problem.

All Static graphs are invoked from the "Graph" menu. Each graph has its own menu bar providing start-up, query, and customization functionality.

---

## Starting SoftBench Static Analyzer Graphs

The Static Graphs are invoked from the "Graph" menu of SoftBench Static Analyzer:

Function	Description
"start"	Invokes the graph without displaying any nodes. If the graph is already running it moves to the foreground.
"Display Symbol()"	Displays the identifier provided in the "Symbol ()" input box on the graph.
"All Items"	Displays all items of a specific symbol type on the graph. For example, you might want to display all files on the file graph. These menu items are also available from the root popup menu of each graph.

For complete descriptions of the menus, use SoftBench Online Help.

## Making Graphical Static Queries

This section covers the common features of these graphs. For more information on using generic SoftBench Graph Window features, see Appendix A, "Using SoftBench Graph Windows," on page 323.

### General Static Graph Features

The Static Graphs have a consistent user interface, with several common features:

Feature	Description
Displaying Nodes	<p>Display nodes (files, functions, variables, classes) on each of the graphs. This can be done by entering an identifier in the "Symbol ( )" input box and selecting <b>Display</b>, or by choosing the "All <i>item</i>" option under the "Graph" menu or on the root popup menu.</p> <p>When you enter an identifier into the "Symbol ( )" input box, Static Graph cannot determine any scoping information. If you want to refer to a specific instance of the identifier, select it in a textual Static query result. Choose "Graph: <i>graph type</i> → Display Symbol ( )" from the main SoftBench Static Analyzer window.</p>
Context-Sensitive Queries	<p>Different queries are available for each node type on each graph. For example, a file node has file-related queries on the File Graph and function-related queries on the Call Graph.</p>
Popup Menus	<p>The graph popup menus provide query actions that are specific to the node or arc type selected. Select a node or arc with the left mouse</p>

button, then activate the popup menu with the right mouse button. If no node or arc is selected, a menu of general graph operations appears. The "Selected" menu contains the same commands as the context-sensitive popup menu.

### Switching between Graphs

Use "Graph: Switch To *name* Graph" to move between the Static Graphs.

### Display Legend

All graphs use a combination of shapes and colors to distinguish different types of nodes. Graph arcs are distinguished by color and line type. Each graph has a Legend explaining these shapes and colors. The Legend can be toggled on or off from the respective graph's "View" menu.

## Finding Graph Nodes

To find a particular node on a graph, choose "Graph: Find Node..." and enter the node name, or a portion of the name, in the dialog box input area. This feature makes it easy to find nodes on large complex graphs.

## Operating on Static Graph Nodes

Double-click on a node to edit the code represented by the node. A single arc may represent several locations in the code, such as several calls to a function. Double-clicking an arc displays a description dialog that lists all locations represented by the arc. You can select a specific location in the description dialog to edit the indicated source location.

Several other operations are available on graph nodes. Select the node and activate a popup menu with the right mouse button, or use the "Selected" pulldown menu on each Graph Window. Each type of node has its own specialized popup menu for each graph, containing operations such as:

Operation	Description
Describe	This choice brings up a dialog box that provides auxiliary information on the node. For example, a function node's "Describe" dialog box provides a list of function parameters, local variables, the location of the definition, and a list of attributes of the function.
Node Queries	By selecting the node you can perform queries that are specific to that node type and that graph.
Hide Nodes	The node can be hidden from the graph by choosing "Hide Selected". You can also remove everything except your selection by choosing "Hide Unselected".
Copy Nodes	You can copy nodes to other graphs using the "Redisplay Selection" submenu. This feature allows you to view nodes from several different perspectives. As an example, you could copy a class to the Call Graph, and find out all the functions that call any member of the class.

## Switching between Static Graphs

SoftBench Static Analyzer provides the flexibility to switch between graphs quickly, allowing you to view your application from several viewpoints. For example, you can switch from the Static Call Graph to the Static File Graph to see which source files would be affected by a change to that function. Each graph provides menu actions that are unique to either files, functions, variables, or classes. You can also enter identifiers in the "Symbol ()" input box on any graph. You could, for example, enter `main` in the "Symbol ()" input box on the File Graph and select **Display**.

## Displaying Nodes on Another Graph

The popup menus on each graph provide operations that are specific to that graph. You can copy nodes from one graph to another to access the features of the new graph. Select a node, display the popup menu, and choose one of the selections on the "Redisplay Selection" submenu.

For example, if you select a function in the Call Graph, you can find what functions call or are called by the function. If you copy the function to the SoftBench Class Graph/Editor, you can find what *classes* access or are



accessed by the function.

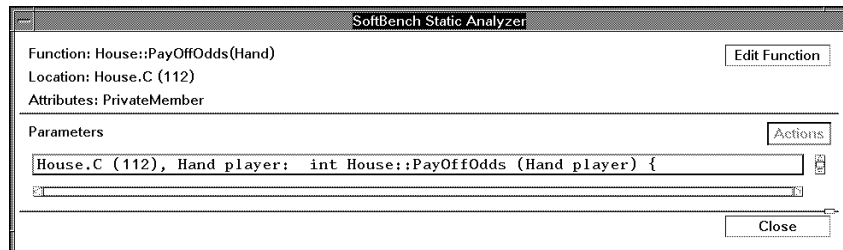
## Using Description Boxes

The Static Graphs provide the ability to display additional detail about identifiers and relationships through description dialog boxes for both nodes and arcs. One of the main advantages of using these dialog boxes in conjunction with the graphical display of your program, is the ability to see more detail without moving back and forth between the main SoftBench Static Analyzer window and the Static Graphs. You can access these dialog boxes with the graph popup menus by selecting a node or arc and choosing "Describe" from the popup menu.

For example, assume you have just generated a call graph and you want to find out more about a particular function. Select the function node and choose "Describe" from the popup menu. See Figure 10-1 for an example of the description dialog box for function `PayOffOdds`. The dialog box displays the parameters and local variables of the function.

**Figure 10-1**

### "Function Description" Dialog Box



You can perform several actions on any item in the "Describe" dialog box by selecting the item and then selecting **Actions**. This menu allows you to:

- Edit or browse the corresponding line of source code with the editor. Double-click on a line, or select the line and choose "Edit" from the **Actions** menu.
- Copy an item from the list to another Static Graph. Choose the "Redisplay Selection" submenu to copy the item to one of the other Static Graphs.

You can also use the graph description dialog box to determine the functions and global variables defined in a source file:

1. Select a source file node.

2. Choose "Describe" from the popup menu.

The "Describe" dialog box shows all functions and any global variables defined in that source file.

Arc description boxes show additional detail on the relationships between nodes. For example, you might be interested in determining all places where a particular function calls another function. Select the arc and choose "Describe" from the popup menu to see occurrences of calls.

## Setting Breakpoints for SoftBench Debugger

From any Static Graph you can set breakpoints on functions, member functions, and call sites. To set a breakpoint from the Static Call Graph:

1. Start SoftBench Debugger from the main SoftBench window.
2. Display the desired identifier on the Call Graph.
3. To set a breakpoint on a function:
  - a. Select the function node with the left mouse button.
  - b. With the right mouse button, choose "Set Breakpoint" from the "Function Actions" popup menu.
4. To set a breakpoint on a particular function call:
  - a. Select the arc representing the call in which you are interested. For example, if you want a breakpoint on a particular call of `printf` within `main`, select the arc between `main` and `printf`.
  - b. With the right mouse button, choose "Describe" from the "Arc Actions" popup menu.
  - c. Select the particular call you want in the "Describe" dialog box. You may double-click on the individual calls to examine the code in your configured editor.
  - d. Select **Actions**, and choose "Set Breakpoint".

## Saving Static Graph Images to Files

SoftBench Static Analyzer allows you to save your graph images in several file formats, scales, and page sizes. From any graph window choose "Graph: Save Image...". You cannot reload saved images into SoftBench Static Analyzer.

However, you can print a saved image or incorporate it into other documentation.

For complete descriptions of the options, see Appendix A, “Using SoftBench Graph Windows,” on page 323.

## Simplifying Graph Displays

The Static Graphs can display many relationships between items in your program simultaneously. To use the graphs effectively, you may want to display only a subset of these relationships. You can filter nodes out based on topics such as inheritance, containment, type of function call, declarations, and many others, depending on the graph. To apply filters to new nodes and arcs that are added to the graph, choose "View: Filters...". The filters take effect immediately. When you make changes, such as settings filters, SoftBench Static Analyzer saves the changes automatically.

The main SoftBench Static Analyzer window also offers filtering mechanisms. See "Simplifying Query Results" on page 266.

## Reducing Graph Complexity

For large software applications, you can control graph complexity by reducing the number of relationships displayed on the graph or by hiding relationships of lesser interest after they are displayed.

For example, when you display all functions on the Call Graph, the resulting graph contains all functions and all "Calls" and "Calls Within" relationships. Once you see the entire graph, you may decide to concentrate on a small subsection of the call hierarchy and hide the rest of the graph:

1. Choose "Graph: All Functions" to display the initial call graph of all functions.
2. Press Shift, then select the node at the top of the subsection of nodes that you want to retain. The Shift key causes SoftBench Static Analyzer to select all nodes in the hierarchy below the selected node.

Alternatively, you can select multiple nodes by pressing the Control key as you individually select nodes, or you can drag the mouse and draw a box around the nodes you want to select.

3. Choose "Hide Unselected" from the popup menu.

In this example, you selected the nodes to retain. You can also hide nodes using reverse logic and select the nodes you want to hide, then choose "Hide Selected".

You can further reduce complexity by displaying fewer relationships initially. For example, you can turn off all initial queries and subsequently display only the relationships of interest to you:

1. From the Call Graph, choose "Options: Queries On New Nodes..." and disable all queries. The default queries are "■ Calls" and "■ Calls Within". Without any default queries, no arcs are drawn.
2. Save these changes by selecting **OK**.
3. Choose "Graph: All Functions".
4. Select the functions you want to query on by dragging the left mouse button and drawing a box around those functions. Alternatively, select multiple functions by pressing the Control key while you select function nodes.
5. Post the "Function Actions" popup menu.
6. Choose the desired query.

## Filtering Sourceless Nodes

A sourceless node is a node for which there is no source code available, for example, header files. There may be times when you want to see calls to functions you have written and exclude calls to library functions you did not write. For example, you might want to exclude calls to functions defined in X11/Motif libraries. You can apply this filter to the Static Call Graph, Class Graph/Editor, or Query Graph.

In the same way, on the SoftBench Class Graph/Editor you can display classes that are declared but not defined.

To exclude these calls from the Call Graph, Class Graph/Editor, or Query Graph:

1. Choose "View".
2. Deselect the "■ Display Sourceless Nodes" toggle button.
3. Select **OK**.

## Customizing Static Graphs

The Static Graphs have many options that you can configure. Each graph has an "Options" and a "View" menu that help you customize the graph for your needs. For example, you can turn off the graph legend, or specify default queries to be applied to any new node that is added to the graph. Filters and other customizations are also available. See SoftBench Online Help for full descriptions.

### Removing the Graph Legend

The legend provides useful information about the types of nodes and arcs displayed on the graph. To remove the legend and see more of the graph area, turn off the "View:  Display Legend" toggle button. The legend disappears from the graph immediately.

### Viewing Multiple Graphs

You can view more than one Static Graph simultaneously. For example, you can view the Call Graph and File Graph to see the relationships between functions and files as they are presented in those graphs.

To view multiple Static Graphs:

1. From the main SoftBench Static Analyzer window, choose "Options: Behavior Settings..." which displays the "Behavior Settings" dialog box.
2. Deselect the "Collect Graphs Into A Stack" toggle button. To set this option select **OK**. A dialog box appears, indicating the change takes effect when the tool restarts. Select **OK**.
3. To make this option active you need to restart SoftBench Static Analyzer. Choose "File: Exit".
4. Restart SoftBench Static Analyzer.

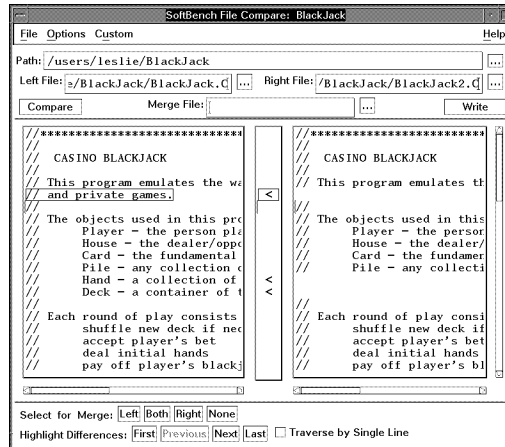
# 11 Using SoftBench File Compare

With SoftBench File Compare you can compare two text files by putting them next to each other in text windows and scrolling through them in a matching manner. You can then merge selected contents into a single new file. If you wish to change a file, you must modify the file using your pre-configured editor.

## Understanding the SoftBench File Compare Window

You can access the SoftBench File Compare window from the SoftBench main window using: "File: Compare...", or by selecting the file Compare icon from the tool bar. If files are selected, they are provided as arguments to File Compare, allowing you to specify files to compare before the tool starts. Otherwise you may enter the file names directly into the tool. The SoftBench File Compare window looks like this:

**Figure 11-1** SoftBench File Compare Window



## Understanding the Menu Bar

The menu bar at the top of the application window consists of the menu items "File", "Options", "Custom", and "Help". Refer to SoftBench Online Help for detailed descriptions of these menu items.

## Understanding the "Working Directory" Input Box

The "Path" input box displays the current working directory.



## Using the "Left File" and "Right File" Input Boxes

These two input boxes are for the names of the files to be compared. As with other SoftBench tools, you can use ... to help you find the files you want.

**Compare** starts a comparison between the two files. The comparison may take a few seconds or more, depending on the size of the files being compared.

## Using the "Merge File" Input Box

This input box is for the name you give to the new, merged file. SoftBench File Compare creates the file if you select the **Write** button. If the file already exists, selecting **Write** causes the contents to be overwritten.

**Write** writes out a merged version of the two files to the file specified, using the lines you have selected from each file.

## Understanding the Text Areas

The two text areas display the two files being compared. Scroll bars allow top-to-bottom scrolling of both files, and left-to-right scrolling in each file.

## Reading the Gutter Column

The gutter column between the two file view windows displays up to three characters. The middle character indicates whether the corresponding lines in the file view windows are the same, different, or appear in only one of the files. The left and right characters indicate which version of a difference has been selected for inclusion into a merged file.

SoftBench File Compare uses these symbols for the middle character:

- | (vertical bar) indicates lines that differ from each other.
- > indicates a line that exists only in Right File.
- < indicates a line that exists only in Left File.
- (blank gutter) indicates identical lines.

The left and right columns of the gutter use a plus sign (+) or minus sign (-) in the appropriate column to indicate which version you want

included in the new file:

**Table 11-1**                      **SoftBench Static Analyzer Display Options**

<b>Left Column</b>	<b>Right Column</b>	<b>Selection</b>
Blank	Blank	No Decision
+	-	Left File
-	+	Right File
+	+	Both Files
-	-	Neither File

SoftBench File Compare initially leaves the left and right columns blank, indicating that you have not made a decision as to which side to save. If you just want to compare and not merge files, you can ignore the left and right columns.

## Selecting Lines for Merging

These buttons enable you to choose which version of the currently highlighted difference you want to put into a merged file. You may want to choose "Options: Move Forward after Selection" to automatically go to the next difference after making a selection.

- Left**                      Marks the Left File version of the highlighted difference as the version you want included in the merged file.
- Both**                     Marks both sides of the currently highlighted difference to be included in the merged file. The text from the left side is written to the output file before the text from the right side.
- Right**                    Marks the Right File version of the highlighted difference as the version you want included in the merged file.
- None**                     Marks neither side of the highlighted difference as the version you want included in the merged file.

SoftBench File Compare automatically copies contents that are identical in both files into the new file.

## Highlighting Differences

These buttons enable you to change which difference between the left and right files SoftBench File Compare displays and highlights.

<b>First</b>	Moves the current selection to the first difference between the two files and highlights it.
<b>Previous</b>	Moves the current selection to the previous difference between the two files and highlights it.
<b>Next</b>	Moves the current selection to the next difference between the two files and highlights it.
<b>Last</b>	Moves the current selection to the last difference between the two files and highlights it.

If you select "Options: View Unresolved Differences Only", these buttons move the current selection highlight to the next difference where you have not made a choice regarding which version to put into the merged file.

## Traversing by Single Line

The "■ Traverse by Single Line" button enables you to move line-by-line when merging one line at a time.

## Comparing Two Files

To compare two files:

1. Enter the name of the first file for comparison into the "Left File" box, and the name of the second file into the "Right File" box. You can begin the file name with "/" to override the current working directory, or enter just the name of the directory path and file relative to the current working directory.

You can also use the ... buttons to open a navigation dialog box to find the files you wish to compare. The filter option allows you to specify the types of files you wish to see in the "Files" list box. For example, if you enter "\*.txt" in the "Filter" box, and select **Filter** (refreshes the "Files" list box), only files ending with ".txt" appear in the "Files" list box.

2. Select **Compare** after choosing your left and right files.

Copies of the two files appear, one in each file view window, with the first set of differences highlighted. SoftBench File Compare adds blank lines as placeholders to allow the files to be vertically aligned. The center of the gutter displays the appropriate symbol:

- | (vertical bar) indicates lines that differ from each other.
- > indicates a line that exists only in Right File.
- < indicates a line that exists only in Left File.
- A blank gutter indicates identical lines.

If you want to visually compare the two files without making a new file, use the scroll bars and the **First**, **Previous**, **Next**, and **Last** buttons.

You may find viewing the differences easier if you choose "Options: View Common Lines on Left Only". When you select **Compare** again, SoftBench File Compare displays any identical lines in both files being compared only in the left file view window.

---

## Merging Compared Files

Merging files is similar to comparing files except that you decide what to put in a new file, what to name the new file, then you write your selections to the new file. Follow these steps to create a merged file:

1. Enter the names of the files you wish to merge in the "Left File" and "Right File" input boxes, or use the ... buttons to open a navigation dialog to find the files you wish to compare.
2. Enter a name for the new file in the "Merge File" input box. Alternatively, select the ... button to open the navigation dialog to find the directory where you want to write the merge file. The new file name can be entered at any time.
3. Set any options from the "Options" menu.
4. Select **Compare**.
5. SoftBench File Compare highlights the first set of differences between the files. Use the "Select For Merge" and the "Highlight Differences" buttons to work through your two files. You may want to choose "Options: Move Forward after Selection" to automatically go to the next difference after making a selection.

SoftBench File Compare automatically copies identical contents in both files into the new file.

You can select "■ Traverse by Single Line" to move line-by-line when merging one line at a time.

6. Continue making selections as SoftBench File Compare highlights each set of differences. When you reach the end of the files, select **Write** to send the merged version of the two files to the file name you indicated.

---

## If Something Goes Wrong

Table 11-2

Condition	Explanation
No file specified message appears when you select <b>Compare</b> .	You must enter a file name in the "Left File" or "Right File" input box.
Could not open file: (filename) message appears when you select <b>Compare</b> .	The file name in the "Left File" or "Right File" input box does not exist or is not readable.
sdiff command failed to execute message appears when you select <b>Compare</b> .	Make sure your system has the sdiff command and that your path includes the directory in which sdiff resides.
Could not write to file: (filename) message appears when you select <b>Write</b> .	A merged file cannot be created, probably because the directory that you specified has no write permission.
No file specified message appears when you select <b>Write</b> .	You must enter an output file name in the "Merge File" input box before choosing <b>Write</b> .
No choice was selected for one or more differences message appears when you select <b>Write</b> .	Select <b>Left</b> , <b>Both</b> , <b>Right</b> , or <b>None</b> for all remaining differences found by SoftBench File Compare. Choosing "Options: View Unresolved Differences Only" may facilitate this process.

## For More Information

- On editing files, refer to Chapter 4, “Using SoftBench Editors,” on page 117.
- On SoftBench File Compare, see the *softcom(1)* reference page, available by selecting "Help: Show Man Page".

Using SoftBench File Compare  
**For More Information**



# 12 Using SoftBench Message Monitor

SoftBench Message Monitor provides a window to SoftBench tool communication. It displays all messages sent through a particular Broadcast Message Server. It can also create and send messages to other tools. SoftBench Message Monitor is the companion tool to SoftBench Encapsulator, which provides the SoftBench integration library (`libsoftbench`). Use SoftBench Message Monitor to help debug of the message interface provided by this library.

## Starting SoftBench Message Monitor

You can start SoftBench Message Monitor from the main SoftBench window. First you need to add the tool to the toolbar, then start the tool:

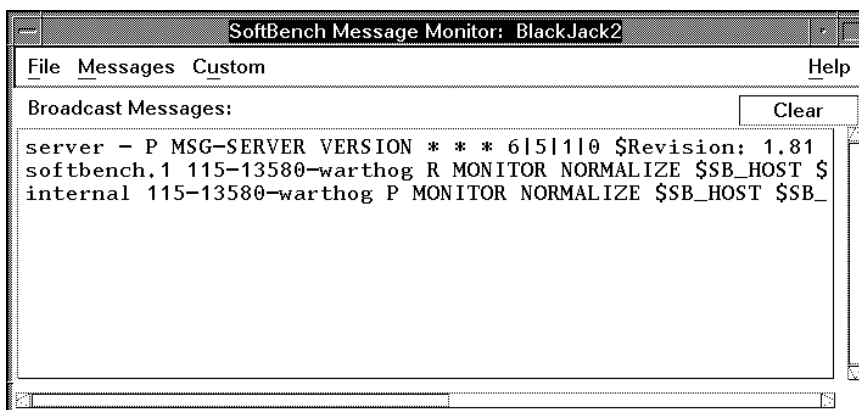
1. In the main SoftBench window, choose "Options: **T**oolbar Setup...".
2. From the "Available Tools" list in the "Toolbar Setup" dialog box, select MessageMonitor. Notice the "Message Monitor" icon in the middle of the dialog box.
3. Select **Add to Toolbar**. This action adds MessageMonitor to the "Tools on the Toolbar" list on the right.
4. Select **OK** to confirm the addition and close the dialog box.
5. Select the "Message Monitor" icon in the SoftBench toolbar. This action starts SoftBench Message Monitor.

---

## Understanding the SoftBench Message Monitor Window Area

SoftBench Message Monitor has a menu bar and a message display.

**Figure 12-1** SoftBench Message Monitor



### Understanding the Menu Bar

The menu bar at the top of the application window contains pulldown menus to initiate commands. SoftBench Message Monitor contains four menus: "File", "Messages", "Custom", and "Help" menus.

### Understanding Broadcast Messages

The "Broadcast Messages" area displays messages in a scrollable list. SoftBench messages have the following components:

Tool_name	Identifies the particular tool instance (the tool name followed by a dot and a unique number) that sent the message. Broadcast Message Server assigns this field when you start the tool.
-----------	---

Message-id	A unique string identifying a particular message and any responses from the tool handling the request.
Message-type	Indicates whether this particular message is a "Request" (R), or a "Notify" (P for pass and F for fail).
Tool_class	Identifies the class of tool that sent, or is to handle, the message. Two examples are EDIT and DEBUG.
Command	Identifies the actual task requested or performed.
Project	Defines the location of a project (defaults to \$SB_HOST and \$SB_DIR).
Operand	Defines the operand identification for the message.
Data	Contains any data or arguments that provide additional information.

For messages in this output area, a "\*" in a field indicates that the field is not applicable for this message. A "-" in a field indicates that a value for this field is relevant to the message, but that only the receiver knows what the value should be.

## Clearing the Broadcast Message Area

**Clear** clears the "Broadcast Messages" area of all accumulated messages.

## Composing and Sending a Message

You can compose and send messages from SoftBench Message Monitor using "Messages: Send Message...". Here is a sample session:.....

1. Choose "Messages: Send Message...".
2. Select  Tool Request".
3. Enter EDIT in the "Tool Class" input box to request an edit.
4. Enter WINDOW in the "Command" input box to bring up the file in an edit window.
5. "Host" and "Dir" identify the group of tools that communicate. You should not change these two fields. "Oper" identifies the file to be edited.

Be sure that the "Host", "Dir" and "Oper" fields have the appropriate format. By default, the "Host" and "Dir" fields are filled in based on the tool's current project. If this tool is not associated with a project, it defaults to the current working directory. To create a proper "Oper", type in the path of the file you want to edit and select the **Expand** button to put the file in the proper message format.

6. Select **Send**.
7. Select **Close** to remove the "Send Message" dialog box.

Figure 12-2

### "Send Message" Dialog Box

The screenshot shows a dialog box titled "Message Monitor - Send Message". At the top, there are three radio buttons: "Tool Request" (which is selected), "Pass Notify", and "Failure Notify". Below these are several input fields. On the left, there are two stacked input boxes: "Tool Class" containing the text "EDIT" and "Command" containing "WINDOW". On the right, there are three stacked input boxes: "Host" containing "warthog.sde.hp.com", "Dir" containing "h/projects/BAAa02453", and "Oper" containing "ers/leslie/BlackJack". Each of these three boxes has a small "Expand" button to its right. Below the input fields are two larger text areas labeled "Data:" and "Reply Data:". At the bottom of the dialog, there are two buttons: "Send" on the left and "Close" on the right.

## Using SoftBench Message Monitor

### Composing and Sending a Message

A message resembling the following appears in the SoftBench Message Monitor Broadcast Messages area:

```
Softmsg.nn 29-25966-tycho R EDIT WINDOW <host> <dir> <oper>
```

Assuming the appropriate editor successfully edits the file, it sends the following message in response to the request: For SoftBench vi Editor:

```
softvi.2 29-25966-tycho P EDIT WINDOW <host> <dir> <oper>
```

For SoftBench XEmacs:

```
softemacs.3 29-25966-tycho P EDIT WINDOW <host> <dir> <file>
```

Notice that the above examples contain no real data. The data is in the form:

```
<executable>.<nn> <msgId> <type> <tool> <command> <host> <dir> <file> <data>
```

where *nn* is a unique number assigned by the Broadcast Message Server (BMS) to identify this invocation of this tool.

## Logging Messages

If you are integrating with SoftBench and want to see actual message formats, you can exercise SoftBench tools with SoftBench Message Monitor running. Tools may send several messages for a single action, filling the "Broadcast Message" area quickly. These messages can be saved by logging them to a file.

### Specifying a Log File

The log file can be specified by choosing "File: Set Logfile Name...", and specifying a filename in the resulting dialog box.

### Starting and Stopping Message Logging

To start message logging

1. Choose "File: Set Logfile Name...", and specify a filename in the resulting dialog box. The default is "\$HOME/.softbench/*projectname*.msglog".
2. Set the "File:  Enable Logging" toggle button.

SoftBench Message Monitor logs all messages to the logfile.

To disable message logging, clear the "File:  Enable Logging" toggle button. Once disabled, SoftBench Message Monitor closes the logfile. If you enable logging again, SoftBench Message Monitor logs to a new copy of the named file.

Using SoftBench Message Monitor  
**Logging Messages**



# 13      **Using SoftBench with SQL**

Relational Database Management System (RDBMS) products support embedded SQL statements in source code by using a preprocessor to translate the SQL into the source language. SoftBench assists developers using relational database embedded SQL in their software applications.

## **Determining Supported Environments**

SoftBench SQL Supports the system platforms, RDBMS products, and language environments found in the list below.

- System Platforms:
  - HP-UX
- RDBMS Products:
  - Informix
  - Oracle
- Languages:
  - C
  - C++

---

## Configuring SoftBench with SQL

SoftBench fully supports embedded SQL source files in both project and external build models. For project build, SoftBench provides packages to include in your build configuration instructions.

### Using Default SQL File Types

SoftBench recognizes an embedded SQL source file type by the file extension. Each file extension results in unique language-specific operations. Table 13-1 shows the default file extensions used for a particular RDBMS. See SoftBench Online Help for additional information regarding file types.

**Table 13-1** SQL File Extensions for C Language

RDBMS	Embedded SQL File Extensions
Informix	.ec
Oracle	.pc
C++ Oracle	.pC

### Using SQL with Project Build

The basic steps in defining **targets** that use SQL are:

1. Create the embedded SQL source files and add them to the **project**:
  - If you create the files from scratch in a SoftBench editor started from the main SoftBench window, then new files automatically become members of the project when you save them.
  - If you have existing files, you can add them to the project by choosing "Project: Add File(s) to Project...".
2. Set up a **build configuration** that works for the targets in this application:
  - a. Choose "Builder: Manage Build Configurations...".

- b. Select the build configuration that most closely describes the type of target that you need to create. For example, to create a C executable that uses the Oracle database, select the "OracleCExecutable" build configuration.
    - c. Expand each section of the dialog box to check the appropriateness of the build instructions. Add libraries, include files, compiler flags, etc. as needed for your application.
    - d. Select **Save As...** and provide a new build configuration name.
    - e. Select **Close** when you have customized the build configurations that you need.
  3. Define the targets that you need:
    - a. Choose "Target: New..." to open the "Define New Target" dialog box.
    - b. Enter the target name.
    - c. Optionally, select a directory other than the **local workspace root** for the target's location.
    - d. Select the build configuration for building the target.
    - e. Select **Add** to add the target to the "Pending Targets" list.
    - f. Repeat steps a through e for any other targets in the project.
    - g. Select **OK** when you have added the targets that you need.
    - h. If you need to add any special build instructions for specific targets, select the target name, then Choose "Target: Modify Properties...". Select **Customize Build Configuration...** Make any changes to the build configuration that you need, then select **OK**.
  4. Associate embedded SQL files with the targets by selecting the embedded SQL files, then choosing "File: Link Source to Target...".

When you build the targets, SoftBench generates and manages the necessary build instructions automatically. You can preview the build by choosing "Target: More Build Actions → Preview Build". Or you can run the build and see whether your build completes successfully. If you encounter build problems, either fix the embedded SQL source code errors or the build configuration, whichever is appropriate.

## Using SQL with External Build

If your Makefile was created with SoftBench version C.05.xx or earlier, you need to migrate your application to project build or use external build to build your application. With external build, you must maintain your Makefile manually.

To use your Makefile or build script, choose "Builder: Use External Build Command...". Complete the dialog box and select **Build**. All necessary commands for using SQL should be contained within the Makefile.

## Updating RDBMS Versions

When you update your RDBMS version, the names and locations of the SQL libraries that need to be linked to your SQL application may change. Since SoftBench supports two build models, **project build** and **external build**, your RDBMS version update process depends on the build model you use.

**Project Build**      SoftBench provides **packages** for managing libraries and include files logically as a group. SoftBench ships with default packages for embedded SQL. If you update to a new version, you need to update the definition of the SQL package.

Build configurations use packages to produce targets. If you use the default SQL packages initially in your build configurations, then the update process requires you to save the revised package under a new name. (You cannot modify the packages shipped with SoftBench.) Consequently, you need to update the build configurations that include SQL packages as well.

**External Build**      Manually update each Makefile or build script that uses SQL to point to the new SQL library names and locations.

To update the version of SQL libraries linked to your application:

1. Install the new RDBMS version.
2. Set all normally required RDBMS environment variables. See your RDBMS documentation for specific details about environment variables used by your RDBMS.

Once you install the RDBMS and the runtime environment is ready, the

update process varies based on how you build your project. For project build:

1. If preprocessor flags change, you need to manually edit the transform file for the preprocessor. See SoftBench Online Help for details.
2. Start SoftBench.
3. From the menu bar, choose "Builder: Manage Packages...".
4. From the drop-down list, select the desired SQL package.
5. Expand the sections of the dialog box to check the current package definition. Make any necessary changes for use with the new RDBMS version. The most likely changes involve libraries. If you changed preprocessor flags, you may need to change transform flags as well.
6. When you complete your changes, select **Save** if the package is a "User" package and you want to maintain the same package name. Select **Save As...** if the package is a "System" package or you want a different package name.
7. Select **Close** to close the dialog box.
8. If you changed the name of the SQL package, choose "Builder: Manage Build Configurations..." to change the build configurations that use the SQL package. Select each relevant build configuration, switch to the new package, and save the changes.

To continue using SoftBench version C.05.xx or earlier Makefiles with external build, you need to edit a macros file in `/opt/softbench/config/buildt/include`. You can determine the RDBMS version for which the macros file is currently configured by examining the comment line in the file.

You must have root privilege to edit the macros file. Select the file to edit based on your specific RDBMS:

- Choose `c.informix.macros` for Informix applications.
- Choose `c.oracle.macros` for Oracle applications.

For other external builds, manually edit your Makefiles or build scripts.

## Using SQL with SoftBench Tools

You can use embedded SQL source files with Builder, the SoftBench editors, SoftBench Debugger, SoftBench Static Analyzer and SoftBench CodeAdvisor.

### Using SQL Preprocessor Wrappers

Working with your original SQL source code is often simpler than working with the expanded SQL statements that the RDBMS sends to the compiler. To provide line numbers that reflect your original source files, SoftBench provides a preprocessor wrapper which the compiler uses to insert line number directive information into the object file. SoftBench automatically invokes these preprocessor wrappers when a file with an SQL type extension appears in a **project build**. Builder, SoftBench CodeAdvisor, SoftBench Static Analyzer, and SoftBench Debugger use the line number information to point back to the original SQL embedded source code.

For **external build** projects, SoftBench tools refer back to the line numbers in the file specified within your object file. Refer to your RDBMS preprocessor documentation for information on generating line numbers through your Makefile.

### Debugging with SQL

SoftBench Debugger performs debugging of applications using the original source code (with embedded SQL statements). SoftBench Debugger treats embedded SQL statements as one single statement in the source language for breakpoints and stepping operations. SoftBench Debugger sets breakpoints at the beginning of an SQL statement.

## Debugging Expanded SQL Statements

With SoftBench Debugger you typically debug your program using the original source code with the SQL statements. However, you may want to debug your source code after the SQL statements have been expanded by the SQL preprocessor. To debug the generated source file instead of the embedded SQL file:

- Project build
1. Select the target and choose "Target: Display on Graph".
  2. On the target graph's **Display Dependencies** option menu button, select "Source File Dependencies".
  3. Select the generated source code node you want, then choose "Modify Properties..." on the popup menu.
  4. In the "Modify Intermediate File Properties" dialog box, remove the debug compile mode option, then select **OK**.

By overriding the normal compile mode for debugging, the compiler does not create line number directive information for the embedded source code file, but it does generate debug information in the object file for the generated source code file.

To debug expanded SQL statements for all files, create a "NoDebug" version of the SQL package when you set up your targets' build configurations. Using a text editor, comment out the `APPEND DEBUGMODE -g` line in the `Proc` package. See SoftBench Online Help topic "Sharing Packages with Team Members" for more information.

Old SoftBench Makefile Edit the Makefile and remove the compile mode for the `SQLDEBUG` macro.

Other Makefiles and build scripts Edit the Makefile using whatever control mechanisms the RDBMS vendor provides.



## Editing and Rebuilding with SQL for SoftBench Debugger

The `make` program does not follow dependencies beyond the source code (that is, the `.c`) level. So when using external build, you may need to remove the generated source code file after an edit to correctly rebuild.

### Project build

1. Select **Edit** in the SoftBench Debugger Edit Area or choose "File: Edit" to start your configured editor.
2. Make the source code change in the embedded SQL file.
3. Save the changes.
4. Choose "File: Build".

### Old SoftBench Makefile

Follow the steps for project build. If you have changed your Makefile or your original source code (for example, the `.ec` or `.pc` file), you may need to delete the generated source code file (for example, the `.c` file). If the generated source code file exists, SoftBench does not recompile the original source code file.

### Other Makefiles and build scripts

This situation may require the same actions as an old SoftBench Makefile. What you need to do depends on how your build process handles dependencies.

In all cases, SoftBench takes care of unloading and reloading the executable when SoftBench Debugger is running.

## Using SoftBench CodeAdvisor and SoftBench Static Analyzer with SQL

Both SoftBench CodeAdvisor and SoftBench Static Analyzer rely on analysis data generated during the build process. SoftBench uses the preprocessor wrapper to provide line number information based on the embedded SQL files rather than the generated source files.

---

## **For More Information**

- On packages, build configurations, project build, and external build, see the SoftBench Online Help.

# A **Using SoftBench Graph Windows**

The various SoftBench graph tools visually help you with programming tasks. This appendix describes basic graph operations in the Graph Windows. For tool-specific information, refer to:

- “Using the Target Graph” on page 66
- Chapter 10, “Using Static Graphs,” on page 283
- Chapter 8, “Using SoftBench Debugger Data Graph Window,” on page 235

---

## Accessing SoftBench Graph Windows

SoftBench Graph Windows offer graphical views useful with several of the SoftBench tools. A Graph Window may be opened from:

- the main SoftBench window, Builder page
- SoftBench Static Analyzer
- SoftBench Debugger

Table A-1 shows how to access the various SoftBench graphs.

**Table A-1**      **Accessing SoftBench Graphs**

Starting From	Procedure
Main SoftBench window	<ol style="list-style-type: none"><li>1. Select <b>Expand &gt;&gt;</b> to expand the main window and display the target graph.</li><li>2. Select a target, then use the <b>Display Dependencies</b> menu button to select the level of dependencies that you want to see.</li></ol> <p>Note: You can display dependencies only in a project that uses <b>project build</b>.</p>
SoftBench Static Analyzer	<ul style="list-style-type: none"><li>• Choose "<b>Graph: Static Graph</b> → <b>Start</b>" to display an empty graph. You can add information to the graph using the "<b>Symbol()</b>" input box together with the <b>Display</b> button.</li><li>• Enter data in the "<b>Symbol()</b>" input box of the main SoftBench Static Analyzer window and choose "<b>Graph: Static Graph</b> → <b>Display Symbol</b>".</li><li>• Choose one of the "<b>Graph: Static Graph</b>" submenu commands such as "<b>All Functions</b>" or "<b>All Classes</b>" to graph all entities of a certain type.</li></ul>

**Table A-1**      **Accessing SoftBench Graphs**

<b>Starting From</b>	<b>Procedure</b>
SoftBench Debugger	<ol style="list-style-type: none"><li data-bbox="733 340 1276 401">1. Enter an expression or a variable in the "()" input box.</li><li data-bbox="733 418 1276 479">2. Choose "Show: Data Graph()" or "Show: Data Graph Indirect()".</li></ol>

To end the browsing session for SoftBench Debugger Data Graph Window, select the **Close** button. To end a browsing session for the Static Graphs, select "Graph: Exit" from any Static Graph window. The tool discards all information gathered during the session.

## Using Graph Window Areas

You use the Graph Area to create a visual image. From most tool graph windows you can:

- Use popup action menus.
- Control the arrangement of nodes (not available on the target graph).
- Save an image to a file for printing.
- Scroll and zoom the graph area to manipulate your view.
- Clear the Graph Area and hide selected or unselected nodes.

## Using Popup Menus

Pressing the right mouse button within the Graph Area brings up a popup menu. The specific popup menu depends on the types of nodes or arcs currently selected. For more information on individual popup menu items see the respective SoftBench Online Help.

## Using Save Options for the Graph Image

SoftBench allows you to save your graph images in several file formats, scales, and page sizes. The saved file is only a graphic image of the graph. You cannot restore or reload a graph from a saved file, but you can print using your usual printer commands. For more information about printing these files, refer to your printer or plotter manual.

Saving the image created in the graph in a printable format is useful when creating pictures used in documenting your software program. Many word processing programs allow you to load saved graphs directly into your document.

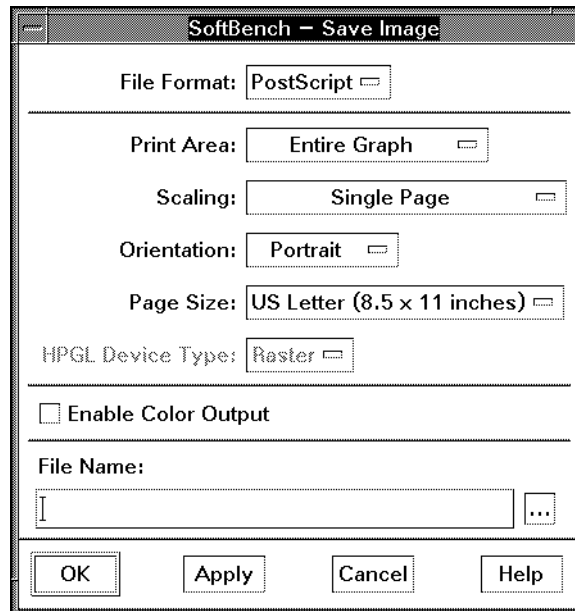
Table A-2 describes the formats in which you can save a graph image.

**Table A-2** **Graph Image File Formats**

Option	Used for
xwd	Importing graph images into a word processing or graphics application for further editing and reporting. This file format cannot be sent to a printer or plotter directly.
HP-GL	Sending graph images to a plotter or printer that handles HP-GL emulation.
PostScript	Sending graph images to a PostScript-capable printer or to import images into a word processing or graphics application.

Access the "Save Image" dialog box (See Figure A-1) by choosing "Graph: Save Image...".

**Figure A-1** **"Save Image" Dialog Box**



## Using Graph File Image Save Options

The tools provide the following options in the "Save Image" dialog box (see Figure A-1):

- Save a graph image in "xwd", "HP-GL", or "PostScript" format.
- Save an image of the entire graph area or just the visible portion of the graph.
- Scale the graph image to fit on a single page or allow it to be printed actual size on multiple pages. The multiple page option prints the graph from left to right then top to bottom.
- Choose between "Landscape" and "Portrait" mode.
- Choose from several page sizes.
- Choose to print the graph in grayscale or color.
- If you select "HP-GL" format, choose whether to save the image in a format suitable for "Plotter" type devices like a pen plotter or "Raster" type devices like a laser printer.

SoftBench Static Analyzer saves the arc labels from a Static Graph only if you set the "View:  Display Arc Labels" toggle button. SoftBench saves the graph legend from a Static Graph or Target Graph only if the legend is displayed. Control the display of the legend with "View:  Display Legend" in SoftBench Static Analyzer, or "Graph:  Display Graph Legend" in the main SoftBench window.

## Saving HP-GL and PostScript Images

To save your HP-GL or PostScript file:

1. Select HP-GL or PostScript from the "File Format" option menu. This sensitizes the option menus used for other options. When neither of these options is selected, the related options are not sensitive and may not be modified.
2. Select "Entire Graph" or "Visible Portion Only" from the "Print Area" option menu.
3. Select "Single Page" or "Multiple Pages" from the "Scaling" option menu.
4. Select "Portrait" or "Landscape" from the "Orientation" option menu.
5. Select a page size appropriate for your printer from the "Page Size" option menu.
6. Set the " Enable Color Output" toggle button to print in color.



7. If you selected HP-GL, make sure the "HP-GL Device Type" option menu reflects the hardware on which you intend to print the image.
8. Enter your filename, which can also include the path name. For example, use `savegraph.hpgl` or `savegraph.ps`. Optionally, select the ... button to access a file selection dialog box for specifying the file name.
9. Save the image. Selecting **Apply** saves the file and leaves the dialog box up for further actions. Selecting **OK** saves the file and closes the dialog box. Selecting **Cancel** closes the dialog box without performing any actions. If you select **Cancel**, the dialog retains its current settings until changed during the same session. Settings in this dialog are not saved between sessions.

Many programs that convert the HP-GL or PostScript files to other formats convert only the graphical objects and do not convert text. If processed in this manner, the arc and node labels may be lost when you print the file.

### **Saving xwd Images**

The `xwd` option saves only the image on your screen. If necessary, adjust your image size or location in the Graph Area or resize the window. Make sure the graph window is fully exposed and displaying the information you wish to save. Portions of windows obscuring the graph will appear in the image. Black and white color schemes print best. When using SoftBench color schemes, it may be necessary to convert to black and white before printing. Check with your local system administrator for conversion utilities.

To save your `xwd` file:

1. Select `xwd` from the "File Format" option menu. When you select this option, the "Print Area", "Scaling", "Orientation", and "Page Size" option menus are not sensitive and may not be modified.
2. Enter your filename in the "File Name" input box. The filename can include the path name. For example, use `savegraph.xwd`. Optionally, select the ... button to access a file selection dialog box for specifying the file name.
3. Save the image. Selecting **Apply** saves the file and leaves the dialog box up for further actions. Selecting **OK** saves the file and closes the dialog box. Selecting **Cancel** closes the dialog box without performing any actions. If you select **Cancel**, the dialog retains its current settings

until changed during the same session. Settings in this dialog are not saved between sessions.

You can use the `xpr` utility to convert an `xwd` file to a printable format.

## Using Vertical and Horizontal Scrolling

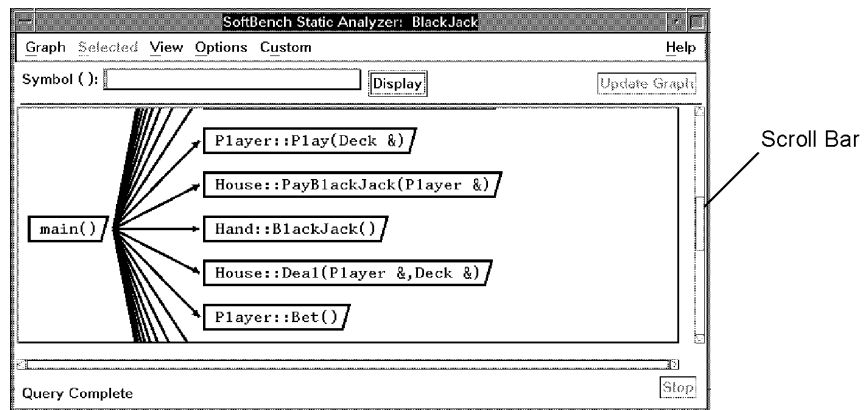
If the scroll bar fills the entire scroll area, there is no more graph to be displayed. If the scroll bar appears smaller than the scroll bar area, it means the tool has more graph than could fit in the area. The size of the bar is proportional to the visible part of the file. Dragging on the scroll bar moves the view in the direction of the drag (see Figure A-2).

You can change the displayed area of the graph in the following ways:

- Move the pointer to the **arrows** (**▲ ▼ < >**) in the scroll bar and keep the left mouse button held down to continuously scroll the graph.
- If you **click** in the empty area between the arrows and the scroll bar, the Graph Area scrolls up or down, left or right one window length. Clicking and holding down the left mouse button continuously scrolls the graph one screen at a time.
- Pressing the middle mouse button in the empty area between the arrows and the scroll bar moves the view to the location in the graph proportional to the location along the scroll bar area at which the button was pressed.

Figure A-2

### Using Scroll Bars



## Zooming In or Zooming Out

Graph Windows provide several zoom settings. You might zoom out for a high level view of the general structure of your graph. Zooming out decreases the size of the objects so you can see more of the graph. When zooming out, the labels become illegible.

You can zoom in to look at the details in the nodes. Zooming in increases the size of the objects and allows you to read a smaller section of the graph.

You can change the size of your graph by selecting "Zoom In" or "Zoom Out" from the "Graph " menu in the Target Graph and Debug Data Graph Window or the "View " menu in the Static Graphs. You can also use the graph popup menu when no graph objects are selected.

You can change the initial zoom setting by changing the `.initialZoomlevel` resource. The *softbench(1)*, *softstatic(1)*, or *softdebug(1)* manual pages document this resource. Use SoftBench Online Help to read or print the manual pages by selecting "Help: Show Man Page..." from the appropriate tool.

## Clearing the Graph Area

You can hide selected or unselected objects or completely clear the Graph Area. You only remove the node or arc from your screen when you hide or clear. The tool does not change the object represented by the node or arc.

You can hide selected or unselected objects using the "Hide" menu commands found on the popup menus.

When you hide an arc, the tool does not automatically remove the nodes even though the tool hides the relationship. In contrast, when you hide a node, the tool automatically hides all arcs to or from the node.

You can clear the Graph Area by choosing "Graph: Clear Graph" in the Target Graph and Data Graph, or "View: Clear Graph" in the Static Graphs.

---

## Understanding Nodes and Arcs

All graphs display nodes and arcs within the Graph Area. Nodes represent entities within your code, and arcs represent the relationships between the entities. The following pages describe how you can select, move, or hide nodes and arcs.

### Reading Graph Area Nodes

The Graph Area includes **nodes** corresponding to entities, and **arcs** corresponding to the relationship between two entities. Table A-3 shows what entities the nodes represent in each graph window. For more detailed discussions about nodes and arcs, refer to the applicable graph chapters.

**Table A-3**      **Graph Area Nodes**

Target Graph	Static Graph	Data Graph Window
<ul style="list-style-type: none"><li>• Include files</li><li>• Source files</li><li>• Intermediate files</li><li>• Build target</li></ul>	<ul style="list-style-type: none"><li>• Functions</li><li>• Function templates</li><li>• Structures</li><li>• Classes</li><li>• Class templates</li><li>• Variables</li><li>• Files</li></ul>	<ul style="list-style-type: none"><li>• Data in memory</li></ul>

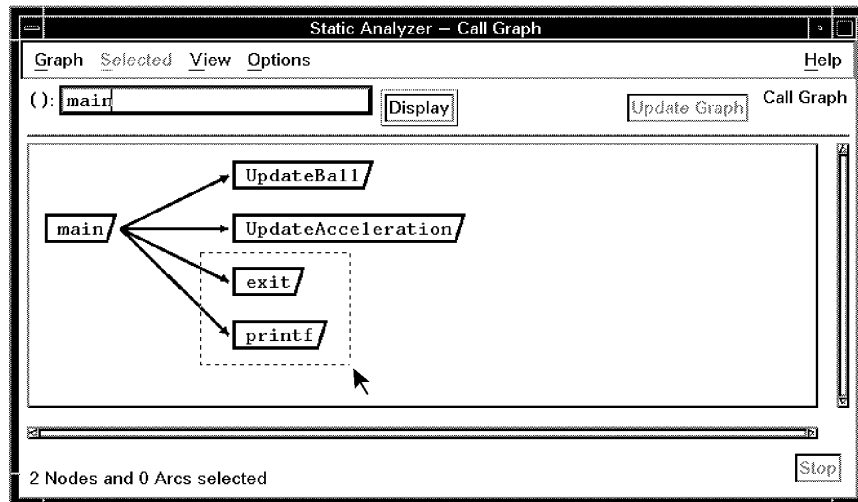
### Selecting Nodes and Arcs

You can select any subset of nodes and/or arcs found within the Graph Area. Refer to Table A-4 for a list of selection methods. When selected, the object's outline changes.

You select a single node or arc by placing the pointer over the object and clicking the left mouse button. This clears any other selections.

When you select multiple objects, the tool applies the chosen popup menu action to all of them. The tools provide several ways to select more than one object. The reasons for selection determine the method you choose to use. For example, you may want to remove all nodes in a certain area of the graph. Figure A-3 shows the selection of two nodes within a Static Graph.

**Figure A-3** Selecting Objects by Dragging the Pointer



**Table A-4** Methods for Selecting Multiple Objects

Objects	Procedure
Random or Unconnected	<ol style="list-style-type: none"> <li>1. Select the first object with the left mouse button.</li> <li>2. Hold down the Control key on your keyboard.</li> <li>3. Move the pointer over the next object and click the left mouse button.</li> <li>4. Repeat steps 2 and 3 as desired.</li> </ol>
Whole or Partial Trees	<ol style="list-style-type: none"> <li>1. Place the pointer over the parent node.</li> <li>2. While holding down the Shift key, press the left mouse button. This selects the parent node and all descendent nodes and arcs.</li> </ol>

**Table A-4**      **Methods for Selecting Multiple Objects**

<b>Objects</b>	<b>Procedure</b>
Regions	<ol style="list-style-type: none"><li>1. Depress the left mouse button in a background area of the graph.</li><li>2. Drag the pointer while pressing the left mouse button. A dashed rectangle appears with the point where the left mouse button was pressed as one corner, and the current pointer position as the diagonal corner.</li><li>3. Release the mouse button. All nodes and arcs completely enclosed in the rectangle become selected.</li></ol>

When you select more than one object, you can clear a single object by holding down the Control key and clicking on the object. Clicking the left mouse button on an empty point on the graph clears all objects.

### **Moving Nodes**

On the Static Graphs and Data Graph you can reposition nodes. The Target Graph does not support this action. Choosing "Options:  Full Relayout" from the SoftBench Static Analyzer graph's main menu causes all nodes to be repositioned whenever nodes or arcs are displayed on or removed from the graph. If you add nodes to the graph while the " Full Relayout" toggle button is off, the tool inserts the new nodes into the existing graph or tree. For the Data Graph Window, you must select " Manual Layout" mode to be able to move nodes and " Auto Layout" to have them repositioned automatically.

You can move a node by selecting the node with the middle mouse button and dragging the pointer to the desired new location in the graph. Copies of the affected node and arcs indicate the new locations prior to releasing the mouse button. Upon release, these copies disappear and the actual node and arcs move to the new location.

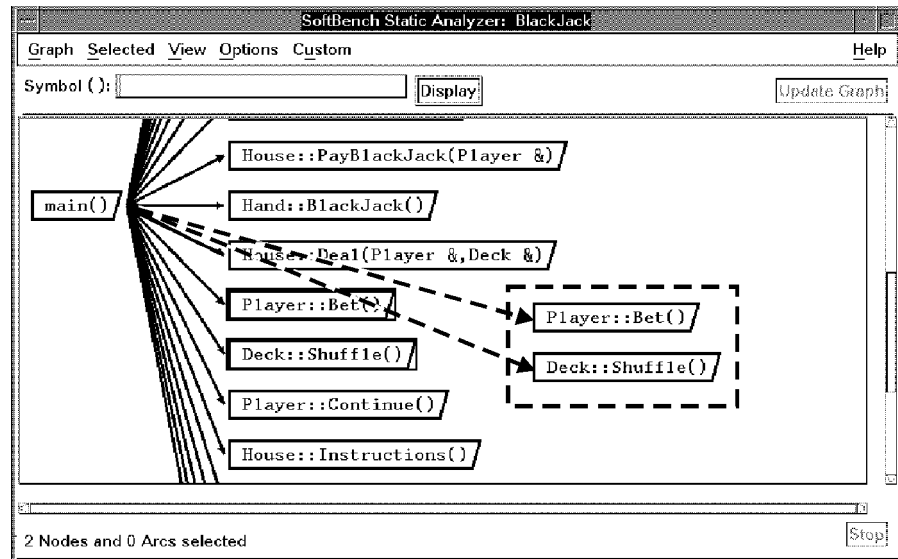
You can move a group of nodes by following these steps:

1. Select the nodes with a multiple node selection method.
2. With the pointer over one of the selected nodes, hold down the middle mouse button and drag the pointer to the desired new location in the

graph. Copies of the selected nodes and attached arcs indicate the new locations prior to releasing the mouse button. Upon release, these copies disappear and the tool moves the actual node and arcs to the new location.

The auto-scroll feature of the graphs allow you to select and move regions of nodes and arcs which may not be completely visible in the current graph window. As you select regions with the mouse and move toward the edge of the graph, the graph scrolls in the opposite direction. See Figure A-4 for an example of moving nodes "Player::Bet()" and "Deck::Shuffle()".

**Figure A-4** Moving Multiple Nodes



---

## Customizing SoftBench Graphs

The Graph Windows layout and display can be controlled so you can easily reposition nodes and arcs.

### Controlling Graph Layout

You can control the layout of the Data Graph Window from SoftBench Debugger by using the radio buttons at the top of the graph window. You control the layout of the SoftBench Static Analyzer Graphs by setting the "Options:  Full Relayout" toggle button. If you add nodes to the graph while you have the " Full Relayout" toggle button turned off, the tool inserts the new nodes into the existing graph or tree.

### Controlling Graph Display

You can control the display of nodes and arcs in the Data Graph Window by using toggle buttons and push buttons, which control what and when information is displayed. You control display of the Target Graph through the **Display Dependencies** menu button. You control display of the SoftBench Static Graphs through the "View" and "Options " menu. Table A-5 shows the buttons and menu items specific to each graph window. For more information on a specific option, see SoftBench Online Help.

**Table A-5**

#### Controlling the Graph Display

<b>Target Graph Area</b>	<b>Static Graph</b>	<b>Data Graph Window</b>
Display Dependencies menu settings:		



**Table A-5 Controlling the Graph Display**

<b>Target Graph Area</b>	<b>Static Graph</b>	<b>Data Graph Window</b>
<ul style="list-style-type: none"> <li>• Target Only (default)</li> <li>• Intermediate File Dependencies</li> <li>• Source File Dependencies</li> <li>• Include File Dependencies</li> </ul>	<ul style="list-style-type: none"> <li>• all "View" menu commands in a Static Graph window</li> <li>• all "Options" menu commands in a Static Graph window</li> <li>• "Options: Behavior Settings..." in the main SoftBench Static Analyzer window</li> </ul>	<ul style="list-style-type: none"> <li>• <input type="checkbox"/> Show Arc Labels</li> <li>• <input type="checkbox"/> Show Non-Followed Arcs</li> <li>• <input type="checkbox"/> Suspend</li> </ul>

### Understanding Window Status Information

The Static Graph and Data Graph Windows display status information on the length of the current query. Use the **Stop** button to terminate a query. The Target Graph does not display query status information or provide a **Stop** button.

## **For More Information**

- About a menu item or window area, use SoftBench Online Help. Move the mouse pointer over the item and press **F1**.
- About the Target Graph, see “Using the Target Graph” on page 66.
- About the Static Graphs, see “Making Graphical Static Queries” on page 286.
- About using the SoftBench Debugger Data Graph Window, see Chapter 8, “Using SoftBench Debugger Data Graph Window,” on page 235.

# **B** **Customizing SoftBench CM Configuration**

SoftBench CM lets you customize your configuration management environment at the user, system, or global level.

---

## Modifying the Configuration Files

The following configuration files come with SoftBench CM and are usually maintained by the SoftBench CM administrator. However, each SoftBench CM user can configure the default `.fmrc` file. The server reads and acknowledges modifications to these files every 30 seconds. The system writes a record to the `/var/opt/softbench/cm/msglog` file every time it reads a configuration file.

**Table B-1** Server Configuration Files

File Name	Description
<code>cm.mapping</code>	Determines where archive files are stored on the server(s).
<code>cm.permission</code>	Determines user access rights to the archive files.
<code>cm.option</code>	Determines logging and debug levels.
<code>cm.nameperm</code>	Determines user modification rights on symbolic names.

The server configuration files can exist in two different directories on your server:

- `/opt/softbench/config`

When SoftBench CM is installed on your server, SoftBench CM places the configuration files in this directory. The contents of the files in this directory provide global configuration of your SoftBench CM environment. All machines are configured according to the settings in these files.

- `/etc/opt/softbench/config`

To configure a specific system differently from the global configuration, you can copy the configuration files into this directory and make the changes you want. The contents of the files in this directory only affect the system on which they reside.

SoftBench CM looks in both directories for the configuration files, but only uses one. The contents of the files found in the

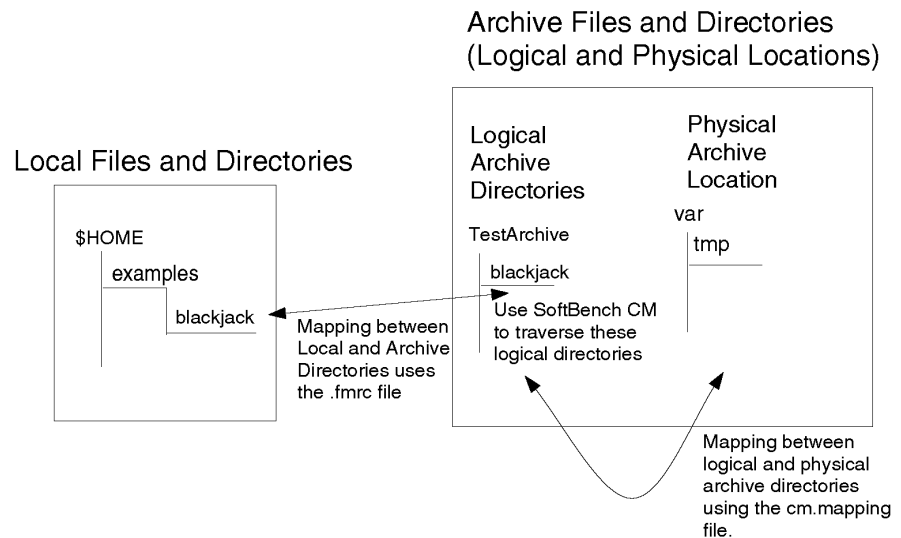
`/etc/opt/softbench/config` directory override the contents of the files found in `/opt/softbench/config` directory. Comment lines in these files begin with a `"#"` character.

## Configuring Where Archive Files are Stored

The `cm.mapping` file determines where archive files are physically stored on the server. This file is created during installation and contains default entries that *MUST* be modified to map to your archive file structure. Together this file and the mapping file determine the relationship between local files and directories and the logical and physical location of archive files and directories (see Figure B-1).

Each line in this file contains two fields separated by tabs or spaces. The first field is the *logical* archive directory prefix. The second field is the corresponding physical directory prefix.

**Figure B-1** File Mapping Between Local and Archive Files



## Customizing SoftBench CM Configuration

### Modifying the Configuration Files

The `cm.mapping` file initially contains the following entry for the `/TestArchive` directory:

```
# TestArchive entry
/TestArchive /var/tmp
```

In this entry, `/TestArchive` is the logical archive path by which the SoftBench CM client and server communicate, and `/var/tmp` is the physical path where the example files are stored on the SoftBench CM server.

Every file managed by SoftBench CM should have exactly one logical path name. You should not map multiple, logical archive path names to the same physical file or directory. When creating or updating the `cm.mapping` file, create the physical directories and set their ownership to `softcm`. Otherwise, the SoftBench CM server may not be able to access these directories. For example, if you add the following entry to the `cm.mapping` file:

```
#logical path  physical path
/project_95    /data/project_srcs
```

you would also need to do the following from the command line:

```
mkdir      /data/project_srcs
chown softcm /data/project_srcs
chgrp 10000 /data/project_srcs
chmod 700   /data/project_srcs
```

Change the group id number for the "chgrp" entry above if you used a GroupID number other than "10000" for the `softcm` entry in the `/etc/passwd` file.

You can edit the `cm.mapping` file while the SoftBench CM server is running. The server notices the changes and updates its internal data structures automatically.

## Defining User Access to the Server

The `cm.permissions` file determines users' access rights to the SoftBench CM archive files. When you make changes to this file, they take effect within 30 seconds of saving the file.

The `cm.permission` file consists of a series of entries. Each entry contains three types of fields terminated by a semicolon, as follows:

```
f1 f2 f3 ... [ f2 f3 ... ] ... ;
```

where f1, f2 and f3 are three distinct types.

- *User@System*
  - *User* is either \* (indicating any user) or a user name.
  - *System* is a `sh(1)` wild card pattern for a [full domain] machine name, or a machine address or address range in dot notation (see *inetd.sec(4)*).
- Permission symbols (see Table B-2).  
 Each option is specified by a lower case letter and each grouping by an upper case letter.
- `/path`.

**Table B-2 Options Used in the `cm.permission` and `cm.option` Files**

Option	Description
a	List the contents of a directory.
b	Show the history of a file.
c	Create a new file.
d	Create a new directory.
e	Rename a directory.
f	Check out a revision.
i	Check in a new revision.
l	Create a lock for later check in.
m	Modify a comment on an existing revision.
n	Create a new symbolic revision name.
o	Obsolete a revision.
p	Modify the mode of a file.
q	Delete a file and all of its revisions.
s	Change the state of a revision.
t	Change the descriptive text for a file.

**Table B-2 Options Used in the cm.permission and cm.option Files**

Option	Description
u	Delete a lock on a revision.
v	Rename a file.
w	Break a lock which is owned by another.
x	Delete a directory.
Y	Move a symbolic revision name.
z	Delete a symbolic revision name.
S	Superaccess: allows all access rights.
R	Read access to archive files Equivalent to "abf".
M	Modify archive files Equivalent to "cdilmnptuwX".
D	Delete archive files, revisions and symbolic revision names. Equivalent to "oqz".
A	Administration: Rename files and directories. Modify symbolic revision name values and state values. Equivalent to "ensvy".
0	Set low priority for this entry.
2	Set high priority for this entry.
3	Set highest priority for this entry.
-	Ignored placeholder.

The permission symbols come in three groups:

- lower case letters control access to individual operations
- upper case letters control access to groups of operations
- numbers set relative priorities for the associated entry

These priorities, coupled with the best match criteria, apply a single set of permissions to a given access.

SoftBench CM normally uses the best matching entry (longest /path



prefix match, then most specific user/host match) to determine access rights. However, by specifying priorities, you can use a shorter path for a specific user. For example, in the following entries:

```
*@host R /earth ;
john@host S / ;
jane@host 2S / ;
```

the user "john" has "R" access to /earth because /earth is more specific than "/" and "S" access to the rest of the archive. The user "jane" has "S" access to the whole archive, including /earth because the "2" specifies an increased priority.

The `/opt/softbench/sbin/checkperm -d` command displays the permissions file entries in sorted order. For any given access, the first entry that matches the requesting `user@host` and `/path` determines the access.

## Recommended Format for Permissions File

You can order fields within an entry in two formats:

### Format 1. Place user field first and permissions field second.

```
user1@host.domain.hp.com
    RM /project1 /project2
    S  /project3      # system admin
    R  /              # read-only default
;
user2@hp* S  /project1
    RM /project2 /project3
;
```

### Format 2. Place path field first and permissions field second.

```
/      R    *@*
;
/project1 RM  user1@host* user2@15.1.1-40.* user3@*
        S   user4@host1.domain.hp.com
;
/project2 RM  user1@host*
        RMD user2@host1*
;
```

Use the pattern `RMDA` or `-abf-cdilnptuw-x-oqz-esvy` for permissions and replace the disallowed permission bits or groupings with "-" characters. This makes it easier to understand which permissions are deleted and given. The permission letters can be in any order.

If the `cm.permission` file is missing, or no match is found, no access is allowed. You should specify a default entry such as:

```
*@15.*      -abf-cdilmnptuw-x-oqz-esvy  /  
            ;
```

## Setting Logging and Debug Options

The `cm.option` file specifies two archive options:

`LogOptions`      The recommended (default) setting for logging is to log every modification operation into the `/var/opt/softbench/cm/activityLog` file. The option letters are position independent (see Table B-2).

```
LogOptions:  abf-cdilmnptuw-x-oqz-esvy  # Logging options (full list)  
LogOptions:  cdilmnptuw-x-oqz-esvy    # Logging options (recommended)
```

`Debug`            Enables writing of debugging information into the file `/var/opt/softbench/cm/msglog`. Debug levels range from 1 to 5. Higher numbers result in more output.

```
Debug: 1
```

Use the `cron` utility to trim the log files periodically because the log files have no maximum file length.

## Controlling Client Machine Access to the SoftBench CM Server on HP-UX

The `inetd.sec` file determines the list of client machines that have access to the archive server machine.

- If `inetd.sec` is missing or does not contain an entry for `softcm`, all users have access to the archive files, subject to the `cm.permission`. For more information, see “Defining User Access to the Server” on page 342.
- If `inetd.sec` exists, but is inaccessible to the `softcm` user, SoftBench CM denies all access.
- Otherwise, the `softcm` service entry determines access according to the standard `inetd.sec` constructs. (See the *inetd.sec(4)* reference page for more information.)

You can edit the `inetd.sec` file while running the archive server. The server notices the change and updates its internal data structures immediately.

## Performing SoftBench CM Administrator Tasks

SoftBench CM requires a few maintenance tasks to ensure a smooth-running configuration management environment.

### Migrating Archive Files From RCS

You can import individual RCS files into SoftBench CM using the `-ARCS` option of `fci`. SoftBench CM can also assume management of an existing hierarchy of RCS files.

To migrate archive files from RCS:

1. Edit the `/opt/softbench/config/cm.mapping` or `/etc/softbench/config/cm.mapping` file so that the desired logical path is mapped to the existing directory hierarchy of RCS files.
2. Make certain that all RCS locks in the hierarchy are removed.
3. Change the owner (and group) of all files and directories in the hierarchy to "softcm".

### Migrating Archive Files From SCCS

To migrate files from existing SCCS archives to SoftBench CM archives:

1. Create the new SoftBench CM archive hierarchy using the `futil -M` command.
2. Use the utility `sccstorcs` to convert the SCCS files to GNU RCS format.

Run `sccstorcs` without any arguments to create the `KeyWordEdit` script. This script lets you decide which SCCS keywords convert to RCS keyword equivalents. Most users use the default keywords in this script.

3. Run `"sccstorcs s.file"` to convert files in the current directory.
4. Run `"find directory -type f -name 's.*' -print | xargs sccstorcs"` to convert files in a directory hierarchy. (See the `sccstorcs(1)` man page for more information on `sccstorcs`)

5. Run `"fci -ARCS file,v"` to create the initial SoftBench CM archive files.
6. Begin using SoftBench CM for file versioning.

This conversion process preserves the history and contents from the SCCS files.

## Modifying the Lockinfo File

The file `/var/opt/softbench/cm/lockinfo` holds all archive file lock information and is created and maintained automatically. The SoftBench CM server accesses this file for all lock or unlock information requests. It is recommend that you *DO NOT* modify this file. If you must modify it, follow these steps:

1. Shut down the SoftBench CM server.  
Failing to shutdown the server could cause data corruption and loss of information.
2. Modify the `lockinfo` file.
3. Run the `fixinfo` program on the `lockinfo` file:  

```
cd /var/opt/softbench/cm  
/opt/softbench/sbin/fixinfo lockinfo outOK outBAD
```
4. If `"bad = 0"` prints, copy the `outOK` file onto `lockinfo`:  

```
cp outOK lockinfo
```
5. If there are bad records, edit the `outBAD` file to try to fix the problems.
6. Run `fixinfo` on the fixed file.
7. Concatenate all the "OK" files together and recheck before restarting the SoftBench CM server process.

## Creating Archive Backups

SoftBench CM requires no special setup or precautions to backup the SoftBench CM archive files, configuration files, or log files. Follow the normal backup procedures for your SoftBench CM server.

## Moving Archive Storage Locations

As a project grows and you add more files to an archive, the archive may

exceed the size of the disk. Several options exist for moving archive locations:

To move an entire archive hierarchy to another disk:

1. Stop the `cmserver` process.
2. Copy or move the hierarchy.
3. Modify the `cm.mapping` file to reflect the new physical location.

To move a portion of an archive hierarchy (subhierarchy) to another disk:

1. Stop the `cmserver` process.
2. Copy or move the hierarchy.
3. Modify the `cm.mapping` file to reflect the new physical location.
4. Add a symbolic link or leave an empty directory in the location from which you moved each subhierarchy.

This lets you browse the parent directory to view the moved subhierarchy.

To move an archive hierarchy to another machine:

1. Stop the `cmserver` process.
2. Copy or move the hierarchy.
3. Modify the `cm.mapping` file to reflect the new physical location.
4. Copy lock information for the portion of the archive hierarchy that you are moving (see “Modifying the Lockinfo File” on page 349).
5. Add a symbolic link or leave an empty directory in the location from which you moved each subhierarchy.
6. Have each user replace or update the `usehost` line in their mapping file to reflect the move.

If the global `fmrc` file `/opt/softbench/config/fmrc` is used, then the system administrator can edit its `usehost` entries for all users. This is usually most practical for sites mounting SoftBench CM from a central server.

All changes should be made such that the logical paths to files are not modified. This way, users are less affected by the move and locks are preserved. Changing the logical hierarchy during a move can cause loss of lock information.

## Troubleshooting

SoftBench CM uses archive log files for quick resolution of problems you may encounter.

### Using the Archive Status and Error Log

The SoftBench CM server daemon keeps the file `/var/opt/softbench/cm/msglog` open in append mode for writing various status and error messages. Look at this file when unexpected events occur.

Log files can grow without bounds if left unchecked. Use an automated process, such as a cron job, to truncate the log files.

Each message in the log files begins with a date stamp. The most common messages include the "Started" and "Terminated" status messages, the configuration file reading messages, and the "Access denied" messages. Some RCS error messages might also show up in the file. These messages are more detailed than the command error messages, so consult the server's `msglog` file if the command error message is not clear.

### Using the Archive Activity Log

The `/var/opt/softbench/cm/activityLog` file tracks modifications made to archive files. The SoftBench CM server writes a line for every activity that has been selected for logging in the `cm.option` file. The `activityLog` file can be moved, truncated, or deleted without affecting SoftBench CM. If deleted, the system recreates the file automatically at runtime.

Customizing SoftBench CM Configuration  
**Performing SoftBench CM Administrator Tasks**



# C Using Regular Expressions

This appendix describes some of the more common types of pattern matching you can perform using Basic Regular Expressions in SoftBench. For the complete list and description of all of the basic regular expressions available, see *regexp(5)*.

Unless you specify otherwise, SoftBench searches for matches anywhere within a line.

## Pattern Matching

Table C-1 describes the following types of pattern matching:

- Match any character.
- Treat a special character as a literal.
- Specify a list of matching values.
- Match the beginning of a line.
- Match the end of a line.
- Match entire lines.
- Exclude specified values.
- Match a range of values.
- Repeat a single character any number of times.
- Repeat an expression any number of times.
- Treat a composite expression within parentheses as a single expression to be repeated any number of times.

**Table C-1**

### **Description of Special Characters in Regular Expressions**

<b>To...</b>	<b>Use...</b>	<b>Example</b>
Match a particular string	<i>match_string</i>	To match the string <i>st</i> , enter:  <i>st</i>  In response, <i>SoftBench</i> matches any string containing the string <i>st</i> (for example, <i>stock</i> and <i>lists</i> ).

**Table C-1 Description of Special Characters in Regular Expressions**

<b>To...</b>	<b>Use...</b>	<b>Example</b>
Match any character	.	<p>To match a string beginning with the characters <code>st</code>, followed by any character, followed by <code>ck</code>, enter:</p> <pre>st.ck</pre> <p>In response, <code>SoftBench</code> matches such strings as <code>stack</code> or <code>stocks</code>. To match a string that includes a period, refer to the description of the backslash character (<code>\</code>).</p>
Treat a special character as a literal.	\	<p>Treats any special character following it as a literal character. For example, to match a string that includes a period, such as <code>st.</code>, enter:</p> <pre>st\.</pre>

**Table C-1**      **Description of Special Characters in Regular Expressions**

<b>To...</b>	<b>Use...</b>	<b>Example</b>
Specify a list of valid matching values.	<code>[valid_values]</code>	<p>To match either <code>stack</code> or <code>stock</code>, enter: <code>st[ao]ck</code></p> <p>To specify the string <code>ab</code>, followed by any digit except <code>5</code>, enter: <code>ab[012346789]</code></p> <p>Note that to treat an opening or closing bracket as a literal string, it must be the first character after the opening bracket. For example, to match all the occurrences of <code>ab]</code> and <code>ab6</code>, enter: <code>ab[ ]6]</code></p> <p>To match all occurrences of <code>ab[</code> and <code>ab6</code>, enter: <code>ab[ [6]</code></p>

**Table C-1**      **Description of Special Characters in Regular Expressions**

<b>To...</b>	<b>Use...</b>	<b>Example</b>
Exclude specified values.	[ <b>^</b> ]	<p>When the caret (<b>^</b>) appears first within the list of bracketed valid values, it is used to exclude all of the characters that follow it. For example, to match <b>ab</b> or any character other than <b>5</b>, enter:</p> <pre>ab[<b>^</b>5]</pre> <p>Otherwise, <b>^</b> represents a literal character. For example, to match <b>ab^</b> or <b>ab5</b>, enter:</p> <pre>ab[5<b>^</b>]</pre>

**Table C-1**      **Description of Special Characters in Regular Expressions**

<b>To...</b>	<b>Use...</b>	<b>Example</b>
Match a range of values	[-]	<p>To match <code>ab</code> followed by any digit, enter:</p> <pre>ab[0-9]</pre> <p>To match any digit except 5, enter:</p> <pre>[1-46-9]</pre> <p>Note that if you need to match the dash (<code>-</code>), do not place it between any two characters that could be interpreted as a range of values. For example, <code>a-z</code> could be interpreted as a range of values, while <code>z-a</code> would not. To avoid confusion, place it as the first or last character in your string. For example, to search for a dash (<code>-</code>), followed by any digit, enter:</p> <pre>[-0-9]</pre>
Match the beginning of a line	<code>^match_expression</code>	<p>To match all occurrences of <code>stack</code> that occur at the beginning of a line, enter:</p> <pre>^stack</pre>

**Table C-1 Description of Special Characters in Regular Expressions**

To...	Use...	Example
Match the end of a line	<i>match_expression</i> \$	To match all occurrences of <code>stack</code> that occur at the end of a line, enter:  <code>stack\$</code>
Match entire lines	^ <i>match_expression</i> \$	The following regular expression would match all occurrences of <code>stack_push</code> as long as it appeared on a line by itself:  <code>^stack_push\$</code>
Repeat a single character	<i>single_character</i> *	To match <code>ab</code> , followed by any number of <code>c</code> 's, followed by a <code>d</code> (for example, <code>abcd</code> , <code>abccd</code> , and <code>abcccd</code> ), enter:  <code>abc*d</code>  Also note that "any number of" includes zero occurrences of a character. Therefore, the string "abd" would also match.
Repeat a <i>[match_string]</i>	<i>[match_string]</i> *	To match <code>ab</code> , followed by any sequence of digits, followed by <code>c</code> , enter:  <code>ab[0-9]*c</code>

**Table C-1**      **Description of Special Characters in Regular Expressions**

<b>To...</b>	<b>Use...</b>	<b>Example</b>
Treat a composite expression within parenthesis as a single expression to be repeated	<i>(composite_expression)*</i>	To match ab followed by a string repeated any number of times (for example, ab1, ab2ab3, ab2ab3, ab4), enter:  <code>(ab[0-9])*</code>



# **D** **Customizing SoftBench for Native Language Support (NLS)**

This appendix describes how to configure your system so that SoftBench correctly handles non-USASCII data. It also tells you how to access human interface localizations if you have purchased a localized version of SoftBench. If you use USASCII data exclusively, you can skip this appendix.

## Preparing to Use NLS in SoftBench

SoftBench was developed to take full advantage of the flexibility of the X11 Window System in supporting non-USASCII text handling and localized human interface preferences. Strings for menu alternatives, button labels, and user messages have been placed in resource (`app-defaults`) files and message catalogs to provide dynamic access and localizability.

You must complete the following steps to use NLS in SoftBench. If you have purchased a localized version of SoftBench, additional installation and customization directions may have been shipped with it. Follow those directions, too. If there are any discrepancies, follow the directions that came with your localized version.

1. Make sure that SoftBench is properly installed.
2. Determine whether or not you need the NLS-related environment variable called `LANG`, and then determine where to set it. “Setting the LANG Environment Variable” on page 364 describes how to do this.
3. Execute the following command:

```
ls -l /opt/softbench/config/types/lang
```

where *lang* is the value you use for the `LANG` environment variable.

If the directory is not found, you will still be able to run SoftBench, but SoftBench will issue a warning message each time it is started. To avoid this warning, you can link in a nonlocalized file.

Execute the following command (as superuser) to link in a nonlocalized directory:

```
ln /opt/softbench/config/types/C /opt/softbench/config/types/lang
```

4. Verify that your X server font search path is set to load the fonts for the character set representation that you will use. SoftBench supports both:

- the Roman8 and ISO 8859.1 encodings for eight-bit languages
- two-byte EUC and Shift-JIS for multi-byte languages

Use the `xset(1)` command to determine and change the font search

path. Be sure that your value of `LANG` and your X server font search path are consistent. See “Setting the `LANG` Environment Variable” on page 364 for more information on `LANG` and character-set representation.

5. Set the `*Scheme` resource using `xrdb` or specify it in your `$HOME/.Xresources` file to indicate the color and font scheme file you wish to use. See the documentation shipped with your localized version of SoftBench to determine the correct value for the `*Scheme` resource.

Note that the default font specifications used by SoftBench support all of the USASCII, Roman8, and ISO 8859.1 characters.

6. Decide whether to remove the conflict between `Extend char` used to enter non-USASCII data and `Extend char` used as "Alt" in SoftBench edit commands. See the "Edit Area Keybindings" topic in SoftBench Online Help to determine which edit commands use "Alt," and see “Rebinding Alt” on page 366 below for instructions on removing the conflict.
7. If you prefer to minimize your use of the mouse, you probably like to use menu mnemonics and keyboard accelerators whenever possible. SoftBench menu mnemonics and keyboard accelerators may prevent the input of non-USASCII characters used in your native language environment. You can customize your SoftBench environment to remove these conflicts. See the “SoftBench Mnemonics and Non-USASCII Character Inputs” on page 367 and “SoftBench Keyboard Accelerators and Non-USASCII Character Inputs” on page 369 sections later in this chapter for details.
8. If you plan to use SoftXEmacs with non-USASCII data, see "Setting Your Meta Key Binding" in SoftBench Online Help for a discussion of your options.
9. Follow any other directions that come with your localized version.

---

## Setting the LANG Environment Variable

You must set the LANG environment variable to make use of native language support. Setting the LANG environment variable directs SoftBench to use the language-sensitive routines for character handling, and will control where X11 applications look for resource (app-defaults) files.

You can set LANG to most of the values supported by HP-UX. The file `/usr/lib/nls/config` contains a list of the supported values. SoftBench supports the ISO 8859.1 character encoding, as well as the Roman8 character encoding. It also supports the EUC (Extended UNIX Code) representation for Japanese, in addition to Shift-JIS for Asian languages. (SoftBench does not support middle-Eastern languages.)

The LANG environment variable determines which encoding your invocation of SoftBench will use. For example:

```
LANG=german  
Specifies the HP Roman8 encoding
```

```
LANG=german.iso88591  
Specifies ISO 8859.1
```

Use the following values for LANG in Japanese environments:

```
ja_JP.SJIS  
SJIS encoding
```

```
ja_JP.eucJP  
EUC encoding
```

LANG can be set to provide for either global (system-wide) or local scope of effect. You should choose the arrangement that is best for your work. The following table describes the possible means of setting the environment variable.

**Table D-1**

To have LANG apply to:	Set LANG in:
All the software you run on your system	<code>\$HOME/.profile</code> if you use <code>sh</code> or <code>ksh</code> <code>\$HOME/.login</code> if you use <code>csh</code> <code>\$HOME/.dtprofile</code> if you use <code>CDE</code> <code>\$HOME/.vueprofile</code> if you use <code>VUE</code>

**Table D-1**

<b>To have LANG apply to:</b>	<b>Set LANG in:</b>
Only the X11 applications that you run	\$HOME/.x11start
Only the current SoftBench session	The command line from which you start SoftBench.

### **Converting from One Encoding Method to Another**

Files saved using one encoding method will not be interpreted properly if LANG specifies another encoding when reading the files. This may be a problem particularly if you display SoftBench remotely. The ISO 8859.1 and EUC encodings are common on non-HP platforms. Use the *iconv(1)* routines to convert files from one encoding method to another.

---

## Rebinding Alt

To replace the Alt keybindings used by the editor with alternate bindings using **ESC**, include the following lines in your `$HOME/.Xresources` file or use `xrdb` to set them:

```
*CodeEdit*extendKey:      esckey
*Edit*extendKey:          esckey
*ListMgr*extendKey:      esckey
```

Note that **ESC** works differently from **Extend char**:

To enter Alt-A using **Extend char**: Press **Extend char** and hold it down while pressing **A**.

To enter Alt-A using **ESC**: Press **ESC**, release it, and then press **A**.

Note also that setting the three `*extendKey` resources shown above replaces only the Alt command bindings used in edit commands. All other uses of Alt, such as mnemonics used for menu operations, continue to require **Extend char**. See the “SoftBench Mnemonics and Non-USASCII Character Inputs” on page 367 section for further instructions.

If you remap the Alt key, you introduce a conflict between **ESC** used as the Alt key and **ESC** used to close a dialog box. To resolve this conflict, change the `osfCancel` virtual keybinding from **ESC** to **CTRL-ESC** by adding the following line:

```
*osfCancel: Ctrl <Key> Escape
```

to your `$HOME/.Xresources` or `$HOME/.motifbind` file.

If your `.motifbind` file (or your `.Xresources` file) does not contain a Motif virtual-keybindings table, add the contents of the file

```
/opt/softbench/newconfig/opt/softbench/config/examples/C1405A_RX.motifbind
```

For more information on `osfCancel` and Motif virtual keys, see the *VirtualBindings(3X)* and *mwm(1X)* reference pages. For more information on SoftBench edit area keybindings, see “Edit Area Keybindings” in SoftBench Online Help.

## SoftBench Mnemonics and Non-USASCII Character Inputs

All SoftBench tools' menus have mnemonics. A mnemonic is a single character used to identify each menu name (the menus are displayed at the top of the tool's main window). The mnemonic is indicated visually by the underlined letter in the menu button. For example, **F** in the "**F**ile" menu is the mnemonic.

For USASCII keyboard users, a menu with a mnemonic can be posted (displayed) by holding down the Alt key and then pressing the mnemonic key simultaneously. For example, Alt-F will post the "**F**ile" menu. However, if you use Asian language input, and therefore have customized your environment using the steps outlined in the previous sections, you will need to press and release the Alt key once, and then press the Alt key plus the mnemonic character simultaneously to post the menu.

If you use non-Asian, non-USASCII character inputs, then you can post the menu using the mnemonics in the same manner as the USASCII keyboard users. However, the mnemonic character may conflict with your non-USASCII character inputs and prevent you from entering certain characters. If a conflict occurs, you need to further customize your SoftBench environment by changing or removing the problem mnemonics. Follow the instructions given below.

### Changing or Removing Menu Mnemonics

1. Bring up all SoftBench tools. Identify all SoftBench menus that have mnemonics that conflict with your non-USASCII character inputs. (You need to check only the menus that appear at the top of the tool and not the menu commands that exist within each menu pane.)
2. For each of the menus you identified in step 1, find the corresponding menu label and menu mnemonic resource definitions. They will either be in a file in

`/opt/softbench/app-defaults`

or in the file

`/opt/softbench/menus/.../C`

3. To change the value of the mnemonic, pick a letter in the menu label that is neither a mnemonic for another top-level menu in the same tool nor used with **Alt** to input a non-USASCII character. To remove the mnemonic, pick a letter that does not occur in the menu label. Use the letter you have selected as the new value of the menu mnemonic resource.
4. Modify the mnemonic definition using your selected new value. Be sure to begin the definition with the tool class name (for example, `Softcm`) even if the tool class was not specified in the original mnemonic definition. Merge the new mnemonic definition into your X resource database through `xrdb` to test the change.
5. Restart the SoftBench tools that contained the conflicting mnemonic(s).
6. Verify that the mnemonic is changed or removed and that your change allows you to input the character that the old mnemonic had blocked.
7. To make the change permanent, append the new resource definition to your `.Xresources` or `.Xdefaults` file.

Removing the mnemonic from a particular menu will disable the use of the **Alt-key** key combination to post that menu, hence enabling you to input your non-USASCII character. However, you will still be able to post that menu using the **F10** key in combination with the keyboard traversal (arrow) keys, then select the menu commands within that posted menu using the mnemonics for those commands. And, of course, you can always choose your menu selections using the mouse.



## SoftBench Keyboard Accelerators and Non-USASCII Character Inputs

Some of the SoftBench tools' menu commands have keyboard accelerators. Accelerators are keystrokes (such as **Alt-Shift-E** for "File: Edit...") that can be used to invoke the menu command without having to pull down the menu itself. These SoftBench keyboard accelerators may conflict with the key(s) used for entering multibyte characters using Input Method Servers. For example, you will not be able to invoke "File: Edit..." using **Alt-Shift-E** if the Alt key is used for accessing the Input Method Server for entering a multibyte character. If such a conflict exists, and you wish to eliminate that conflict, you can do so by customizing the keyboard accelerators directly.

### Customizing Keyboard Accelerators

Keyboard accelerators are defined in the files:

```
/opt/softbench/app-defaults/toolclass  
/opt/softbench/menus/.../C
```

Select the keyboard accelerator resources that are in conflict for each of the tools and modify them to your liking.

For example, suppose you wish to customize all of the SoftBench File Compare tool's keyboard accelerators so that it uses the Function keys, such as **F2**, instead of **Alt-Shift-SomeKey** keystrokes.

1. Copy the appropriate SoftBench File Compare tool's resource specifications into your `$HOME/.Xresources` file:

```
Softcom*fileedit.accelerator:  Shift Meta<Key>E  
Softcom*fileedit.acceleratorText: Shift+Alt+E  
Softcom*filecontext.accelerator: Shift Meta<Key>D  
Softcom*filecontext.acceleratorText: Shift+Alt+D
```

2. Replace the `accelerator` resource with the keystroke you desire, and the `acceleratorText` resource with the actual label you expect to see in the menu pick. For instance:

```
Softcom*fileedit.accelerator:  <Key>F2  
Softcom*fileedit.acceleratorText: F2  
Softcom*filecontext.accelerator: <Key>F3  
Softcom*filecontext.acceleratorText: F3
```

*Choose your accelerator resource values carefully.* Be careful not to select a value for an accelerator resource that conflicts with edit keybindings. For information on SoftBench edit area keybindings, see "Edit Area Keybindings" in SoftBench Online Help. Also be careful not to select a value that conflicts with Motif Text and TextField widgets, nor with the Motif style guide keybindings (such as **F1** for help).

## Starting Your Localized SoftBench

Once all of the customizations explained above have been performed, you are ready to run your localized SoftBench system. You can start SoftBench by typing "softbench" in a shell window.

If your SoftBench system does not operate as you expect, try logging out and logging back in. If SoftBench still fails, check for and correct any configuration errors. If SoftBench still does not operate as you expect, contact your Hewlett-Packard systems engineer for assistance. If you have purchased a fully localized system and have followed the installation and customization instructions for NLS and see non-localized messages as you run SoftBench, report what you were doing and what the messages say (to help determine the cause of the error).

## Remote Execution Hosts and NLS

You can invoke localized SoftBench applications on any remote execution host that has a similarly configured localized SoftBench installation. The value of `LANG` on the invocation host is passed to the remote host when the application is started. The environment variables do not contain any host information, however. Thus, the message catalogs, application resource files, etc., must be in the same locations on both systems unless the `$HOME/.softenv` file is used on the remote execution host to specify a different location.

## Editing in SoftBench

All characters that can normally be entered from an HP keyboard into an HPterm window can also be entered into SoftBench Edit Areas. However, you may need to modify your input method or customize your SoftBench environment.

If you have difficulty entering non-USASCII characters into SoftBench XEmacs, see "Setting Your Meta Key Binding" in SoftBench Online Help.

If you have difficulty entering non-USASCII characters into the remainder of SoftBench, you have two choices. You can either enable the character input each time you need to enter it, or you can enable it permanently by performing the customizations described in "Preparing to Use NLS in SoftBench" on page 362. In making your choice, you should consider the location of **ESC**, and how often and where the key-conflicts occur for your language.

### Character Input Example

On the French keyboard, Alt-W (Alt is the **Extend char** to the *left* of the space bar) enters the ~ character. This conflicts with the editing command "copy-selection". However, pressing **CTRL-Q** before typing the character overrides the command binding and allows the input of the character into the SoftBench edit area. That is, to enter ~ into an edit area on a French keyboard, you would type:

**CTRL-Q** Alt-W



## Glossary

**Accelerator** A key or key combination that performs the same function as a menu choice, but without the need to drop down the menu. It is accessible anywhere within the window. For example, **Shift-Alt-E** is identical to choosing "File: Edit...".

**Alternate Source Root** One or more directories which are the top of a tree for additional source locations. You can use alternate source roots to provide additional directory trees which SoftBench searches to find project files when doing an external build. If your environment maintains identical file structures for different versions of the project, then alternate source roots allow you to put only the files you need in your **local workspace** and find the remaining files in alternate locations. This works like the `VPATH` environment variable for *make(1)*.

**Analysis File Set** Used by SoftBench Static Analyzer, the analysis fileset identifies the set of projects (files and directories when in standalone mode) which identify the Static database files and source files to open.

**Archive** The location in SoftBench CM where file revisions are stored. The archive contains historical information about files, including file revisions, when revisions were made, what changes were made, and who made them.

**Arcs** The lines that connect two nodes in a graph. Normally includes an "arrowhead" to indicate the direction of flow, inheritance, and so on.

**Arrows** Graphical arrows usually seen in scroll areas. Clicking on these arrows scrolls the text. Arrow keys on the keyboard move the cursor in an edit area.

**Breakpoint** A "hook" placed in your executable program by SoftBench Debugger to halt execution at a specific line. Indicated by a breakpoint annotation (⊞ in an octagon) in the Annotation Margin. When your program is paused at a breakpoint, control returns to the debugger, so you can examine variables, modify program status, or perform other operations.

**Buffer** An area in memory used for temporary storage. The edit areas in your configured SoftBench editor and SoftBench Debugger are buffers.

You edit a copy of the original file in this temporary storage area, and the original file is not changed unless you save the edits.

**Build** An action appropriate for projects and targets. When you choose "Build" with a target file selected, SoftBench begins with the source files and transforms them as necessary to create the end result, the target.

With **project build**, SoftBench uses build dependency information and build configuration information to build the target. With **external build**, SoftBench uses the build command that you supply to build the target.

See also **Compile**.

**Build Configuration** A build configuration is a complete set of build instructions to produce a target of a particular type, such as a C++C++ executable, a C shared library, or an Oracle C executable. A build configuration includes libraries, include directories, compiler and compiler options, defines, etc.

SoftBench ships with a basic set of build configurations. Additionally, users can create their own build configurations to support third party or in-house libraries. Each target of a project is associated with a build configuration, but build configurations can be used by many targets, across many projects. You can create and change build configurations in two ways:

- To create new build configurations or make changes that can affect many targets, choose "Builder: Manage Build Configurations...", select the build configuration that you want, and make the changes.
- To make changes that affect only one target, select the target, then choose "Target: Modify Properties...". In the "Modify Target Properties" dialog box, select **Customize Build Configuration....** Changes made through the "Customize Build Configuration" dialog box are unnamed and can be used only by the target associated with the customization.

For example, if most of your C executables are built in a certain way, but one needs a special library, customizing the build configuration may be more efficient than creating another named build configuration for just one C executable. However, once you customize a build configuration, changes to the underlying build configuration have no impact on the



customized build configuration.

To see the complete list of available build configurations, choose "Builder: Manage Build Configurations..." on the SoftBench main window, then explore the drop-down list of build configuration names.

**Check In** The process of moving file changes from the local file system into the archive file system. When you check a file in as unlocked, the local copy of the file is read-only.

**Check Out** The process of retrieving a copy of an archive file into a mapped, local file system. When you check out a file, you can edit the file without affecting any other files. You can check a file out with read-only ("Check Out Unlocked") or read-write ("Check Out Locked") permissions.

**Child Process** A program called from another program. Every HP-UX process except the root process is a child of some other process. When two programs interact, one is usually the parent and one is the child.

**Click** Consecutively pressing a mouse button and releasing it. The click action is used to move the edit cursor (move the pointer to the desired cursor position, and click), and to select a button (move the pointer to the button and click).

**Clipboard** An area in memory used for cutting and pasting text. When you select a cut or copy function in an Edit Area, the text is placed in a Clipboard. When you paste text, the text in the Clipboard is placed into the Edit Area.

**Compile** An action appropriate for source files and object files. When you choose "Compile" with either source or object files selected, SoftBench transforms the source file into the object file.

See also **Build**.

**Configuration Management** A process that lets development teams identify and control changes to the components of their projects. This method of control allows teams to build any saved version of their product in a consistent, repeatable manner.

**Dependency** The relationship between files and targets when the

creation of a target is dependent upon the existence of another file or target.

SoftBench supports three types of dependency relationships: a subproject relationship, a build order dependency between two targets in the same project, and a source-to-target relationship between source files and the target derived from them.

The first two types of dependency relationships both provide a build order dependency. When the dependency crosses projects (through the subproject relationship), you can control whether the subproject is automatically rebuilt with the "■ Build Subprojects" toggle button.

**Double-click** Press and release a mouse button twice in rapid succession.

**Drag and drop** You complete a drag action by pressing and holding down a mouse button while moving the mouse on your desktop (and the pointer on the screen). You complete a drop action by releasing the mouse button after the object has been "dragged" to a new position.

**Edit Area** An editable area used for displaying or entering text data in your configured SoftBench editor and SoftBench Debugger.

**Embedded SQL Source Code** Structured Query Language source code that exists within your source language environment. Preprocessors translate the embedded SQL to source code statements prior to compilation.

**Environment Variable** Also called "Shell Variable", an environment variable is a named variable that is passed to all processes created by the current shell. Your shell stores information about who you are and what you are doing, and some of your preferences.

Some examples of environment variables are:

**PATH** Typically set in your `.login` or `.profile` file, this variable is used to locate executable programs.

**DISPLAY** Tells X11 where to locate the X server process for I/O.

See your shell reference page (*cs(1)*, *sh(1)*, or *ksh(1)*) for information on

setting and reading environment variables.

**Execution Host** The computer on which a process executes. For example, you can specify a remote compile host for building projects or a remote debug host for debugging executables. The remote execution host can be different from the computer that runs the main SoftBench window or SoftBench Debugger. SoftBench must be installed on the remote execution host.

See *Installing SoftBench* for more information on how to configure your system to run your tools over distributed systems.

**External Build** SoftBench's external build model means that users have their own make utilities, such as their own `imake`, `make`, or `nmake` files or scripts. To use external build:

1. Select the "( ) External Build" radio button in the "Create Project" dialog box or the "Modify Project Properties" dialog box. (Choose "Project: New -> Create..." or "Project: Modify Properties" respectively.)
2. In the same dialog box, specify the build directory and default build command for building the entire project.
3. Choose "Builder: Use External Build Command..." in the SoftBench main window to post the "External Build Command" dialog box and provide build instructions for the various targets in the project.
4. Select **Save as Target...** in the "External Build Command" dialog box to save the build instructions for later reuse from the project browser.

Projects that use external build model do not take advantage of SoftBench's **project build** and the supporting build configurations and packages. With project build, SoftBench handles the build process without a need for the user to create and edit a Makefile, and SoftBench provides automatic generation of a Makefile when it is needed.

Both external build projects and project build projects can use the "External Build Command" dialog box to execute shell commands or build software which is not in a project.

**File Revision** A particular instance of a file in a series of changes to

that file. The file and its series of changes are stored in an archive.

**File Server** A computer system that maintains source, version control, or other sets of files. The system can be centralized so that project team members have access to the same versions of common files.

**File Set** SoftBench supports two types of file sets:

- project file set — the set of files included in the project
- analysis file set — the set of files used by SoftBench Static Analyzer

**Files View** The view of your project provided in the main SoftBench window when you select the "Files" tab on the **project browser**.

**Filter** A mechanism to determine which rule violations are displayed. Available only in SoftBench CodeAdvisor, part of C++C++ SoftBench.

**Greyed-Out** A greyed-out menu choice or button is one that is inactive and cannot be selected. It appears in a lighter (half-tone) color.

**Help** Get help by choosing items from the "Help" menu, or by pointing to a screen item of interest with the mouse and selecting the Help key (F1).

If a dialog box containing "There is no help for this item." appears, move the pointer to a nearby area and try again.

**Input Box** A text area which can accept typed keystrokes. Also refers specifically to an area in a dialog box in which you enter text.

For example, when you choose "File: Edit...", the file selection dialog box has two input boxes, one for a file filter and one for a file name. You can type the directory path in the "Filter" input box, then select **Filter**. You can type the file name in the "Edit File" input box, then select **OK**.

**Intercept** A SoftBench Debugger feature that monitors certain events, such as signals, and notifies the user when they occur. Events that can be intercepted include operating-system signals, the loading or removing of images from a program's address space, and the termination of the program. By default, all operating-system signals are intercepted. "Execution: Signals/Intercepts..." allows you to review and modify these settings.

**Intermediate File** A file that is derived from a source file and serves as an intermediate step between the source file and the target. For example, object files (\*.o files) are intermediate files between source files and executable files (targets). Generated source files, such as source files resulting from transforming embedded SQL source, are also intermediate files.

SoftBench determines what intermediate files you need based on the **build configuration** associated with the target.

*Recommendation:* Do not add intermediate files as project files in the project. Let the build process handle the creation and modification of these files.

**Keyboard Focus** The window or window area receiving keyboard events, which is controlled by the window manager. See *Installing SoftBench* for information on setting resources for controlling the keyboard focus policy.

**Local Workspace Root** The local workspace root is the top-level directory of the local workspace, the file system hierarchy where the user places working versions of files for editing and building. Project descriptions include only the relative paths of project files, so that project descriptions can be shared with team members or across the network. Typically, the value of local workspace root changes from one user to the next.

Local workspace root is defined during project creation. Its initial value is the directory in which SoftBench is started. You can change a project's local workspace root by choosing "Project: Modify Properties..." in the SoftBench main window.

**Location** The currently viewed line, procedure, and file for SoftBench Debugger.

**Lock** A mechanism used by configuration management systems to prevent other users from making changes to a file. When you check out a file from configuration management, you can lock it so only you can modify and save the file.

Also a mechanism used to prevent simultaneous access to a project. SoftBench locks projects when they are open and unlocks them when they

are closed.

**Makefile** A control file that specifies rules for building targets. Makefiles can contain the following types of information:

- macro definitions
- file dependency information
- executable commands

Makefiles help you maintain up-to-date versions of projects that result from many operations on a number of files. In SoftBench, you can use **project build**, which frees you from maintaining your own Makefiles, or you can use **external build**, which allows you to use and maintain your existing Makefiles.

**Mapping** The relationship between your local files and directories and the corresponding configuration management archive files and directories. When files are checked out of the archive, writable copies are placed in the local directory that is mapped to that archive directory.

**Menu** A pull-down, graphical selection device consisting of a list of actions or options to select. In SoftBench, there is a menu under each entry in the **menu bar**. These may contain cascading submenus (denoted by "->" in the menu item label).

**Menu Bar** The horizontal line of menus at the top of a SoftBench tool. These are referred to by the word at the head of the menu, such as the "File" menu, the "Edit" menu, or the "Help" menu.

**Mnemonic** A single letter character that provides a shortcut for selecting a menu command from the keyboard. Mnemonics are indicated by underlined characters in the menus. For example "File: Exit..." can be invoked by pressing and holding down the Alt key while pressing the F key, and then pressing the x key.

**Monitor** One of several SoftBench Debugger functions (**breakpoints**, **watchpoints**, **tracepoints**, and **intercepts**) that monitor the progress of a debugged program and notify the user when a predetermined event occurs.

**Nodes** The items of interest in a graph. In the target graph, nodes represent targets, intermediate files, source files, and include files. In Static Graph nodes represent identifiers (variable, procedure, or class

names) and files, and are represented by various shapes in the graphic display. In SoftBench Debugger's Data Graph Window, nodes represent variables.

**Package** A package is a compiler, third-party or in-house library, or utility that is used to build a target. Packages can include library and include directory information, defines and compiler flags, and precompiler or code generator specifications.

Packages provide a short-cut way to use third-party or in-house libraries in many build configurations. First, define the package; then include the package in all appropriate build configurations. If the package's build instructions need to change, you modify the package, and all build configurations which use the package update automatically.

SoftBench ships with a basic set of packages, or users can create their own packages. Examples of shipped packages include Motif, X11R5, Encapsulator, RogueWave, and Oracle. To see the complete set of shipped packages choose "Builder: Modify Packages..." in the SoftBench main window, then explore the drop-down list of package names.

**PC** Program location where execution stopped in SoftBench Debugger. The statement at this location will, by default, be the next statement to be executed when execution resumes.

PC stands for Program Counter.

**Popup Menus** Menus which are displayed by selecting mouse button 3 (usually the right mouse button) over a designated area of the screen. All graphs and the Builder and CodeAdvisor output browser contain popup menus.

**Primary Selection** Several lines of text highlighted by using the left mouse button and dragging it over the desired text. This works in any X application. You can then paste this text in an Editor window using the Shift-Middle mouse button combination.

**Project** A project is a named set of files and one or more sets of build instructions and dependencies that produce a set of related targets. A project is always an entity to itself, that is, it does not dynamically inherit options or build information from any other project. Projects are related by the parent/subproject build dependency relationship. Project

names contain regular characters. Control characters, spaces, and punctuation characters special to the shell are not allowed in project names.

**Project Browser** The area in the main SoftBench window that displays project data. In the **Projects view**, the project browser displays projects and subprojects. In the **Files view**, the project browser displays files and targets in the current project.

**Project Build** A project build means that users let SoftBench manage their build instructions. Users specify source-to-target dependencies, target build order dependencies, and **build configurations** for building the targets. SoftBench manages this information, builds the targets in the project, and generates the supporting Makefile, if the user requests it.

Some users have a highly-tuned build process and may not want to transition to the project build model. For that reason, SoftBench also supports an **external build** model, allowing you to use your existing build process within SoftBench.

**Project Description Data** Project description data is SoftBench-created information that describes a project, its file set, its build configurations, options, and settings. Description files in a project root should not be edited by the user, except in rare instances such as creating a new **transform**.

See also **Project Root**.

**Project Root** The project root is the file system location where SoftBench saves and retrieves **project description data**. The default project root is the place where all new project descriptions are written. (User source and target files are found in the **local workspace root**, not the project root.

The default value for project root is `$HOME/.softbench`. You can change the default value by choosing "Options: Set Default Project Root..." in the SoftBench main window.

**Projects View** The view of your project provided in the main SoftBench window when you select the "Projects" tab on the **project browser**.



**Push Buttons** Independent buttons found in top-level tool windows and in dialog boxes, which when pushed initiate frequently used system actions.

**Query** A request for cross-reference information about some part of your project in SoftBench Static Analyzer.

**Radio Buttons** A graphical user interface construct consisting of several buttons representing several choices. Only one button may be selected at any time. When a button is selected, all other buttons are automatically deselected.

**Revision history** A file kept by SoftBench CM that contains information about the content of each file revision, as well as the author, check-in date, check-in time, and a log message for each change.

**Scope** The region of source code over which a name's declaration is active. Scoping is the ability to uniquely identify a program identifier in SoftBench Static Analyzer based on its source code position.

**Signal** A software interrupt sent from the operating system to a program. This can inform the program of any asynchronous event. Signals are used for segment violation, divide by zero, or other hardware problems, and they can be sent as a job control mechanism (stop, continue, kill).

**Single Step** To execute a program one statement or instruction at a time, to allow you to look at the values of variables and other information between steps in SoftBench Debugger.

**SQL** Structured Query Language, the defacto standard database query language used to perform operations on a Relational Database Management System.

**SQL Preprocessor** A script or utility that converts Structured Query Language source lines in your code to the native language you are using in your application. This script or utility usually is provided by the database vendor.

**Static Database** A file that contains cross-reference information that SoftBench Static Analyzer requires to answer queries. The `Static.sadb` file is generated by a parser when the `Static compile` mode is set during

a build, or during an "Analyze File Set" operation.

**Subproject** A subproject is a **project** with a defined build dependency relationship to a parent project. Examples of subprojects include:

- a project to build a library that will be linked into a parent project executable
- a project to build an executable used to generate files or targets in the parent project

Subprojects are projects in their own right, and do not inherit options or build instructions from the parent project (except at creation time if they are created through the "File: Convert to Project..." process). A subproject relationship does not imply "inclusion"; that is, the file set of the subproject is not a subset of the parent project file set.

**Symbolic Name** A special name or code for a file that associates it with a certain release. If subsequent revisions of this file form a branch, the symbolic name applies to the entire branch. Also known as username or tag.

**Synchronous Mode** The program interacts with the X Window System in such a way that events are not buffered, but are written/read as they happen.

**Target** A project file that is the end product of a build; for example, a linked executable, shared library, archive library, or message catalog.

Targets share the following characteristics:

- A target belongs to only one project. You can, however, use the same target name in more than one project.
- SoftBench builds targets in no specific order unless you create a build order dependency between targets.
- You can (and usually do) define targets before they exist on the file system.
- Targets have a physical file system location, usually the **local workspace root**, which can be changed by modifying the target's properties.
- Project build targets have an associated build configuration which provides build rules and tool interactions.
- Project build targets have source-to-target dependencies that you create by choosing "File: Link Source to Target...". The target is

- derived from its associated source files.
- External build targets have an associated build command which can produce the target.
- Targets are the end product of a build, as distinguished from **intermediate files** (.o files or generated .C files) which are produced as an intermediate step in the build.

Not every target is an executable file, and not every executable file in a project is a target. SoftBench treats files that are added via the "Project: Add File(s) to Project..." operation as files. It treats files that are added via the "Target: New..." operation as targets.

**Toggle Buttons** Independent buttons which set the (off|on) state of selected options. If selected, the button remains selected until you press the button again.

**Token** A unit of text, delimited by spaces or punctuation. Within text, a token is a "word". Within source code, a token is an "identifier" or a "constant".

**Tracepoint** A SoftBench Debugger feature that "traces" the execution of a program, notifying the user when certain points in the program are executed.

**Transform** A transform takes a specific type of file, processes it, and produces a new type of file. A transform is the basic building block that allows SoftBench to build a target. Transforms are used by **packages** which are in turn used by **build configurations**.

**Unlock** You can unlock files when you check them into the archive. Unlocked archive files are available for check out by any user with the appropriate permissions. If you remove the lock on a locked file and do not check in the file, you will cancel any changes you made to that file.

**Update** The process of retrieving the most current set of archive files onto your local system. This process updates your mapped, local file system so that it matches the most current version files.

**Version** One instance of a file in a series of changes A particular instance of a file in a series of changes to that file that are stored in the SoftBench CM archive.

**Watchpoint** A debugger **monitor** that "watches" the value of a variable

or memory range, and notifies the user if the value changes. This is especially useful if variables in your program change value unexpectedly.

**Working Directory** The directory in which your SoftBench processes run. In most cases, SoftBench uses the **local workspace root** as the working directory. You can override this location for a target by changing the target's properties. You can change the target's physical file system location, or you can set a "Working Directory" value in a target's execution and debug properties.

To change either of these values, select the target, then choose "Target : Modify Properties..." in the main SoftBench window.

**X Resources** File entries that allows you to customize an X application environment. Each resource usually consists of the application name, the resource name, and the value to be assigned to the specific resource.

**Symbols**

\$HOME/.softbench/bmsinit, 80  
 \$HOME/.softbench/projectname.msglog, 311  
 ( ) input box, 160, 182  
 ( ) Silent, 194, 202  
 ( ) Verbose, 194, 202  
 ), debugging, 175  
 .softdebugrc file, 228  
 .underscorevfp, 243  
 .underscorevptr, 243  
 .Xdefaults file, 176  
 `Messages menu  
   Send Message, 309

**A****Actions menu**

Cancel Check Out (Discard Changes), 103  
 Check In from Local Directory, 94, 110  
 Check Out to Local Directory, 93  
 Check Out to Local Directory Check Out,  
   103  
 Check Out to Local Directory Locked, 103  
 Check Out to Local Directory Unlocked, 103  
 Create New Mapping, 93, 98  
 Show Local Server Information, 92, 95  
 Show Set Up Instructions, 92  
 Show/Modify Mappings, 99, 113  
 Update to Local Directory Current  
   Directory Only, 104  
 Update to Local Directory Recursive  
   (Directories Only), 104  
 Update to Local Directory Recursive (Files  
   and Directories), 104  
 Activate All (Break menu), 194  
 Add Base Class(Edit menu), 139  
 Add Build Order Dependency (Target menu),  
   61  
 Add Existing Source Files, 53  
 Add File(s) to Project (Project menu), 38, 51,  
   53, 58, 64  
 Add File(s) to Project(Project menu), 94  
 Add Source Directories (File menu), 168, 169  
 Adding  
   mappings, 100  
   source directories in Debugger, 169  
   targets to a project, 51, 53  
   tools to toolbar, 79, 80  
 Administering SoftCM, 339  
 Adopting  
   a process for debugging, 222

Analysis file set, 255, 257  
   customizing, 255, 256  
   including parents and subprojects, 255  
   multiple projects, 255  
   status, 257  
   synchronizing, 256  
 Analyze File Set (File menu), 255  
 Annotation Margin, 161, 189, 190, 194, 211  
 app-defaults, 364  
 Archive, 90  
   backup, 349  
   breaking file locks, 107  
   browsing, 95  
   browsing archive files, 95, 102  
   cancelling file check out, 103  
   checking in files, 110  
   checking out archive files, 102  
   creating archive directories, 97  
   creating initial files, 102  
   customizing the display filter, 109  
   deleting archive files or directories, 106  
   description, 90  
   directory, 97  
   filtering display, 109  
   location, 100, 341, 349  
   locking archive files, 106  
   mapping, 90, 93, 98  
   moving, 349  
   permissions, 342  
   sorting display, 109  
   specifying file revisions, 113  
   unlocking archive files, 107  
   updating local directories, 103  
   viewing archive file content, 104  
   viewing locked files, 109  
   viewing revision history, 107  
 Archive server, 90, 95  
   accessing, 346  
   deleting, 101  
   modifying, 100  
 Arcs  
   Data Graph Window, 240  
   selecting, 332  
 Arguments, in Debugger, 167  
 Assembly  
   instructions window, 212  
   language, 211, 218  
   level breakpoints, 211  
   registers, 213  
 Assembly Instructions (Show menu), 211, 218

---

# Index

## B

- B (Breakpoint) annotation, 161, 189, 190, 193
- Backup
  - archives, 349
- Behavior Settings (Options menu), 257, 266, 267
- Block
  - specifying, 177
  - traces, 204
- bmsinit, 80, 120
- Boldface font, 5
- Break File Lock(File menu), 107
- Break menu, 191, 192, 197
  - Activate All, 194
  - Clear All, 195
  - Set, 192
  - Set at (), 191
  - Set at Class (), 197
  - Set At Hex Address (), 211
  - Set at Instance (), 197
  - Set at Overloaded (), 197
  - Set at Procedure Entry (), 192
  - Set at Template (), 197
  - Show, 189, 193
  - Suspend All, 194
- Breaking archive locks, 107
- Breakpoints, 188, 189
  - assembly level, 211
  - changing, 193
  - clearing, 194
  - commands at, 192, 195
  - Cplusplus, 197
  - debugging with, 189
  - exception handling, 225
  - listing, 189
  - on threads, 193
  - setting, 190
  - setting from Static Analyzer, 290
  - setting from your editor, 131
  - setting group, 197
  - setting in annotation margin, 162
  - SQL, 319
  - threads, 221
  - viewing, 193
  - viewing and modifying, 197
- Broadcast Message Server, 305
- Broadcast Messages area, 307
- Browser submenu
  - Find String, 75
  - Load Browser from File, 75
  - Print/Save Output, 75
- Browsing and fixing errors, 74
- Browsing archive servers, 95
- Browsing archives, 95
- Buffer
  - in XEmacs Editor, 122
- Build
  - changing type, 56
  - dependencies, 38, 51, 55, 61
  - error browsing, 74
  - handling errors, 73
  - options, 70
  - order, 55
  - package, 63
  - preview, 72
  - project, 38, 70
  - remote, 75
  - selected target, 69
  - server, 75
  - starting, 44
  - subproject, 44, 55, 73
  - targets and projects, 70, 75
  - troubleshooting, 86
  - vs. compile, 73
- Build (File menu in Debugger), 39
- Build (Target menu), 69, 70
- Build configuration, 22, 23, 38
  - customizing, 61, 62, 63
  - modifying, 52
  - selecting, 60
  - SQL, 315
- Build control area, 44
- Build model
  - changing, 56
  - external build, 39, 50, 52, 64
  - project build, 38, 50, 60
- Build package, 24, 62
- Build Project (File menu), 130
- Build Settings (Options menu), 75
- Builder, 32
- Builder menu
  - Browser, 74
  - Manage Build Configuration, 62, 71
  - Manage Package Information, 63
  - Use External Build Command, 72
- Builder page, 46, 47
- Building
  - for debugging, 159
  - from your editor, 130
  - with Static data, 254

- Button area, 44
- Buttons, 162
  - customizing, 228
- C**
- Call graph, 284
- Call stack, 186
- Calling functions, 186
- Cancel Check Out (Discard Changes)
  - (Actions menu), 103
- Case sensitivity, 178
- CDE
  - dragging files, 77
  - integrating with SoftBench, 82
  - starting SoftBench from, 43
  - workspaces, 36
- Change Working Directory (File menu), 167
- Changing
  - build type, 56
  - data, 181
  - source code, 207
  - variables, 181, 183
- Character constants, 185
- Characters, non-USASCII, 367
- Check In from Local Directory (Actions menu), 94, 110
- Check Out to Local Directory (Actions menu), 93
- Checking code, 39, 44, 76, 150
- Checking in
  - files to archive, 110
  - symbolic names, 112
- Checking out
  - archive files, 102
  - symbolic names, 113
- Child process, 222
- Class
  - break, 197
  - commands, 224
  - creating, 139
  - deleting, 140
  - editing, 134, 140
  - hierarchy, 138
- Class Graph/Editor, 32, 133, 284
  - analysis data, 134, 135, 136
  - calling your editor, 140
  - configuration management, 142
  - creating classes, 139
  - data unsynchronized, 135
  - deleting components, 140
  - editing classes, 139
  - editing components, 140
  - starting, 138
  - synchronizing, 136
  - templates, 141
  - troubleshooting, 143
  - undo list, 136
  - use model, 134
  - with other editors, 136
- Classes (Show menu), 260
- Classification () (Symbol menu), 263
- Clear All (Break menu), 195
- Clear All (Trace menu), 206
- Clear All (Watch menu), 203
- Clearing Graph Area, 331
- Client-server architecture, 90
- Clock icon, 46, 190
- Cloning projects, 27, 53
- Close, 46
- Close (Project menu), 83
- Code understanding, 39
- CodeAdvisor, 32, 39, 145
  - accessing, 150
  - checking programs, 150
  - file set, 151
  - from command line, 149
  - page, 46, 48, 76
  - preparation, 149
  - rule groups, 150
  - rules, 147
  - SQL, 321
  - starting, 76
  - terminating, 152
  - use model, 149
  - violations, 151
  - with external build, 149
- Collapsing group breakpoints, 197
- Colors
  - on target graph, 68
- Command line, 36, 115
  - starting SoftBench from, 43
- Compare (File menu), 296
- Comparing files, 295, 300
- Compatibility
  - regenerating Static data, 254
- Compile
  - errors, 73
  - files, 73
  - mode options, 44, 70
  - server, 75
  - vs. build, 73

---

# Index

- Compile (File menu), 73, 130
  - Compile mode
    - external build, 72
    - override, 71
  - Compile options, 159
    - changing, 71
  - Compiler flags
    - defining, 71
    - external build, 72
  - Compiling
    - for debugging, 159
    - from your editor, 130
  - Computer font, 5
  - Configuration files, 340
  - Configuration management, 27
    - choosing tool, 80
    - Class Graph/Editor, 142
    - from Debugger, 207
    - from your editor, 131
  - Configuration Management submenu (File menu), 38, 131
  - Configuring
    - an editor, 119
    - Program Editor, 119
    - SoftCM, 92
    - toolbar, 79
    - vi Editor, 119
  - Constants, 185
  - Controlling Graph Layout, 336
  - Convert to Project (File menu), 54
  - Converting files to a project, 54
  - Copy
    - block of text in vi Editor, 129
    - data between tools, 37
    - in vi Editor, 129
    - multiple files in vi Editor, 129
  - core file, 217
  - Correcting errors, 207
  - Cplusplus Class Graph/Editor, 133
  - Cplusplus debugging, 223
  - Cplusplus Settings (Options menu), 225
  - CPU Registers, 213
  - Create Class(Edit menu), 139
  - Create Data Member(Edit menu), 139
  - Create Member Function(Edit menu), 139
  - Create New Mapping(Actions menu), 93, 98
  - Create(Directory menu), 93, 97
  - Create(File menu), 93, 102
  - Creating
    - a mapping, 98
    - an archive directory, 97
    - archive files, 102
    - files with your editor, 121
    - files within a project, 57
    - projects, 38, 50, 55
    - subprojects, 55
  - Current Location, 161, 178
  - Current project, 35
    - area, 44
    - in title bar, 36
  - Custom menu
    - Edit Menus, 77, 78
  - Customizable buttons in Debugger, 162
  - Customize Build Configuration, 52
  - Customize Build Configuration dialog box, 38
  - Customize File Set (File menu), 255, 256
  - Customizing, 339
    - build configuration, 52
    - buttons, 228
    - Debugger, 228
    - environment, 78, 83
    - popup menus, 228
    - SoftBench, 81
    - SoftBench Graphs, 336
    - Static Analyzer, 275
    - Static Graph, 294
    - tool preferences, 80
  - Cut
    - multiple files in vi Editor, 129
- ## D
- d compiler option, 197
  - Data
    - changing, 181, 183
    - examining, 181, 182
  - Data Graph ( ) (Show menu), 236
  - Data Graph Indirect ( ) (Show menu), 236
  - Data Graph Window, 235
    - arcs, 240
    - Display Control Area, 238
    - embedded arrays, 248
    - exiting, 236
    - general graph operations, 323
    - Graph Area, 238, 241
    - Layout Control Area, 237
    - node values, 247
    - nodes, 239
    - popup menus, 239
    - starting, 236
    - suspending, 244
    - use models, 245



- Window Control Area, 240
- Data members
  - displaying, 243
  - setting values, 243
  - viewing values, 247
- Data Value Show" menu), 183
- DDE
  - Busy, 161
  - communication, 162
  - Reference, 155
  - variables, 183
- DDE commands, 181
  - declare, 180
  - list decl, 184
  - on breakpoints, 192, 195
  - on buttons, 229
  - on intercepts, 215
  - on traces, 204, 205
  - on watchpoints, 200
  - print, 184
  - prop lang, 184
- Deadlock, 175
- Debug (Target menu), 39, 77
- debug compiler option, 197
- Debug Running Process (File menu), 222
- Debugger, 155
  - adding source directories, 169
  - breakpoints, 189, 193
  - calling functions, 186
  - class commands, 224
  - configuration management, 207
  - core file, 217
  - Cplusplus, 223
  - Current Location, 161
  - customizing, 228
  - debugger variables, 183
  - dynamic libraries, 209
  - edit area, 207
  - exception handling, 224
  - forked processes, 219
  - inline functions, 197
  - interrupting a program, 172
  - loading executable, 165
  - main toolface tear apart, 163
  - nested classes, 224
  - no source available, 172
  - object identification, 224
  - optimized code, 227
  - parameterized types (templates), 224
  - preparing a program, 159
  - program interaction, 174
  - redirecting I/O, 175
  - running, 39
  - running a program, 165, 190
  - runtime environment, 165
  - setting breakpoints, 193
  - setting breakpoints from Static Analyzer, 290
  - signals and events, 214
  - source languages, 184
  - specifying location, 177
  - SQL, 319
  - starting editor, 207
  - stepping through a program, 171
  - stop after count, 195
  - synchronizing source files, 208
  - terminal I/O, 175
  - threads, 220
  - undebuggable code, 171, 172
  - variable specification, 178
  - watchpoints, 199
  - window areas, 160
  - with external build, 169
  - with project build, 169
  - X windows, 175
- Debugging
  - expanded SQL source, 320
  - from your editor, 131
  - running programs, 222
  - SQL, 319
- debugui.buttons file, 229
- Declaration of an identifier, 262
- Default Load/Rerun Settings (Options menu), 165, 167, 168, 175
- Define Targets, 51, 53
- Definition of an identifier, 262
- Delete(Directory menu), 106
- Delete(File menu), 106
- Deleting
  - a mapping, 101
  - a query's history, 271
  - archive files and directories, 106
  - archive servers, 101
  - breakpoints, 194
  - Class Graph/Editor components, 140
  - traces, 206
  - watchpoints, 202
- Dependencies
  - defining, 68
  - showing, 61, 68

---

# Index

- source-to-target, 51, 61
  - target-to-target, 61
  - Description Boxes, 289
  - Directory
    - archive, 97
    - working, 167
  - Directory menu
    - Create, 93, 97
    - Delete, 106
  - Display Filter(Options menu), 109
  - Display legend
    - Static Graph, 287
    - target graph, 328
  - Display on Graph (Target menu), 66
  - Display Results (View menu), 266
  - Distributed data, 81
  - Dynamic libraries, debugging, 209
- E**
- Edit Area syntax, 126
  - Edit menu
    - Add Base Class, 139
    - Create Class, 139
    - Create Data Member, 139
    - Create Member Function, 139
    - Modify, 140
    - Undo, 140
    - Update Buffer, 208
  - Edit Menus (Custom menu), 77, 78
  - Editing
    - Cplusplus classes, 134, 139
    - Cplusplus components, 140
    - Cplusplus structures, 134
    - fixing build errors, 74
    - graphical, 134
  - Editor
    - accessing Static Analyzer from, 130
    - alternate, 119
    - building from, 130
    - calling from Class Graph/Editor, 140
    - calling other tools, 130
    - checking files in and out, 131
    - choosing tool, 80
    - Class Graph/Editor, 133
    - compiling from, 130
    - configuration management from, 131
    - configuring, 119
    - debugging from, 131
    - multiple files in vi Editor, 127
    - multiple files in XEmacs Editor, 124
    - starting, 121
    - starting from build error, 74
    - starting from Debugger, 207
    - starting from Static Analyzer, 253, 264
    - starting from Static Graph, 287
    - starting from target graph, 69
    - synchronizing multiple, 136
    - within Debugger, 207
    - XEmacs Editor, 122, 126
  - Ellipses, 5
  - Embedded arrays, 248
  - Enable Images/Libraries (Execution menu), 209
  - Encapsulation, 80
  - Entering identifiers, 260
  - Environment
    - customizing, 78, 83
  - Environment variables, 167
    - LANG, 362, 364
    - PATH, 42
  - Error
    - browsing, 47, 48, 74
    - correcting, 207
    - handling, 73
    - interpretation, 74
    - invoking editor, 74
    - source of, 48
  - Error messages
    - build, 86
    - File Compare, 302
    - in Class Graph/Editor, 143
    - in Debugger, 231
  - Events, debugging, 214
  - Examining
    - data, 181, 182
    - variables, 181, 182
  - Exception handling, 224, 225
  - Executable
    - loading, 165
  - Execution
    - breakpoints, 189, 193
    - host, 81
    - traces, 204
    - watchpoints, 199
  - Execution menu
    - Enable Images/Libraries, 209
    - Get Current Program Info, 167
    - Signals/Intercepts, 214
    - Threads, 187, 220
  - Exit (File menu), 83

- Exit SoftBench (Project menu), 83
- Expanding group breakpoints, 197
- Expressions
  - printing, 182
  - using, 184
- External build, 26, 39, 50, 52
  - changing to project build, 56
  - CodeAdvisor, 149
  - Debugger, 169
  - defining targets, 64
  - makefile, 64
  - SQL, 317
- F**
- File
  - adding existing to project, 51, 53, 58
  - adding groups to project, 58
  - check in, 110
  - check out, 102
  - comparing, 295, 300
  - creating in project, 57
  - creation with editor, 121
  - dependencies, 51
  - icons, 59
  - in XEmacs Editor, 122
  - linking sources to targets, 61
  - mapping, 168
  - merging, 295, 301
  - open with editor, 121
  - specifying, 177
  - synchronization, 208
  - version contents, 90
  - versioned, 90
- File ( ) (Visit menu), 191
- File Compare, 295
  - difference indicators, 297
  - gutter, 297
  - menu bar, 296
  - merge indicators, 298
  - merge selections, 298
  - specifying files, 296, 297
  - troubleshooting, 302
  - window, 296
- File graph, 284
- File menu
  - Add Source Directories, 168, 169
  - Analyze File Set, 255, 257
  - Break File Lock, 107
  - Build Project, 130
  - Change Working Directory, 167
  - Compare, 296
  - Compile, 73, 130
  - Configuration Management Check Out
    - Locked, 110
  - Configuration Management submenu, 38, 131
  - Convert to Project, 54
  - Create, 93, 102
  - Customize File Set, 255, 256
  - Debug Running Process, 222
  - Delete, 106
  - Exit, 83
  - Free Running Process, 222
  - Iconify Windows, 164
  - Link Source to Target, 38, 51, 61
  - Load Corefile, 217
  - Load Executable, 165, 176
  - Lock File, 106
  - New, 38
  - Normalize Windows, 164
  - Open, 38, 64, 110
  - Rerun, 165, 175
  - Save, 57, 58, 118
  - Save As, 118
  - Save Out Of Project, 118
  - Save Out Of Project As, 118
  - Set Break Point, 131
  - Set Logfile Name, 311
  - Show Revision History, 105, 107
  - Static Analysis, 252
  - Unload Executable, 37
- File menu in Debugger
  - Build, 39
- File Set
  - status, 257
- File set
  - customizing for non-project, 256
  - default, 255
  - filtering, 257, 266
  - information area, 258
  - non-project mode, 256
  - searching subdirectories, 273
  - status, 258
  - updating, 256, 264
  - updating data without building, 255
- File types
  - add files by type, 59
  - SQL, 315

---

# Index

- use new types, 64
- File" menu
  - Load New Executable, 37
- Files view, 44, 59
- fileSetFile resource, 274
- Filter Results (View menu), 266, 268
- Filtering
  - archive display, 109
  - buttons, 48
  - Cplusplus queries, 268
  - queries by file set, 257, 266
  - query results, 266
  - sourceless nodes, 293
  - Static Graphs, 292
  - violations, 152
- Filters (View menu), 292
- Find Node (Graph menu), 287
- Find String (Output browser menu), 75
- Follow All Recursively, 241
- Follow Recursively, 243
- Font
  - boldface, 5
  - computer, 5
  - italic, 5
  - typewriter, 5
- fork, 219
- Fork Behavior (Options menu), 219
- Forked processes, debugging, 219
- Frame
  - in XEmacs Editor, 122
- Free Running Process (File menu), 222
- Function
  - calling, 186
- Functions (Show menu), 260
- G**
- Get Current Program Info (Execution menu), 167
- Global Variables (Show menu), 260
- GNU RCS versioning system, 90
- Granularity
  - traces, 204
  - watchpoints, 199
- Graph menu
  - Class EditorAll Classes, 138
  - Find Node, 287
  - Hide commands, 69
  - Save Image, 291
  - Update Status of Nodes, 68, 72
- Graph Windows, 323
  - clearing, 331
  - customizing, 336
  - display control, 336
  - display dependencies, 336
  - layout, 336
  - moving nodes and arcs, 334
  - nodes, 332
  - popup menus, 326
  - printing the image, 326
  - saving the image, 326
  - scrolling, 330
  - selecting nodes and arcs, 332
  - starting, 324
  - targets and dependencies, 66, 69
  - zooming, 331
- Graphical queries, 252
- Group breakpoints
  - setting, 197
  - viewing and modifying, 197
- H**
- Handling events, 214
- Handling signals, 214
- Help, 84, 85, 124, 126
- Help menu
  - On Item, 85
  - Show Man Page, 85
  - Tool Overview, 85
  - Using Help, 85
- Hex, printing, 183
- Hide
  - node, 288
  - violations, 152
- Hide commands (Graph menu), 69
- History menu, 271
- HP-GL
  - saving images, 328
- HP-GL file format, 326
- I**
- Iconify Windows (File menu), 164
- Iconize Project button, 36, 44
- Icons
  - changing on toolbar, 79
  - for file existence, 59
- Identifier
  - classification, 262
  - entering, 258, 260
  - invalid, 263
  - relationships, 261

- Identifier Matching Rules (Options menu), 263
- Images
  - saving PostScript and HP-GL, 328
- Index
  - vi Editor, 128
- Informix, 314
- Inherited values, 225
- Input, 174
- Instance break, 197
- Integration, 32
  - calling other tools from editor, 130
  - with CDE, 82
- Intercepts, 214
- Intermediate file, 58
  - access to commands, 67
- Interpreting error messages, 74
- ISO 8859.1 characters, 363
- Italic font, 5
  
- J**
- Japanese, customizing SoftBench for, 362
  
- K**
- Kernel code
  - debugging in, 172, 222
- Keycaps, 5
  
- L**
- LANG variable, 362, 364
- Language, source
  - in Debugger, 184
- Learning SoftBench, 40
- Library, system, 172
- libsoftbench, 305
- Line number
  - specifying, 177
- Link Source to Target (File menu), 38, 51, 61
- Linking files, 61
- Load Browser from File (Output browser menu), 75
- Load Corefile (File menu), 217
- Load Executable (File menu), 165, 176
- Load New Executable (File menu), 37
- Local workspace root, 22, 50
  - changing, 56
- Location
  - specifying, 177
- Lock File (File menu), 106
- Lock project, 35
- Locking archive files, 106
  
- Locks
  - breaking, 107
  - viewing locked files, 109
- Log file, 351
  - changes to archive files, 107
  - configuring, 346
- Logging messages, 311
  
- M**
- Machine
  - instructions, 211
  - registers, 213
- main(), 179
- Make Subproject Of (Project menu), 55
- Makefile, 64
- Man page
  - vi Editor, 126
- Manage Build Configuration (Builder menu), 62, 71
- Manage Package Information (Builder menu), 63
- Managing environment, 78, 83
- Mapping
  - adding, 100
  - creating, 98
  - deleting, 101
  - file, 99
  - local and archive directories, 98
  - modifying, 100
  - symbolic name list, 113
  - view list, 99
- Mappings, 168
- Menu
  - bar, 44, 296
- Merging files, 295, 301
- Message format, 310
- Message Monitor, 305
  - configuring on toolbar, 306
  - logging messages, 311
  - message format, 310
  - sending messages, 309
  - starting, 306
  - window, 307
- Migration
  - RCS, 348
  - SCCS, 348
- Mnemonic, 367
- Modify Properties (Target menu), 38, 52, 63, 71, 165
- Modify(Edit menu), 140
- Modifying

---

# Index

- an identifier, 262
- archive server, 100
- breakpoints, 193
- mappings, 100
- projects, 56
- watchpoints, 201
- Module
  - specifying, 177
- More Build Actions submenu (Target menu), 72
- More File Types button, 64
- Moving archives, 349
- Multiple files
  - vi Editor, 127
  - XEmacs Editor, 124

## N

- Native Language Support, 362
- Nested classes, 224
- Networked environment, 81
- New (File menu), 38
- New (Target menu), 39, 51, 53, 60, 64
- New Clone (Project menu), 53
- New Create (Project menu), 38, 51
- NLS, 362
- Node Values dialog box, 241
- Nodes, 239
  - Data Graph Window, 239
  - displaying more information, 289
  - displaying new, 241
  - hiding, 69, 288, 292
  - moving, 334
  - on Graph Windows, 332
  - selecting, 332
  - sourceless, 293
  - types, 67
- Nondebuggable code, 172
- Non-project
  - customizing Static Analyzer file set, 256
  - files in editors, 118
- Non-USASCII characters, 367
- Normalize Project button, 36, 44
- Normalize Windows (Filemenu), 164

## O

- Object identification, 224
- On Item (Help menu), 85
- Online Help, 84, 85
- Online Tutorial, 40
- Open (File menu), 38, 64

- Open(File menu), 110
- Open/Out-of-Date, 257
- Opening files with your editor, 121
- Optimized code
  - debugging, 227
- Options
  - compiler, 159
- Options menu, 78, 225, 228, 266
  - Behavior Settings, 257, 266, 267
  - Build Settings, 75
  - Default Load/Rerun Settings, 165, 167, 168, 175
  - Display Filter, 109
  - Fork Behavior, 219
  - History Menu Size, 280
  - Identifier Matching Rules, 263
  - Queries On New Nodes, 138
  - Save All Settings, 109, 228, 229
  - Set Default Project Root, 28
  - Stack Settings, 179, 186
  - Tool Preferences, 80, 92, 119
  - Toolbar Setup, 77, 79
  - User Configurable Buttons, 228
  - View Unresolved Differences Only, 299
  - Window Configuration, 163
- Oracle, 314
- Output, 174
- Output browser, 47, 48, 74
- Output browser menu
  - Find String, 75
  - Load Browser from File, 75
  - Print/Save Output, 75
- Overloaded break, 197

## P

- Package, 24, 62, 63
  - SQL, 317
- Parameterized types, 224
- Parameters, 167, 217
- Parent process, 222
- Pasting
  - data between tools, 37
  - multiple files in vi Editor, 129
  - text in vi Editor, 129
- PATH variable, 42
- Pattern Match () (Symbol () menu), 261, 263
- Pattern Match () (Symbol menu), 263
- PC
  - (Program Counter) arrow, 161, 171, 172, 181, 192

- location, 161, 172, 220
  - Permissions, 342
  - Planning your project, 31
  - Popup menu, 239, 287, 326
    - customizing, 228
    - Show Data Members, 140
    - Show Member Functions, 140
  - PostScript
    - file format, 326
    - saving images, 328
  - Preferences, setting, 80
  - Preprocessor
    - wrapper, 319
  - Prerequisites to using SoftBench, 42
  - Preview Build (More Build Actions submenu), 72
  - Previewing a build, 72
  - Print/Save Output (Output browser menu), 75
  - Printing
    - data, 181, 182
    - dialog box, 326
    - expressions, 182
    - graph images, 290, 326
    - hex or string values, 183
    - output browser, 75
    - query results, 271
  - Procedure
    - calling, 186
  - Procedure ( ) (Visit menu), 178, 181, 191
  - Program
    - checking using CodeAdvisor, 150
    - preparation for CodeAdvisor, 149
  - Program Editor
    - configuring, 119
  - Project, 21
    - accessing multiple, 35
    - adding files, 38, 51, 53, 58
    - building, 38, 70, 75
    - changing build type, 56
    - changing Local Workspace Root, 56
    - changing name, 56
    - checking code, 76
    - cloning, 27, 53
    - converting, 55
    - creating, 38, 50, 55
    - creating files, 57
    - creating subprojects, 54, 55
    - current, 35, 44
    - defining targets, 38, 39
    - external build model, 39, 50, 52, 64
    - file set, 22
    - files in editors, 118
    - Iconize Project button, 36, 44
    - locking, 35
    - modify definition, 56
    - name, 22, 50
    - Normalize Project button, 36, 44
    - planning, 31
    - project build model, 38, 50, 60
    - restructuring, 54
    - search path, 53
    - sharing, 27
    - specifying on startup, 83
    - static queries, 255
    - version control, 27
  - Project browser, 44
  - Project build, 25
    - changing to external build, 56
    - Debugger, 169
    - SQL, 315
  - Project menu
    - Add File(s) to Project, 38, 51, 53, 58, 64, 94
    - Close, 83
    - Exit SoftBench, 83
    - Make Subproject Of, 55
    - New Clone, 53
    - New Create, 38, 52
  - Project Search Path, 53
  - projectrc file, 228
  - Projects view, 36, 44
- ## Q
- ### Queries
- browsing to source code, 264
  - deleting history, 271
  - displaying results, 258
  - filtering, 292
  - filtering for Cplusplus, 268
  - filtering results, 266
  - graphical, 286
  - information about, 259
  - initial, 293
  - pattern matching, 263
  - printing results, 271
  - redisplaying results, 271
  - reference relationships, 261
  - saving results, 271
  - scoping, 268, 270

---

# Index

- selecting text, 261
- simplifying results, 266, 269
- textual, 260
- textual and graphical, 252
- valid identifiers, 263
- Queries On New Nodes(Options menu), 138
- Query graph, 284
- Query Result Area, 259
- Querying scope, 181, 182, 183
- Queue Length, 240

## R

### RCS

- migration, 348

### RDBMS, 313

- supported, 314
- versions, 317

### Redirecting I/O, 167

### Redisplaying query results, 271

### Reference relationships, 261

### References (Static menu), 130

### Refresh Files View (View menu), 59

### Registering

- Program Editor, 119
- tools, 80

### Registers, 213, 217

- Show Registers Dialog, 213

- Tracing, 213

### Registers (Show menu), 212

### Remote

- compilation server, 75
- data access, 81
- display, 82
- execution, 75, 81, 82, 372

### Removing tools from toolbar, 79

### Reordering tools on toolbar, 79

### Rerun (File menu), 165, 175

### Resources, 81

- buttonConfigFile, 229
- fileSetFile, 274
- synchronize, 176

### Restoring session, 83

### Reusing tool windows, 37

### Revision

- naming, 113
- setting, 111

### Revision history

- viewing, 107

### Rule Group Help, 48, 76, 150

### Rule group selection, 48, 150

### Rules, 147

- violations, 151

- run(), 178

- Running program, debugging, 222

- Runtime environment, 165

## S

- Save (File menu), 57, 58, 118

- Save All Settings (Options menu), 109, 228, 229

- Save As (File menu), 118

- Save Image (Graph menu), 291

- Save Image dialog box, 327

- Save Out Of Project (File menu), 118

- Save Out Of Project As (File menu), 118

### Saving

- graph images, 290, 326
- non-project files in editors, 118
- output browser, 75
- project files in editors, 118
- query results, 272

### SCCS

- migration, 348

### Scope

- querying, 181, 182, 183
- troubleshooting, 269

### Scoping Used, 259

### Screen areas, 44

### Scrolling

- Graph Windows, 330

### Security, 342, 346

### Selecting

- rule groups, 150
- text, 129, 261, 269

### Send Message (Messages) menu, 309

### Session

- restoring, 83
- stopping, 82

- Set (Break menu), 192

- Set (Trace menu), 205

- Set (Watch menu), 200

- Set at () (Break menu), 191

- Set at Class () (Break menu), 197

- Set at Entry () (Watch menu), 200

- Set At Hex Address () (Break menu), 211

- Set at Instance () (Break menu), 197

- Set at Overloaded () (Break menu), 197

- Set at Procedure Entry () (Break menu), 192

- Set at Template () (Break menu), 197

- Set Breakpoint (File menu), 131

- Set Default Project Root (Options menu), 28

- Set Logfile Name (File menu), 311

- Setting



- breakpoints, 190
  - data member values, 243
  - PATH, 42
  - revision, 111
  - revision state, 111
  - watchpoints, 200
  - Sharing
    - projects, 27
    - subprojects, 29
  - Show (Break menu), 189, 193
  - Show (Trace menu), 206
  - Show (Watch menu), 201
  - Show Data Members(Popup menu), 140
  - Show Local Server Information(Actions menu), 92, 95
  - Show Man Page (Help menu), 85
  - Show Member Functions(Popup menu), 140
  - Show menu, 252, 260, 265
    - Assembly Instructions, 211, 218
    - Classes, 260
    - Data Graph (), 236
    - Data Graph Indirect (), 236
    - Data Value Print Format", 183
    - Functions, 260
    - Global Variables, 260
    - Registers, 212
    - Source Files, 260
    - Stack, 181, 186, 218
  - Show On Graph
    - (Values popup menu), 248
  - Show Registers Dialog, 213
  - Show Revision History(File menu), 107
  - Show Set Up Instructions(Actions menu), 92
  - Show/Modify Mappings(Actions menu), 99, 113
  - Signals, 217
    - debugging, 214
    - ignoring, 216
  - Signals/Intercepts, 188, 214
  - Signals/Intercepts (Execution menu), 214
  - Single step, 171
  - SoftBench
    - accessing tools from editor, 130
    - Customizing, 78
    - main window, 44
    - multiple sessions, 35
    - resources, 81
    - starting, 43
    - starting on a remote system, 82
    - using, 41
  - SoftBench vi Editor
    - accessing Static Analyzer from, 130
    - building from, 130
    - calling other tools, 130
    - checking files in and out, 131
    - compiling from, 130
    - configuration management from, 131
    - copying text, 129
    - debugging from, 131
    - pasting text, 129
    - selecting text, 129
    - starting from Static Analyzer, 253
  - SoftBench XEmacs
    - starting from Static Analyzer, 253
  - softbenchrc file, 81
  - SoftCM
    - accessing, 342, 346
    - administration, 339, 349
    - browsing archive files and directories, 95
    - cancelling file check out, 103
    - capacity, 90
    - checking in files to archive, 110
    - checking out archive files, 102
    - command line, 115
    - configuring in SoftBench, 92
    - creating archive directories, 97
    - creating archive files, 102
    - creating initial archive files, 102
    - customizing, 339
    - deleting archive files or directories, 106
    - licenses, 90
    - lockinfo file, 349
    - logging, 346, 351
    - mapping local and archive directories, 98
    - permissions, 342
    - starting SoftBench CM, 92
    - troubleshooting, 351
    - updating local directories, 103
    - use model, 90
    - viewing archive file content, 104
  - softstatic command, 281
  - Software lifecycle, 251
  - Sort Results (View menu), 266
  - Source
    - breakpoints, 189
  - Source File Area, 161
  - Source Files (Show menu), 260
  - Source locations
    - Debugger, 168
  - Specifying files
    - File Compare, 296, 297
-

- SQL, 313
  - build configuration, 315
  - debugging, 319
  - file types, 315
  - in project build model, 315
  - packages, 317
  - packages,nodebug, 320
  - preprocessor wrapper, 319
  - SoftBench CodeAdvisor, 321
  - Static Analyzer, 321
  - updating versions, 317
  - with external build, 317
- SQLDEBUG Makefile macro, 320
- Stack, 217
  - viewing, 186
- Stack (Show menu), 181, 186, 218
- Stack Frame, 161
- Stack Settings (Options menu), 179, 186
- Standalone
  - customizing Static Analyzer file set, 256
  - Static Analyzer, 273
  - tools, 36
- Standard I/O, 174
- Starting
  - Class Graph/Editor, 138
  - CodeAdvisor, 150
  - Data Graph Window, 236
  - editor, 121
  - editor from target graph, 69
  - Graph Windows, 324
  - Message Monitor, 306
  - SoftBench, from CDE, 43
  - SoftBench, from command line, 43
  - SoftBench, on a remote system, 82
  - SoftBench, with a project, 83
  - SoftCM, 92
  - Static Analyzer, 252
  - Static Graph, 285
  - tools, 77
- Starting tools
  - in multiple projects, 35
  - starting one tool from another, 34
- State
  - setting, 111
- Statement breakpoints, 189
- Static Analysis (File menu), 252
- Static Analyzer, 32, 251
  - access from other tools, 39
  - accessing from your editor, 130
  - accurate results, 256, 273
  - backward compatibility, 254
  - Cplusplus filters, 268
  - customizing, 275
  - customizing analysis file set, 255, 256
  - default file set, 255
  - generating static code, 254
  - generating static data, 273
  - incomplete data, 254
  - Menu bar, 258
  - query information area, 259
  - Query Results Area, 259
  - running, 39
  - scoping queries, 268, 270
  - searching subdirectories, 273
  - SQL, 321
  - standalone mode, 273
  - starting, 252
  - starting editors, 264
  - Static.sadb file, 275
  - subprojects, 255
  - textual queries, 260
  - troubleshooting, 276
  - window, 258
  - with compile errors, 254
- Static compile mode, 254
- Static constructors, 225
- Static database, 148, 254, 255, 273
  - location, 56, 275
  - used from Class Graph/Editor, 134
- Static Graph, 283
  - customizing, 294
  - displaying nodes on another graph, 288
  - finding nodes, 287
  - general graph operations, 323
  - graphical queries, 286
  - graphs in separate windows, 294
  - hiding nodes, 292
  - printing, 290
  - removing graph legend, 294
  - saving images, 290
  - simplifying display, 292
  - starting, 285
  - starting editors, 287
  - switching between graphs, 288
- Static menu
  - References, 130
- Static.sadb file, 275
- Staticfileset file, 274
- Status line, 46

- stderr, 167, 174
- stdin, 167, 174
- stdout, 167, 174
- Stopping
  - Data Graph Window, 236
  - SoftBench, 82
  - tools, 82
- String constants, 185
- Strings, printing, 183
- Subproject, 22
  - build order, 73
  - building, 55, 73
  - converting, 55
  - creating, 55
  - sharing, 29
- Suspend All (Break menu), 194
- Suspending Data Graph Window, 244
- Switching Static graphs, 287
- Sybase, 314
- Symbol
  - menu
    - Pattern Match (), 263
- Symbol () input box, 258, 260, 261
- Symbol () menu
  - Pattern Match (), 261
- Symbol menu, 252, 260, 261, 265
  - Classification (), 263
  - Pattern Match (), 263
  - pattern matching, 263
- Symbolic name list, 113
- Symbolic names, 112
  - default specifier, 112, 113
  - defining, 113
  - on check in, 113
  - on check out, 112
- synchronize resource, 176
- Synchronizing
  - analysis file set, 256, 264
  - files, 208
- Synchronous mode, 175
- Syntax
  - Debugger, 177
- System library, 172
- T**
- Target, 22, 23
  - add, 51, 53
  - building, 69, 70, 75
  - defining, 51, 53, 60
  - defining for external build, 64
  - graph, 47, 66, 69
    - selecting build configuration, 60
    - showing dependencies, 61
    - specifying dependencies, 61, 68
- Target graph, 66
  - colors, 68
  - hiding nodes, 69
  - node types, 67
  - starting editor, 69
  - with external build, 47, 67
  - zooming, 69
- Target menu
  - Add Build Order Dependency, 61
  - Build, 69, 70
  - Debug, 39, 77
  - Display on Graph, 66
  - Modify Properties, 38, 52, 63, 71, 165
  - More Build Actions submenu, 72
  - New, 38, 39, 51, 53, 60, 64
- Tearing apart main Debugger toolface, 163
- Template break, 197
- Templates, 224
- Terminal Emulator window, 175
- Textual queries, 252
- Threads
  - breakpoints on, 193
  - debugging, 220, 221
  - stack, 187
- Threads (Execution menu), 187, 220
- Tool Overview (Help menu), 85
- Tool Preferences (Options menu), 80, 119
- Tool Preferences(Options menu), 92
- Toolbar, 44, 77
  - changing tools, 79
  - troubleshooting, 86
- Toolbar Setup (Options menu), 77, 79
- Tools
  - access from editor, 130
  - adding new, 80
  - copy and paste across tools, 37
  - displaying current project, 36
  - encapsulating, 80
  - instances, 36, 37
  - list, 32
  - multiple instances, 35
  - preferences, 80
  - registering, 80
  - starting, 77
  - starting one from another, 34
- Trace menu, 204

---

# Index

- Clear All, 206
  - Set, 205
  - Show, 206
- Traces, 188, 204
  - clearing, 206
  - commands at, 204
  - creating, 204
  - DDE commands, 205
  - granularity, 204
  - Registers, 213
  - viewing, 205
- Transform, 24
- Troubleshooting, 351
  - build, 86
  - File Compare, 302
  - in Class Graph/Editor, 143
  - in Debugger, 231
  - Static Analyzer, 276
  - toolbar, 86
- types file
  - using nonlocalized, 362
- Typewriter font, 5
- U**
- Undebuggable code, 172
- Understanding code, 39
- Undo(Edit menu), 140
- Unload Executable(File menu), 37
- Unlocking archive files, 107
- Untitled project, 43
- Update Buffer (Edit menu), 208
- Update Status of Nodes (Graph menu), 68, 72
- Use External Build Command (Builder menu), 72
- Use models, 32, 39
  - Class Graph/Editor, 134, 141
  - Debugger Data Graph Window, 245
  - multiple projects, 35
  - reusing tool windows, 37
  - single set of cooperating tools, 34
  - SoftCM, 90
  - standalone tools, 36
  - starting one tool from another, 34
  - starting tools from SoftBench, 77
- User Configurable Buttons (Options menu), 228
- User Program I/O Area, 162, 167, 174, 175, 176
- Uses of an identifier, 262
- Using Help (Help menu), 85
- Using SoftBench, 41
- V**
- Values
  - showing on Data Graph Window, 248
- Values Display (Watch menu), 201
- Variables
  - changing, 181, 183
  - DDE, 183
  - examining, 181, 182
  - specifying, 178
  - tracing, 199
- Verifying fixes, 207
- Version control, 27
  - choosing tool, 80
  - Class Graph/Editor, 142
  - from your editor, 131
- Versioned files, 90
- vi Editor, 126
  - clipboard, 129
  - configuring, 119
  - Cursor, 127
  - editing multiple files, 127
  - Editor Index, 128
  - Mouse pointer, 127
  - non-project files, 118
  - Pointer, 127
  - project files, 118
  - save non-project files, 118
  - save project files, 118
  - Text cursor, 127
- View menu, 78, 266
  - Display Results, 266
  - Filter Results, 266, 268
  - Filters, 292
  - Refresh Files View, 59
  - Sort Results, 266
- View Unresolved Differences Only (Options menu), 299
- Viewing
  - archive file contents, 102
  - breakpoints, 193
  - data, 181, 182
  - mappings, 99
  - multiple Static Graphs, 294
  - revision history, 107
  - traces, 205
  - watchpoints, 201
- Violation, 151
  - browsing, 48

filtering, 48, 152  
Violation Help, 49, 76, 151  
Visit menu  
File (), 191  
Procedure (), 178, 181, 191

## W

Watch menu, 200  
Clear All, 203  
Set, 200  
Set at Entry (), 200  
Show, 201  
Values Display, 201  
Watchpoints, 188, 199  
changing, 201  
clearing, 202  
collapsing, 202  
creating, 200  
DDE commands, 200  
expanding, 202  
granularity, 199  
hiding, 202  
setting, 199, 200  
viewing, 201

## Window

area, 44  
in XEmacs Editor, 122  
reusing, 37  
SoftBench File Compare, 296  
status information, 337

Window Configuration (Options menu), 163

Windowed applications, debugging, 175

Working directory  
from Debugger, 167  
program, 167

## X

X applications, debugging, 175  
XEmacs Editor, 122  
accessing Static Analyzer from, 130  
buffers, 122  
building from, 130  
calling other tools, 130  
checking files in and out, 131  
compiling from, 130  
configuration management from, 131  
debugging from, 131  
editing multiple files, 124  
files, 122

frames, 122  
help, 124, 126  
non-project files, 118  
preferences, 119  
project files, 118  
reference information, 122  
resource file, 119  
save non-project files, 118  
save project files, 118  
web site, 122  
windows, 122, 125  
xrdb, 81, 176  
XtAppInitialize function, 176  
xwd  
file format, 326  
saving images, 329

## Z

Zooming, 69  
Graph Windows, 331