

# *Sun WorkShop TeamWare: Users Guide*



2550 Garcia Avenue  
Mountain View, CA 94043  
U.S.A

**Part No: 802-5953-10**  
**Revision A, December 1996**

Copyright 1996 Sun Microsystems, Inc., 2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Portions of this product may be derived from the UNIX<sup>®</sup> system, licensed from Novell, Inc., and from the Berkeley 4.3 BSD system, licensed from the University of California. UNIX is a registered trademark in the United States and other countries and is exclusively licensed by X/Open Company Ltd. Third-party software, including font technology in this product, is protected by copyright and licensed from Sun's suppliers.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and FAR 52.227-19.

Sun, Sun Microsystems, the Sun logo, Solaris, and Sun WorkShop TeamWare, are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK<sup>®</sup> and Sun<sup>™</sup> Graphical User Interfaces were developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

X Window System is a trademark of X Consortium, Inc.

THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.



# *Contents*

---

Preface.....	xiii
<b>1. QuickStart .....</b>	<b>1</b>
Basic Concepts .....	1
Parent and Child Workspaces .....	2
Source Code Control System .....	3
Changing Names.....	3
Versioning.....	7
Freeze-pointing .....	9
Distributed Make.....	11
<b>2. Introduction.....</b>	<b>13</b>
How to Use the Documentation Supplied with TeamWare ...	14
<b>3. Introduction to TeamWare Configuring .....</b>	<b>15</b>
Coordinating the Work of Software Developers .....	15
Copy-Modify-Merge Model .....	16
Copy-Modify-Merge Example .....	18

---

Default Configuring . . . . .	19
Workspace . . . . .	20
Copying Files between Workspaces . . . . .	25
Resolving Conflicts . . . . .	33
<b>4. Introduction to Merging . . . . .</b>	<b>35</b>
Differences Defined . . . . .	36
Difference . . . . .	36
Current, Next, and Previous Difference . . . . .	36
Resolved and Remaining Difference . . . . .	36
Graphical Overview . . . . .	37
Merging Window . . . . .	37
Merging Glyphs . . . . .	37
<b>5. Starting a Project . . . . .</b>	<b>41</b>
Moving an Existing SCCS-Based Project to Sun Workshop TeamWare . . . . .	41
Moving a Non-SCCS Software Development Project to Workshop TeamWare . . . . .	42
Converting an RCS Project to Workshop TeamWare . . . . .	42
Further User Notes . . . . .	43
Starting a New Software Development Project with Workshop TeamWare . . . . .	43
Configuring a Workspace Hierarchy . . . . .	44
File Transfer Considerations . . . . .	45
Product Release Considerations . . . . .	47
Coordinating Access to Source Files . . . . .	48

---

Branches .....	49
<b>6. TeamWare Configuring User Interfaces .....</b>	<b>51</b>
TeamWare Configuring Command-Line Interface.....	52
TeamWare Configuring Graphical User Interface .....	53
Starting TeamWare Configuring.....	53
TeamWare Configuring Windows .....	54
Configuring Window.....	55
Customizing the Configuring Program Using Properties .	57
Accelerators .....	60
<b>7. TeamWare Configuring Workspace .....</b>	<b>61</b>
The Workspace Metadata Directory .....	62
Creating a Workspace .....	64
Using Workspace Create.....	64
Using the Bringover Create Transaction.....	64
Deleting a Workspace .....	64
Moving and Renaming a Workspace.....	65
A Note About Moving Workspaces.....	65
Reparenting a Workspace .....	66
Two Ways to Reparent Workspaces.....	66
Reasons to Change a Workspace's Parent.....	67
Controlling Access to Workspaces.....	71
Viewing and Changing Access Control Values .....	74
How to Notify Users of Changes to Workspaces.....	75
Viewing and Changing Notification Entries .....	76

---

Notes About Registering Notification Events . . . . .	77
Viewing Workspace Command History . . . . .	78
Ensuring Consistency through Workspace Locking . . . . .	80
Configuring Environment Variables . . . . .	81
The CODEMGR_WS Variable . . . . .	82
The CODEMGR_WSPATH Variable . . . . .	82
<b>8. Copying Files between Workspaces . . . . .</b>	<b>83</b>
Configuring Transaction Model . . . . .	83
General File Copying Information . . . . .	84
SCCS History Files . . . . .	85
Viewing Transaction Output . . . . .	85
Specifying Directories and Files for Transactions . . . . .	85
Copying Files from a Parent to a Child Workspace (Bringover) . . . . .	89
Creating a New Child Workspace (Bringover Create) . . . . .	90
Notes about the Bringover Create Transaction . . . . .	92
Updating an Existing Child Workspace (Bringover Update) . . . . .	94
Notes about the Bringover Update Transaction . . . . .	96
Bringover Action Summary . . . . .	99
Copying Files from a Child to a Parent Workspace (Putback) . . . . .	99
Updating a Parent Workspace Using Putback . . . . .	100
Notes about the Putback Transaction . . . . .	102
Putback Action Summary . . . . .	104
Reversing Bringover and Putback Transactions with Undo . . . . .	105
Notes about the Undo Transaction . . . . .	106

---

How the Undo Transaction Works. . . . .	106
Renaming, Moving, or Deleting Files . . . . .	108
Renaming Files. . . . .	108
Deleting Files . . . . .	112
Notes about Renaming Files. . . . .	113
<b>9. Resolving Conflicts. . . . .</b>	<b>115</b>
Conflict Resolution Process . . . . .	115
Changing Names . . . . .	116
Detecting Conflicts. . . . .	116
Detecting Conflicts during Bringover Update Transactions	116
Preparing Files for Conflict Resolution. . . . .	117
Resolving Conflicts . . . . .	117
Resolve Transaction. . . . .	118
<b>10. Administering the Workspace . . . . .</b>	<b>123</b>
Starting a Project with the Configuring Program . . . . .	123
Moving an Existing Project. . . . .	123
Starting a New Project. . . . .	124
Structuring Your Workspace Hierarchy . . . . .	124
File Transfer Considerations. . . . .	126
Product Release Considerations. . . . .	128
<b>11. How the Configuring Program</b>	
<b>Merges SCCS Files . . . . .</b>	<b>131</b>
Merging Files That Do Not Conflict . . . . .	132
Merging Files That Conflict . . . . .	133

---

An Example of Merging .....	134
<b>12. Configuring Example .....</b>	<b>143</b>
Creating Workspaces .....	144
Putting Back Changes .....	145
Updating a Workspace .....	146
Resolving Conflicts .....	147
<b>13. Error and Warning Messages .....</b>	<b>149</b>
Error Messages .....	150
Warnings Messages .....	165
<b>14. Performing Basic SCCS Functions with Versioning .....</b>	<b>171</b>
Typical Sessions .....	171
Changing Names .....	172
Putting a Project Under SCCS Control .....	172
Working with a Project Under SCCS Control .....	173
Getting Help on the GUI or the CLI .....	174
Commands: Manipulating Files .....	174
Checking Out and Checking In Files .....	175
Editing a Checked-Out File .....	176
Checking in a New File .....	176
Unchecking Out a File .....	177
View Option: Changing Versioning Properties .....	177
Changing the Main File List Display .....	177
Defining an Editor .....	178
Changing the Double-Click Action .....	178



---

Changing the History Graph Display . . . . .	179
Changing the History Information Display . . . . .	179
<b>15. Starting and Loading Merging . . . . .</b>	<b>181</b>
Changing Names . . . . .	182
Starting Merging from Sun WorkShop . . . . .	182
Starting Merging from Sun WorkShop TeamWare. . . . .	182
Loading Files from the Merging Window. . . . .	182
Starting Merging from the Command Line . . . . .	183
Basic Startup Command . . . . .	184
Command-Line Synopsis . . . . .	184
Loading Two Files at Startup . . . . .	186
Loading Three Files at Startup . . . . .	187
Loading Files from a List File. . . . .	187
Saving the Output File . . . . .	188
<b>16. Examining Differences. . . . .</b>	<b>189</b>
Moving Between Differences . . . . .	189
Resolving Differences . . . . .	189
Automatic Merging . . . . .	190
An Example . . . . .	190
Starting Merging . . . . .	192
Examining Differences . . . . .	193
Saving Output File. . . . .	194
<b>17. Introduction to FreezePointing. . . . .</b>	<b>197</b>
Changing Names . . . . .	197

---

How FreezePointing Works .....	198
Starting FreezePointing.....	200
Creating a Freezepoint File.....	201
Viewing or Modifying a Freezepoint File .....	202
Recreating (Extracting) a Source Hierarchy .....	202
Notes about Using FreezePointing .....	204
Details about the Freezepoint File .....	204
What is a SMID?.....	205
Why are SMIDs Necessary? .....	205
SMID/SID Translation .....	205
Translating SIDs to SMIDs .....	206
Translating SMIDS to SIDS.....	206
<b>18. Troubleshooting Versioning and FreezePointing.....</b>	<b>209</b>
Troubleshooting Checklist .....	209
Reporting Problems.....	210
Error Messages.....	210
Display Problems .....	211
<b>19. Building Programs in Sun Workshop TeamWare .....</b>	<b>213</b>
Building a TeamWare Target.....	213
Building With Default Values.....	214
Building With NonDefault Values.....	214
Modifying a TeamWare Target .....	215
Fixing Build Errors.....	215

---

<b>20. Using DistributedMake</b> .....	<b>219</b>
Basic Concept of Distributed Make .....	219
Configuration Files .....	220
The DMake Host .....	220
The Build Server .....	223
What You Should Know About DMake Before You Use It . . . .	223
DMake's Impact on Makefiles .....	223
How to Use DMake .....	229
Notes .....	230
Controlling DistributedMake Jobs .....	230
Getting Help on the GUI or the CLI .....	231
<b>21. Dealing With Release Matters</b> .....	<b>233</b>
Integrating Your Changes .....	233
Performing Master Builds .....	233
Establishing Nightly Builds .....	234
Performing Releases .....	234
Organizing a Release .....	234
How the Release Process Works .....	234
Index .....	241



## *Preface*

---

### *Introduction*

The *Sun Workshop TeamWare Users Guide* describes how to use the *Sun Workshop TeamWare* code management tools. The concepts and information discussed apply to both command line and graphical user interfaces.

### *Who Should Use This Book*

Sun Workshop Teamware Users Guide is directed towards the software developer, but also addresses integrators, administrators and release engineers in their tasks involving code management.

As a software developer, you typically acquire code from a code integration area or integration workspace. You then:

- Add new features to your program module
- Test and debug the program
- Put the code back in the implementation or integration workspace from which it was acquired

The Configuring section of this guide is primarily addressed to the software developer. It also addresses the needs of integrators, administrators, and release engineers.

The Versioning and Freezepointing section of this guide explains how to use Versioning for controlling files and monitoring changes on concurrent software development projects. Versioning is a graphical user interface (GUI) to the

---

source code control system (SCCS). It also explains how to use Freezepointing, a tool that allows you to create snapshots of a project at various key junctures. These snapshots, or freezepoints, enable you to recreate the project at a particular state in its development cycle. Use this section if you write programs coded in ASCII text source. This section assumes that you are familiar with programming constructs and processes. You need not have previous experience with SCCS.

The Building and DMake sections of this guide are a supplement to the standard `make` documentation. They describe how to use Building and Distributed Make to make the process of building programs more efficient. Use these sections if you maintain programs using the `make` utility and wish to speed up the build process. These sections also assume that you are familiar with the standard `make` utility.

This manual assumes that you are familiar with the SunOS operating system, the UNIX® source code control system (SCCS), and with general programming terminology.

## *Compatibility*

See the online `readme` file for specific operating environment information.

## *Before You Read This Book*

You should have *TeamWare* installed on your system. See the *Installing Developer Products on Solaris* manual for information on how to install the *TeamWare* software.

## *How This Book Is Organized*

Chapter 1, “QuickStart”—provides instructions for quickly getting started using the *TeamWare Code Management Tools*.

Chapter 2, “Introduction”—provides a full introduction to the *Sun Workshop TeamWare* product, and provides you with ways to get the best information.

Chapter 3, “Introduction to TeamWare Configuring”—presents an overview of the Configuring portion of *TeamWare*. Basic concepts are discussed that are vital to understanding how Configuring works.

---

Chapter 4, “Introduction to Merging” —presents an overview of the Merging portion of TeamWare, and describes the graphical interface.

Chapter 5, “Starting a Project” — presents information on how to start a new project, and how to move existing work projects into the TeamWare environment.

Chapter 6, “TeamWare Configuring User Interfaces”— describes the Configuring user interfaces.

Chapter 7, “TeamWare Configuring Workspace”—describes the Configuring workspace and the associated commands.

Chapter 8, “Copying Files between Workspaces”— describes the TeamWare Configuring transactions used to transfer files between workspaces.

Chapter 9, “Resolving Conflicts”—explains how you resolve conflicts between files in parent and child workspaces.

Chapter 10, “Administering the Workspace”—discusses those considerations to be made when starting a project with the Configuring Program.

Chapter 11, “How the Configuring Program Merges SCCS Files”—describes how TeamWare Configuring manipulates SCCS history files during file transfer transactions.

Chapter 12, “Configuring Example”—contains an example that demonstrates the TeamWare Configuring bringover, putback, and resolve transaction cycle.

Chapter 13, “Error and Warning Messages”—lists TeamWare Configuring error messages and warnings. Each message is defined, and a possible remedy is provided.

Chapter 14, “Performing Basic SCCS Functions with Versioning” —describes common SCCS functions such as checking out and editing a file, checking in a new file, and displaying delta differences, using the Versioning service. It covers the basic operational tasks and walks you through step-by-step instructions.

Chapter 15, “Starting and Loading Merging” —presents Merging, a tool that allows you merge files to resolve conflicts. It provides an introduction to the graphical interface and a tour of the command line options.

---

Chapter 16, “Examining Differences” — examines the process of using Merging to resolve differences between files, including automatic merging, and provides a detailed example.

Chapter 17, “Introduction to FreezePointing”—presents Freezepointing, a tool that allows you to create snapshots of a project. It provides an overview of the graphical interface and shows you how to use this tool in conjunction with the other TeamWare development tools.

Chapter 18, “Troubleshooting Versioning and FreezePointing”—provides a problem checklist to consider before calling the Sun Support hotline. It also gives information on how to report a problem, as well as a list of error messages—their meanings and what to do next.

Chapter 19, “Building Programs in Sun Workshop TeamWare”—presents a discussion of the build process with specific targets, as well as hints on fixing build errors.

Chapter 20, “Using DistributedMake”—describes DMake, a tool that allows you to distribute builds over several hosts concurrently. The operation of DMake is described, and instructions given on how to distribute your build efficiently.

Chapter 21, “Dealing With Release Matters”—provides a summary of the post-build process, and points to a source for further help in handling release and integration matters.

“Glossary”—provides a clear explanation of the special terms used in this manual.

## *How to Get Help*

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

- **Online Help** – A new help system containing extensive task-oriented, context-sensitive help. To access the help, choose Help ► Help Contents. Help menus are available in all *Sun Workshop TeamWare* windows.



- 
- **TeamWare Documentation** – A set of online manuals. These manuals make up the complete documentation set for *Sun WorkShop TeamWare* and are available using AnswerBook™ or (at the user's option) using a browser. To access the online manuals, choose Help ► TeamWare Manuals in any TeamWare window.
  - **Video Demonstration** – This demo provides a general overview of *Sun WorkShop TeamWare* and describe how to use *Sun WorkShop TeamWare* manage code. To access, choose Help ► Demos in the TeamWare main window.
  - **Release Notes** – The Release Notes contain general information about *Sun WorkShop TeamWare* and specific information about software limitations and bugs. To access the Release Notes, choose Help ► Release Notes.

## *Related Books*

Sun WorkShop TeamWare provides comprehensive documentation. The following books are available in online and printed forms (except where noted).

Access online books by choosing Help ► Online Books or Help ► *Sun WorkShop TeamWare* Contents from the main WorkShop Help menu. Many of the *Sun WorkShop TeamWare* tools and utilities are linked to detailed, context-sensitive, on-item help, which in turn is linked to the relevant sections of the *Sun WorkShop TeamWare* help volume. This release of *Sun WorkShop TeamWare* also provides a set of video demonstrations accessible from the main *Sun WorkShop TeamWare* Help menu. Some documents are available with all *Sun WorkShop TeamWare* products, others are not (as noted).

---

## *Sun WorkShop TeamWare Documentation*

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

Manual Pages (man pages)—*TeamWare* has the following manual pages:

*Table P-1* TeamWare Manual Pages

---

codemgr(1)	rsc2ws(1)	access_control(4)
codemgrtool(1)	sccsmerge(1)	args(4)
bringover(1)	teamware(1)	children(4)
def.dir.flp(1)	twbuild(1)	freezeptfile(4)
dmake(1)	twconfig(1)	notification(4)
filemerge(1)	twfreeze(1)	conflicts(4)
freezept(1)	twmerge(1)	history(4)
freezepttool(1)	twversion(1)	locks(4)
make(1)	vertool(1)	nametable(4)
maketool(1)	workspace(1)	parent(4)
putback(1)	ws_undo(1)	putback.cmt(4)
resolve(1)		

---

## *Ordering Additional Hardcopy Documentation*

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

<http://sunexpress.usec.sun.com>

---

## Sun on the World Wide Web

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

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

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

## What Typographic Changes Mean

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

Table P-2 Typographic Conventions

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

---

## *Shell Prompts in Command Examples*

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

*Table P-3* Shell Prompts

<b>Shell</b>	<b>Prompt</b>
C shell prompt	machine_name%
C shell superuser prompt	machine_name#
Bourne shell and Korn shell prompt	\$
Bourne shell and Korn shell superuser prompt	#

Use this chapter to quickly get started using *Sun WorkShop TeamWare™*. For more details, see the other parts of this guide. Also refer to the online help for immediate assistance in completing specific tasks.

This chapter contains the following sections:

<i>Basic Concepts</i>	<i>page 1</i>
<i>Versioning</i>	<i>page 7</i>
<i>Freeze/pointing</i>	<i>page 9</i>
<i>Distributed Make</i>	<i>page 11</i>

## *Basic Concepts*

*TeamWare* is based on a concurrent development model called *Copy-Modify-Merge*. Isolated (per developer) workspaces form the *TeamWare* model. A workspace is a specially designated UNIX directory and its subdirectories. With *TeamWare*, you *copy* source from a central workspace into your own workspace, *modify* the source, and *merge* your changes with changes made by other developers in the central workspace.

Besides providing isolated workspaces, *TeamWare* enables you to easily and intelligently copy files between workspaces and then merge changes that exist between corresponding files. The intelligent copy feature enables you to copy

project files in groups that you (or the project administrator) determine are logically linked, and automatically determines differences between files in the originating workspace and the destination workspace.

*TeamWare* further assists the concurrent development process by determining whether differences exist between the files in the central workspace and your workspace. If differences are found to exist, *TeamWare* commands prevent you or another developer from copying over those changes; *TeamWare* then provides sophisticated window-based tools that help you to merge these differences.

### *Parent and Child Workspaces*

When you copy files from a central workspace to create a new workspace, a special relationship is created between the central workspace and the new one. The central workspace is considered the *parent* of the newly created *child* workspace. You can acquire files from any Configuring workspace in this manner, and workspaces can have an unlimited number of children. The portion of the file system that you copy from the parent workspace is determined at the time you copy it. You can copy the entire contents of the parent to the child, making it a clone of the parent, or you can copy only portions of the file system hierarchy that are of interest to you. The *TeamWare* Configuring transaction used to copy files from a parent workspace to a child workspace is called *Bringover*.

When development and testing are complete in the child, you copy changes in files that were modified or added in the child, back into the parent workspace. Once the altered files are present in the parent, they can be copied by other children or passed up another level to the parent's parent workspace. The *TeamWare* Configuring transaction for copying changes in files from a child workspace to a parent workspace is called *Putback*.

If any of the files you attempt to put back are changed in *both* the parent and child workspace, the files are in *conflict*. If this is the case, Configuring will block the transaction. You must then use the *Bringover* transaction to bring over the changed information from the parent and use the *Resolve* transaction to resolve the conflict in the child workspace before you can put your work back to the parent.

## Source Code Control System

TeamWare Configuring acts only upon files under the source code control system (SCCS). When considering Configuring file transfer transactions, remember that source files are derived from SCCS deltas and are identified by SCCS delta IDs (SIDs). When a file is copied by either a Putback or Bringover transaction, Configuring acts upon (copies or merges) the file's SCCS history file (also known as the "s-dot-file"). How Configuring manipulates and merges the history files is described in Chapter 11, "How the Configuring Program Merges SCCS Files."

## Changing Names

The current release of TeamWare uses new command names, so Table 1-1 summarizes the correspondences for you. Note that the old commands still work, however this manual uses the new commands and GUI names.

*Table 1-1* Correspondences Between Old and New TeamWare Commands

Old Command	New Command	Old Tool Name	New GUI Name
codemgrtool	twconfig, teamware	CodeManager	Configuring
vertool	twversion	VersionTool	Versioning
filemerge	twmerge	FileMerge	Merging
maketool	twbuild	MakeTool	Building
freezepttool	twfreeze	FreezePoint	Freezepointing

### ▼ Getting Started

You can use TeamWare Configuring through either a graphical user interface (GUI) or command-line interface (CLI). The following procedures use the GUI. For information about the CLI, refer to the `bringover(1)` and `putback(1)` manual pages.

---

**Note** – Before you begin using TeamWare Configuring on your project, you must know the path name of the workspace from which you are to bring over your work.

---

▼ **Creating a New Workspace**

1. **From a command prompt start the Configuring GUI:**

```
% twconfig &
```

2. **If the workspace from which you must obtain your files is not automatically loaded, load the workspace using the Load item from the File menu.**
3. **Once you load the workspace, use the Bringover Create transaction to create your own workspace. Your workspace is a child of the original workspace. You initiate the transaction by dragging and dropping the parent workspace icon into an open area of the pane. This activates the Bringover Create version of the Transactions window.**
4. **In the Bringover Create Transactions window, enter the child workspace path name in the text field labeled: To Child Workspace Directory.**
5. **In the Directories and Files text pane, create the list of directories and files you wish to bring over to your workspace from the parent workspace. Select all files to bring over by accepting the default "." or choose File ► Add Files to create the Directories and Files list. *Optionally:* Select the Preview option to verify your transaction before you actually transfer any files.**
6. **Click on the Bringover button at the bottom of the window to initiate the transaction.**
7. **View transaction output in the Transaction Output window.**  
For more information about the Bringover Create transaction, see "Creating a New Child Workspace (Bringover Create)" on page 90."

▼ **Changing Files in the Child Workspace**

1. **To change files in a child workspace, you need to start Versioning. Refer to "Starting Versioning" on page 7.**
2. **The files must then be checked out, edited, and checked back in under SCCS. Refer to "Checking Files In and Out of SCCS" on page 8.**



---

## ▼ Putting Back Changes to the Parent

- 1. Update the parent workspace with the changes you make.**  
This Configuring transaction is called Putback.
- 2. Initiate Putback transactions by dragging and dropping your child workspace icon onto the parent workspace icon. This step activates the Putback version of the Transactions window.**  
Configuring automatically fills in the names of the parent and child workspaces in the Putback Transaction window and includes the same directories and files that you included when you created the child workspace.
- 3. Type a comment in the Comments text window. *Optionally:* Select the Preview option to verify your transaction before you transfer any files.**
- 4. Click on the Putback button at the bottom of the window to initiate the transaction.**
- 5. Delete selected workspaces.**
- 6. View transaction output in the Transaction Output window.**  
For more information about the Putback transaction, see “Updating a Parent Workspace Using Putback” on page 100.”

## ▼ Updating the Child Workspace

If any of the files have changed in the parent since you brought them over, the Putback transaction is blocked. In this case, you will have to use the Bringover Update transaction to bring those changes into your child workspace.

- 1. Resolve any conflicts.**
- 2. Test.**
- 3. Put them back to the parent.**  
A popup window advises you that the transaction is blocked.

- 4. Initiate the Bringover Update transaction by clicking on Bringover now in the popup window. *Optionally:* Select the Preview option to verify your transaction before you transfer any files.**

This activates the Bringover Update version of the Transactions window. In the Bringover Update Transactions window, Configuring automatically fills in the names of the parent and child workspaces, and includes the same directories and files that you included when you created the child workspace.

- 5. Click on the Bringover button at the bottom of the window to initiate the transaction.**

- 6. View your transaction output in the Transaction Output window.**

For more information about the Bringover Update transaction, see “Updating an Existing Child Workspace (Bringover Update)” on page 94”.

## ▼ Resolving Conflicts

If any of the files you changed in your child workspace were also changed in the parent workspace, they are in *conflict*. If Configuring discovers any conflicts during the Bringover Update transaction, it automatically activates a popup window advising you of this.

- 1. Initiate the Resolve transaction by clicking on Resolve now in the popup window.**

This step activates the Resolve version of the Transactions window. Configuring automatically alters the workspace icon to alert you that a workspace contains unresolved conflicts. Configuring automatically:

- Lists the path names of the files that are in conflict in the Resolve Transaction window
- Starts the Merging program, loading the first file in the list

Merging displays two text files (the versions of the file from the parent and child workspaces) for side-by-side comparison, each in a read-only subwindow. Each version is shown in comparison (using glyphs) to the version that existed before the changes were made. Beneath them, Merging displays a subwindow that contains a merged version. The merged version contains selected lines from either or both deltas.

Merging automatically merges the files for you in the bottom window. If you disagree with the choices made by the program:

2. Use the **Left** and **Right** buttons to accept the changes found in the left or right window.
3. When you merge the files, select the **Save** button to save the file.  
If there are more files in the Transactions window conflict list, Configuring automatically loads the next file in the list into Merging.

For more information about resolving conflicts and merging files, see Chapter 9, “Resolving Conflicts”.

## Versioning

Versioning is a GUI to SCCS that enables you to manipulate files and perform SCCS functions without having to know SCCS commands. It provides a way to check files in and out, and to display a file’s delta history and show differences between deltas. With Versioning, you can do the following:

- Check out a version of the file for editing
- Check in files
- Retrieve copies of any version (delta) of a file
- Visually peruse the branches of an SCCS history file
- Back out changes to a checked-out copy
- Display differences between selected deltas using Merging
- Display the version log summarizing executed commands
- Create new SCCS files

### ▼ Starting Versioning

To start Versioning, enter the command as shown:

```
demo% twversion &
```

---

**Note** – Versioning can also be started directly from the Configuring GUI by double-clicking on a workspace icon.

---

To use Versioning, select a file (or group of files) in the File List pane and choose a menu item to operate on it. Commands are located in the:

- Commands menu

- View menu
- File List pane floating menu

Following are two examples that describe how to use Versioning to check out and check in files, and to view and compare a file's delta history.

## ▼ Checking Files In and Out of SCCS

### 1. From a command prompt start Versioning:

```
% twversion &
```

2. If the directory that contains your file is not automatically loaded, you can type the directory path name (followed by Return), in the Directory text field.
3. Click on a file icon to select a file; use the ADJUST mouse button to extend the selection.
4. Choose either Checkout ► Default or Checkout ► Check Out and Edit from the Commands menu. As the files are checked out a check mark appears in their icons.
5. When you are ready to check the files back in, select the file(s) and choose Check In from the Commands menu.  
This activates the Check In popup window.
6. Enter a comment in the text window that describes your changes and click on Check In to complete the check in process.  
The check mark is removed from the file icon as the files are checked in.

## ▼ Viewing and Comparing a File's Delta History

1. To view a graph of a file's delta history, select the file's icon in the main window and choose File History from the View menu.
2. Select two deltas in the graph.

### 3. Choose Use Merging from the Differences menu.

Merging displays the two deltas side by side, marking differences with glyphs. For more information about Merging, see Chapter 11, “How the Configuring Program Merges SCCS Files” in this guide.

## Freeze-pointing

During software development it is often useful to create *freeze-points* of your work at key junctures. These freeze-points serve as “snapshots” of a project that enable you to recreate the state of the project at key development points. With the Freeze-pointing program, you preserve these freeze-points using small storage resources. You can use Freeze-pointing through two functionally equivalent user interfaces:

- Graphical user interface (`twfreeze`)
- Command-line interface (`freezept`)

The concepts discussed in this section apply to both the GUI and the CLI. Descriptions and examples are included for the GUI only. For information on the CLI, see the manual pages: `twfreeze` (1), `freezept`(1), and `freezeptfile`(5).

Freeze-pointing enables you to create freeze-point files from Configuring workspaces. Freeze-pointing files are text files that list the default deltas in SCCS history files contained in the workspace. When you later recreate (extract) the files, Freeze-pointing uses those entries as pointers back to the original history files and to the delta that was the default at the time the freeze-point file was created.

---

**Note** – The recreated files will not contain the original SCCS histories; only the g-files represented by the default deltas from the original hierarchy are recreated. The default delta is the delta that would be retrieved using the SCCS `get` command with no options specified.

---

## ▼ Starting the Freezepointing GUI

To start Freezepointing, at a shell command prompt type `twfreeze` followed by the ampersand symbol (`&`):

```
demo% twfreeze &
```

## ▼ Creating and Extracting Freezepoints

### 1. From a command prompt start the Freezepointing GUI:

```
% twfreeze &
```

The pane below the Control area is used for both creating and extracting freezepoints. Switch between the Create and Extract panes by choosing the appropriate item from the Category menu. The Create pane is the default and is displayed when you start Freezepointing.

### 2. Enter the path name of a freezepoint file.

Freezepointing automatically inserts the file name `freezepoint.out`.

### 3. Delete it and replace it with a path name of your choosing.

### 4. Enter the path name of the source workspace.

This is the workspace that you are “freezepointing.”

### 5. In the Directories and Files text window, compose a list of directories, or files, or both that you wish to freezepoint.

### 6. Choose File ► Add Files to create the Directories and Files list

### 7. Click Create to execute.

### 8. To extract a freezepoint, choose Extract from the Category menu.

This changes the pane from Create to Extract.

### 9. Type the path name of an existing freezepoint file.

### 10. Specify the path name of the Destination Directory.

This the directory into which the newly extracted files are placed.

**11. Click on Extract to execute.**

For more information about Versioning, see Chapter 14, “Performing Basic SCCS Functions with Versioning” in this guide.

## *Distributed Make*

DistributedMake (DMake) marks the evolution of the `make` utility into a powerful and flexible tool that permits you to take full advantage of the potential of today’s networks and powerful multiprocessor workstations. Using DMake, you can concurrently distribute the process of building large projects, consisting of many programs, over a number of workstations and, in the case of multiprocessor systems, over multiple CPUs.

You execute `dmake` on a *DMake host* and distribute jobs to *build servers*. You can also distribute jobs to the DMake host, in which case it is also considered to be a build server. DMake distributes jobs based on makefile targets that DMake determines (based on your makefiles) can be built concurrently. You can use any machine as a build server that meets the following requirements:

- From the DMake host (the machine you are using) you must be able to use `rsh`, without being prompted for a password, to remotely execute commands on the build server. For example:

```
demo% rsh build_server which dmake
/opt/SUNWspro/bin/dmake
```

---

**Note** – For more information about the `rsh` command see the `rsh(1)` man page or the system AnswerBook.

---

- The `bin` directory in which the DMake software is installed must be accessible from the build server.

---

**Note** – For more information see the `share(1M)` and `mount(1M)` man pages or the system AnswerBook.

---

- The `bin` directory in which the DMake software is installed must be in your execution path when you `rsh` to the build server. Be sure this directory is added to the `PATH` variable in your `.cshrc` file (or equivalent), *not* in your `.login` file. You can verify this as follows:

```
demo% rsh build_server which dmake
/opt/SUNWspr0/bin/dmake
```

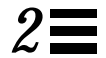
- The source hierarchy you are building must be accessible from the build server.

From the DMake host you can control which build servers are used and how many DMake jobs are allotted to each build server. The number of DMake jobs that can run on a given build server can also be limited on that server.

For more information about DMake see Chapter 20, “Using DistributedMake.”



## Introduction



*Sun Workshop TeamWare* (TeamWare) is a source management product that provides you and your team with:

- software configuration management
- version control
- change control
- integration support
- release support

You can access TeamWare through the command-line interface (CLI) or through several graphical-user interfaces (GUIs). The GUIs and a description of what they do is in the following table:

*Table 2-1* TeamWare GUI Descriptions

Configuring	Software configuration management, previously called CodeManager.
Versioning	Version control. Versioning is a GUI for SCCS. Previously called VersionTool.
Building	Serial, parallel, or distributed make. Building is a GUI for the make utility. Previously called Maketool
Freezepointing	Obtaining a snapshot or freezepoint. Previously called FreezePoint.
Merging	Selectively merging two files. Previously called FileMerge.

## *How to Use the Documentation Supplied with TeamWare*


There are several kinds of documentation provided with TeamWare. Each kind of documentation is designed to help you use TeamWare more easily. An explanation of the TeamWare documentation is provided in the following table.

*Table 2-2* TeamWare Documentation

<b>Type of Documentation</b>	<b>Best Use</b>
Online help	Read online help when you want quick information on how to complete a task with TeamWare. For example, if you want to know how to set up a workspace using a GUI, use online help.
Online books	
• <i>Sun Workshop TeamWare Users Guide</i>	Read the <i>Workshop TeamWare Users Guide</i> (this book) if you want detailed information about TeamWare use.
• <i>Sun Workshop TeamWare Solutions Guide</i>	Read the <i>Workshop TeamWare Solutions Guide</i> to see how others have used TeamWare to be more productive.
On-Item Help (spot help)	Read on-item (spot) help to find out about menus and window displays in the GUIs.
Manual pages (man pages)	Read the manual pages when you want quick information about completing a task with the CLI. For example, if you want to know how to set up a workspace using the CLI, use man pages.
Video	View the video to quickly get an idea of how to use all of TeamWare.

Refer to “How to Get Help” on page xvi of the Preface to see how to access the various types of information.

# *Introduction to TeamWare Configuring*

3 

This chapter contains the following sections:

<i>Coordinating the Work of Software Developers</i>	<i>page 15</i>
<i>Copy-Modify-Merge Model</i>	<i>page 16</i>
<i>Default Configuring</i>	<i>page 19</i>

## *Coordinating the Work of Software Developers*

Managing large programming projects involves coordinating the work of developers who share common and interdependent files.

If developers have private copies of the source code, the changes they make to the source base are difficult to track when all of the code is finally (or even periodically) merged. Often the incompatible changes are subtle, and they can affect the entire project. Preparing the code for a final build and release can be difficult.

One solution is to allow serial access to the common files, one developer at a time. This approach eliminates conflicts due to changes that are made simultaneously. Unfortunately, this approach produces a productivity bottleneck because only one programmer at a time has access to the code.

Developers often change the way source files are grouped and used to build the intermediate and final product. A developer must know what source files, header files, and libraries are required to build a particular program. Often a developer copies a set of files, then later finds that it is incomplete. Only after

repeated failed attempts to build the program is the developer able to determine which files are required to successfully build the program. Also, changes not only occur to files, but often to the file system structure as well. New files and directories are constantly created, renamed, and deleted.

Maintaining a consistent, buildable set of sources in preparation for a product release is also difficult on a large software project. When developers integrate their work directly into the mainline source hierarchy, a set of sources that built correctly one day can be made incompatible the next.

Another problem common to large software projects is the inability to recreate the product at a certain stage of development (for example, a past release). Preserving source code deltas becomes difficult when different copies of files are changed concurrently. Developers generally do not take the time to apply more than one delta. To accurately represent concurrent development, SCCS branch deltas must be used. When deltas are collapsed together, or when parallel deltas are represented sequentially, the true history of the file is lost.

Sometimes development of a feature is begun for a given release and later (often quite near the release date) a decision is made to include the feature in a different release. Backing out the changes and then including them in a different release can be difficult.

## *Copy-Modify-Merge Model*

Sun WorkShop TeamWare assists in the development and release of large software projects. TeamWare is based on a concurrent development model called *Copy-Modify-Merge*. Isolated (per developer) workspaces form the basis of the TeamWare Configuring model. With TeamWare Configuring, you (the developer) *copy* source you want to change from a central workspace into your own workspace, *modify* the source to your liking, and then *merge* your changes with changes made by other developers in the central workspace.

---

**Note** – A workspace is simply a specially designated SunOS™ directory and its subdirectories.

---

The inconvenience of merging changes is outweighed by the productivity increase that results from developers working concurrently. TeamWare Configuring is designed to minimize (and in some cases, eliminate) the inconvenience of merging changes.

---

Besides providing isolated workspaces, TeamWare Configuring enables you to easily and “intelligently” copy files between workspaces and then merge changes that exist between corresponding files. Configuring’s “intelligent” copy feature enables you to copy project files in groups that you (or the project administrator) determine are logically linked; it also automatically determines for you whether differences exist between the files in the originating workspace and the destination workspace.

You copy project files from a central workspace into your own private workspace, make changes to files (or the file system), and then copy your changes back to the central workspace. You can group source files, header files, libraries, and so on, together in logical units that are copied in unison; TeamWare Configuring further assists the concurrent development process by determining whether differences exist between the files in the originating workspace and the destination workspace. If differences *are* found to exist, Configuring prevents you (or another developer) from copying over those changes. Configuring then provides sophisticated window-based tools that help you to merge these differences.

### Copy-Modify-Merge Example

The following is a Copy-Modify-Merge concurrent development model employed by TeamWare Configuring. This example describes a common software development scenario where two developers are working simultaneously on the same or related parts of a project.

Table 3-1

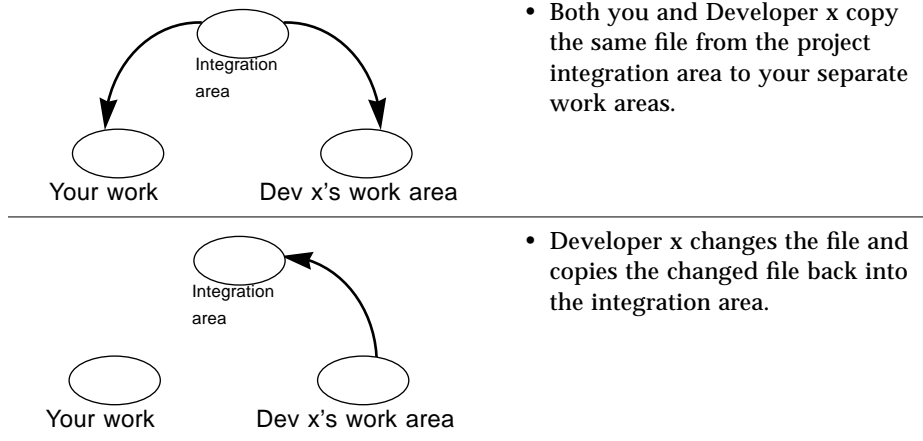
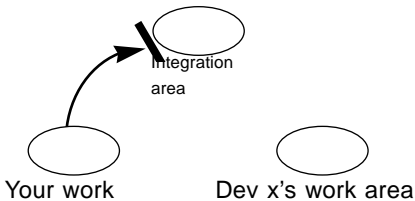
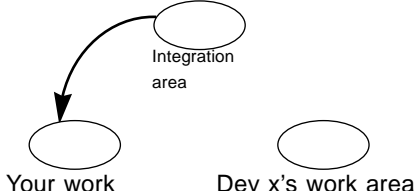
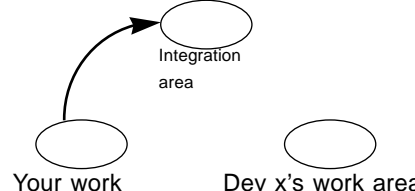


Table 3-1

	<ul style="list-style-type: none"> <li>You modify the same file in your work area and attempt to copy the file back into the integration area. TeamWare Configuring blocks your attempt to copy since it would overwrite Developer x's changes.</li> </ul>
	<ul style="list-style-type: none"> <li>Configuring informs you of the conflicting changes and you copy the file containing Developer x's changes from the integration area to your work area.</li> </ul>
	<ul style="list-style-type: none"> <li>With Configuring's assistance, you resolve the conflicts, merge the changes, test the changes, and successfully copy the file back to the integration area.</li> </ul>

## Default Configuring

TeamWare Configuring can be customized in ways that modify its default behavior; many of those customizations are discussed in Chapter 7, "TeamWare Configuring Workspace." All source files in a Configuring project are maintained under the UNIX SCCS. *TeamWare Configuring only copies files that are under SCCS.* Within your workspaces, you use SCCS in the normal way. For example, you:

- Create files
- Create deltas
- Edit files
- Add comments
- Check in files using SCCS commands

SCCS history files are in SCCS subdirectories, as they would be if the project were not using Configuring. When you copy files between workspaces and merge files that have changed, TeamWare Configuring manages SCCS history files for you, preserving all comments and deltas.

## *Workspace*

The *workspace* is the basis of the Configuring system. The workspace provides the isolation in which developers work concurrently with other developers programming in other workspaces. Project files are propagated between workspaces by Configuring commands. The workspace is a directory and its subdirectory hierarchy. When the workspace is created, Configuring creates a special subdirectory under the workspace, called `Codemgr_wsdata`, to store workspace information.

A Configuring project is created in a top-level workspace from which all others are derived. When other workspaces are created from the original workspace, the original file system hierarchy is recreated to form the new workspace. In the following example, work is begun by a developer in a workspace whose top-level directory is `boatspex`. The workspace exists under the directory `/usr/src/ws`.



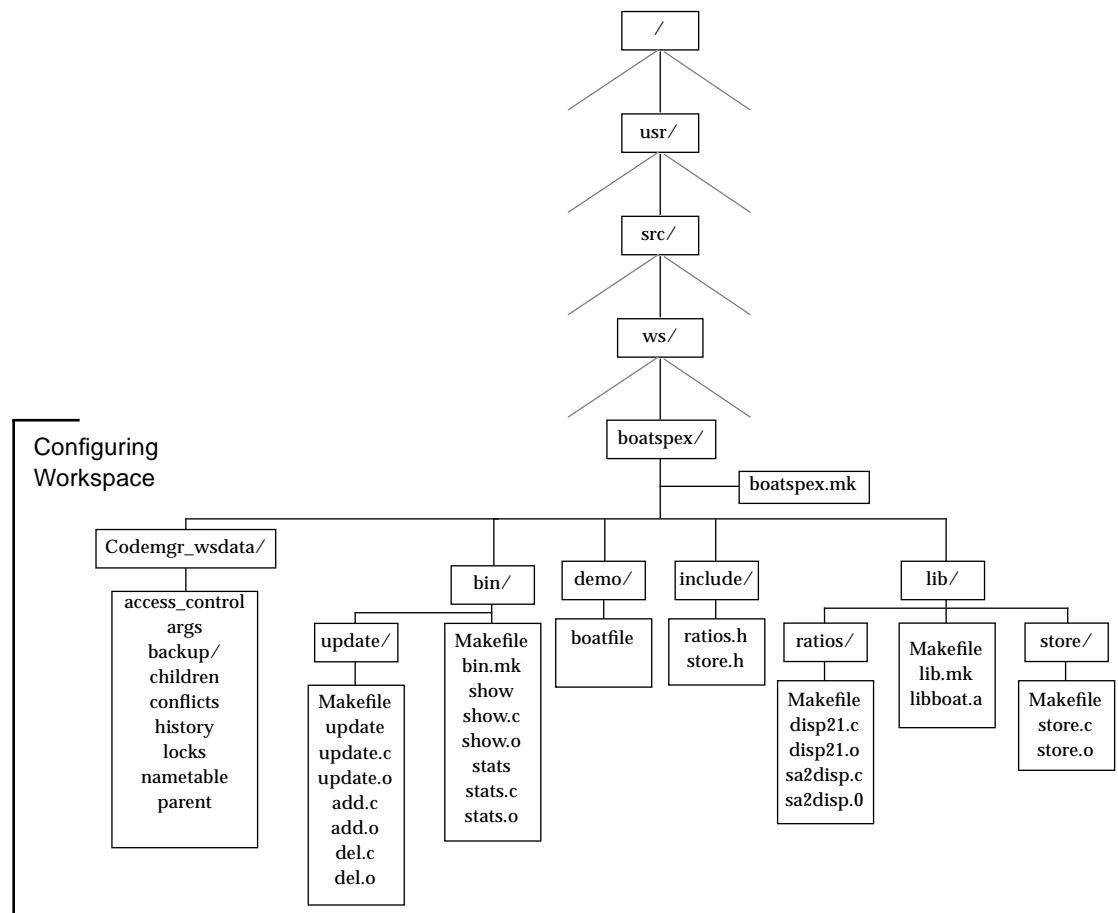


Figure 3-1 Project File System Hierarchy

If you are assigned to work on the Boatspex project you create a copy of the original workspace in a file system of your choice; the workspace portion of the file system in the new workspace is identical to that of the original workspace. If you create the new workspace in your home directory, it appears something like Figure 3-2.

**Note** – If you were only working on a portion of the project, you could copy only that portion.

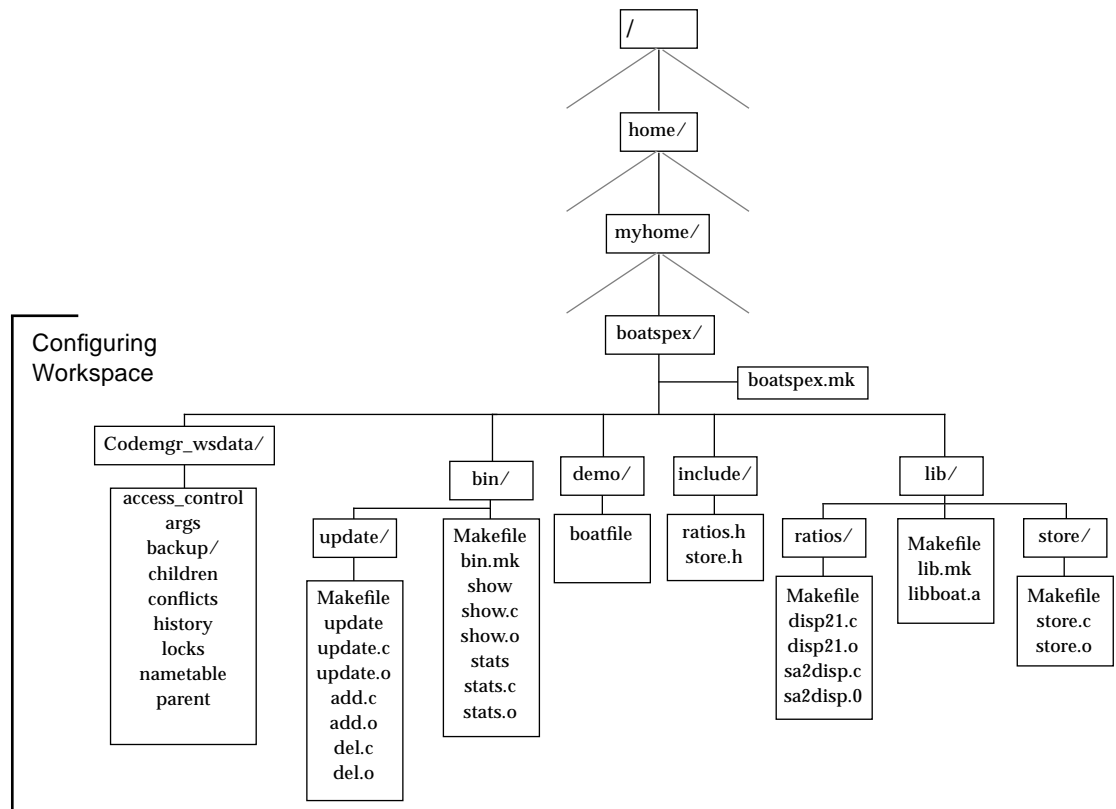


Figure 3-2 Your New Workspace

The directories previous the workspace directory (boatspex) are variable. They change depending on where in the file system you locate the workspace. Below the workspace directory, the file system is a duplicate of the original workspace.

### Parent and Child Relationship

When you copy files from a workspace to create a new workspace, a special relationship is created between the original workspace and the new one. The original workspace is considered the *parent* of the newly created *child* workspace. You can acquire files from any Configuring workspace in this manner, and workspaces can have an unlimited number of children. The

---

portion of the file system that you copy from the parent workspace is determined at the time you copy it. You can copy the entire contents of the parent to the child, making it a clone of the parent, or you can copy only portions of the file system hierarchy that are of interest to you. The Configuring transaction used to copy files from a parent workspace to a child workspace is called Bringover.

---

**Note** – If you use the Bringover transaction to copy files to a workspace that does not already exist, the transaction creates a new child workspace and then copies files to it. This special case is called a *Bringover Create* transaction. You use the *Bringover Update* transaction to update an existing child workspace.

---

The parent and child relationship is special because project data is exchanged only between parent and child workspaces. All files contained in a child workspace were either brought over from a parent workspace or created in the child workspace. When development and testing are complete in the child, you can copy the files that were modified or added in the child back into the parent workspace. Once the altered files are present in the parent, they can be copied by other children or passed up another level to the parent's parent workspace. The Configuring transaction for copying files from a child workspace to a parent workspace is called Putback.

---

**Note** – Unless the child is itself a parent, in which case new files can also be copied to it from its children.

---

Workspace hierarchies are formed by repeating Bringover transactions to create child workspaces. The hierarchy of parent and child workspaces forms a pathway through which data is moved throughout the project.

In the following example, a project is originally created in a workspace and then a three-level workspace hierarchy is created by means of the Bringover transaction. The original workspace is considered to be the parent of the integration workspace and, conversely, the integration workspace is considered to be the child of the original workspace. Developers (Jon, Jack, and Jill) then use the Bringover Create transaction, shown in Figure 3-3 on page 24 to create child workspaces from the integration workspace, which forms a three-tiered hierarchy of workspaces.

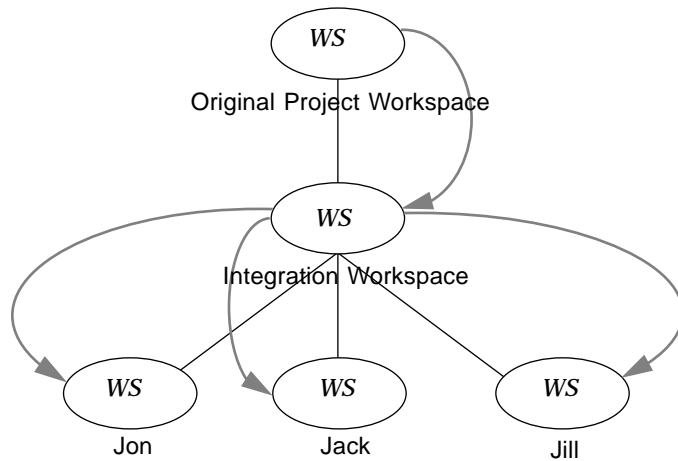


Figure 3-3 Using the Bringover Create Transaction to Create a Workspace Hierarchy

In this hierarchy, files can be disseminated from Jon’s workspace to its “sibling” workspaces owned by Jack and Jill. Jon uses the Putback transaction to copy modified files from his workspace into the common parent (step 1) and then Jack and Jill use the Bringover Update transaction to copy the files from the parent into their workspaces (step 2), shown in Figure 3-4.

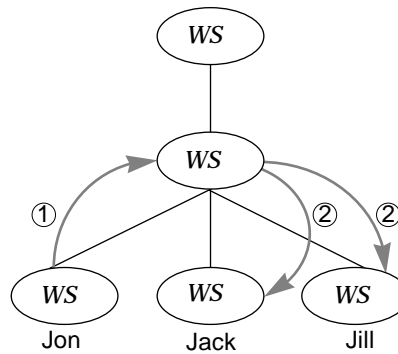


Figure 3-4 Copying Files between Workspaces

## *Reparenting*

Parent and child relationships can be changed. Configuring permits child workspaces to be “reparented” to new parent workspaces. Reasons that you might want to reparent a workspace include the following:

- To reorganize workspace hierarchies
- To populate a new project hierarchy (new top-level workspace)
- To move a feature into a new release
- To apply a bug fix to multiple releases

Refer to “Reparenting a Workspace” for more information.

## *Codemgr\_wsdata Directory*

Every Configuring workspace contains a directory named `Codemgr_wsdata` that is a subdirectory of the workspace top-level (root) directory. This directory contains text files that Configuring uses to log its actions, and store temporary and permanent data. You can view and alter these files using standard text utilities. Refer to “The Workspace Metadata Directory” for more information.

## *Modifying Files*

Since Configuring workspaces are simply directories within the SunOS file system, all your usual tools and utilities can be used on files and directories in workspaces. Your normal edit/compile/debug process is not altered by Configuring.

## *Copying Files between Workspaces*

Once you make and test modifications in a child workspace, you must disseminate them to the rest of the developers working on the project and ultimately to an integration/release workspace.

Every developer in a project needs up-to-date data with which to work. If a modification is made to a module in one part of the project, it could have profound implications for the testing of a different module in another part of the project. Perhaps even more important is the sharing of information between developers working on the same or closely related modules.

Newly modified files (or groups of files) are transferred between parents and children up and down the workspace hierarchy in order to keep workspaces consistent. The decision as to when the data is ready for dissemination is, of course, left to the developer's discretion.

The Putback and Bringover transactions are generally applied to groups of files so that files need not be specified individually. Configuring provides the means for you (or your project administrator) to specify groupings of files that should logically be copied together. Three examples of this type of grouping are as follows:

- Directories
- Files required to build a particular program
- All of the child workspace

How files are grouped for Bringover and Putback transactions between workspaces is discussed in detail in Chapter 8, "Copying Files between Workspaces."

Bringover and Putback transactions are always initiated from within the child workspace. Both transactions are viewed from the perspective of the child workspace—not the parent's.

### *Source Code Control System Files*

When considering Bringover and Putback transactions, remember that source files are derived from SCCS deltas and are identified by SCCS delta IDs (SIDs). When a file is copied by either a Putback or Bringover transaction, Configuring is manipulating the file's SCCS history file (also known as the s-dot-file).

When a file is copied from one workspace to another, Configuring decides how to manipulate the SCCS history file used to derive the file. If the file does not exist in the target workspace, Configuring copies the history file from the source workspace to the target. In the more complicated case—when the file (and thus the SCCS history file) exists in both the source and the target—the SCCS history files must be merged to maintain the file's delta, administrative, and comment history.

Remember, files consist of both the file derived from the latest delta and its predecessors by the SCCS `get` command and the SCCS history file from which it is derived. When files are copied from workspace to workspace, SCCS history files are adjusted appropriately.

### *Bringover and Putback Transactions*

When you initiate a Bringover Update or Putback transaction, Configuring must make a number of determinations before taking any action. Copying files indiscriminately from one workspace to another could overwrite work that you or another developer want to keep. Configuring must check all files specified for transfer to determine where they stand in relationship to each corresponding file in the other workspace.

For example, suppose a file was modified in the parent (perhaps put back from another child) since it was last brought over into your child. You have modified your copy of the same file in your child workspace. When you attempt to put back that file (or a group of files that contains that file) from your child workspace to the parent, Configuring will not allow your Putback transaction to proceed because it would cause the revised version of the file in the parent to be overwritten by the version of the file from your child. In this case, Configuring blocks your attempt to put back the files into the parent and informs you of the conflicting change.

When a Putback or Bringover Update transaction is blocked, none of the files in the group are copied, even those that don't conflict.

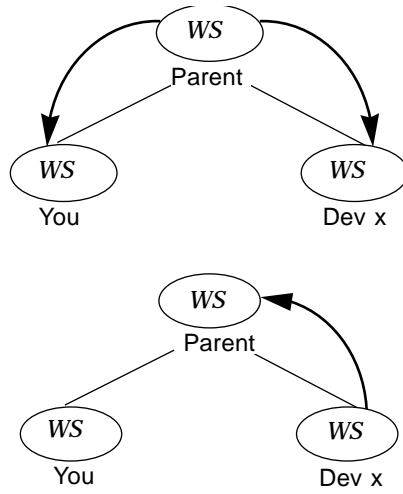
The conflicts between your versions of the files and the versions in the parent must be resolved in your (child) workspace. *Conflicts are always resolved in the child workspace to preserve the integrity of the parent.*

You use the Bringover Update transaction to copy the conflicting files from the parent to your workspace, and using Configuring's merge tool, you merge your changes with those made by the other developer. After testing the changes you then put back the merged files to the parent workspace.

### *Relationships between Files in Parent and Child Workspaces*

The previous example describes only one of four possible states of relationship that can exist between corresponding files in parent and child workspaces. The relationship between files in parent and child workspaces governs the way that Configuring behaves when you attempt to copy files via Putback and Bringover Update transactions. Following are descriptions of the four cases and the action Configuring takes in each case:

Figure 3-5 Keeping Work Synchronized

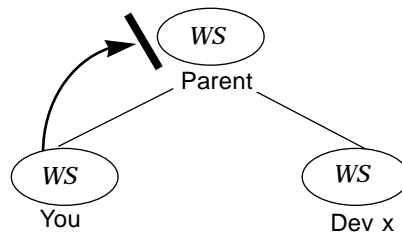


- Both you and Developer x bring over the same file to your workspaces.

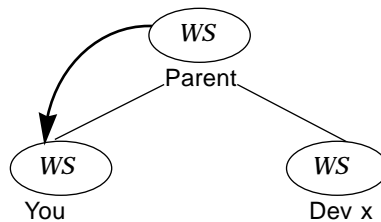
- Developer x changes the file and puts the changed file back into the parent.



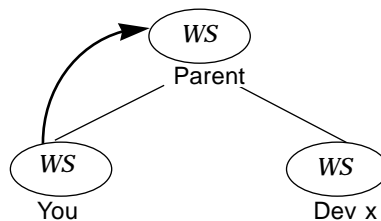
Figure 3-5 (Continued) Keeping Work Synchronized



- You change the same file in your workspace and attempt to put the file back into the parent. Configuring blocks the Putback.



- Configuring notifies you of the conflicting changes and you bring the file over to your workspace (actually, the SCCS history files are merged).

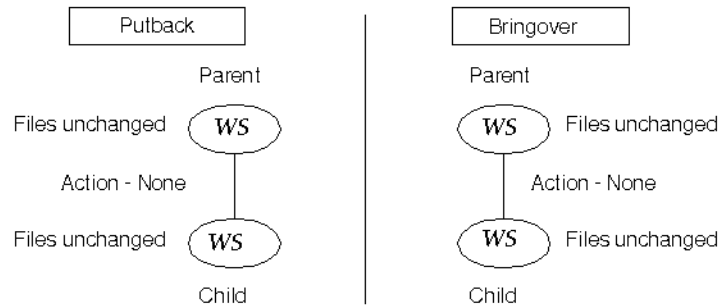


- You resolve the conflict, test the changes and successfully put back the file back to the parent workspace.

### Case 1

Neither the files in the parent nor the corresponding files in the child have been modified since they were put back into the parent or brought over into the child

In this case no action is required by Configuring in either case. The files are exactly the same in both the parent and child.

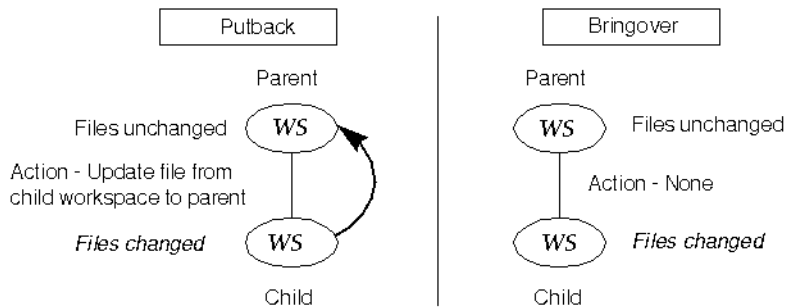


**Case 2**

The specified files were not modified in the parent since they were brought over from the parent into the child or put back from the child into the parent. The corresponding files *were* modified in the child.

In this case when you use the Putback transaction to copy the file to the parent, the changed files are updated from the child into the parent, replacing the corresponding files in the parent. This new data is available for acquisition by other children of that parent or to be further propagated up to the parent's parent workspace.

When you use the Bringover Update command in this case, no action is taken because copying the file from the parent would overwrite changes made in the child.

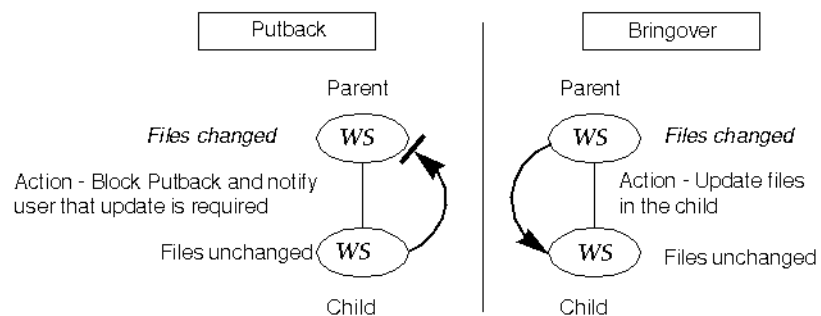


### Case 3

One or more files in the parent were modified since their corresponding files were brought over into the child or put back into the parent from that child. The corresponding files in the child were not modified.

In this case the parent's copy of the file being put back from the child was modified (probably by one of its other children) since it was last brought over to the child; the corresponding file in the child was not modified since it was last brought over into the child.

When Configuring detects this situation during the Putback transaction, it cannot update the parent workspace until the child workspace is updated by means of the Bringover Update transaction. Even if the changes are in files that you have not altered (remember you're copying *groups* of files), they might impact the changes you have made. In this case, the Putback transaction is blocked and the user is notified. It is the user's responsibility to execute the Bringover Update transaction in order to update the child workspace.



### Case 4

Corresponding files were modified in both the parent and child workspaces.

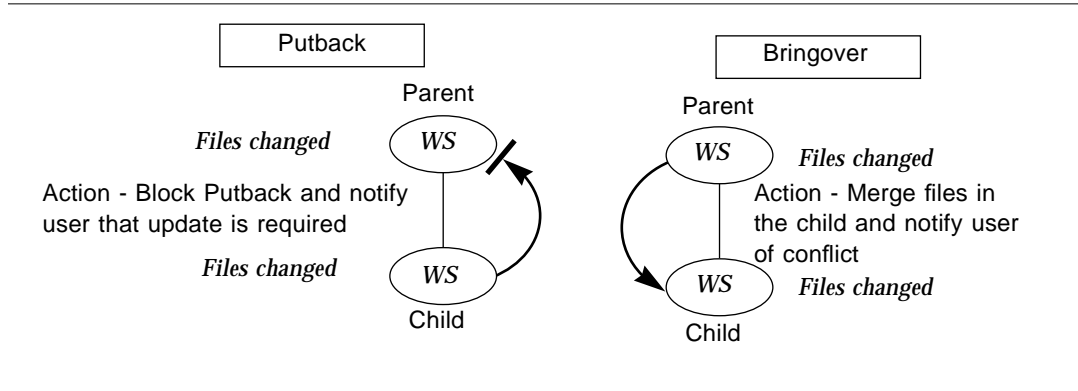
This is the most complicated of the four cases. Configuring cannot allow the file to be put back from the child into the parent because the transaction will obscure modifications there. Likewise, Configuring cannot allow the file to be brought over from the parent into the child because the transaction will overwrite modifications there.

As in case 3 above, Configuring blocks the Putback transaction and notifies the user. When the user attempts to update the child workspace by means of the Bringover transaction, Configuring detects that the file in the child has also

been changed; the file cannot be updated without overwriting the newly created work in the child. In this case Configuring merges the parent and child SCCS history files for the conflicting file in the child workspace.

Configuring merges the parent and child SCCS history files together in the child workspace; the SIDs that were created in the child are renamed and placed on an SCCS branch off of the current line of work brought down from the parent. Although it is a branch, the child's SCCS version tree remains the default for any additional deltas so that work on the file may proceed in the child as if nothing had changed. The merge process places all needed deltas in the SCCS history file so that the conflicting files can be merged at the user's discretion. All SCCS comments are preserved in this process since the entire SCCS delta history is preserved.

At this point the conflict between the parent and child versions of the file is still open. Work can continue on the branch that contains the deltas created in the child; any new deltas will be added to the branch. *However, the user must resolve the conflict before the group of files that contain the conflicting file(s) can successfully be put back to the parent.* Conflict resolution is discussed in the next section.



## Summary

The following two tables summarize, first, the action taken by Configuring during a Putback transaction in each of the four cases described above and secondly, for the Bringover transaction.

Table 3-2 Putback Transaction

Case	File in Parent	File in Child	Action by Configuring
1	Unchanged	Unchanged	None
2	Unchanged	Changed	Update file in parent
3	Changed	Unchanged	Block Putback, notify user
4	Changed	Changed	Block Putback, notify user

Table 3-3 Bringover Transaction

Case	File in Parent	File in Child	Action by Configuring
1	Unchanged	Unchanged	None
2	Unchanged	Changed	None
3	Changed	Unchanged	Update child (extend SCCS files)
4	Changed	Changed	Merge SCCS history files and notify user of conflict

## Resolving Conflicts

During the Putback transaction, Configuring may determine that a file in the parent has been modified since it was last put back from that child or brought over into the child. In that case it blocks the Putback so that the changes are not overwritten and then notifies the user of the potential conflict.

Generally the owner of the child workspace will then attempt to update the child by bringing over the changed file. If, during the Bringover Update transaction, Configuring determines that the corresponding file in the child has *also* been modified since it was last brought over, a conflict exists.

Conflicts arise when corresponding files in both the parent and child have been modified. If Configuring were to overwrite either of the files, a loss of data would result. Before the specified file can be put back or brought over the user must resolve any conflicts.

When Configuring detects a conflict during the Bringover Update transaction, as described in the previous section, it then does the following:

- Merges the parent and child SCCS history files for the conflicting files in the child workspace
- Notifies the user of the conflict
- Assists the user in resolving the conflict

---

**Note** – All conflicts are resolved from within the child workspace and from the perspective of the child workspace.

---

In the case of most conflicts, the options available to the user for resolving conflicts are:

- Install the latest delta from the parent as the resolved version in the child.
- Accept the latest delta from the child as the resolved version of the file. Since it has been through the resolve process, its Putback transaction will no longer be blocked in the parent.
- Merge the contents of latest delta from the parent with that of the child.

Configuring provides tools that aid in resolving conflicts, however, the conflicts must be resolved by the user. Refer to Chapter 9, “Resolving Conflicts,” for a detailed discussion about conflict resolution.

# Introduction to Merging



This chapter deals with the basic concept of Merging files, and is organized into the following sections:

<i>Differences Defined</i>	<i>page 36</i>
<i>Graphical Overview</i>	<i>page 37</i>

For explicit details on using Merging, together with an example, see Chapter 15, "Starting and Loading Merging."

Merging loads and displays two text files for side-by-side comparison, each in a read-only text pane. Merging marks lines that differ between the two files and displays a merged version in a third text pane. When automatically activated, the merged version contains two types of lines:

- Lines that are common to both input files (these lines always appear in the output file)
- Marked lines that are different in each file (these lines appear as the result of the default automerge process)

You can edit the merged version and save it as an output file.

At the time you load the two files to be merged, you can also specify a third file, called the *ancestor* of the two files (which are called its *descendants*). When you have specified an ancestor file, Merging marks lines in the descendants

that are different from the ancestor and produces a merged file based on all three files. To automatically merge (automerge) the two input files, you must specify an ancestor file.

## *Differences Defined*

Merging operates on *differences* between files. Although you probably have a good intuitive grasp of what a difference is, the following describes how Merging recognizes and classifies differences.

### *Difference*

When Merging discovers a line that differs between the two files to be merged (or between either of the two files and the ancestor), it marks with glyphs the lines in the two files. Together, these marked lines are called a *difference*. While Merging is focusing on a difference, it highlights the glyphs.

### *Current, Next, and Previous Difference*

The difference on which Merging is focusing at any given time is called the *current* difference. The difference that appears immediately later in the file is called the *next* difference; the difference that appears immediately earlier in the file is called the *previous* difference.

### *Resolved and Remaining Difference*

A difference is *resolved* if either you or Merging accept the changes to a line. Differences are resolved one of two ways:

While focusing on a difference, you can accept a line from one of the original files, or you can edit the merged version by hand. When you indicate that you are satisfied with your changes (by clicking on a command button), the current difference is then resolved.

If the Auto Merge feature is on, Merging resolves differences automatically. For more information on how Merging resolves differences, see the discussion in "Merging Glyphs."



---

After a difference is resolved, Merging identifies it by changing its associated glyphs from solid to outline font. Merging then automatically advances to the next difference (if the Auto Advance property is on) or moves to the difference of your choice.

A *remaining* difference is one that has not yet been resolved.

## *Graphical Overview*

The graphical interface for Merging consists of one main window, in which you do most of your work, and pop-up windows for handling files and setting properties. Descriptions of the graphical interface are found in the online help. Pull down the Help menu on the upper right of the Merging menu bar to access Help Contents.

### *Merging Window*

The Merging window at startup shows the left and right text panes at the top displaying the files to be compared; the text pane at the bottom displays a merged version of the two files that you can edit.

### *Merging Glyphs*

When files are loaded in the text panes, glyphs appear to indicate the disparities. There is a difference between two files being merged without a common ancestor, and two files that have a common ancestor (this case is actually a three-way merge). The meaning of the glyphs in each case is slightly different, as explained below.

#### *Two Input Files*

When only two files have been loaded into Merging, lines in each file are marked by glyphs to indicate when they differ from corresponding lines in the other file:

- If two lines are identical, no glyph is displayed.
- If two lines are different, a vertical bar (|) is displayed next to the line in each input text pane, and the different characters are highlighted.

- If a line appears in one file but not in the other, a plus sign (+) is displayed next to the line in the file where it appears, and the different characters are highlighted.

### *Three Input Files*

When an ancestor file has been specified for the two files to be merged, lines in each descendant are marked according to their relationship to the corresponding lines in the common ancestor:

- If a line is identical in all three files, no glyph is displayed.
- If a line is not in the ancestor but was added to one or both of the descendants, a plus sign (+) is displayed next to the line in the file where the line was added, and the different characters are highlighted.
- If a line is present in the ancestor but was removed from one or both of the descendants, a minus sign (-) is displayed next to the line in the file from which the line was removed, and the different characters are highlighted and in strikethrough.
- If a line is in the ancestor but has been changed in one or both of the descendants, a vertical bar (|) is displayed next to the line in the file where the line was changed, and the different characters are highlighted.

Resolved differences are marked by glyphs in outline font.

Table 4-1 summarizes the automerging algorithm. Ancestor is the version of a text line that is in the ancestor file; Change 1 is a change to that line in one of the descendants; Change 2 is another change, different from Change 1. Only when a line is changed differently in the left and right descendants does automerging fail.

*Table 4-1 Automerging Rules Summary*

<b>Left Descendant</b>	<b>Right Descendant</b>	<b>Automerged Line</b>
Ancestor	Ancestor	Ancestor
Change 1	Ancestor	Change 1
Ancestor	Change 2	Change 2
Change 1	Change 1	Change 1
Change 1	Change 2	No Automerge

---


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

You can disable automatic merging in the Properties window. When automatic merging is disabled, Merging constructs a merged file using only lines that are identical in all three files and relies on you to resolve the differences.

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



## Starting a Project

5 

When you begin to use *Sun Workshop TeamWare*, you are probably moving an existing software development project to *TeamWare* or you may be starting a new software development project. This chapter contains information about:

<i>Moving a Non-SCCS Software Development Project to Workshop TeamWare</i>	<i>page 42</i>
<i>Starting a New Software Development Project with Workshop TeamWare</i>	<i>page 43</i>
<i>Product Release Considerations</i>	<i>page 47</i>

### *Moving an Existing SCCS-Based Project to Sun Workshop TeamWare*

To move an existing SCCS-based software project to TeamWare do the following:

- 1. Ensure that all SCCS history files (s-dot files) are in directories named SCCS and that these files are directly beneath directories containing source files.**  
Configuring works only on files under SCCS version control.
- 2. Be sure that your project directory structure is current and organized.**
- 3. Execute the Create Workspace command item in the File menu, specifying the top-level directory as your workspace. The Create Workspace command creates the Codemgr\_wsdata directory under the top-level directory.**

- 4. Begin using the Bringover Create transaction to form a workspace hierarchy. See “Configuring a Workspace Hierarchy” on page 44 for guidelines regarding workspace hierarchies.**

If your project is structured so that compilation units can be easily grouped on a directory basis during transfer operations, you can use the default Configuring FLP. See “Grouping Files for Transfer Using File List Programs” on page 86 for a description of the default FLPs.

If your project requires files to be grouped for transfer operations in special ways, you will have to write your own FLP(s).

## *Moving a Non-SCCS Software Development Project to Workshop TeamWare*

To move an existing project into Workshop TeamWare do the following:

- 1. Use the Create Workspace command item in the File menu to create your project’s top-level directory (with its Codemgr\_wsdata directory)**
- 2. Traverse to the directory where the existing development files live.**
- 3. Start Workshop Versioning and select Check In New.**  
Select the files you want to include in the development project, and enter an initial comment.

## *Converting an RCS Project to Workshop TeamWare*

`rcs2ws` is a program that produces a CodeManager workspace from an RCS source hierarchy. It converts a project developed in RCS (Revision Control System) and works its way down through the hierarchy to convert the RCS files to SCCS.

`rcs2ws` operates on RCS files under the parent directory and converts them to SCCS files, then puts the resulting SCCS files into a workspace. If a workspace doesn’t already exist, it will be created. The parent directory hierarchy is unaffected by `rcs2ws`.

To convert files, `rcs2ws` invokes the RCS `co` command and the SCCS `admin`, `get` and `delta` commands. These commands will be found using the user’s `PATH` variable. If the SCCS commands can’t be found, then they will be sought in the `/usr/ccs/bin` directory.

## ▼ How to Use rcs2ws

### 1. Type `rcs2ws -p [parentdir] -n [files or directory]` at the prompt.

The `-p` option is required to name the RCS source hierarchy, the `-n` option allows you to see what would be done without actually doing anything. This will specify the value of the shell environment variable as `CODEMGR_WS`. If the current directory is contained within a workspace, the containing workspace is used as the child workspace. If workspace does not exist, `rcs2ws` will create it.

If you wish to designate the child workspace, use the `-w` option. See `man rcs2ws (1)`. If all is as you wish, go on to Step 2.

### 2. Reenter the `rcs2ws` command using only the `-p` option.

Normally, SCCS gets (g-files) are extracted after the files are converted from RCS. If you want to halt the process, do the conversion using the `-g` option. (`rcs2ws -g`)

## *Further User Notes*

- The “.” directory may be used to specify that every RCS file under *parentdir* should be converted.
- Relative file names are interpreted as being relative to *parentdir*.
- Directories are searched recursively.
- `rcs2ws` does not convert RCS keywords to SCCS keywords. Keywords are treated as text in the SCCS delta.
- `CODEMGR_WS` variable contains the name of a user’s default workspace. The workspace specified by `CODEMGR_WS` is automatically used if the `-w` option is not specified.

## *Starting a New Software Development Project with Workshop TeamWare*

To start a new project with Workshop TeamWare do the following:

1. Use the **Create Workspace** command item in the **File** menu to create your project’s top-level directory (with its `Codemgr_wsdata` directory)

- 2. Proceed as you would to set up an SCCS-based development hierarchy. Ensure that all SCCS history files (“s-dot-file”) are in directories named SCCS located directly beneath directories that contain source files.**
- 3. Begin using the Bringover Create transaction to form a workspace hierarchy. See “Configuring a Workspace Hierarchy” on page 44, for guidelines regarding workspace hierarchies.**
- 4. The default Configuring FLP groups files recursively by directory; if you intend to use that FLP, be sure to arrange files in compilation units accordingly. If your project requires that files be grouped differently during transfer, be sure to arrange your project hierarchy in such a way that it works well with the FLP(s) you will create.**

### *Configuring a Workspace Hierarchy*

How you configure a project workspace influences the inter-workspace file-transfer process and the way you prepare product releases. This section will help you choose the workspace hierarchy best suited for your project. You can change any decisions you make now regarding workspace hierarchies by using the Configuring workspace reparenting feature. See “Reparenting a Workspace” on page 66 for details. You can also refer to the *Sun WorkShop TeamWare: Solutions Guide* for examples.

A workspace hierarchy is a chain of parent and child workspaces that is two or more layers deep. The number of layers in a hierarchy bears no relation to the number of workspaces comprising it. A parent workspace and its child comprise two layers. A parent workspace and three children also comprise two



layers. A parent workspace and its child and grandchild comprise three layers. Figure 5-1 depicts a “flat” (three-tiered) hierarchy, and Figure 5-2 shows a “multitiered” (four-tiered) hierarchy.

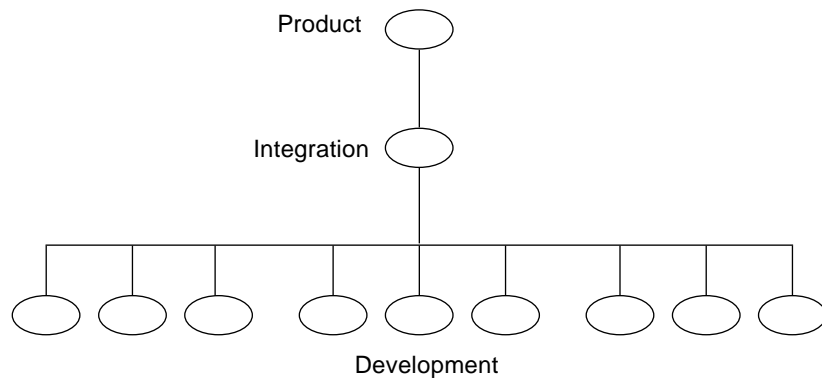


Figure 5-1 A “Flat” (Three-Tiered) Hierarchy

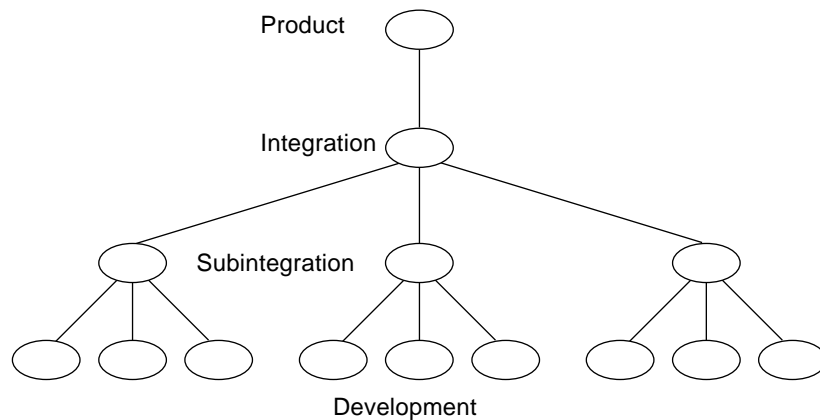


Figure 5-2 A “Multitiered” (Four-Tiered) Hierarchy

## ***File Transfer Considerations***

The way in which you set up your workspace hierarchy can have an impact on the transfer of files up and down the hierarchy.

### ***File System Accessibility***

In order to transfer (Bringover/Putback) files between workspaces, both the parent and the child must be mounted on the same file system. The automounter can be used to connect file systems.

### ***Flat Hierarchy vs. Multitiered Hierarchy***

To properly design your workspace, you need to be aware of the advantages and disadvantages of flat and multitiered hierarchies.

#### ***Advantages of a Flat Hierarchy***

A flat workspace hierarchy is one in which many developers put back files to a single integration workspace. The advantage of a flat hierarchy is that all developers have immediate access to one another's work. The moment that Jack (a developer) puts back his work to the integration workspace, Jon (another developer) can use the Bringover Update transaction to have immediate access to the changes made by Jack.

#### ***Disadvantages of a Flat Hierarchy***

The disadvantage of a flat hierarchy is that time is often wasted because the integration workspace changes frequently, requiring developers to do frequent Bringover transactions, builds, and tests in order to keep their source base up-to-date. There is a cumulative effect of doing Putback transactions; the first developer to do a Putback resolves only one set of changes, the next developer resolves two, and so on till the last developer, who must resolve all of the changes that have been made within her development group.

#### ***Advantages of a Multitiered Hierarchy***

The amount of time required for a developer to put her work back to the integration workspace can be sharply reduced by interposing a tier of subintegration workspaces between the integration and development level workspaces.

Whenever a developer puts back work to an integration workspace, there is some chance that the next developer to do a Putback transaction will not be able to put back their changes until they bring over the earlier changes, rebuild the modules, and test the new changes with their own—the more Putbacks that occur the higher the potential for conflict.

---

When many developers work on a project, the Bringover, rebuild, test cycle can become onerous and time consuming. If smaller groups of developers working on related portions of code integrate into a subintegration workspace, that workspace will be more stable and require fewer builds and less testing. Of course when the subintegration workspaces are themselves put back to their common integration area, changes made in the other development workspaces will have to be integrated. Experience has shown, however, that doing larger integrations, less frequently, is more efficient.

### ***Disadvantages of a Multitiered Hierarchy***

The disadvantages of multiplying subintegration workspaces are as follows:

- Each new workspace consumes disk space.
- Developers who ought regularly to be looking at one another's work may find it harder to do so because they do not put back to the same integration workspace
- Integration of the subintegration workspaces to the higher integration workspace can become more complicated than more frequent, smaller integrations.

## ***Product Release Considerations***

When you plan your project hierarchy structure, consider how you plan to release your product. There are a number of ways that you can structure workspace hierarchies to facilitate the preparation of major, minor, and patch releases. The following discussion presents some ideas for you to consider; your product may not lend itself to this model, or your product may have considerations that suggest an alternate scheme. The *Sun WorkShop TeamWare: Solutions Guide* presents several ways to structure project hierarchies.

You should consider dedicating a workspace as a product release staging area for each release. “Hang” the release workspace off a top-level “product” workspace. The product workspace should be located hierarchically above the workspaces in which normal development integration is done. Locating the product workspace this way permits you to begin development of your next release without corrupting the current release.

After the files are transferred to the product workspace, you use the Bringover transaction to transfer the files down to the release workspace. The release workspace can be used to make masters and can serve as an area in which to save work for subsequent releases if necessary.

Figure 5-3 shows a hierarchy that contains a product workspace and release workspaces for six different releases.

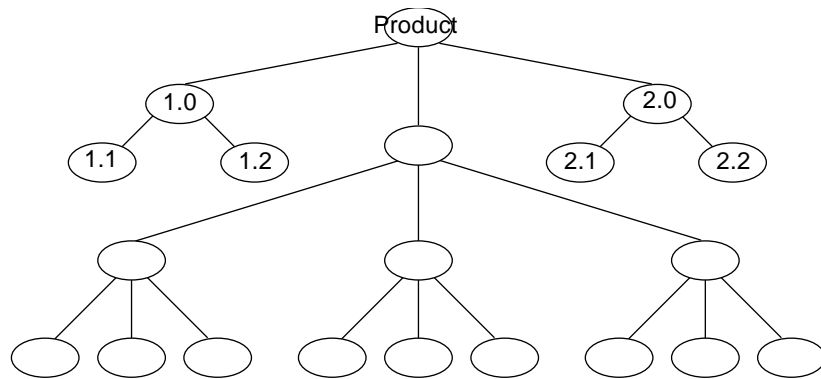


Figure 5-3 Hierarchy with Product and Release Workspaces

---

**Note** – You can use the reparenting feature to transfer data between release workspaces directly. See “A Reparenting Example” on page 68 for details. Refer also to the *Sun WorkShop TeamWare: Solutions Guide* for workspace hierarchy strategies.

---

## Coordinating Access to Source Files

Coordinating write access to source files is important when changes will be made by several people. Maintaining a record of file updates allows you to determine when and why changes were made.

The source code control system (SCCS) allows you to control write access to source files and monitor changes made to those files. The SCCS allows only one user at a time to update a file, and it records all changes in a history file.

---

Versioning is a GUI to SCCS. Versioning allows you to manipulate files and perform most of the basic SCCS functions without having to know SCCS commands. It provides an intuitive method for checking files in and out, as well as displaying and moving through the history branches.

With Versioning, you can do the following:

- Check in files under SCCS
- Check out and lock a version of the file for editing
- Retrieve copies of any version of the file from SCCS history
- Visually peruse the branches of an SCCS history file
- Back out changes to a checked-out copy
- Inquire about the availability of a file for editing
- Inquire about differences between selected versions using Filemerge
- Display the version log summarizing executed commands

Versioning helps you perform these tasks and expedites the progress of concurrent development projects.

## *Branches*

You can picture the deltas applied to an SCCS file as *nodes* of a tree with the initial version of the file as the *root*. The root delta is numbered 1.1 by default. These two parts of the SCCS delta ID (SID) are the release and level numbers. Successive deltas (nodes) are named 1.2, 1.3, and so forth. This structure is called the *trunk* of the SCCS delta tree. It represents the normal sequential development of an SCCS file.

It may be necessary to create an alternative *branch* on the tree. Branches can be used to keep track of alternate versions developed in parallel, such as for bug fixes.

The SID for a branch delta consists of four parts: the release and level numbers and the branch and sequence numbers, or *release.level.branch.sequence*. The *branch* number is assigned to each branch that is a descendant of a particular trunk delta; the first branch is 1, the next 2, and so on. The *sequence* number is assigned, in order, to each delta on a particular branch. Thus, 1.3.1.1 identifies the first delta of the first branch derived from delta 1.3. A second branch to this delta would be numbered 1.3.2.1 and so on.

The concepts of branching can be extended to any delta in the tree. The branch component is assigned in the order of creation on the branch, independent of its location relative to the trunk. Thus, a branch delta can always be identified from its name. While the trunk delta can be identified from the branch delta's name, it is not possible to determine the entire path leading from the trunk delta to the branch delta.

For example, if delta 1.3 has one branch, all deltas on that branch will be named 1.3.*n*. If a delta on this branch has another branch emanating from it, all deltas on the new branch will be named 1.3.2.*n*. The only information that can be derived from the name of delta 1.3.2.2 is that it is usually the second chronological delta on the second chronological branch whose trunk ancestor is delta 1.3. In particular, it is *not* possible to determine from the name of delta 1.3.2.2 all of the deltas between it and its trunk ancestor (1.3).

# *TeamWare Configuring User Interfaces*

## **6**

You can work with TeamWare Configuring in two ways:

- Use the command-line interface (CLI).
- Use the Configuring graphical user interface (GUI).

Both interfaces are included with TeamWare to accommodate different computing styles. Complete TeamWare Configuring functionality is implemented in both interfaces. The interfaces can be used interchangeably. You can simultaneously use the GUI for some functions and the CLI for others. Both interfaces employ the same underlying Configuring functionality and command structure; the difference is an easy-to-use, graphical, interface for the GUI.

The concepts in this chapter generally apply to both the GUI and the CLI. Except in cases where there are special considerations regarding the CLI, descriptions and examples are included for the GUI only—information specific to the CLI can be obtained online through the manual pages. The remainder of this chapter is an introduction to the Configuring CLI and GUI.

This chapter is organized as follows:

<i>TeamWare Configuring Command-Line Interface</i>	<i>page 52</i>
<i>TeamWare Configuring Graphical User Interface</i>	<i>page 53</i>
<i>Starting TeamWare Configuring</i>	<i>page 53</i>

## TeamWare Configuring Command-Line Interface

The TeamWare Configuring CLI is accessible from any Solaris shell. The CLI is useful when you are not working on a window-based system.

Like SCCS commands, all CLI commands may be executed through a central “umbrella” command. The individual commands may also be executed directly by specifying the individual command name.

The umbrella command `codemgr` enables you to list Configuring commands (Code Example 6-1). You can list Configuring commands with their use summaries by executing `teamware` without specifying any arguments. You can achieve the same results by executing `codemgr` with the `help` subcommand.

*Code Example 6-1* `codemgr` Umbrella Command

```
example% codemgr
    bringover ...
    codemgrtool
    help
    putback ...
    resolve ...
    ws_undo ....
    workspace ....
```

To use the umbrella command to execute commands, type `codemgr` followed by the name of the subcommand you wish to execute. For example:

```
% codemgr bringover -w my_child -p their_parent /usr/ws/project
```

You can also execute the commands directly (without typing `codemgr`). For example:

```
% bringover -w my_child -p their_parent /usr/ws/project
```

---

**Note** – You must use the individual command name when you access TeamWare manual pages.

---



---

TeamWare provides several ways to reduce typing long command-lines, including environment variables and argument files that store previously specified arguments. See the respective manual pages for details.

## *TeamWare Configuring Graphical User Interface*

The TeamWare GUI (Configuring) is a tool that enables you to view workspace hierarchies and to execute menu-based commands on workspaces and their contents. Key features include the following:

- Graphical display of workspaces in a Workspace Graph pane. This feature enables users to:
  - Conveniently view workspace hierarchies.
  - Use the mouse to select workspace icons.
  - Execute menu-based commands on selected workspaces and their contents.
- Menu lists that reduce the need for you to remember and type command names, options, and arguments.
- Facilities to customize the GUI to meet your individual style and needs.
- Online Help to assist you at all levels, including explanation of error messages.

---

**Note** – TeamWare windows, menus, and buttons are documented online using the Help feature. With Help, you can obtain information regarding any object on the screen. Therefore, this section does not discuss these objects in detail; rather, it serves as a guide to other aspects of Configuring. Throughout the rest of this chapter, Configuring tasks such as Bringover and Putback transactions and conflict resolution are discussed in detail.

---

## *Starting TeamWare Configuring*

To start TeamWare Configuring, at a shell command prompt type `twconfig` followed by the ampersand symbol (&) as shown. After a moment, the Configuring window appears.

```
demo% twconfig &  
demo%
```

## *TeamWare Configuring Windows*

Configuring consists of a base window and a number of pop-up windows. Within this base window is a menu bar from which you can choose commands, menu items, and window items to help you accomplish your tasks. The online Help contains a full explanation of TeamWare Configuring Windows.

You navigate the file system with point-and-click windows. Select files and directories by moving the mouse pointer over icons and clicking SELECT. You can make multiple selections using two different methods:

- Use the ADJUST mouse button to extend the selection to multiple files or directories.
- Press SELECT in an open area of the pane and drag a bounding box diagonally until the desired group of icons is enclosed, then release SELECT.

When you make your selection, select the button at the bottom of the chooser window to make your choice effective. You can also choose a file or directory by typing its name in the Name text field and selecting the Add Files to List button (or typing Return).

You can navigate down through the file system hierarchy by double-clicking SELECT on any directory icon. To move hierarchically upward, double-click SELECT on the directory icon. To move directly to a directory, enter its path name in the Name text field and select the Load Directory button.

---

**Note** – Mouse button action is different in the Motif/CDE interfaces. To make a selection, click the left mouse button to select the first item, then extend the selection by clicking the left mouse button on another item while pressing the Shift key (this action is called Shift-SELECT). If you press Shift-SELECT on an item that is already selected, that item is deselected.

---

---

**Note** – A check mark in a file icon indicates that the file is checked out from SCCS.

---

---

## *Configuring Window*

When you start the Configuring program, the base window appears. In working with Configuring, you select workspace icons in the Workspace Graph pane and then choose commands that act upon the selected workspaces and the files they contain.

### *Workspace Graph Pane*

In the Workspace Graph pane each workspace is represented by a workspace icon. Parent and child relationships are depicted by lines connecting workspaces. The path name of the workspace top-level (root) directory is displayed beneath the icon.

### *Loading Workspaces into the Workspace Graph Pane*

When Configuring is started, it checks the directory (or directories) specified by the environment variable CODEMGR\_WSPATH to determine if it contains any workspaces. If workspaces are found, they are loaded into the Workspace Graph pane. If CODEMGR\_WSPATH is not set, Configuring attempts to load workspaces from the directory in which it was started. To load additional workspaces, use the Load Workspaces window from the File menu.

### *Layout*

Workspace hierarchy graphs are automatically created in the Workspace Graph pane by the Configuring program as you load workspaces (using the Load menu). Hierarchies are displayed either vertically or horizontally starting from the upper-left corner and distributed to the right as space permits. You can choose the orientation using the Orientation item from the View menu. Vertical orientation is the default. Layout is done automatically—you are not able to change the layout by moving icons with the mouse.

### *Workspace Name Fields*

Beneath the workspace icon is a text field that contains the name of the workspace root directory. You can choose to have workspace names displayed one of two ways:

- Using the absolute (full) path name of the root directory
- Using the truncated (short) name of root directory

Choose the display style you prefer using the Name item from the View menu. You can change the path name of a workspace by editing the name text field.

***Dragging and Dropping Workspace Icons***

You can accomplish two types of operations by directly manipulating icons on the Workspace Graph pane. You can “drag and drop” workspace icons to initiate both Bringover and Putback transactions and to reparent workspaces.

- **Interworkspace transactions**—If you select and drag a workspace and drop it on top of another icon, the Configuring program will initiate one of the following transactions: Bringover Create, Bringover Update, Putback. You determine which transaction is initiated by which icon you drag, and where you drag it; Table 6-1 summarizes these actions. For more information about interworkspace transactions, see “Copying Files between Workspaces” on page 83.

*Table 6-1* Workspace Drag and Drop Action

<b>Drag:</b>	<b>To:</b>	<b>Action</b>
Any workspace icon	Open area	Activate Bringover Create transaction window
Parent workspace icon	Child workspace icon	Activate Bringover Update transaction window
Child workspace icon	Parent workspace icon	Activate Putback transaction window
Any workspace icon	A nonrelated (not a parent or child) workspace icon	Activate pop-up notice to determine actions

- **Reparenting**—To use the drag and drop facility to change a workspace’s parent, press and hold the SHIFT key while you select and drag the workspace icon on top of its new parent’s icon. If you drag the icon to an open area of the Workspace Graph pane, the workspace will be orphaned (have no parent). The display is automatically adjusted to reflect the new relationship. For more information about reparenting workspaces, see “Reparenting a Workspace” on page 66.

---

**Note** – You are prompted to confirm the reparent operation.

---

### ***Double-Click Action***

When you double-click the SELECT mouse button when the pointer is over a workspace icon, the TeamWare utility Versioning is automatically started (with the selected workspace automatically loaded). See Versioning online Help and the section in this manual on Versioning for instructions on using Versioning.

If you double-click SELECT when the pointer is over the icon of a workspace that contains unresolved conflicts, Configuring automatically activates the Resolve transaction window. Conflicted files from the selected workspace are automatically loaded and ready for processing.

You can customize Configuring double-click behavior using the Configuring pop-up window under the Properties button.

### ***Configuring Window Control Area***

See online Help for detailed descriptions of Configuring Windows.

## ***Customizing the Configuring Program Using Properties***

Using the Tool Properties window, you can customize the behavior of:

- Configuring window functions
- Bringover/Putback transactions
- Resolve transaction

You activate the Tool Properties window by choosing the Configuring item from the Options menu. The Category menu on the Properties window enables you to switch between the Configuring, Bringover/Putback, and Resolve panes.

### ***Configuring Defaults Files***

When you change Configuring behavior using the Tool Properties window, you can use the Set Default button to preserve the changes in defaults files in your home directory. The defaults files are consulted by Configuring when it is started, your changes are used as the default values.

Changes made in the Resolve pane of the Tool Properties window are written to the file `~/ .codemgr_resrc`. This file is a standard SunOS runtime configuration file.

Changes made in the Configuring and Bringover/Putback panes of the Tool Properties window are written to the file `~/.codemgrtoolrc`. This file is an OpenWindows XDefaults format file.

### *Configuring Pane*

The Configuring pane of the Tool Properties window enables you to change the behavior of the Configuring base window. The specific properties are described in Table 6-2.

*Table 6-2* Configuring Tool Properties

<b>Property</b>	<b>Description</b>
Default Directory	Directory to which Configuring actions are relative.
Double-click Action	Specify the commands you want launched when you double-click SELECT on: standard workspace icons, icons of workspaces that contain conflicts. Specify the path names required to execute the commands based on the current working directory and your search path. By default, the standard workspace command is Versioning ( <code>vertool</code> ); by default, the Resolve Transaction window ( <code>&lt;resolve_pane&gt;</code> ) is activated for conflicted workspaces.
Load Workspaces	Select this check box if you want the parent and children of workspaces you load in the Workspace Graph pane automatically loaded with them. By default this box is not checked.
Orientation	Choose the Horizontal setting if you want the workspace hierarchy displayed horizontally from left to right in the Workspace Graph pane. Choose the Vertical setting if you want workspace hierarchy displayed vertically from top to bottom. By default the Vertical setting is in effect. This property corresponds to the Orientation item on the View menu in the main Configuring window.
Workspace Names	Choose the Short setting if you want workspaces labelled with the shortest possible name in the Workspace Graph pane. Choose Full if you want workspaces labelled with absolute path names. By default the Full setting is in effect. This property corresponds to the Names item on the View menu in the main Configuring window.

### *Bringover/Putback Pane*

The Bringover/Putback pane of the Tool Properties window enables you to change the behavior of the Bringover and Putback panes of the Transactions window. The specific properties are described in Table 6-3.

*Table 6-3* Bringover/Putback Tool Properties

<b>Property</b>	<b>Description</b>
Auto Load	Causes Configuring to reread the Codemgr_wsdata/args file and load it into the File List pane whenever a new workspace is selected. You might choose to deselect this property when you want to use the same file list for a number of transactions involving different workspaces.
Auto Display	Automatically displays the Transaction Output window during transaction execution.
Auto Bringover Update	If a Putback transaction is blocked, automatically initiates a Bringover transaction to update the child workspace.

### *Resolve Pane*

The Resolve pane of the Tool Properties window enables you to change the behavior of the Resolve pane of the Transaction window. The specific properties are described in Table 6-4.

*Table 6-4* Resolve Tool Properties

<b>Property</b>	<b>Description</b>
Start Merging (auto load)	Causes Merging(FileMerge) to start automatically when the Resolve transaction pane is chosen.
Auto Advance	Causes the next file in the list to be automatically loaded into Merging after the current file is resolved.

Table 6-4 Resolve Tool Properties (Continued)

Property	Description
Prompt for Checkin Comments	A default comment is automatically supplied during checkin after you resolve a file. This property causes you to be prompted for an additional comment that is appended to the standard comment.
Use Existing Merging	If this property is set, an already running Merging process is reused during subsequent resolve operations.
Auto Save (when no unresolved diffs)	If this property is set, <i>and</i> all the changes in the file can be “automerged,” the files will also be saved and checked in; you need not select the Merging Save button.

## Accelerators

Table 6-5 summarizes the various accelerators available for Configuring functions.

Table 6-5 Summary of Configuring Accelerators

Accelerator	Action	Where to Find More Information
Drag and drop workspace icon	Activate Bringover/Putback transaction window	“Dragging and Dropping Workspace Icons”
SHIFT + drag and drop workspace icon	Reparent workspace	“Dragging and Dropping Workspace Icons”
Click SELECT on workspace icon name field	Rename workspace	“Workspace Name Fields”
Double-click SELECT on workspace icon	Launch a tool. User configurable, Versioning is the default	“Double-Click Action”
Double-click SELECT on an icon of a workspace that contains conflicts	Launch a tool. User configurable, Resolve window is the default	“Double-Click Action”



## TeamWare Configuring Workspace

---



As discussed in Chapter 3, “Introduction to TeamWare Configuring“, the *workspace* forms the basis of the Configuring system. The workspace provides isolation in which you (a developer) work in parallel with other developers programming in other workspaces. For an introduction to the Configuring workspace, refer to “Workspace” on page 20 of this manual.

This chapter discusses specific aspects of workspaces and the Configuring commands you use to configure, create, manipulate, and administer them, and contains the following sections:

<i>The Workspace Metadata Directory</i>	<i>page 62</i>
<i>Creating a Workspace</i>	<i>page 64</i>
<i>Deleting a Workspace</i>	<i>page 64</i>
<i>Moving and Renaming a Workspace</i>	<i>page 65</i>
<i>Reparenting a Workspace</i>	<i>page 66</i>
<i>Controlling Access to Workspaces</i>	<i>page 71</i>
<i>How to Notify Users of Changes to Workspaces</i>	<i>page 75</i>
<i>Viewing Workspace Command History</i>	<i>page 78</i>
<i>Ensuring Consistency through Workspace Locking</i>	<i>page 80</i>
<i>Configuring Environment Variables</i>	<i>page 81</i>

## The Workspace Metadata Directory

A Configuring workspace is a directory hierarchy that contains a directory named `Codemgr_wsdata` in its root directory. the Configuring program stores data (metadata) about that workspace in `Codemgr_wsdata`. Configuring commands use the presence or absence of this directory to determine whether a directory is a workspace.

All data stored in the `Codemgr_wsdata` directory is contained in ASCII text files that can be edited by users. Table 7-1 briefly describes each of the files and directories contained in the metadata directory. Information regarding the format of these files is available in the `man(4)` page for each file.

*Table 7-1* Contents of the `Codemgr_wsdata` Metadata Directory

File/Dir Name	Description
<code>access_control</code>	The <code>access_control</code> file contains information that controls which users are allowed to execute Configuring transactions and commands for a given a workspace. When workspaces are created, a default access control file is also created. See Section , “Controlling Access to Workspaces,” on page 71.
<code>args</code>	The <code>args</code> file is maintained by the Configuring Bringover and Putback transaction commands and contains a list of file, directory, and FLP arguments. Initially, the <code>args</code> file contains the arguments specified when the workspace was created. If you explicitly specify arguments during subsequent Bringover or Putback transactions, the commands determine if the new arguments are more encompassing than the arguments already in the <code>args</code> file; if they are, the new arguments replace the old.
<code>backup/</code>	The <code>backup</code> directory is used to store information that Configuring uses to “undo” a Bringover or Putback transaction. See Section , “Reversing Bringover and Putback Transactions with Undo,” on page 105.
<code>children</code>	The <code>children</code> file contains a list of the workspace’s child workspaces. The names of child workspaces are entered into the workspace’s <code>children</code> file during the Bringover Create transaction. Configuring consults this file to obtain the list of child workspaces. When you delete, move, or reparent a workspace, Configuring updates the <code>children</code> file in its parent.

Table 7-1 Contents of the Codemgr\_wsdata Metadata Directory (Continued)

File/Dir Name	Description
conflicts	The <code>conflicts</code> file contains a list of files in that workspace that are currently in conflict. See Chapter 9, “Resolving Conflicts,” for more information about conflicts and how to resolve them.
history	The <code>history</code> file is a historical log of transactions and updated files that affect a workspace. See “Viewing Workspace Command History” on page 78 for more information.
locks	To assure consistency, Configuring locks workspaces during Bringover, Putback and Undo transactions. Locks are recorded in the <code>locks</code> file in each workspace; Configuring consults that file before acting in a workspace. See “Ensuring Consistency through Workspace Locking” on page 80.
nametable	The <code>nametable</code> file contains a table of SCCS file names (path names relative to that workspace) and a unique number represented as four 32-bit hexadecimal words. Each entry in the table is terminated by a newline character. The <code>nametable</code> file is used by Configuring during Bringover and Putback to accelerate the processing of files that have been renamed. If this file is not available, Configuring rebuilds it automatically during the next Putback or Bringover transaction. See “Renaming, Moving, or Deleting Files” on page 108.
notification	The <code>notification</code> file is edited by users to register notification requests. This facility permits Configuring to detect events that involve that workspace and to send electronic mail messages in response to the event. See Section , “How to Notify Users of Changes to Workspaces,” on page 75.
parent	The <code>parent</code> file contains the path name of the workspace’s parent workspace and is created by the Bringover Create transaction, or by the Reparent command if the workspace was originally created with the Create Workspace command (and thus had no parent). Configuring consults this file to determine a workspace’s parent. When you delete, move, or reparent a workspace, Configuring updates the <code>parent</code> file in its children.
putback.cmt	The <code>putback.cmt</code> file is a cache of the text of the comment from the last <i>blocked</i> Putback transaction. When a Putback transaction is blocked, the comment is discarded. Configuring caches the comment in <code>putback.cmt</code> so that you can retrieve the original text when you reexecute the transaction.

## *Creating a Workspace*

You can create a workspace in one of two ways:

- *Explicitly* by means of the Create Workspace item in the File menu on the main Configuring window
- *Implicitly* by using the Bringover Create transaction to copy files into a nonexistent child workspace, in which case the child workspace is created and then populated with the files specified as part of the transaction

### *Using Workspace Create*

The Workspace Create item in the Configuring File menu is used to create new workspaces. Type the name of the new workspace's root (top-level) directory in the Workspace Directory text field and click on the Create Workspace button.

If the workspace you are creating already exists as a directory hierarchy, Configuring converts it to a workspace by simply adding the `Codemgr_wsdata` directory in the root directory and displaying its icon in the Workspace Graph Pane.

If the directory does not already exist, Configuring creates both the root directory and the `Codemgr_wsdata` directory.

### *Using the Bringover Create Transaction*

Use the Bringover Create transaction (on the Transactions menu) to copy files from a parent workspace to a nonexistent child workspace; the child is automatically created as part of the transaction. See "Creating a New Child Workspace (Bringover Create)" on page 90 for details.

The online Help provides further assistance in Creating New Workspaces. To access the help, choose Help ► Help Contents ► Starting a Project.

## *Deleting a Workspace*

You delete workspaces by selecting their icons in the Workspace Graph Pane and then invoking the Delete ⇒ item from the Configuring Edit menu.

Two menu items are provided that delete workspaces:

- Sources and Codemgr\_wsdata Directory  
Recursively deletes the contents of the workspaces.
- Codemgr\_wsdata Directory only  
Changes their status to nonworkspace directories by deleting only the Codemgr\_wsdata directory and removing their icons from the Workspace Graph Pane.

In either of these cases Configuring automatically updates records in parent and child workspaces to reflect the deletion of the workspace.

When you choose the Sources and Codemgr\_wsdata Directory command, you are prompted to confirm your decision.

## *Moving and Renaming a Workspace*

Since workspaces are directories, you move them by changing their path names. There are two ways that you can move/rename a workspace:

- By editing its name in the Workspace Graph pane  
Select the name field by moving the pointer over a portion of the text and click SELECT. This selects the text for editing. Use standard text editing to change the name; type Return to enter your changes. Click SELECT in an empty portion of the pane to deselect the text.
- By using the Rename command item from the Configuring Edit menu  
The path name of the selected workspace is changed to the name that you type in the New Workspace Name text field.

In addition to changing the workspace path name, both methods also update the appropriate data files in the parent and child workspaces to contain the new name. These data files are discussed in “The Workspace Metadata Directory” on page 62.

## *A Note About Moving Workspaces*

---

**Caution** – Do not use the SunOS `mv` command to rename or move workspaces.

---

The Configuring Rename command updates files in the workspace's parent and children, as well as logging the event in the `Codemgr_wsdata/history` file.

If you inadvertently use the `mv` command to move/rename a workspace and discover that it has become "disconnected" from its parent and children, you can use the Rename command to reconnect it.

For example, if you used the `mv` command to rename a workspace from A to B:

**1. Use the Rename command to rename B to C.**

This causes Configuring to update the workspace's new name (C) in the parent and child workspaces. To save time, be sure to use a path name on the same device.

**2. Use the Rename command to change C back to B.**

Everything should be reconnected.

## *Reparenting a Workspace*

As discussed in Chapter 3, "Introduction to TeamWare Configuring," "Parent and Child Relationship" on page 22, the parent/child relationship is the thread that connects the workspace hierarchy. Configuring provides the means for you to change this relationship at your discretion.

This section discusses how you can explicitly change a workspace's parent. It is also possible for you to implicitly change a workspace parent "on the fly" (for the duration of a single command) by specifying the new parent's path name as part of a Bringover Update or Putback transaction. See the descriptions of the Bringover Update and Putback transactions in "Copying Files between Workspaces" on page 83 for more information.

The following sections describe:

- Two methods that you can use to change a workspace's parent
- Some reasons why you might want to change a workspace's parent
- An example of using the rename feature

## *Two Ways to Reparent Workspaces*

This section describes two completely equivalent ways to reparent workspaces.

### *Drag and Drop Workspace Icons*

You can change a workspace's parent by selecting its icon in the Workspace Graph Pane, pressing and holding the SHIFT key, and dragging it on top of its new parent's icon. The display is automatically adjusted to reflect the new relationship.

---

**Note** – You are prompted to confirm the change.

---

You may also “orphan” a workspace by selecting its icon, pressing SHIFT, and dragging it to an open area on the Workspace Graph. The workspace no longer has a parent: the display is automatically adjusted to reflect its new status.

### *The Parent Command*

You can change a workspace's parent by selecting its icon in the Workspace Graph Pane and then choosing the Parent command item from the Edit menu. This activates the Parent pop-up window.

When the window is initially activated, the New Parent Workspace Directory text field contains the name of the current parent; edit that line so that it contains the name of the new parent file. Click SELECT on the Parent button. The Workspace Graph Pane is automatically adjusted to reflect the new relationship.

If you do not specify a parent workspace in the New Parent Workspace Directory text field, the workspace is orphaned—it has no parent. The Workspace Graph Pane is automatically adjusted to reflect its new status.

## *Reasons to Change a Workspace's Parent*

Reasons why you might want to permanently or temporarily change a workspace parent are as follows:

- To populate a new project hierarchy (new top-level workspace)

You may be completing Release 1 of your product and see the need to begin work on Release 2. In this case you might:

- Create a new (empty) Release 2 workspace by means of the Create Workspace command item.

- Use either of the two methods described above to make the Release 2 workspace the new parent of the Release 1 workspace.
- Use the Putback transaction to copy files to the Release 2 workspace.
- Reparent the Release 1 workspace to its original parent.

- To move a feature into a new release

If a feature intended for a particular release is not completed in time, the workspace in which the feature was being developed can be reparented to the following release's integration workspace. A similar use of reparenting is described in the example in the next section.

- To apply a bug fix to multiple releases.

The workspace in which work was done to correct a bug is reparented from hierarchy to hierarchy; the Configuring Putback transaction is used to incorporate the changes into the new parent. An example of this use of reparenting is included in the next section.

- To reorganize workspace hierarchies
  - You can add additional levels to the hierarchy.
  - You can remove levels from the hierarchy (do not specify a new parent during reparenting).
  - You can reorganize workspace branches within the project hierarchy.
- To adopt an orphan workspace if its `Codemgr_wsdata/parent` file is deleted

If, for some reason a file is orphaned (for example, its parent is corrupted or its own `Codemgr_wsdata/parent` file deleted) you can use the reparenting feature to restore its parentage.

### *A Reparenting Example*

Often a bug is fixed in a version of a product and a patch release is made to distribute the fixed code. The code that was fixed must usually be incorporated into the next release of the product as well. If the product is developed using Configuring, the patch can be incorporated relatively simply by means of reparenting.

In the following example, a patch is developed to fix a bug in Release 1.0 of a product. The patch must be incorporated into Release 2.0, which has begun development.



1. The workspace in which the patch was developed (or the workspace from which it is released) is cloned by means of the Bringover Create transaction. The reason the workspace is cloned is that it will be altered by its interaction with its new parent (Bringover transaction to synchronize it with its new parent).
2. Either of the two reparenting methods are used to change the cloned workspace's parent from 1.0patch to 2.0. (Figure 7-1)



Figure 7-1 Patch Workspace Reparented to New Release

3. The workspace is then updated from its new parent, and any new work is brought over from 2.0. (Figure 7-2A)
4. The fixes made for the patch are merged in patch with the files from 2.0 and are put back into the 2.0 workspace where they are now available to workspace 2.0child. (Figure 7-2B)

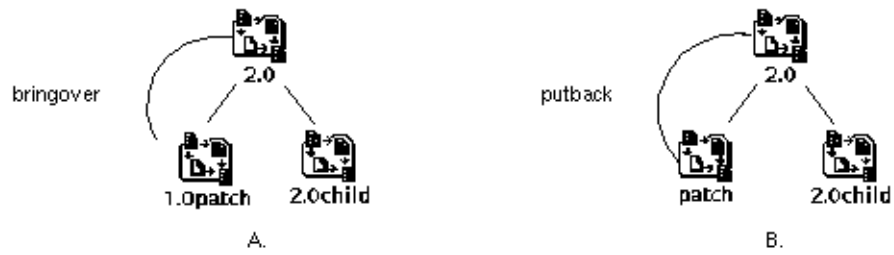


Figure 7-2 Files Brought Over, Merged, and Incorporated into the New Release

- Files are brought over to 2.0child, and patch is deleted by means of the Delete ⇒ Sources and Metadata item from the Edit menu. (Figure 7-3)



Figure 7-3 Patched Files Brought Over into 2.0child; patch Deleted

For more detailed examples of reparenting, refer to *Sun WorkShop TeamWare: Solutions Guide*, in particular, workspace hierarchy strategies.

## Controlling Access to Workspaces

Configuring permits you to control the access that users have to your workspaces. Table 7-2 lists and describes the eight types of access over which you can exercise control.

Table 7-2 Operations Over Which You Have Access Control

Type of Access	Description
bringover-from	Controls which users may bring over files <i>from</i> this workspace
bringover-to	Controls which users may bring over files <i>to</i> this workspace
putback-from	Controls which users may put back files <i>from</i> this workspace
putback-to	Controls which users may put back files <i>to</i> this workspace
undo	Controls which users may “undo” commands executed in this workspace
workspace-delete	Controls which users may delete this workspace
workspace-move	Controls which users may move this workspace
workspace-reparent	Controls which users may reparent this workspace
workspace-reparent-to	Controls which users may reparent other workspaces to this workspace

Prior to taking any of the actions listed above, the Configuring program consults a file in the `Codemgr_wsdata` directory named `access_control` to determine whether the user taking the action has access permission to the workspace for that purpose. The `access_control` file is a text file that contains a list of the eight operations and corresponding values that stipulate who is permitted to perform those operations. The `access_control` file is automatically created at the time the workspace is created and is owned by the creator of the workspace.

To view and change access permissions, use the Options menu. Choose the Workspace item, then use the Category button to choose the Access Control pane of the Workspace Properties pop-up window (see “Viewing and Changing Access Control Values” on page 74).

Table 7-3 shows the default contents of `access_control` after you create a workspace:

*Table 7-3* Default Access Control Permissions

<b>Operation</b>	<b>Permissions</b>
bringover-from	
bringover-to	creator
putback-from	
putback-to	
undo	
workspace-delete	creator
workspace-move	creator
workspace-reparent	creator
workspace-reparent-to	

---

**Note** – Creator permission indicates that Creator’s login name appears.

---

You can express which users have or do not have access to a workspace in a number of ways. Table 7-4 shows all of the value types you can specify to control access to your workspaces and what the entries mean.

Table 7-4 Workspace Access Control Values

Value	Meaning
@engineering	All users in the net group named <code>engineering</code> can execute this operation
~@engineering	No users from the net group named <code>engineering</code> can execute this operation. Note that “~” denotes negation.
@special ~user2 @engineering	All users in the net groups <code>special</code> and <code>engineering</code> can execute the operation; <code>user2</code> cannot (unless <code>user2</code> is in the <code>special</code> netgroup). “~” denotes negation.
user1 user2	The users <code>user1</code> and <code>user2</code> can execute the operation.
“~”	No user can execute the operation.
creator	Only the user who created the workspace can execute the operation. Note that the creator’s login name actually appears.
(no entry)	Any user may execute the operation.

**Note** – If a user is listed as having both access permission *and* restriction, the first reference is used.

**Note** – Performance may degrade when net groups are included in the access control file. The time required to look up group membership can add several seconds to the execution of a given operation.

## *Viewing and Changing Access Control Values*

- ▼ To view the access control status of a workspace, do the following:
  1. Select a workspace icon in the base window Workspace Graph pane.
  2. Choose the Workspace item from the Options button menu.
  
- ▼ To change the access control status of a workspace, do the following:
  1. Select a workspace icon in the base window Workspace Graph pane.
  2. Choose the Workspace item from the Options button menu.
  3. Use the **SELECT** mouse button to select an access line in the global Access Control list, then select the **Edit** button to activate the Access Control Edit pop-up.

The operation you selected before clicking on the Edit button is automatically selected for you.
  4. Optionally, use the **Operation** menu in the Access Control Edit pop-up to select an operation type.
  5. Choose the type of permission you wish to allow:
    - None: No users have permission
    - All: All users have permission
    - Specify: Use the Permissions list to construct a list of users and netgroups that are to be granted or denied permission
  6. If you choose to specify individual and/or group permissions, construct your entry using:
    - The Name text field to enter the name of the user or netgroup
    - The Type setting to specify whether the entry is a user or a netgroup
    - The Access setting to specify whether the specified user/netgroup is granted or denied permission
  7. Select the **Insert** button to enter your entry into the Permissions list.
  8. Select the **Apply** button to enter your selection into the global Access Control list.

9. In the Workspace Properties pop-up, select the Set Default button to write the changes to the `access_control` file.

## *How to Notify Users of Changes to Workspaces*

You can request Configuring to notify you (through an electronic mail message) when a variety of Configuring events occur in a workspace. Notification requests are entered in the file named `notification` in the `Codemgr_wsdata` directory.

A notification request consists of the following items:

- An address to which mail is sent.
- The event for which you want notification triggered.
- An optional list of directories and files whose changes of status trigger notification. The list is bracketed by BEGIN/END statements.

The following is an example of a `notification` file that contains three requests:

```
chip@mach1 bringover-to
BEGIN
dir1/foo.cc
dir2
END
biff@mach2 bringover-to putback-to
BEGIN
.
END
biff@mach2 workspace-move
```

In the first entry, the user `chip@mach1` requests to be notified when the file `dir1/foo.cc` and *any* file in the directory `dir2` (path names are relative to the workspace root directory) are brought over to the workspace.

---

**Note** – File and directory entries for each event are bracketed by BEGIN/END statements. An empty list, a missing list, or a list that consists of only the “.” character indicate that all files and directories in the workspace are registered for notification.

---

In the second entry, user `biff@mach2` requests to be notified when any file in the workspace is brought over to, or put back to, the workspace. The “.” character represents all files in the workspace.

In the third entry, `biff@mach2` requests to be notified if the workspace is moved. Events that involve entire workspaces (delete, move, reparent) do not accept directory/file lists.

Table 7-5 lists the events for which you can register notification requests:

Table 7-5 Notification Events

Event Name	Description
bringover-from	Send mail whenever files are brought over <i>from</i> the workspace in which the notification file is located.
bringover-to	Send mail whenever files are brought over <i>to</i> the workspace in which the notification file is located.
putback-from	Send mail whenever files are put back <i>from</i> the workspace in which the notification file is located.
putback-to	Send mail whenever files are put back <i>to</i> the workspace in which the notification file is located.
undo	Send mail whenever a transaction is “undone” in the workspace in which the notification file is located.
workspace-delete	Send mail if the workspace in which the notification file is located is deleted.
workspace-move	Send mail if the workspace in which the notification file is located is moved.
workspace-reparent	Send mail if the workspace in which the notification file is reparented.
workspace-reparent-to	Send mail if the workspace becomes the new parent of an existing workspace.

### Viewing and Changing Notification Entries

To view and change notification entries, select a workspace icon in the Workspace Graph pane and choose the Workspace item from the base window Options menu. Use the Category menu to choose the Notification pane. The requests contained in the `notification` file described on the previous page will be similar to those displayed in the Workspace Properties pop-up.



Use the items in the Edit menu to modify, create and delete notification entries. Choosing the Entry and Create menu items activates the Notification Edit pop-up.

To create a new request, choose (with no items selected) the Create item in the Edit menu. The Notifications Edit pop-up is activated—use this window to specify the following request information:

- The mail address to which notification mail is sent
- The event about which notification mail is sent
- The files the notification event applies to:
  - Any file in the workspace (All)
  - Specific directories and/or files (Specify). If you choose to specify files/directories, create the list of directories and files in the Files text pane. To create the list, activate the Add Files chooser by clicking on the Add files to List button. Delete files from the list using the Delete button.

Modify an existing entry, by selecting it in the Notification pane of the Workspace Properties pop-up and choosing the Entry item from the Edit menu. Use the Notification Edit pop-up to modify the entry.

Apply changes to the Notification Edit pop-up changes to the global Notification list, by clicking on the Apply button.

Apply changes to the notifications file, by clicking on the Set Default button in the Workspace Properties pop-up.

### *Notes About Registering Notification Events*

- The following events involve entire workspaces and thus do not require a directory/file list:
  - workspace-delete
  - workspace-move
  - workspace-reparent
  - workspace-reparent-to
- When a directory is specified in the list, all files hierarchically beneath it are automatically registered.
- The mail address can be any valid mail address, including aliases.

## Viewing Workspace Command History

Configuring commands are logged in the text file `Codemgr_wsdata/history`. Commands that affect a single workspace are logged only in that workspace; interworkspace transactions are logged in both the source and destination workspaces.

---

**Note** – Although command entries are logged in both the source and destination workspaces, the list of changed files is entered only in the destination directory.

---

You can view the contents of this file to track or reconstruct changes that have been made to a workspace over time. Log entries consist of the underlying command-line entries and do not correspond to GUI menu item names. If you have any questions about the meaning or syntax of a command, refer to its `man` page for details. Table 7-6 lists the GUI operations and the corresponding CLI command that is entered in the history log.

*Table 7-6* Corresponding GUI and CLI Commands

---

GUI Menu Item	Corresponding CLI Command
Create Workspace	<code>workspace create</code>
Rename	<code>workspace move</code>
Parent	<code>workspace parent</code>
Bringover Create	<code>bringover</code>
Bringover Update	<code>bringover</code>
Putback	<code>putback</code>
Undo	<code>ws_undo</code>
Resolve	<code>resolve</code>

---

---

**Note** – In active workspaces, the `Codemgr_wsdata/history` file can grow very quickly. You may want to periodically prune the file to reduce its size.

---

The following portion of a history file was generated during a Bringover Update transaction; entries are described in Table 7-7. This entry is taken from the history file in the child; the corresponding entry in the parent is identical except that file status messages are not included.

```
COMMAND bringover -w /home/sponge3/larryh/ws/man_pages -p
/home/sponge3/larryh/ws/manpages man trans/man
update: man/Makefile
update: man/man5/access_control.5
create: man/man5/notification.5
create: man/man1/codemgr.1
rename from: man/man1/def.dir.flg.1
to: man/man1/def.dir.flp.1
update: man/man1/def.dir.flp.1
create: man/man1/codemgrtool.1
rename from: man/man1/fileresolve.1
to: deleted_files/man/man1/fileresolve.1
update children's name history:
deleted_files/man/man1/fileresolve.1
rename from: man/man1/resolve_tty.1
to: deleted_files/man/man1/resolve_tty.1
update: deleted_files/man/man1/resolve_tty.1
create: trans/man/man1/codemgr_acquire.1
create: trans/man/man1/codemgr_prepare.1
CWD /tmp_mnt/home/sponge3/larryh/temp
RELEASE Beta 1.0
HOST croak
USER larryh
PARENT_WORKSPACE (/home/sponge3/larryh/ws/manpages)
(sponge:/export/home/sponge3/larryh/ws/manpages)
CHILD_WORKSPACE (/home/sponge3/larryh/ws/man_pages)
(sponge:/export/home/sponge3/larryh/ws/man_pages)
START (Mon Jul 13 13:31:16 1992 PDT) (Mon Jul 13 20:31:16 1992 GMT)
END (Mon Jul 13 13:32:08 1992 PDT) (Mon Jul 13 20:32:08 1992 GMT)
STATUS 0
```

Table 7-7 History File Entry Descriptions

Entry	Description
COMMAND	Underlying command line issued for the operation. File status messages as displayed in the Transaction Output window are included only in the destination workspace history file.
CWD	Name of the current working directory when the command was executed.
RELEASE	Release number of the TeamWare software
HOST	Name of the system from which the command was executed.
USER	Login name of the user who executed the command.
PARENT_WORKSPACE	The path name of the parent workspace specified in two formats: host-specific and <i>machine:pathname</i> .
CHILD_WORKSPACE	The path name of the child workspace specified in two formats: host-specific and <i>machine:pathname</i> .
START	Time the command started execution, both locally and as measured by Greenwich Mean Time (GMT).
END	Time the command completed execution, both locally and as measured by Greenwich Mean Time (GMT).
STATUS	Exit status of the command: 0 = Normal completion, any other value indicates an error condition, warning, or other status.

### Ensuring Consistency through Workspace Locking

To assure consistency, the Configuring transactions—Bringover, Undo, and Putback—lock workspaces while they are working in them. These locks only affect Configuring transactions; other commands such as SCCS programs, are

---

not affected. Locks are recorded in the `Codemgr_wsdata/locks` file in each workspace; the Configuring transaction commands consult that file before acting in a workspace. Two types of locks are used:

- A *read-lock* is used when a command must assure that a workspace does not change while it is examining its contents.

Read-locks may be obtained concurrently by a number of commands; no Configuring command can write to the workspace while a read-lock is in force. A read-lock is obtained during a Bringover transaction in the parent when its files are examined in preparation for copying to the child, and during a Putback transaction in the child when its files are examined in preparation for copying to the parent.

- A *write-lock* is used when a command must assure that a workspace does not change while it is writing to it.

Only one write-lock may be obtained for a workspace at any time. When a write-lock is in force, only the Configuring command that owns the lock can write to the workspace; other commands cannot obtain read-locks from the workspace. A write-lock is obtained during a Bringover transaction for the child when files are copied into it, and during a Putback transaction for the parent when files are copied into it.

If a Configuring command is unable to remove its lock after completion (for example, the system crashes), you must remove the lock yourself before Configuring commands will again be able to read and/or write in the workspace. You can use the Configuring GUI to view and delete active locks for a workspace, or you can edit the file directly.

To view and delete locks using the Configuring GUI, select a workspace icon from the Workspace Graph pane and choose the Workspace item from the main Props menu. Use the Category menu to choose the Locks pane.

To delete locks, select the line that contains the lock and click on the Delete button. To apply the deletion to the `locks` file, click on the Set Default button.

## *Configuring Environment Variables*

Configuring consults environment variables to direct some of its actions.

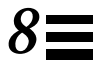
### *The CODEMGR\_WS Variable*

If you do not explicitly specify a workspace as the focus of a Configuring command, many of the commands will consult the shell environment variable `CODEMGR_WS` to determine a default workspace as the focus of their action. If you have a workspace that is the primary focus of your work, use of the variable will allow you to execute the commands without specifying the workspace argument.

### *The CODEMGR\_WSPATH Variable*

When it is started, Configuring automatically loads workspaces from directory path names specified in the `CODEMGR_WSPATH` variable.

## Copying Files between Workspaces



Chapter 3, “Introduction to TeamWare Configuring”, describes copying files up and down the parent/child hierarchy. This chapter describes how you use Configuring to copy files.

The chapter covers the following topics:

<i>Configuring Transaction Model</i>	<i>page 83</i>
<i>General File Copying Information</i>	<i>page 84</i>
<i>Copying Files from a Parent to a Child Workspace (Bringover)</i>	<i>page 89</i>
<i>Copying Files from a Child to a Parent Workspace (Putback)</i>	<i>page 99</i>
<i>Reversing Bringover and Putback Transactions with Undo</i>	<i>page 105</i>
<i>Renaming, Moving, or Deleting Files</i>	<i>page 108</i>

An example demonstrating these transactions can be found in Chapter 12, “Configuring Example” on page 143.

### *Configuring Transaction Model*

Configuring is designed so that all interworkspace transactions (Bringover Create, Bringover Update, Putback, Undo, and Resolve) are based upon the same user model; that model is described in Figure 8-1. The ways in which the transactions differ are described later in this chapter (the Resolve transaction is described in Chapter 9, “Resolving Conflicts”).

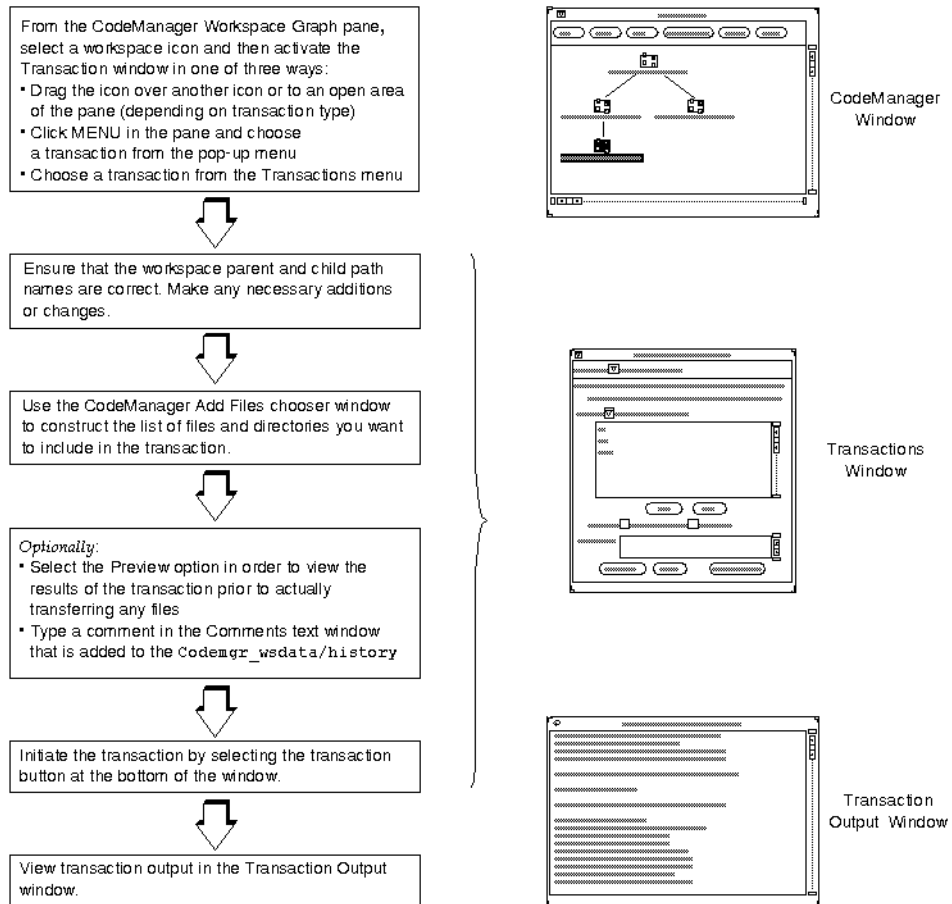


Figure 8-1 Configuring Transaction Model

## General File Copying Information

This section contains background information about copying files between workspaces.



---

## *SCCS History Files*

When considering Configuring file transfer transactions, it is important to remember that source files are actually derived from SCCS deltas and are identified by SCCS delta IDs (SIDs). When a file is said to be copied by either a Putback or Bringover transaction, Configuring actually acts upon (copies or merges) the file's SCCS history file (also known as the "s-dot-file").

The means by which Configuring manipulates and merges the history files is described in detail in Chapter 11, "How the Configuring Program Merges SCCS Files." For specific technical information, refer to `sccsfile(4)`.

## *Viewing Transaction Output*

Output from Configuring transaction commands is viewed in the Transaction Output window. This window is activated automatically when you invoke one of the transactions. You can also activate it yourself by choosing the Show Output button in any of the Transactions window layouts. Check the online Help for details on TeamWare windows.

---

**Note** – Configuring transactions are implemented through command-line based programs; some portion of the output contains messages related to the command-line implementation. This manual describes only messages that apply to the actual transactions. If you are interested in more information about the underlying command-line based programs, please refer to the appropriate `man` pages.

---

## *Specifying Directories and Files for Transactions*

When you copy files between parent and child workspaces using the Bringover and Putback transactions, you must specify the directories and files you wish included in the transaction. The Bringover Create, Bringover Update, and Putback layouts of the Transactions window contain a File List pane. The File List pane is a scrolling text window in which you construct the list of file and directory names to be included in the transaction. You can accept the default "." convention to bringover or putback all files in a workspace.

### *Grouping Files for Transfer Using File List Programs*

In addition to explicitly specifying files for transfer, you can execute programs that generate that list for you — such a program is called a *File List Program* (or *FLP*). An FLP generates a list of files to `stdout`; the Bringover and Putback transactions read the list of files from `stdout` and include them in the transaction.

Configuring is shipped with a default FLP named `def.dir.flp`. The FLP `def.dir.flp` recursively lists the names of files that are under SCCS control in *directories* that you specify in the File List pane (see next section). The files generated by this (or any) FLP are included for transfer with *files* that you also specify in the File List pane.

If you want to use your own FLPs during a transaction, you can specify their path names in the File List pane. The File List pane is used for both specifying file/directory lists and for specifying FLPs. Use the abbreviated menu immediately above the pane to change between the two modes. Add FLPs to the list using the point-and-click chooser window that is activated by choosing the Add FLPs to List item in the File menu (located below the File List pane). See “Add Files Chooser” on page 88 for more information.

---

**Note** – You can create your own FLPs that generate lists of files that are useful for your project.

---

### *Constructing Directory and File Lists in the File List Pane*

Configuring attempts to provide you with a useful initial list of directories and files in the File List pane. You are free to modify the list in any way you wish. The initial list is constructed differently for each type of transaction:

---

<b>Bringover Create</b>	The initial list is empty.
<b>Bringover Update</b>	The initial list is retrieved from the <code>Codemgr_wsdata/args</code> file in the child workspace. This file contains a list of arguments specified during previous Bringover and Putback transactions.
<b>Putback</b>	The initial list is retrieved from the <code>Codemgr_wsdata/args</code> file in the child workspace. This file contains a list of arguments specified during previous Bringover and Putback transactions.

---

---

Every workspace contains a `Codemgr_wsdata/args` file that is maintained by the Configuring Bringover and Putback transaction commands. The `args` file contains a list of file, directory, and FLP arguments. Initially, the `args` file contains the arguments specified when the workspace was created. If you explicitly specify arguments during subsequent Bringover or Putback transactions, Configuring determines if the new arguments are more encompassing than the arguments already in the `args` file; if the new arguments are of a wider scope, the new arguments replace the old.

---

**Note** – You can edit the `args` file at any time to change its contents.

---

### ***Selecting Files in the File List Pane***

Once a list of files and directories exists in the File List pane, you can include or exclude any of them for a given transaction. To be included in a transaction, the file or directory name must be *selected*. You can select or deselect any number of names by moving the pointer over them and clicking SELECT. You can select or deselect the entire list by choosing the Select List or Unselect List items from the Edit menu.

### ***Loading and Saving Default Lists***

You can reload the default list from the workspace `args` file at any time by choosing the “Load List from Defaults” item from the File menu. This feature is useful if you find that you’ve made changes to the list that you do not want to keep; you can use Load List from Defaults to revert the list to its default state.

If you change the default list and wish to make the new list the default in the workspace `args` file, choose the “Save List to Defaults” item from the File menu. This is especially useful if you have eliminated files or directories from the list. If you add files, Configuring automatically adds them to the `args` file for you as part of a Bringover or Putback transaction.

### ***Changing the Contents of the File List Pane***

You *add* files and directories to the File List pane by using the point-and-click, Configuring Chooser. See “Add Files Chooser” on page 88 for details.

You *delete* files and directories from the File List pane using:

- The Clear List and Delete items from the Edit menu
- The Clear All Choices item from the File List pane pop-up menu

---

**Note** – You can specify the “.” directory as the sole item in the file list to designate that the entire workspace be copied to the child. Enter the “.” character using the Name text field in the Configuring Chooser.

---

### ***Add Files Chooser***

You can use the Add Files chooser to conveniently add directories and files to the Transaction window File List pane. The Configuring chooser is a pop-up window that contains a point-and-click chooser pane that you can use to search for and select directories and files. Activate the chooser window using the Add Files to List item in the File menu.

---

**Note** – The Configuring Chooser is also used to add FLPs to the File List pane. The appropriate version of the chooser is automatically invoked when you change the File List pane mode using the abbreviated menu immediately above the pane.

---

Use the chooser to navigate down through the file system hierarchy by double-clicking SELECT on any directory icon. Double-click SELECT on the directory icon to move hierarchically upward in the file system. To move directly to a directory, enter its path name in the Name text field and select the Load Directory button.

---

**Note** – The chooser does not permit you to navigate outside of the workspace file system.

---

To add a file or directory to the File List pane:

---

**1. Select files and directories by moving the pointer over any file or directory icon and clicking SELECT.**

You can extend the selection to include any number of additional files and directories by moving the pointer over them and clicking ADJUST.

You can select entire groups of files by clicking and holding SELECT in an empty portion of the chooser and dragging the bounding box to surround any number of icons. When you release the button, all the files within the bounding box are selected.

You can also add a file to the File List pane by specifying its path name in the Name text field. If you type Return, the entry will be entered immediately; you may also enter it by choosing the Add Files to List button.

**2. Select the Add File to List button to add the file to the File List pane.**

---

**Note** – A check mark in a file icon indicates that the file is checked out from SCCS.

---

## *Copying Files from a Parent to a Child Workspace (Bringover)*

All Configuring file transfer transactions are performed from the perspective of the child workspace; hence Bringover transactions “bring over” groups of files from the parent to the child workspace. There are two types of Bringover transactions:

---

Bringover Create	Copy groups of files from a parent workspace to a nonexistent child workspace; the child is created as a result of the Bringover Create transaction.
Bringover Update	Copy files to an existing workspace; the contents of the child are updated as result of the Bringover Update transaction.

---

**Note** – You can use the Bringover Update and Create transactions to import directories and files from directories that are not Configuring workspaces. You cannot Putback files to directories that are not workspaces.

---

## *Creating a New Child Workspace (Bringover Create)*

You use the Configuring Bringover Create transaction to copy groups of files from a parent workspace to a child workspace that is created as a result of the Bringover transaction. You can display the Bringover Create layout of the Transactions window by any of the following methods:

- Drag and drop a workspace icon onto an empty space in the Workspace Graph pane.
- Select a workspace icon and choose the Bringover >> Create item from the Transactions menu.
- Select a workspace icon and choose the Bringover >> Create item from the Workspace Graph pane pop-up menu.
- Choose the Bringover Create item from the Category menu if the Transactions window is already displayed.

The Bringover Create transaction operates on files that are under SCCS control. When files are said to be copied to the child, the SCCS history file is copied and its g-file (the most recent delta) is created through the SCCS `get` command.

To initiate a Bringover Create transaction, follow these five basic steps:

### **1. Specify the parent workspace.**

If you select a workspace icon on the Workspace Graph pane prior to displaying the Bringover Create window, its path name is automatically inserted in the From Parent Workspace Directory text field. You can edit and change the contents of the text field by hand at any point. You can specify the absolute path name of any accessible workspace; it need not be displayed in the Workspace Graph pane.

---

**Note** – You can also specify the path name of directories that are not workspaces to import directories and files into the new workspace.

---

### **2. Specify the child workspace.**

Type the absolute path name of the child that will be created and populated with files from the parent workspace in the To Child Workspace Directory text field.

---

### 3. Create a list of directory and file names in the File List Pane.

You can copy all or part of the contents of the parent workspace to the child. You specify the directories and files you wish to copy in the File List pane. See “Specifying Directories and Files for Transactions” on page 85 for information about specifying directory and file arguments.

---

**Note** – If you are using your own FLPs to generate file lists, you also specify them in the File List pane. Refer to the *Sun WorkShop TeamWare: Solutions Guide* for examples of how to use CodeManager FLPs.

---

### 4. Select options.

- 
- |   |  |
|---|--|
| <input checked="" type="checkbox"/> Preview | Select this option to preview the results of the transaction. If you invoke the Bringover Create transaction with this option selected, the transaction will proceed without actually transferring any files. You can monitor the output messages in the Transaction Output window (Show Output) as if the transaction were actually proceeding.   |
| <input checked="" type="checkbox"/> Verbose | Select this option to increase the information displayed in the Transaction Output window. By default, a message is displayed for each created, updated, or conflicting file. The Verbose option causes bringover to print a message for all files, including those that are not brought over. If both the Verbose option and the Quiet option are specified, the Quiet option takes precedence. |
-

<input checked="" type="checkbox"/> Quiet	Select this option to suppress the output of status messages to the Transaction Output window (Show Output).
<input checked="" type="checkbox"/> Skip SCCS gets	Select this option to inhibit the automatic invocation of the SCCS <code>get</code> program as part of the Bringover transaction. Normally <code>g</code> -files are extracted after they are brought over. This option improves file transfer performance although it shifts the responsibility to the user to do the appropriate <code>gets</code> at a later time.
<input checked="" type="checkbox"/> Force Conflicts	Select this option to cause all updates to be treated as conflicts. This option is not applicable to the Bringover Create transaction, but is applicable to the Bringover Update transaction.

**5. Select the Bringover button to initiate the transaction.**

*Notes about the Bringover Create Transaction*

- Checked-out files

When, during a Bringover Create transaction, Configuring encounters files that are checked out from SCCS in the parent, it takes action based on preserving the consistency of the files and any changes to the file that might be in-process.

Table 8-1 shows the different actions that Configuring takes when it encounters checked-out files.

*Table 8-1* Effects of Checked-out Files on Bringover Create Transactions

<b>File Checked-out in Parent</b>	<b>Configuring Action</b>
<code>g</code> -file and latest delta differ	<ul style="list-style-type: none"> <li>•Issue a warning</li> <li>•Process file</li> </ul>
<code>g</code> -file and latest delta are identical	<ul style="list-style-type: none"> <li>•Process file</li> </ul>



- As the transaction proceeds, status information is displayed in the Transaction Output window. Messages are displayed as files are processed during the transaction and a transaction summary is displayed when execution is completed.
- If you specify *relative* path names for directory and file names, be aware that they are interpreted as being relative from the top-level (root) directory of the workspace hierarchy (which is assumed to be the same in both parent and child). If you specify these file names using *absolute* path names, the file must be found in one of the two workspaces, or it will be ignored.
- The parent and child workspaces must be accessible through the file system. Either automounter or NFS® mounts can be used.
- Action taken during the Bringover Create transaction can be reversed using the Undo transaction. Refer to Section , “Reversing Bringover and Putback Transactions with Undo,” on page 105 for details.
- While Configuring is reading and examining files in the parent workspace during a Bringover transaction, it obtains a *read-lock* for that workspace. When it is manipulating files in the child workspace, it obtains a *write-lock*.

Read-locks may be obtained concurrently by multiple Configuring commands that read files in the workspace; no commands may write to a workspace while any read-locks are in force. Only a single write-lock can be in force at any time; no Configuring command may write to a workspace while a write-lock is in force. Lock status is controlled by the `Codemgr_wsdata/locks` file in each workspace.

If you attempt to bring over files into a workspace that is locked, you will be so notified with a message that states the name of the user that has the lock, the command they are executing, and the time they obtained the lock.

```
bringover: Cannot obtain a write lock in workspace
"/tmp_mnt/home/my_home/projects/mpages"
because it has the following locks:
    Command: bringover (pid 20291), user: jack, machine: holiday,
time: 12/02/91 16:25:23
    (Error 2021)
```

- Accessibility (by users) to workspaces is controlled by the `Codemgr_wsdata/access_control` file in each workspace. Make sure that “bringover-to” and “bringover-from” access for your workspaces are set appropriately. Refer to Section , “Controlling Access to Workspaces,” on page 71 for more information.
- Configuring records information regarding the Bringover transaction in the `Codemgr_wsdata/history` file. This information can be useful to you as a means of tracking changes that have been made to files in your workspaces. Refer to “Viewing Workspace Command History” on page 78 for further information regarding these files.
- Configuring executes commands during a Bringover transaction and expects to find them in your command search path. Make sure that your `PATH` variable includes the directory in which Configuring commands are installed.

### *Updating an Existing Child Workspace (Bringover Update)*

You use the Configuring Bringover Update transaction to update an existing child workspace. You can display the Bringover Update layout of the Transactions window by any of the following methods:

- Drag and drop a workspace icon on top of the icon of a child workspace.
- Select a child workspace icon and choose the Bringover ⇒ Update item from the Transactions menu.
- Select a child workspace icon and choose the Bringover ⇒ Update item from the Workspace Graph pane pop-up menu.
- Choose the Bringover Update item from the Category menu if the Transactions window is already displayed.

The Bringover Update transaction transfers files that are under SCCS control. When a file exists in the parent workspace but not in the child, its SCCS history file is copied to the child and its g-file (the most recent delta) is created through the SCCS `get` command. When a file exists in both workspaces and has changed only in the parent, Configuring copies the new deltas from the parent to the child. When a file has changed in both workspaces, Configuring moves the *child's* new deltas into an SCCS branch.

To initiate a Bringover Update transaction follow these five basic steps:

---

**1. Specify the child workspace.**

If you select a workspace icon on the Workspace Graph pane prior to displaying the Bringover Update window, its name is automatically inserted in the To Child Workspace Directory text field. You can insert new path names, and edit and change the text field by hand at any point.

**2. Specify the parent workspace.**

The name of the selected child's parent workspace is automatically inserted in the From Parent Workspace text field. The parent workspace name is retrieved from the Configuring metadata file named `Codemgr_wsdata/parent`.

---

**Note** – You can also specify the path name of directories that are not workspaces to import files and directories into the workspace.

---

You can change a child workspace's parent for the duration of a single Bringover Update transaction by specifying the new parent's path name in the From Parent Workspace text field. You change the parent for that transaction only; if you wish to permanently change a workspace's parent, use the Reparent item on the Configuring window Edit menu or drag the child workspace icon over the new parent's icon. See "Reparenting a Workspace" on page 66 for details regarding reparenting workspaces.

---

**Note** – If you enter the child workspace name by hand and no icons are selected in the Workspace Graph pane, Configuring automatically updates the parent field if you rechoose the Bringover Update item in the Category menu.

---

**3. Create a list of directory and file names in the File List Pane.**

You can copy all or part of the contents of the parent workspace to the child. You specify the directories and files you wish to copy in the File List pane. See "Specifying Directories and Files for Transactions" on page 85 for information about specifying directory and file arguments.

---

**Note** – If you are using your own FLPs to generate file lists, you also specify them in the File List pane.

---

**4. Select options.**

<input checked="" type="checkbox"/> Preview	Select this option to preview the results of the transaction. If you invoke the Bringover Create transaction with this option selected, the transaction will proceed without actually transferring any files. You can monitor the output messages in the Transaction Output window (Show Output) as if the transaction were actually proceeding.
<input checked="" type="checkbox"/> Verbose	Select this option to increase the information displayed in the Transaction Output window. By default, a message is displayed for each created, updated, or conflicting file. The Verbose option causes bringover to print a message for all files, including those that are not brought over. If both the Verbose option and the Quiet option are specified, the Quiet option takes precedence.
<input checked="" type="checkbox"/> Quiet	Select this option to suppress the output of status messages to the Transaction Output window (Show Output).
<input checked="" type="checkbox"/> Skip SCCS gets	Select this option to inhibit the automatic invocation of the SCCS <code>get</code> program as part of the Bringover transaction. Normally <code>g</code> -files are extracted after they are brought over. This option improves file transfer performance although it shifts the responsibility to the user to do the appropriate <code>gets</code> at a later time.
<input checked="" type="checkbox"/> Force Conflicts	Select this option to cause all updates to be treated as conflicts. This option is not applicable to the Bringover Create transaction, but is applicable to the Bringover Update transaction.

**5. Invoke the Bringover button to initiate the transaction.**

*Notes about the Bringover Update Transaction*

- Checked-out files

When, during a Bringover Update transaction, Configuring encounters files that are checked-out from SCCS, it takes action based on preserving the consistency of the files and any changes to the file that might be in process.

Table 8-2 shows the different actions that Configuring takes when it encounters checked-out files.

*Table 8-2* Effects of Checked-out Files on Bringover Update Transactions

File Checked-out in Parent	File Checked-out in Child	Configuring Action
g-file and latest delta differ		<ul style="list-style-type: none"> <li>• Issue a warning</li> <li>• Process file</li> </ul>
g-file and latest delta are identical		<ul style="list-style-type: none"> <li>• Process file</li> </ul>
	g-file and latest delta are identical	<ul style="list-style-type: none"> <li>• Uncheckout the file</li> <li>• Process the file</li> <li>• Checkout the file</li> </ul>
	g-file and latest delta differ	<ul style="list-style-type: none"> <li>• Create a conflict</li> </ul>
	g-file is readonly	<ul style="list-style-type: none"> <li>• Issue a warning</li> <li>• Do not process the file</li> </ul>

- As the transaction proceeds, status information is displayed in the Transaction Output window. Messages are displayed as files are processed during the transaction and a transaction summary is displayed when execution is completed.
- Bringover Update transactions often produce conflicts (when files are changed in both the parent and child). When this occurs, you are so notified by messages in the Transaction Output window. See Chapter 9, “Resolving Conflicts,” for details about resolving conflicts.
- If you specify *relative* path names for directory and file names be aware that they are interpreted as being relative from the top-level (root) directory of the workspace hierarchy (which is assumed to be the same in both parent and child). If you specify these file names using *absolute* path names, the file must be found in one of the two workspaces or it will be ignored.
- The parent and child workspaces must be accessible through the file system. Either automounter or NFS® mounts can be used.
- Action taken during the Bringover Update transaction can be reversed using the Undo transaction. Refer to “Reversing Bringover and Putback Transactions with Undo” on page 105 for details.

- While files are read and examined in the parent workspace during the transaction, Configuring obtains a *read-lock* for that workspace. When Configuring manipulates files in the child workspace, it obtains a *write-lock*.

Read-locks may be obtained concurrently by multiple Configuring commands that read files in the workspace; no commands may write to a workspace while any read-locks are in force. Only a single write-lock may be in force at any time; no Configuring command may write to a workspace while a write-lock is in force. Lock status is controlled by the `Codemgr_wsdata/locks` file in each workspace.

If you attempt to bring over files into a workspace that is locked, you will be so notified with a message that states the name of the user that has the lock, the command they are executing, and the time they obtained the lock.

```
bringover: Cannot obtain a write lock in workspace
"/tmp_mnt/home/my_home/projects/mpages"
because it has the following locks:
    Command: bringover (pid 20291), user: jack, machine: holiday,
time: 12/02/91 16:25:23
    (Error 2021)
```

- Accessibility (by users) to workspaces is controlled by the `Codemgr_wsdata/access_control` file in each workspace. Ensure that “bringover-to” and “bringover-from” access for your workspaces are set appropriately. Refer to “Controlling Access to Workspaces” on page 71 for more information.
- Bringover Update transaction information is recorded in the `Codemgr_wsdata/history` file. This information can be useful as a means of tracking changes that have been made to files in your workspaces. Refer to “Viewing Workspace Command History” on page 78 for further information regarding these files.
- Configuring executes a number of programs as part of the Bringover Update transaction and expects to find them in your command search path. Ensure that your `PATH` variable includes the directory in which Configuring commands are installed.

## Bringover Action Summary

Table 8-3 summarizes the actions that Configuring takes during Bringover transactions.

*Table 8-3* Summary of Configuring Action during a Bringover Transaction

File in Parent	File in Child	Action by Configuring
Exists	Does not exist	Create the file in the child
Does not exist	Exists	None
Unchanged	Unchanged	None
Unchanged	Changed	None
Changed	Unchanged	Update file in the child. (Merge SCCS files and extract [via <code>get</code> ] a g-file that consists of the most recent delta.)
Changed	Changed	Merge SCCS history files in the child, create conflict, and notify user of the conflict. Current line of work in the child is moved to an SCCS branch.

## Copying Files from a Child to a Parent Workspace (Putback)

All Configuring file transfer transactions are performed from the perspective of the child workspace; hence the Putback transaction “puts back” groups of files from the child to the parent workspace.

You use the Putback transaction to make the parent and child workspace identical with respect to the set of files that you specify for the Putback transaction. Use the Putback transaction after you make changes and test them in the child workspace. Putting the files back into the parent usually makes them accessible to other developers.

During a Putback transaction, Configuring may find that it cannot transfer files from the child to the parent workspace without endangering the consistency of the data in the parent. If this occurs, no files are transferred and the Putback transaction is said to be *blocked*. A Putback transaction is blocked because:

- A file in either workspace is currently checked out from SCCS.

- A file in the parent workspace contains changes not yet brought over into the child workspace.
- A file conflict in either workspace is currently unresolved.

The Putback transaction transfers files that are under SCCS control. When a file exists in the child workspace but not in the parent, its SCCS history file is copied to the parent and its g-file (the most recent delta) is materialized through the SCCS `get` command. When a file exists in both workspaces and has changed only in the child, Configuring copies the new deltas from the child to the parent. When a file has changed in the parent, or both the parent and child, the Putback transaction is blocked.

### *Updating a Parent Workspace Using Putback*

You can display the Putback layout of the Transactions window by any of the following methods:

- Drag and drop a child workspace icon onto a parent workspace icon.
- Select a workspace icon and choose the Putback item from the Transactions menu.
- Select a workspace icon and choose the Putback item from the Workspace Graph pane pop-up menu.
- Choose the Putback item from the Category menu if the Transactions window is already displayed.

To initiate a Putback transaction, follow these steps:

#### **1. Specify the child workspace.**

If you select a workspace icon on the Workspace Graph pane prior to displaying the Putback window, its name is automatically inserted in the From Child Workspace Directory text field. You can insert new path names, and edit and change the text field at any point.

#### **2. Specify the parent workspace.**

The name of the selected child's parent workspace is inserted in the From Parent Workspace text field. The parent workspace name is retrieved from the Configuring metadata file named `Codemgr_wsdata/parent`.



---

You can change a child workspace's parent for the duration of a single Putback transaction by specifying the new parent's path name in the To Parent Workspace text field. You change the parent for that transaction only; if you wish to permanently change a workspace's parent, use the Reparent item on the Configuring window Edit menu or drag the child workspace icon over the new parent's icon. See "Reparenting a Workspace" for details regarding reparenting workspaces.

---

**Note** – If you enter the child workspace name by hand and no icons are selected in the Workspace Graph pane, Configuring automatically updates the parent field if you rechoose the Putback item in the Category menu.

---

### 3. Create a list of directory and file names in the File List Pane.

You can copy all or part of the contents of the parent workspace to the child. You specify the directories and files you wish to copy in the File List pane. See "Specifying Directories and Files for Transactions" for information about specifying directory and file arguments.

---

**Note** – If you are using your own FLPs to generate file lists, you also specify them in the File List pane.

---

### 4. Select options.

- 
- |   |  |
|---|--|
| <input checked="" type="checkbox"/> Preview | Select this option to preview the results of the transaction. If you invoke the Bringover Create transaction with this option selected, the transaction will proceed without actually transferring any files. You can monitor the output messages in the Transaction Output window (Show Output) as if the transaction were actually proceeding.   |
| <input checked="" type="checkbox"/> Verbose | Select this option to increase the information displayed in the Transaction Output window. By default, a message is displayed for each created, updated, or conflicting file. The Verbose option causes bringover to print a message for all files, including those that are not brought over. If both the Verbose option and the Quiet option are specified, the Quiet option takes precedence. |
-

---

<input checked="" type="checkbox"/> Quiet	Select this option to suppress the output of status messages to the Transaction Output window (Show Output).
<input checked="" type="checkbox"/> Skip SCCS gets	Select this option to inhibit the automatic invocation of the SCCS <code>get</code> program as part of the Bringover transaction. Normally <code>g</code> -files are extracted after they are brought over. This option improves file transfer performance although it shifts the responsibility to the user to do the appropriate <code>gets</code> at a later time.
<input checked="" type="checkbox"/> Auto Bringover	Select this option to cause Configuring to automatically start a Bringover Update transaction to update files in the child if the Putback transaction is blocked.

---

**5. Enter a comment.**

Enter a comment that describes the Putback transaction. This comment is included with the transaction log written into the file called `Codemgr_wsdata/history` in the parent workspace. The comment can be up to 8 Kbytes long.

**6. Invoke the Putback button to initiate the transaction.**

*Notes about the Putback Transaction*

- Checked-out files

When, during a Putback transaction, Configuring encounters files that are checked-out from SCCS, it takes action based on preserving the consistency of the files and any changes to the file that might be in-process.

Table 8-4 shows the different actions that Configuring takes when it encounters checked-out files.

*Table 8-4* Effects of Checked-out Files on Putback Transactions

File Checked-out in Parent	File Checked-out in Child	Configuring Action
g-file and latest delta differ		•Block Putback transaction
g-file and latest delta are identical or g-file does not exist)		•Uncheckout the file •Process the file •Check-out the file
	g-file and latest delta differ	•Block Putback transaction
	g-file and latest delta are identical	•Process the file
	g-file does not exist	•Issue a warning •Process the file •No changes made

- As the transaction proceeds, status information is displayed in the Transaction Output window. Messages are displayed as files are processed during the transaction, and a transaction summary is displayed when execution is completed.
- If you specify relative path names for directory and file names, be aware that they are interpreted as being relative from the top-level (root) directory of the workspace hierarchy (which is assumed to be the same in both parent and child). If you specify these file names using absolute path names, the file must be found in one of the two workspaces or it will be ignored.
- The parent and child workspaces must be accessible through the file system. You can use either automounter or NFS mounts.
- Action taken during the Putback transaction can be reversed using the Undo transaction. Refer to Section , “Reversing Bringover and Putback Transactions with Undo,” on page 105 for details.
- While files are read and examined in the child workspace during the transaction, Configuring obtains a *read-lock* for that workspace. When Configuring manipulates files in the parent workspace it obtains a *write-lock*.

Read-locks may be obtained concurrently by multiple Configuring commands that read files in the workspace; no commands may write to a workspace while any read-locks are in force. Only a single write-lock may be in force at any time; no Configuring command may write to a workspace while a write-lock is in force. Lock status is controlled by the `Codemgr_wsdata/locks` file in each workspace.

If you attempt to put back files into a workspace that is locked, you are notified with a message such as the following that states the name of the user that has the lock, the command they are executing, and the time they obtained the lock.

```
putback: Cannot obtain a write lock in workspace
"/tmp_mnt/home/my_home/projects/mpages"
because it has the following locks:
    Command: bringover (pid 20291), user: jack, machine: holiday,
time: 12/02/91 16:25:23
    (Error 2021)
```

- Accessibility (by users) to workspaces is controlled by the `Codemgr_wsdata/access_control` file in each workspace. Ensure that “putback-to” and “putback-from” access for your workspaces are set appropriately. Refer to “Controlling Access to Workspaces” on page 71 for more information.
- Putback transaction information is recorded in the file called `Codemgr_wsdata/history`. This information can be useful as a means of tracking changes that have been made to files in your workspaces. Refer to “Viewing Workspace Command History” on page 78 for further information regarding these files.
- Configuring executes a number of programs as part of the Putback transaction and expects to find them in your command search path. Make sure that your `PATH` variable includes the directory in which Configuring is installed.

### *Putback Action Summary*

Table 8-5 summarizes the actions that Configuring takes during Putback transactions.

*Table 8-5* Summary of Configuring Action during a Putback Transaction

File in Parent	File in Child	Action by Configuring
Exists	Does not exist	Block Putback and notify user.
Does not exist	Exists	Create the file in the parent.
Unchanged	Unchanged	None.
Unchanged	Changed	Update file in the parent. (Merge SCCS files and extract [via <code>get</code> ] a g-file that consists of the most recent delta.)
Changed	Unchanged	Block Putback, notify user.
Changed	Changed	Block Putback, notify user.
Checked out	Checked out*	Block Putback, notify user.
Unresolved conflict	Unresolved conflict**	Block Putback, notify user.

\*If a file is checked out in either the parent or the child, the transaction is blocked. See Table 8-4 for more information about putting back files that are checked out.

\*\*If a conflict is unresolved in either the parent or the child, the transaction is blocked.

## *Reversing Bringover and Putback Transactions with Undo*

You can reverse (undo) the action of the *most recent* Bringover or Putback transaction in a workspace by using the Undo Transactions window layout. You Undo the Putback or Bringover transaction in the destination workspace (the one in which the files are changed). You can undo a Bringover or Putback transaction as many times as you like until another Bringover or Putback transaction makes changes in that workspace; only the *most recent* Bringover/Putback transaction can be undone.

If a file is updated or found to be in conflict by the Putback or Bringover transaction, the Undo transaction restores the file to its original state. If a file is “new” (created by the Bringover/Putback transaction), then it is deleted.

To initiate an Undo transaction, follow these three basic steps:

**1. Specify the workspace in which to reverse the transaction.**

If you select a workspace icon on the Workspace Graph pane prior to displaying the Undo layout, its name is automatically inserted in the Workspace Directory text field. You can insert a new path name followed by a Return, and edit and change the text field by hand at any point.

**2. Click on the Undo button to initiate the transaction.**

### *Notes about the Undo Transaction*

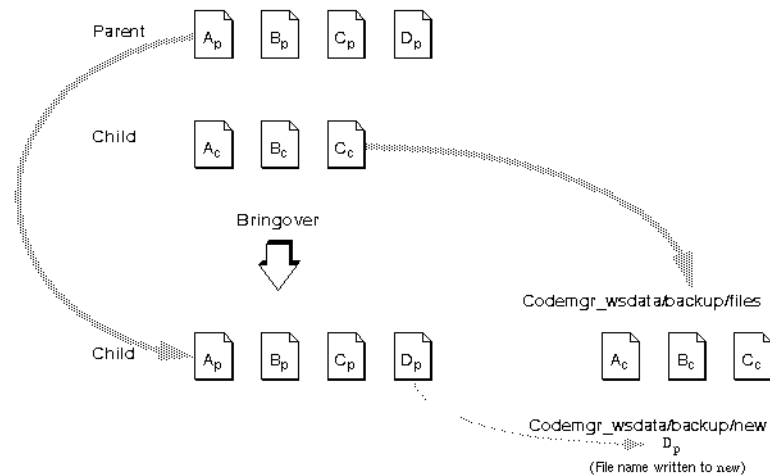
- When it is manipulating files in the specified workspace, Configuring obtains a *write-lock* for the workspace. Only a single write-lock may be in force at any time; no Configuring command may write to a workspace while a write-lock is in force. Lock status is controlled by the `Codemgr_wsdata/locks` file in each workspace. If Configuring cannot obtain the lock, it will display an error message and abort.
- Configuring records information regarding the Undo transaction in the `Codemgr_wsdata/history` file. This information can be useful as a means of tracking changes that have been made to files in your workspaces. Refer to “Viewing Workspace Command History” on page 78 for further information regarding these files.

### *How the Undo Transaction Works*

When the Bringover and Putback transactions update or create files in the destination workspace (the child in the case of Bringover, the parent in the case of Putback), they make backup copies of the originals before they actually make changes to the files. All existing files are copied to the `Codemgr_wsdata/backup/files` directory in the destination workspace, and the names of all newly created files are entered into a file called `Codemgr_wsdata/backup/new`.

When you decide that you would like to cause a workspace to revert to its state before a Bringover/Putback transaction, the Undo transaction does the following:

- Copies the backed-up files from the `Codemgr_wsdata/backup/files` directory over the transferred files



- Deletes files whose names are contained in the Codemgr\_wsdata/backup/new file

The next Bringover/Putback transaction overwrites all data in the Codemgr\_wsdata/backup directory.

**Note** – All files transferred by Configuring are under SCCS control. Usually, only SCCS history files are backed up during Bringover and Putback transactions; if the files are subsequently restored, the Undo transaction extracts the appropriate g-file (most recent delta) from the history file. If, however, a file in the child is checked out (using `sccs edit`) during the Bringover transaction (Configuring permits files to be checked out during a Bringover transaction, but not during a Putback transaction. If a file that is being put back is checked out, an error condition exists). Configuring backs up *both* the g-file and the SCCS history file in order to preserve the work in progress; the g-file and the SCCS history file are copied to the Codemgr\_wsdata/backup/files directory and restored by the Undo transaction.

## *Renaming, Moving, or Deleting Files*

When you rename, move, or “delete” files as described in this section, Configuring tracks those changes so that it knows how to manage the altered files during Bringover and Putback transactions. Although Configuring processes these files automatically, it is helpful for you to understand some of the ramifications of renaming, moving, or deleting files.

---

**Note** – For the purposes of this discussion, the terms “rename” and “move” are considered to be the same action and are referred to only as “rename.”

---

The best way to delete and rename files is to use the move and delete commands available from the Workshop Versioning menu. This section describes the underlying process.

### *Renaming Files*

When you bring over or put back files that you (or another user) have renamed, Configuring must decide whether the files have been newly created or whether they existed previously and have been renamed.

For example, in the following figure, the name of file C in the parent is changed to D. When Configuring brings the file over to the child it must decide which of the following is true:

- D has been newly created in the parent.
- It is the same file as C in the child, only with a new name.



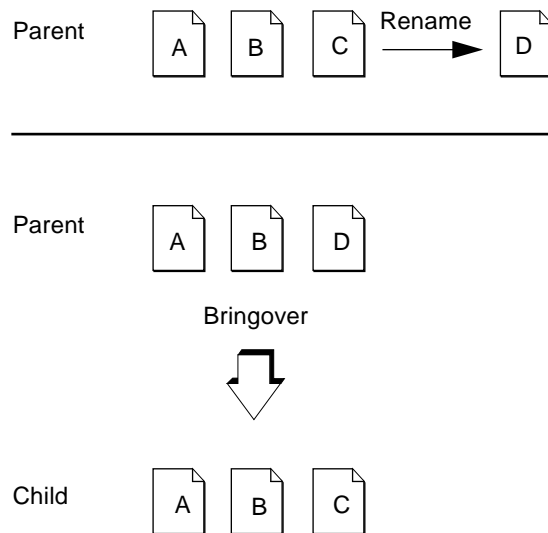


Figure 8-2 File “C” Renamed to “D”

If the same case was the subject of a Putback operation, the same problem would apply: Is “C” new in the child, or has it been renamed from some other file?

The action that Configuring takes is very different in each case. If it is a new file in the parent, Configuring creates it in the child; if it has been renamed in the parent, Configuring renames file “C” to “D” in the child.

Configuring stores information in the SCCS history files that enables it to identify files even if their names are changed. You may have noticed the following message when viewing Bringover and Putback output:

```
Examined files:
```

Configuring examines all files involved in a Bringover Update or Putback transaction for potential rename conditions before it begins to propagate files.

When Configuring encounters renamed files, it propagates the name change to the child in the case of Bringover, and to the parent in the case of Putback. You are informed of the change in the Transaction Output window with the following messages:

```
rename from: old_filename
           to:  new_filename
```

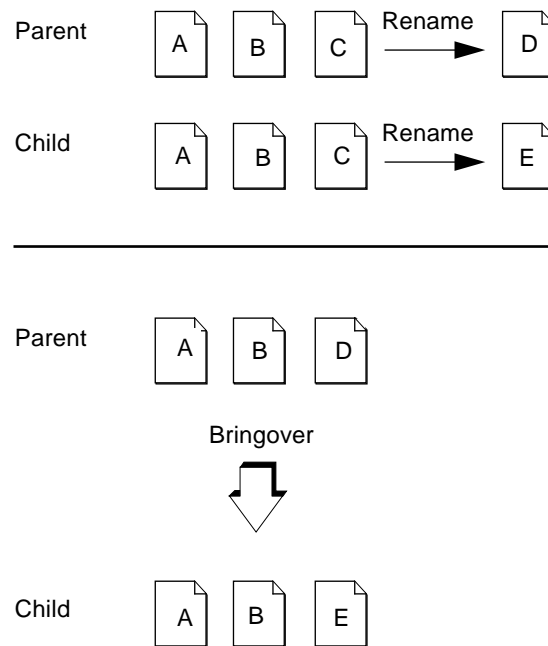
### ***Name History***

Configuring stores information about a file's name history in its SCCS history file. The name history is simply a list of the workspace-relative names that have been given to the file during its lifetime. This information is used by Configuring to differentiate between files that have been renamed and those that are new. When you rename a file, Configuring updates the file's name history during the next Bringover or Putback transaction that includes it. When a name history is updated, you are notified in the Transaction Output window.

```
Names Summary:
    1  updated parent's name history
    1  updated children's name history
```

### ***Rename Conflicts***

In rare cases, a file's name is changed concurrently in parent and child workspaces. This is referred to as a *rename conflict*. For example, the name of file "C" is changed to "D" in the parent, and concurrently to "E" in the child.



**Figure 8-3** File “C” is Concurrently Renamed in both Parent and Child Workspaces

When this occurs, Configuring determines that both “D” in the parent and “E” in the child are actually the same file, but with different names. In the case of rename conflicts:

- Configuring reports the conflict using the name of the file in the child.
- Configuring always resolves the conflict by automatically changing the name of the file in the child workspace to the current (renamed) name in the parent; the name of the file from the parent is *always* chosen, even in the case of a Putback transaction.

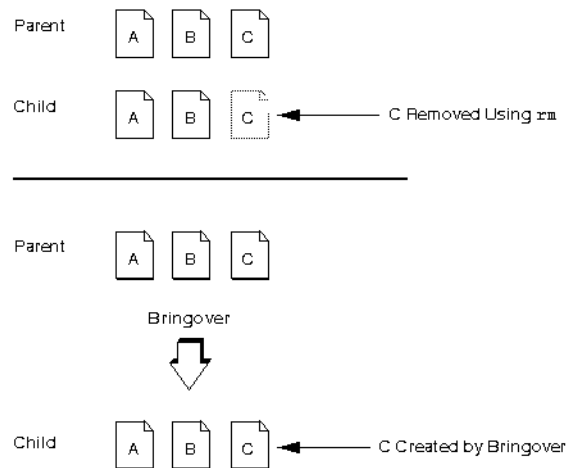
When Configuring encounters a rename conflict, you are notified in the Transaction Output window with the following messages:

```
rename conflict: name_in_child
rename from: name_in_child
           to: name_in_parent
```

## Deleting Files

Deleting files from a Configuring workspace is a little trickier than it first appears. Deleting a file from a workspace with the `rm` command causes Configuring to think that the file has been newly created in the workspace's parent or child.

Take for instance, the following example. The file "C" is removed from the child workspace using the `rm` command; later the Bringover Update transaction is used to update the child.



*Figure 8-4* File "C" Is Removed From The Child Using the `rm` Command, Then Created Again by Bringover

Configuring examines the two workspaces and determines that the file "C" exists in the parent and not in the child — following the usual Configuring rules, it creates "C" in the child.

The recommended method for "deleting" files in workspaces is to rename them out of the way using a convention agreed upon by everyone working on the project. One recommended method is to rename files you wish to "delete" so that they begin with the `.del-` prefix. For example:

```
example% mv module.c .del-module.c
example% mv SCCS/s.module.c SCCS/s..del-module.c
```

---

This method has a number of advantages:


- The file is no longer seen using default SunOS commands such as `ls`.
- Configuring does not recreate the file.
- Configuring propagates the change throughout the workspace hierarchy as a rename, “deleting” the file in all workspaces.
- The file remains available to later reconstruct releases for which it was a part (for example, if it was part of a freeze point (see Chapter 5, “Starting a Project” and Chapter 17, “Introduction to FreezePointing” for more information about freeze points).

### *Notes about Renaming Files*

- When you rename a file, you must rename *both* the *g*-file *and* the SCCS history file.
- During transactions, Configuring processes files individually. When you rename a directory, each file in the directory is evaluated separately as if each had been renamed individually.
- When files are renamed, Configuring propagates the change throughout the workspace hierarchy using the same rules used with file content updates and conflicts.



## *Resolving Conflicts*

9 

This chapter discusses the process by which the Configuring program detects conflicts and then assists you in resolving these conflicts.

<i>Detecting Conflicts during Bringover Update Transactions</i>	<i>page 116</i>
<i>Preparing Files for Conflict Resolution</i>	<i>page 117</i>
<i>Resolve Transaction</i>	<i>page 118</i>
<i>The Merging Program</i>	<i>page 119</i>

### *Conflict Resolution Process*

When files change concurrently in both a parent and child workspace, they are in conflict. Neither the version of the file in the child nor the version in the parent can be copied to the other without overwriting changes. Conflicts are detected during Bringover Update transactions. You must resolve conflicts in the child before the conflicting file(s) can be put back to the parent. The Configuring program assists you in resolving conflicts. Use the following procedures to resolve conflicts if you are using the GUI. If you are using the CLI, see the `resolve(1)` manual page for more information.

## Changing Names

The current release of TeamWare uses new command names, so the following table summarizes the correspondences for you. Note that the old commands still work, however this manual uses the new commands and GUI names.

*Table 9-1* Correspondences Between New and Old TeamWare Commands

Old Command	New Command	Old Tool Name	New GUI Name
codemgrtool	twconfig, teamware	CodeManager	Configuring
vertool	twversion	VersionTool	Versioning
filemerge	twmerge	FileMerge	Merging
maketool	twbuild	MakeTool	Building
freezepttool	twfreeze	FreezePoint	Freezepeating

## Detecting Conflicts

Before you can resolve conflicts, the Configuring program must detect the conflict and prepare the history files of the conflicting files for resolving. These two processes are described in this section.

### *Detecting Conflicts during Bringover Update Transactions*

Usually, the conflict resolution process begins when you attempt to put back files that have changed in both the parent and the child workspaces. The Putback transaction blocks the transfer of files from the child to the parent because the version of the file from the child will overwrite changes made in the parent.

After the Putback transaction is blocked, you must use the Bringover Update transaction to update the child. (If Putback is executed with the Auto Bringover option specified, then the Bringover transaction is initiated automatically by the Configuring program.) If, during the Bringover transaction, the Configuring program determines that the file in the child has *also* changed, a conflict exists. All files included in the Bringover Update transaction that are *not* in conflict are copied or updated normally.



---

## *Preparing Files for Conflict Resolution*

When a conflict is encountered during a Bringover Update transaction, the Configuring program takes special steps to prepare that file so that you can resolve the conflict.

The Configuring program incorporates the deltas created in the parent into the SCCS history file in the child. The parent and child deltas are placed on separate branches in the child SCCS history file. After the deltas are merged, the history file in the child contains:

- Delta(s) created in the parent
- Delta(s) created in the child
- The delta from which the two versions of the file are both descended (their *common ancestor*)

---

**Note** – The Versioning program enables you to view graphical depictions of SCCS delta histories (including branches).

---

Access to the three deltas (common ancestor, parent, and child) in the child enables you to use the Configuring Resolve transaction and the Merging program to compare the parent and child deltas — both to their common ancestor, and to each other.

In addition to merging deltas, the Configuring program adds the name of the conflicted file to the child's `Codemgr_wsdata/conflicts` file. The `conflicts` file is a text file that contains the names of all files in that workspace with unresolved conflicts.

The stage is set for you to resolve the conflicts using the Configuring Resolve transaction.

## *Resolving Conflicts*

The two tools that you use to resolve conflicts are:

- Configuring Resolve Transactions window
- Merging program

## *Resolve Transaction*

The Resolve layout of the Transactions window facilitates resolving conflicts detected during Bringover Update transactions. The Resolve transaction coordinates the merging process, acting as intermediary between you and the file-merging program—Merging.

As previously mentioned, when the Configuring program detects a conflict during a Bringover Update transaction, it does the following:

- Merges new deltas from the parent into the SCCS history in the child
- Enters the file's path name in the child's `Codemgr_wsdata/conflicts` file

To resolve conflicts in a workspace, follow these four steps:

**1. Double-click SELECT on the icon of a workspace that contains conflicted files.**

The Resolve layout of the Transaction window is automatically activated with the names of its conflicted files displayed in the File List Pane.

**2. Select a file in the File List Pane and then invoke the Merging selection button.**

The Configuring program starts the Merging program and begins to process the list of files from the File List Pane. For the next file in the list, the Configuring program extracts the parent delta, the child delta, and the common ancestor from the SCCS history file and passes their path names to the Merging program. (Configuring and Merging communicate via the ToolTalk™ service. The ToolTalk service is a network-spanning, interapplication communication service that allows applications to communicate with other autonomous applications.) The Merging window appears with the files loaded and ready for merging.

**3. Use Merging to resolve the differences between the parent and child versions of the file.**

See “The Merging Program” for more information.

**4. Save the file in Merging.**

After you use Merging to resolve differences between the parent and child versions of the file, the Configuring program creates a new delta in the child SCCS history file and removes the file name from the `conflicts` file. The new delta contains the “Merged Result” you created using Merging.

### *Notes about the Resolve Transaction*

- By default, the Configuring program automatically, sequentially processes the list of files from the File List Pane. After you resolve a conflict, the Configuring program automatically begins to process the next file in the list. If you want to change the behavior so that it individually processes only files that you explicitly select, deselect the Auto Advance check box in the Properties window.
- Conflicts need not be resolved immediately. You can continue to make changes and create new deltas in conflicted files in the child workspace. New deltas are created on a branch; when you finally resolve the conflict, the latest delta is the one merged with the version brought over from the parent. *Conflicts must be resolved before you can put back the files to the parent.*
- When the Configuring program creates the new delta in the child SCCS history file, it includes the following standard comment:

```
Merged changes between workspaces x and y
```

By default, the Configuring program does not prompt you for a comment to append to its comment. If you want to be prompted for comments that are appended to the standard comment, select the Skip Checkin Comments check box in the Properties window.

### *The Merging Program*

This section is a brief introduction to the Merging program as used with the Configuring program. For a more detailed description, refer to the Merging section in this manual.

Merging displays two text files (the parent and child deltas) for side-by-side comparison, each in a read-only subwindow. Beneath them, Merging displays a subwindow that contains a merged version of the two files. The merged version contains selected lines from either or both deltas and can be edited to produce a final merged version.

Each delta in each of the top windows is shown in comparison to the common ancestor delta:

- The child delta may be in the left window labeled “Child vs. Ancestor”
- The parent delta may be in the right window labeled “Parent vs. Ancestor”

The common ancestor is the delta from which both the parent and child deltas are descended. This arrangement permits you to make a three-way comparison—each delta to the common ancestor, and each delta to the other.

Lines in each descendant are marked according to their relationship to the corresponding lines in the common ancestor:

- If a line is identical in all three deltas, then no glyph appears.
- If a line is not in the ancestor but was added to one or both of the descendants, then a plus sign glyph (+) appears next to the line in the delta where the line was added.
- If a line is present in the ancestor but was removed from one or both of the descendants, then a minus sign (-) appears as a placeholder in the delta from which the line was removed.
- If a line is in the ancestor but has been changed in one or both of the descendants, then a vertical bar glyph (|) appears next to the line in the delta where the line was changed.

When Merging discovers a line that differs between either of the two deltas and the ancestor, it marks with glyphs the lines in the two deltas and also in the automatically merged file. Together, these marked lines are called a *difference*. While Merging is focusing on a difference, it highlights the glyphs.

The difference on which Merging is focusing at any given time is called the *current difference*. The difference that appears immediately later in the file is called the *next difference*; the difference that appears immediately earlier in the file is called the *previous difference*.

While focusing on a difference, you can accept a line from either of the original deltas, or you can edit the merged version by hand. When you indicate that you are satisfied with your changes (by clicking on a control panel button), the current difference is said to be *resolved*. After a difference is resolved, Merging changes the glyphs that mark the difference to outline (hollow) font. Merging then automatically advances to the next difference (if the Auto Advance property is on), or moves to another difference of your choice.

In summary, when used with the Configuring program, Merging activity is coordinated by the Resolve transaction window. The Configuring and Merging programs communicate bidirectionally through the ToolTalk service. The Configuring program extracts the parent, child, and common ancestor deltas and starts the Merging program, passing it the names of the files that contain

---

the deltas to be merged. When you complete the merge process using the Merging Save button, the Configuring program creates a new delta in the file's SCCS history file that contains the "Merged Results" and removes the file name from the `conflicts` file.

In the next chapter we consider how to administer the TeamWare workspaces.



# Administering the Workspace

10 

The Configuring program requires little administrative support. However, there are some things to consider when starting out. This chapter contains the following sections:

<i>Starting a Project with the Configuring Program</i>	<i>page 123</i>
<i>Structuring Your Workspace Hierarchy</i>	<i>page 124</i>
<i>Product Release Considerations</i>	<i>page 128</i>

## *Starting a Project with the Configuring Program*

Getting started with the Configuring program is simple. The following sections provide guidelines and strategic issues that you (the project administrator) should consider to maximize the benefit your project receives by using the Configuring program.

### *Moving an Existing Project*

The Configuring program works only with projects that use SCCS for version control. Moving an existing SCCS-based project to the Configuring program is a simple process:

- Ensure that all SCCS history files (“s-dot-files”) are in directories named SCCS located directly beneath directories that contain source files.
- Be sure that your project directory structure is current and organized.

- Execute the Create Workspace command item in the File menu, specifying the top-level directory as your workspace. The Create Workspace command creates the `Codemgr_wsdata` directory under the top-level directory.
- Begin using the Bringover Create transaction to form a workspace hierarchy. See “Structuring Your Workspace Hierarchy” on page 124 for guidelines regarding workspace hierarchies.

If your project is structured so that compilation units can be easily grouped on a directory basis during transfer operations, you can use the default Configuring FLP. See “Grouping Files for Transfer Using File List Programs” for a description of the default FLPs.

If your project requires files to be grouped for transfer operations in special ways, you have to write your own FLP(s).

## *Starting a New Project*

If you are starting a new project:

- Use the Create Workspace command item in the File menu to create your project’s top-level directory (with its `Codemgr_wsdata` directory)
- Proceed as you normally would to set up an SCCS-based development hierarchy. Ensure that all SCCS history files (“s-dot-file”) are in directories named `SCCS` located directly beneath directories that contain source files.
- Begin using the Bringover Create transaction to form a workspace hierarchy. See “Structuring Your Workspace Hierarchy” on page 124, for guidelines regarding workspace hierarchies.
- The default Configuring FLP groups files recursively by directory; if you intend to use that FLP, be sure to arrange files in compilation units accordingly. If your project requires that files be grouped differently during transfer, be sure to arrange your project hierarchy in such a way that it works well with the FLP(s) you will create.

## *Structuring Your Workspace Hierarchy*

The way you structure the workspace hierarchy of your project will have influence on the inter-workspace file-transfer process and on how you prepare product releases. The following discussion will help you make informed choices about the kind of workspace hierarchy best suited for your project.



---

**Note** – Whatever initial decisions you make regarding workspace hierarchies can later be changed by using Configuring’s workspace reparenting feature. See “Reparenting a Workspace” on page 66 for details.

---

A workspace hierarchy is a chain of parent/child workspaces two or more layers deep. The number of layers in a hierarchy bears no relation to the number of workspaces comprising it. A parent workspace and its child comprise two layers. A parent workspace and three children also comprise two layers. A parent workspace and its child and grandchild comprise three layers. Figure 10-1 depicts a “flat” (three-tiered) hierarchy and a “multitiered” (four-tiered) hierarchy.

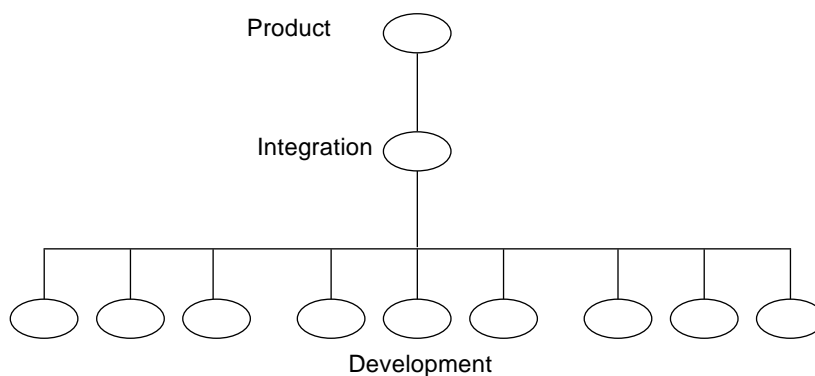


Figure 10-1 A Three-Tiered Hierarchy (Flat)

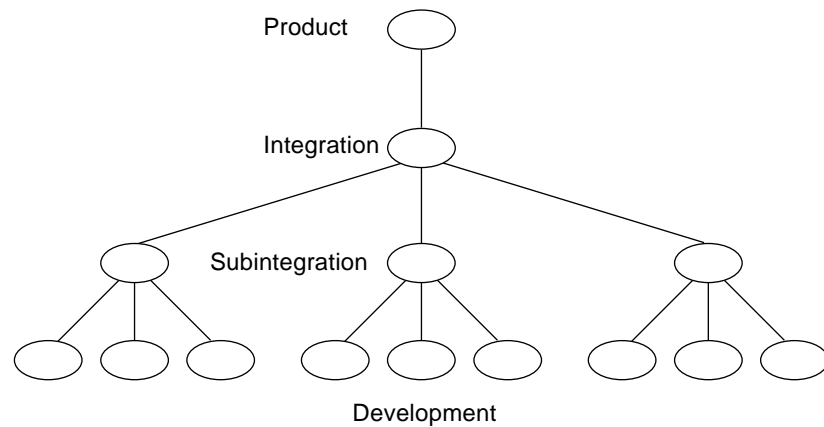


Figure 10-2 A “Multitiered” (Four-Tiered) Hierarchy

### ***File Transfer Considerations***

The way in which you set up your workspace hierarchy can have an impact on the transfer of files up and down the hierarchy. If you have over 2000 files in a single directory under SCCS control, you could have problems using TeamWare. Subdivide workspace directories for better performance.

### ***File System Accessibility***

In order to transfer (Bringover/Putback) files between workspaces, both the parent and the child must be mounted on the same file system. The automounter can be used to connect file systems.

### ***Flat Hierarchy vs. Multitiered Hierarchy***

#### ***Advantages of a Flat Hierarchy***

A flat workspace hierarchy is one in which many developers put back files to a single integration workspace. The advantage of a flat hierarchy is that all developers have immediate access to one another’s work. The moment that

---

Jack (a developer) puts back his work to the integration workspace, Jon (another developer) can use the Bringover Update transaction to have immediate access to the changes made by Jack.

### ***Disadvantages of a Flat Hierarchy***

The disadvantage of a flat hierarchy is that time is often wasted because the integration workspace changes frequently, requiring developers to do frequent Bringover transactions, builds, and tests in order to keep their source base up-to-date. There is a cumulative effect of doing Putback transactions; the first developer to do a Putback resolves only one set of changes, the next developer resolves two, and so on till the last developer, who must resolve all of the changes that have been made within her development group.

### ***Advantages of a Multitiered Hierarchy***

The amount of time required for a developer to put her work back to the integration workspace can be sharply reduced by interposing a tier of subintegration workspaces between the integration and development level workspaces.

Whenever a developer puts back work to an integration workspace, there is some chance that the next developer to do a Putback transaction will not be able to put back their changes until they bring over the earlier changes, rebuild the modules, and test the new changes with their own — the more Putbacks that occur the higher the potential for conflict.

When many developers work on a project, the Bringover, rebuild, test cycle can become onerous and time consuming. If smaller groups of developers working on related portions of code integrate into a subintegration workspace, that workspace will be more stable and require fewer builds and less testing. Of course when the subintegration workspaces are themselves put back to their common integration area, changes made in the other development workspaces will have to be integrated. Experience has shown, however, that doing larger integrations, less frequently, is more efficient.

### ***Disadvantages of a Multitiered Hierarchy***

The disadvantages of multiplying subintegration workspaces are as follows:

- Each new workspace consumes disk space.

- Developers who ought regularly to be looking at one another's work may find it harder to do so because they do not put back to the same integration workspace
- Integration of the subintegration workspaces to the higher integration workspace can become more complicated than more frequent, smaller integrations.

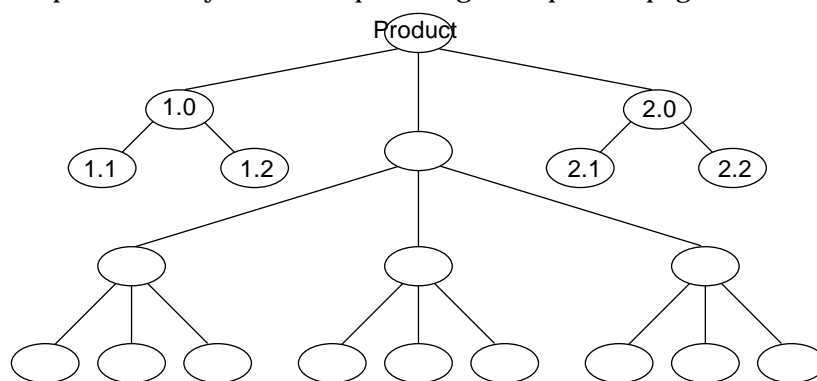
### *Product Release Considerations*

When you plan your project hierarchy structure, you must consider how you plan to release your product. There are several ways that you can structure workspace hierarchies to facilitate the preparation of major, minor, and patch releases. The following discussion presents some ideas for you to consider; your product may not lend itself to this model, or your product may have considerations that suggest an alternate scheme.

It's best to dedicate a workspace as a product release staging area for each release. It's also a good idea to "hang" the release workspace off of a top level "product" workspace. The product workspace should be located hierarchically above the workspaces in which normal development integration is done. Locating the product workspace this way permits you to begin development of your next release without corrupting the current release.

After the files are transferred to the product workspace, use the Bringover transaction to transfer the files down to the release workspace. The release workspace can be used to make masters and can serve as an area in which to save work for subsequent releases if necessary. Figure 10-3 shows a hierarchy

that contains a product workspace and release workspaces for six different releases. You can use the reparenting feature to transfer data between release workspaces directly. See “A Reparenting Example” on page 68” for details.



*Figure 10-3* Product and Release Workspaces



## *How the Configuring Program Merges SCCS Files*

**11** 

This chapter describes the ways the Configuring program manipulates SCCS history files when you copy files between workspaces and resolve conflicts.

<i>Merging Files That Do Not Conflict</i>	<i>page 132</i>
<i>Merging Files That Conflict</i>	<i>page 133</i>
<i>An Example of Merging</i>	<i>page 134</i>

**Note** – This discussion assumes that you are familiar with SCCS, including the concept of branching. SCCS is described in detail in the *Programming Utilities* manual.

When considering Bringover and Putback transactions, remember that source files are derived from SCCS deltas and are identified by SCCS delta IDs (SIDs). When a *file* is copied by either a Putback or Bringover transaction, the Configuring program must manipulate the file's SCCS history file (also known as the "s-dot-file").

When a file is copied (by means of Bringover or Putback transaction) from a source workspace to a destination workspace, it appears that a single file has been transferred. In fact, all of the SCCS information for that file (deltas, comments, and so on) must be merged into the destination SCCS history file. By merging the information from the source into the destination history file, the current version (delta) can be rederived, and the file's entire delta and

comment history are available. (The exception is when the file does not exist in the destination workspace. In this case, the entire history file is copied from the source workspace to the destination workspace.)

### *Merging Files That Do Not Conflict*

If the file in the destination workspace is being updated (the file has changed in the source of a Bringover or Putback transaction and has not changed in the destination). The new deltas from the destination are added to the history file in the destination. The reason that SCCS history files are merged at all in this case, rather than the source history file being copied over the destination history file, is that administrative information (for example, flags and access lists) stored in the destination history file would be overwritten.

To accomplish the merger, the Configuring program determines where the delta histories diverge and adds (to the destination workspace) only the deltas that were created in the source workspace since they diverged. To determine where the histories diverge, the Configuring program compares the delta tables in both the parent and child history files; information used in this comparison includes comments and data such as when and who created the delta. Figure 11-1 contains an example of a Putback transaction where the Configuring program adds deltas 1.3 and 1.4 from the child workspace to the SCCS history file in the parent.



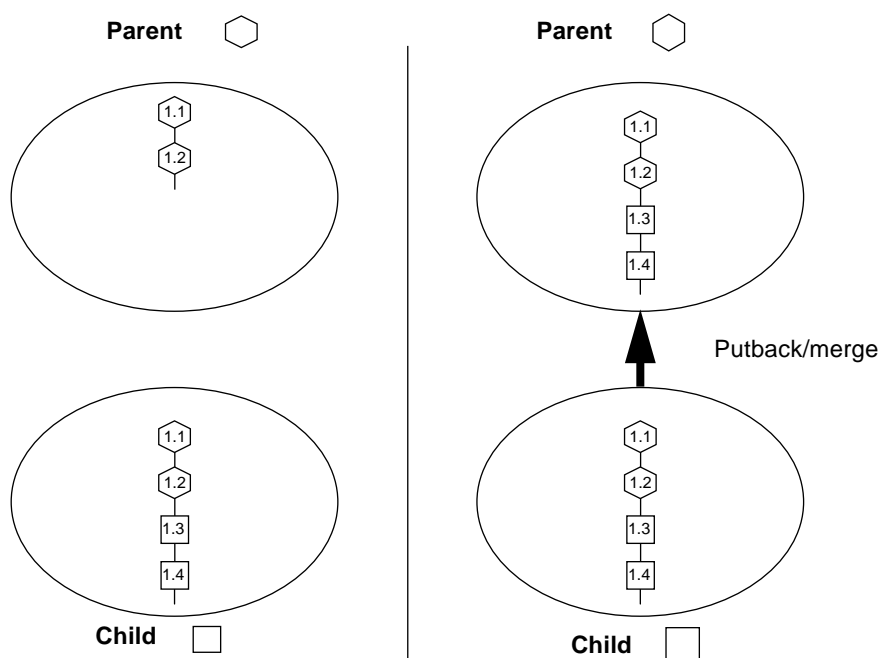


Figure 11-1 Updating a File in the Destination Workspace That Has Not Changed

## Merging Files That Conflict

When you propagate files between parent and child workspaces, often both the version of the file from the parent and the version in your child changed since they were last updated. When that is the case, the parent and child versions of the file are in conflict.

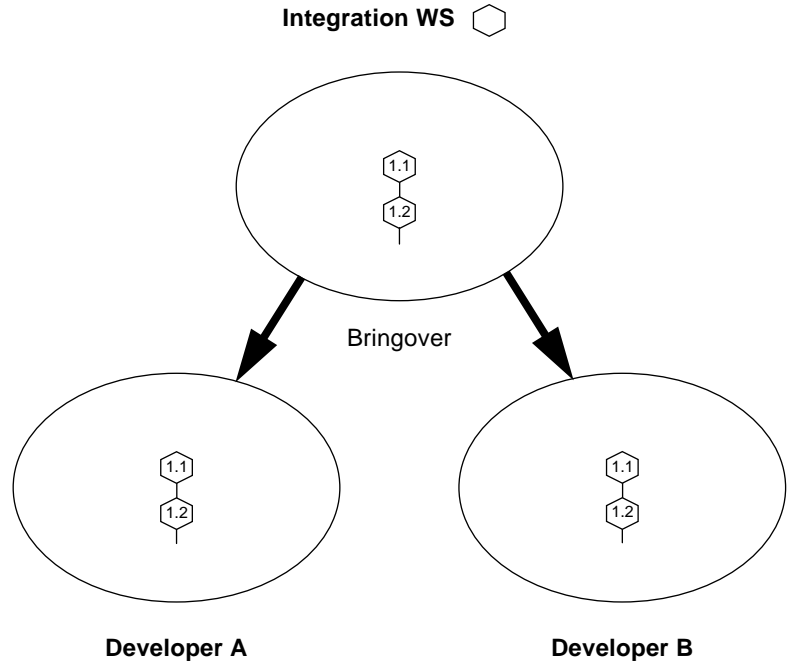
When file contents conflict, the Configuring program aids you in resolving the potentially conflicting changes that were made to the file, and preserves the file's delta, administrative, and comment history. To accomplish this, the Configuring program merges the SCCS deltas from the parent into the history file in the child. Configuring's Resolve transaction is then used to resolve the conflict in the child. See Chapter 9, "Resolving Conflicts", for details on resolving conflicts.

### An Example of Merging

This section illustrates the Configuring merging process. This merge example involves an integration workspace and two child workspaces owned by different developers. The developers bring over copies of the same file from the integration workspace, and independently change the file. The illustrations show how the SCCS history file is manipulated when conflicts occur and when they are resolved. Some notes regarding the following figures:

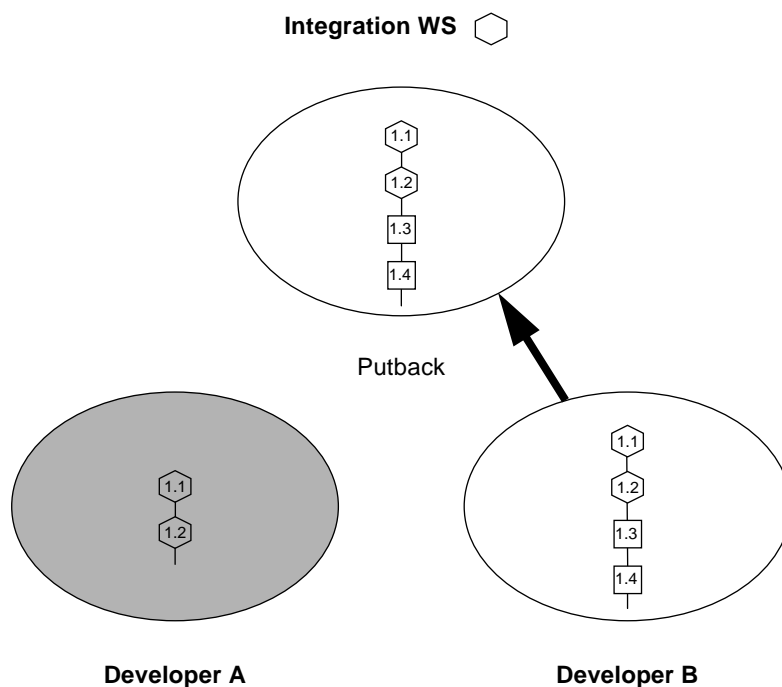
- The default delta (the point at which the next delta is added to the SCCS delta tree) is identified by an unattached descending line.
- You can use the Versioning program to graphically display SCCS delta trees in much the same way they are depicted here.

Both developers copy the same file from the integration workspace with the Bringover transaction. The file is new in both workspaces, so the SCCS history file is copied to both.



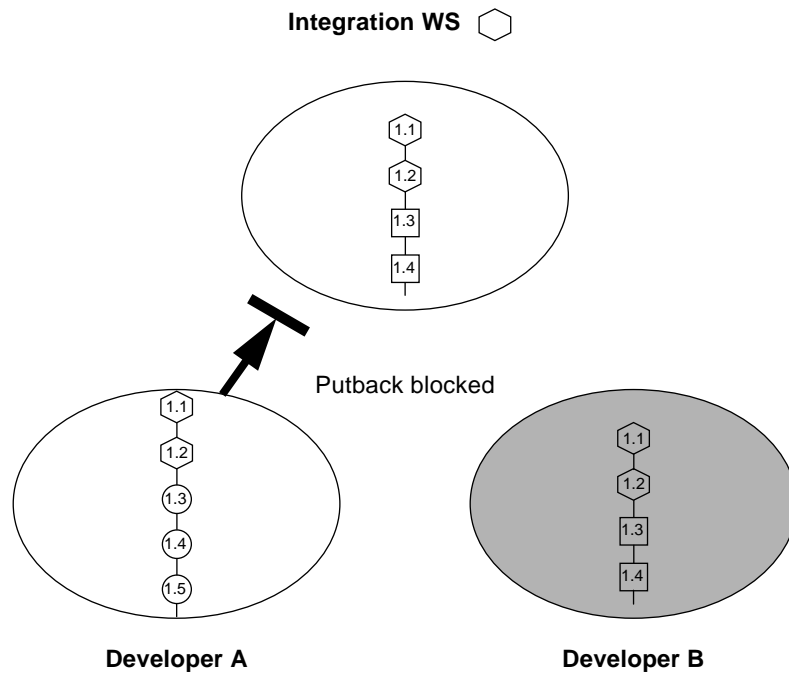
Developer B makes changes to the file, creating two new deltas: 1.3 and 1.4, and then puts the file back into the integration workspace (with the Putback transaction). The Configuring program appends the two new deltas to the parent SCCS delta tree.

Rather than replacing the destination workspace version of the SCCS history file with the source's version, the new deltas are added to the destination SCCS history file to preserve administrative information, such as access lists.



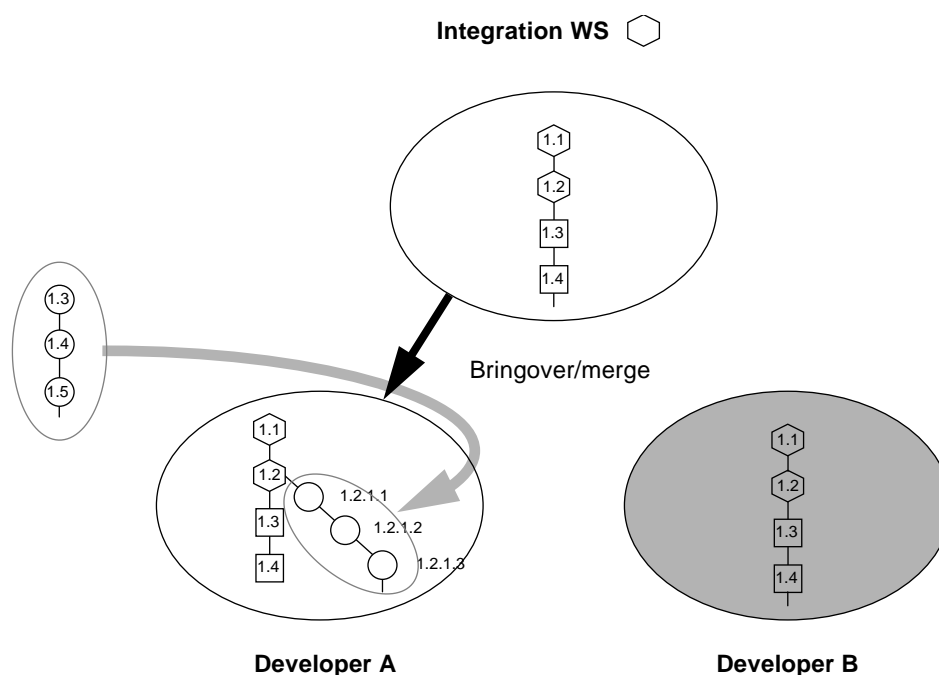
In the meantime, developer A also changes the file (creating three new deltas: 1.3, 1.4, and 1.5) and now attempts to put back the file into the integration workspace.

The Configuring program blocks the Putback of developer A because the files are in conflict. The changes put back by developer B would be overwritten. Developer A must also incorporate the changes made by developer B into his work.



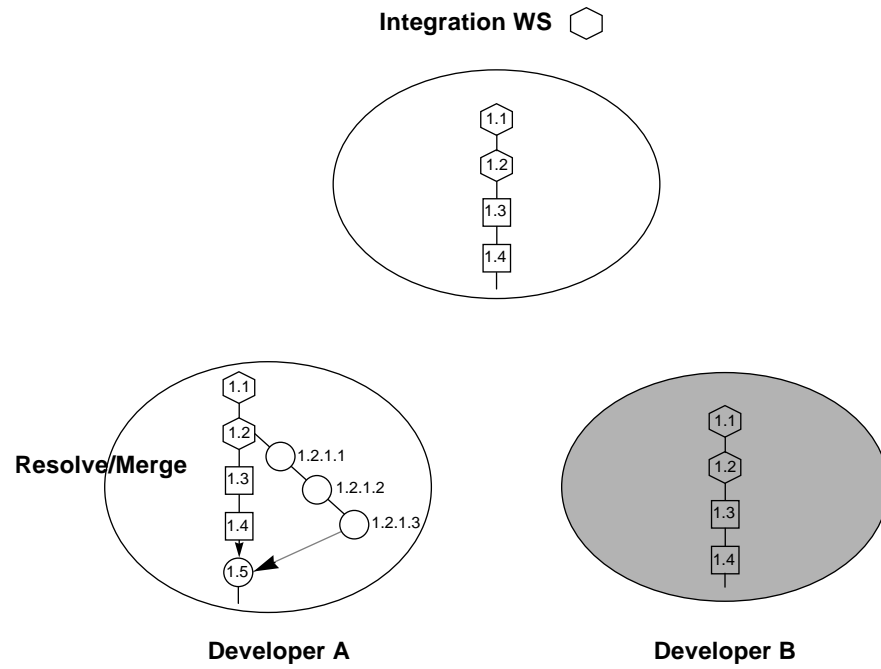
Developer A brings over the file that now contains the changes made by Developer B into his workspace from the integration workspace. The deltas created by Developer B are added into the child SCCS history file by the Configuring program.

The delta tree brought down from the parent is unchanged in the child. The new deltas created in the child are attached as an SCCS branch to the last delta that the child and parent had in common; the deltas from the child are assigned new SIDs accordingly. The deltas are renumbered using the SCCS branch numbering algorithm that derives the SID from the point at which it branches. In this case the branch is attached to SID 1.2; the first delta is renumbered to 1.2.1.1. The last delta created in the child (1.2.1.3, formerly 1.5) is still the default delta. Therefore, any new deltas that developer A creates in the child before the conflict is resolved are added to the child line of work, and not the trunk (the parent line of work).

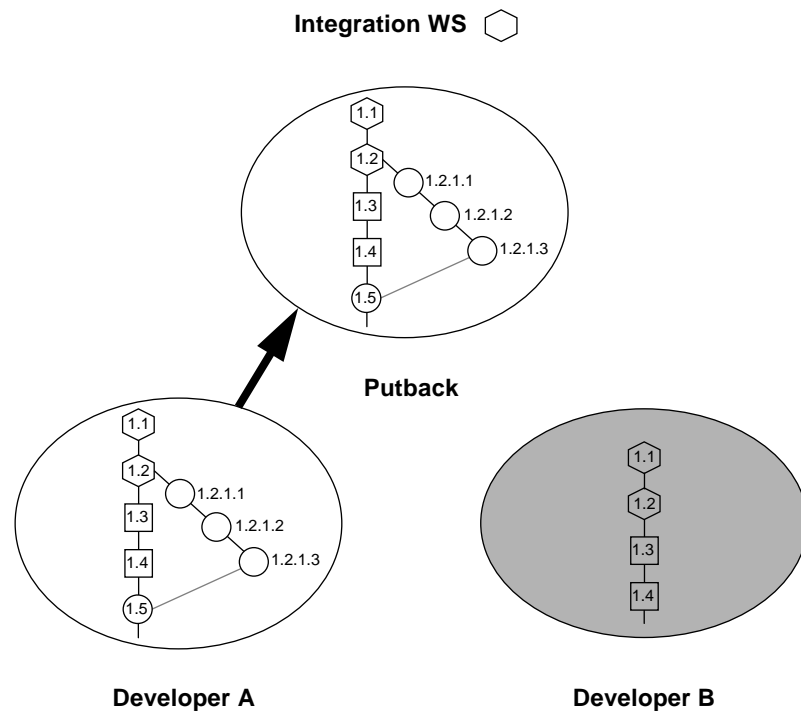


Developer A resolves the conflict in his workspace using the Configuring Resolve transaction (see Chapter 9, “Resolving Conflicts”, for details regarding conflict resolution). Developer A uses the Resolve transaction to help him decide how to merge the versions of the file represented by SIDs 1.2.1.3 and 1.4. When he commits the changes, the Resolve transaction places the newly merged contents into a new delta 1.5. Notes:

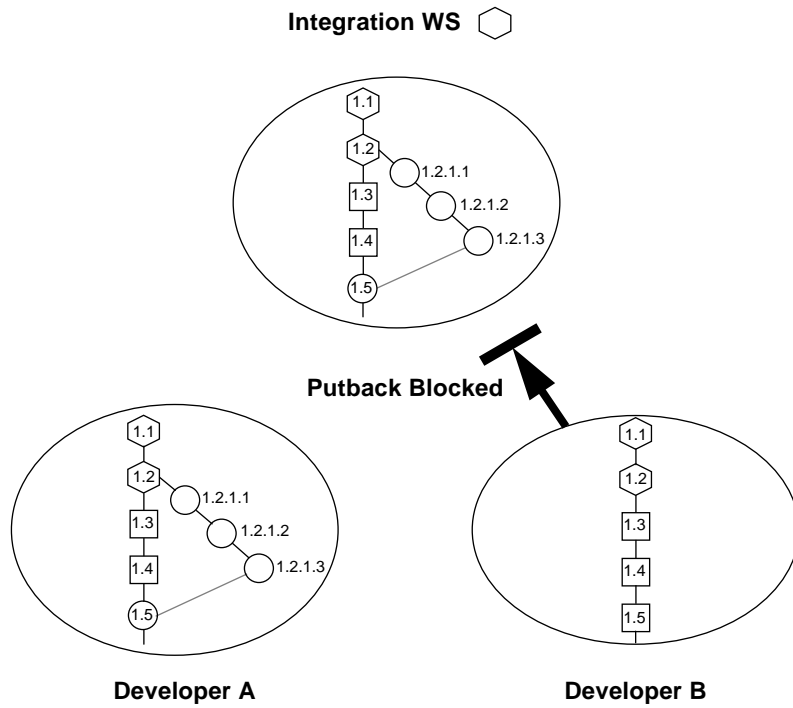
- The new delta, 1.5, is contained in a circle because it is created by developer A.
- The newly created delta is now the default location for any new work created by developer A.



With the conflict resolved, Developer A puts back the file into the integration workspace. The branch and the newly created delta are added to the SCCS history file in the integration workspace.

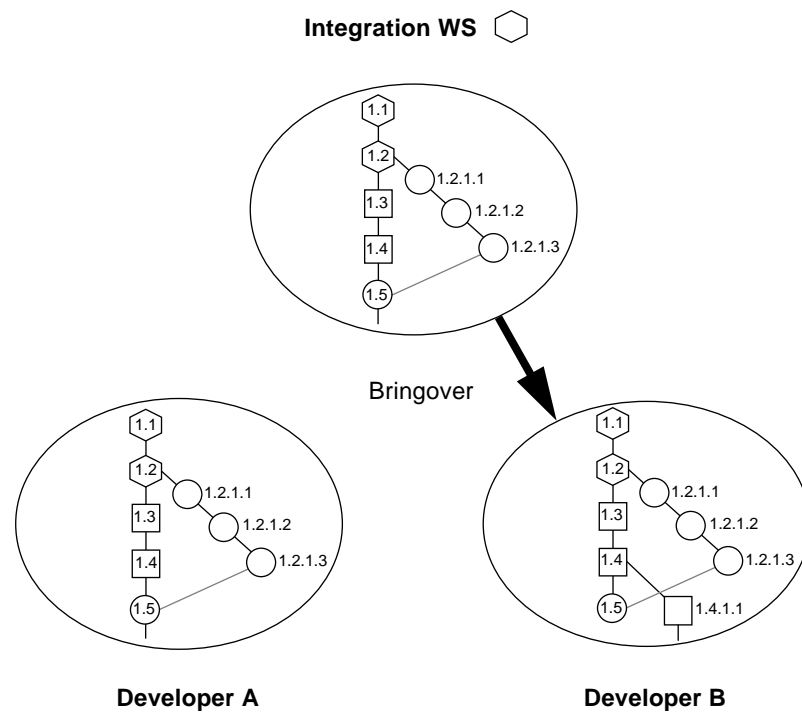


Developer B makes another change to the file in her workspace, creating delta 1.5. She attempts to put back the new work to the integration workspace, but the Putback is blocked because it conflicts with the newly merged delta 1.5 that was put back by Developer A.

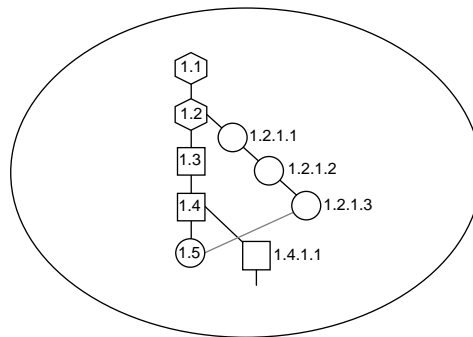


Developer B brings over the changed file into her workspace where its deltas are added into the child SCCS history file and renumbered by the Configuring program.





As in the previous case, the Configuring program appends the delta created by developer B to the last common delta on the delta tree trunk as a branch and rennumbers it appropriately. 1.5 becomes 1.4.1.1. 1.4.1.1 remains the default delta. Any new deltas created in the child before the conflict is resolved will be added to the branch.

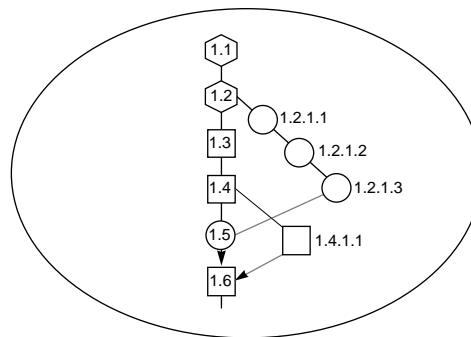


**Developer B**

Using the Configuring Resolve transaction, developer B resolves the conflict merging the differences between 1.5 and 1.4.1.1 to create the new delta 1.6.

Notes:

- The newly created merged contents are added as a new delta to the parent delta 1.6.
- The new delta is owned by the developer who owns the workspace.
- The new delta becomes the default delta, therefore, new work in the child will now be added beneath it.



**Developer B**

This chapter illustrates the basic bringover, putback, resolve cycle using an example.

<i>Creating Workspaces</i>	<i>page 144</i>
<i>Putting Back Changes</i>	<i>page 145</i>
<i>Updating a Workspace</i>	<i>page 146</i>
<i>Resolving Conflicts</i>	<i>page 147</i>

This example employs a simple case to demonstrate:

- Use of the Bringover Create transaction to create two new child workspaces from a common parent
- Use of the Putback transaction to put back changes from one of the child workspaces to the parent
- Use of the Bringover Update transaction to update the other child workspace with those changes
- How to resolve conflicts created during the Bringover Update transaction

For this example, assume two writers (Jane and Bob) are responsible for maintaining the `man` pages for some of the Configuring commands. The main `man` page workspace is named `man_pages`. The writers decide that they will each do their work in separate child workspaces and merge their work in `man_pages`. Figure 12-1 shows the file system hierarchy in the workspace `man_pages`.

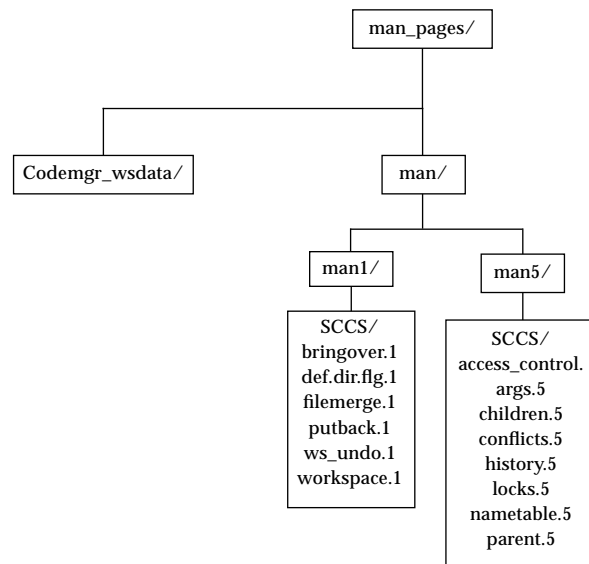


Figure 12-1 The man\_pages Workspace

## Creating Workspaces

Each writer creates his and her own child workspaces. Each child contains the same files as the parent workspace man\_pages.

1. Jane selects the man\_pages icon on the Workspace Graph pane and chooses the Bringover ⇒ Create item from the Transactions menu.

---

**Note** – For accelerator options see “Accelerators” on page 60.”

---

2. The Bringover Create Transactions window is activated. Jane enters the following information:
  - The path name of the child workspace in the To Child Workspace Directory text field
  - The directory /man in the File List pane using the Add Files point-and-click chooser window activated from the File menu .

---

The Transactions window as it is configured to create Jane's child workspace `man_pages_jane`. (Bob repeats the process to create his workspace named `man_pages_bob`.)

---

**Note** – The character “.” (representing all of the workspace) could have been specified in the file list pane instead of `man/.` Since all SCCS files are located beneath `man/.`, the two are equivalent.

---

Notes Regarding the transaction output:

- `Updating names in child workspace's name table`

The name table is a file that assists the Configuring program in tracking file names, it is used to speed up the processing of renamed files.

- `Examined files: 15`

During the initial examination phase of the Bringover transaction, the Configuring program determined that 15 files differed in the parent and child. In this case, since the child is being created and thus contains no files, all files contained in the parent are considered for the transaction.

- `Bringing over contents changes: 15`

The Configuring program has determined that 15 files should be brought over from the parent to the child workspace.

- `Contents Summary:`

Summarizes the results of the transaction.

- `15 Create`

This line indicates that 15 files were created (as opposed to updated) in the child. In the Bringover Create transaction, all transferred files fall into this category.

## *Putting Back Changes*

Bob begins work in his new workspace and makes changes to three files: `bringover.1`, `putback.1`, and `args.5`. He decides that these changes are important and that Jane should have access to them. He uses the Putback transaction to copy the changes back to the common parent workspace `man_pages`.

1. Bob selects the `man_pages_bob` icon on the Workspace Graph pane and chooses the Putback item from the Transactions menu.

---

**Note** – For accelerator options see “Accelerators” on page 60.”

---

2. When the Putback window is activated, Bob chooses the Preview option. By choosing this option, the transaction proceeds without actually copying files. Bob is able to view the output of the transaction without actually altering files; by using this option he is able to confirm that the transaction will proceed the way he expects.

Note that the Configuring program automatically loads the directory `/man` into the File List pane. This is the directory that Bob specified when he created `man_pages_bob`; the value was saved in the workspace `Codemgr_wsdata/args` file. He could change the contents of the File List pane using the items in the File and Edit menus directly below the pane.

## Updating a Workspace

Jane makes changes to the files `putback.1` and `locks.5` in `man_pages_jane`. Before she attempts to put the changes back to the `man_pages` workspace, she wants to update her workspace with the changes that Bob has just put back to `man_pages`. She uses the Bringover Update transaction.

1. Jane selects the `man_pages_jane` icon on the Workspace Graph pane and chooses the Bringover ⇒ Update item from the Transactions menu. For accelerator options see “Accelerators” on page 60.”
2. When the Bringover Update window is activated, Jane chooses the Preview option. By choosing this option, the transaction proceeds without actually copying files. Jane is able to view the output of the transaction without actually altering files. By using this option she is able to determine which files have been changed prior to taking any real action.

The output indicates that:

- `args.5` and `bringover.1` will be updated in `man_pages_jane`
- There will be a conflict created on `putback.1`. The conflict occurs because `putback.1` is changed both in `man_pages` by Bob and in `man_pages_jane` by Jane.

- One file (`locks.5`) is changed only in `man_pages_jane`.
  - The other 11 files are unchanged.
3. None of these changes surprises Jane, so she decides to complete the transaction by reexecuting it with the Preview option deselected. After the transaction completes as expected, the Configuring program automatically presents Jane with the option to resolve the conflict created on `putback.1`.

## *Resolving Conflicts*

Jane decides that she wants to resolve the conflict now and she clicks SELECT on the Resolve now button. The Resolve transaction window is activated.

---

**Note** – If the conflict is left unresolved, the Resolve transaction can be initiated later by either double-clicking SELECT on the `man_pages_jane` icon, or by selecting the icon and choosing the Resolve item from the Transactions menu.

---

---

**Note** – If the “auto load” property is set for the Resolve window, the Merging program begins execution automatically

---

The file in conflict (`man/man1/putback.1`) is listed in the Resolve window File List pane. The file is automatically selected (surrounded by a box) so Jane clicks SELECT on the Merging button. If there had been multiple files in the list, Jane could have deselected any portion of the list. If the Auto Advance property is selected (the default), the Configuring program automatically works its way down through the list of selected files.

The Configuring program starts the Merging program and passes it the name of `putback.1`

Jane works her way through the merging process, accepting Bob’s changes from the right pane and her changes from the left. When all the differences have been resolved, she saves the changes. See Chapter 9, “Resolving Conflicts,” for more information about using Merging.

Jane can now put back her changes to the parent workspace (`man_pages`) following the same procedure that Bob used (“Putting Back Changes”).





## *Error and Warning Messages*

This chapter describes error and warning messages. Error messages are described in "Error Messages" and warning messages are described in "Warning Messages." All Configuring messages are numbered and are listed in numerical order. For each message, the meaning of the message and a possible remedy for the error are provided:

*Table 13-1* Error and Warning Message Numbers

<b>Message Type</b>	<b>Message Number</b>	<b>Page Number</b>
Operating System Error Messages	1000 - 1999	page 150
Configuring Program Error Messages	2000 - 2600	page 150
Configuring Program Warning Messages	2601 - 2632	page 165

## Error Messages

The following table describes error messages, their meaning, and a possible remedy.

Table 13-2 Configuring Error Messages

---

1000 - 1999	System Errors
	Error messages between 1000 and 1999 report errors from operating system calls made by Configuring commands. They consist of a short Configuring message and an appended system error message and number. Refer to operating system documentation for information regarding these errors.
2000	Line too long or unexpected end of file in <i>file_name</i>  Meaning: While reading the <i>file_name</i> , a line was encountered that contained too many characters for a Configuring command to buffer. The maximum line length is 1024 characters.  Remedy: Reduce the size of the long line and re-execute the command.
2001	Must specify a [child]* workspace either with the -w option or via the CODEMGR_WS environment variable  Meaning: The Configuring command could not determine the workspace on which to act. Configuring commands attempt to acquire the workspace path name in the following order: <ol style="list-style-type: none"><li>1. As specified by the command's -w option</li><li>2. As specified by the value of the environment variable CODEMGR_WS</li><li>3. The current directory, if it is hierarchically within a workspace</li></ol> *When the error is reported by Bringover and Putback the word <i>child</i> is included, when reported by Undo and Resolve it is not included.  Remedy: Specify the workspace path name using one of the methods listed above.
2002	Cannot use the -p option to reparent the child of an NSE environment  Meaning: You cannot use the -p option with the CLI <i>bringover</i> and <i>putback</i> commands to reparent a workspace that has an NSE environment as a parent.  Remedy: Use the <i>workspace reparent</i> command to reparent a workspace whose parent is an NSE environment to a Configuring workspace. <i>You cannot reparent such a workspace to another NSE environment.</i>
2003	<i>directory_name</i> is not a workspace  Meaning: The directory specified in the command is not a Configuring workspace. Configuring workspaces are distinguished by the presence of the <i>Codemgr_wsdata</i> directory in the top level directory.  Remedy: Specify a different workspace name or use the CLI <i>workspace create</i> command or GUI File ⇒ Create Workspace command to convert the directory into a workspace.

---

Table 13-2 Configuring Error Messages

---

2004	<p>Workspace <i>workspace_name</i> doesn't have a parent workspace</p> <p>Meaning: A Configuring command (Bringover or Putback) could not complete execution because a parent workspace could not be found for workspace <i>workspace_name</i>.</p> <p>Remedy: Use the CLI <code>workspace parent</code> command or the GUI Edit ⇒ Parent command to re-parent the orphaned workspace.</p>
2005	<p>Parent workspace <i>workspace_name</i> is not visible as it is not mounted on <i>machine_name</i></p> <p>Meaning: The file system that contains the parent workspace is not currently mounted on machine <i>machine_name</i>.</p> <p>Remedy: Mount the file system that contains the parent workspace and re-issue the command.</p>
2006	<p>Filename <i>file_name</i> has too many “..” path components in it</p> <p>Meaning: Relative file names specified to Configuring commands are interpreted as being relative to the root directory of the workspace. If a file name contains “..” components, it is possible for one of the “..” components to reach a directory that is hierarchically above the workspace root.</p> <p>Remedy: Specify the path name with fewer (or no) “..” path name components</p>
2007	<p>Could not get username for uid <i>uid_number</i></p> <p>Meaning: The uid could not be found in the NIS maps or in <code>/etc/passwd</code></p> <p>Remedy: Check NIS server and maps.</p>
2008	<p>No version number in file <i>file_name</i></p> <p>Meaning: When a Configuring command accesses a metadata file (a file in the <code>Codemgr_wsdata</code> directory) it checks the version number written in the file when it was created (for example, VERSION 1). The metadata file <i>file_name</i> does not contain the version string.</p> <p>Remedy: Check the integrity of <i>file_name</i>. The version string may have been removed when the file was edited. If the version string is missing, and the file is not otherwise corrupted, use the <code>workspace create</code> command to create a new workspace. Check the value of the version string for the analogous file in the new workspace and edit that string into <i>file_name</i>.</p>
2009	<p>Command <i>command_name</i> failed, <code>/bin/sh</code> killed by signal <i>signal</i></p> <p>Meaning: A Configuring command attempted to execute <i>command_name</i> and was unable to because the shell was killed by <i>signal</i>.</p> <p>Remedy: Re-execute the Configuring command.</p>
2010	<p>Command <i>command_name</i> failed, could not execute the shell, <code>/bin/sh</code></p> <p>Meaning: A Configuring command could not start a shell. This indicates that some system resource, such as swap space or memory was insufficient.</p>

---

Table 13-2 Configuring Error Messages

	Remedy: Check system resources.
2011	<p>Command <i>command_name</i> killed by signal <i>signal</i></p> <p>Meaning: A command started by a Configuring command received signal <i>signal</i>.</p> <p>Remedy: Re-execute the command. If the error re-occurs, refer to the Solaris documentation for information about the signal.</p>
2012	<p>Command <i>command_name</i> exited with status <i>status</i></p> <p>Meaning: Configuring expects commands it executes to exit with a status of zero indicating successful completion. Configuring considers it an error if a command exits with a non-zero status.</p> <p>Remedy: Refer to the documentation for <i>command_name</i> to determine the meaning of <i>status</i>.</p>
2013	<p>LP <i>FLP_name</i> does not exist in the parent or child workspace</p> <p>Meaning: The file list program (FLP) <i>FLP_name</i> specified for the Bringover or Putback transaction, could not be found in either the parent or child workspace</p> <p>Remedy: Check the path name of the intended FLP and re-execute the transaction.</p>
2014	<p>Could not execute <i>program_name</i></p> <p>Meaning: A Configuring command attempted to execute another program and was unable to do so.</p> <p>Remedy: Ensure that your installation is correct. Ensure that the program is in your search path and that its permissions are set correctly.</p>
2015	<p>Workspace <i>workspace_name</i> already exists</p> <p>Meaning: An attempt was made to create a workspace that already exists.</p> <p>Remedy: Re-execute the command using a different workspace name.</p>
2016	<p>Workspace <i>name</i> does not exist</p> <p>Meaning: The workspace <i>name</i> specified as an argument for a Configuring command could not be found.</p> <p>Remedy: Ensure that the path name was specified correctly.</p>
2017	<p>Can't open file <i>file_name</i> so can't get comments for check in</p> <p>Meaning: Configuring stored checkin comments in a temporary file and was unable to open that file to read the comments.</p> <p>Remedy: Check file permissions and other file system problems that would prohibit opening the file.</p>
2018	<p>Can't reparent a workspace to itself</p> <p>Meaning: An attempt was made (either as part of a transaction, or by using an explicit reparent command) to make a workspace its own new parent.</p>

Table 13-2 Configuring Error Messages

	Remedy: Re-execute the command, specifying a different parent.
2019	<p>Internal error: unknown locktype <i>lock</i></p> <p>Meaning: The workspace lock file (Codemgr_wsdata/locks) is corrupted. An unknown lock value was found.</p> <p>Remedy: Edit the lock file to repair the damage. For more information, see the locks(5) manual page or “Ensuring Consistency through Workspace Locking.”</p>
2020	<p>You must specify a workspace name</p> <p>Meaning: The Configuring command could not determine the workspace on which to act. Configuring commands attempt to acquire the workspace path name in the following order:</p> <ol style="list-style-type: none"> <li>1. As specified by the command's -w option</li> <li>2. As specified by the value of the environment variable CODEMGR_WS</li> <li>3. The current directory, if it is hierarchically within a workspace</li> </ol> <p>Remedy: Specify the workspace path name using one of the methods listed above.</p>
2021	<p>Cannot obtain a <i>type</i> lock in workspace <i>workspace_name</i> because it has the following locks: Command: <i>command (pid)</i>, user: <i>user</i>, machine: <i>machine</i>, time: <i>time</i></p> <p>Meaning: To ensure consistency, Configuring interworkspace commands lock workspaces while they read and write data in them. The command you issued could not obtain a lock because the workspace is already locked. While Configuring is reading and examining files in the parent workspace during a Bringover transaction, it obtains a <i>read-lock</i> for that workspace. When it is manipulating files in the child workspace, it obtains a <i>write-lock</i>. Read-locks may be obtained concurrently by multiple Configuring commands that read files in the workspace. No commands may write to a workspace while any read-locks are in force. Only a single write-lock can be in force at any time; no Configuring command may write to a workspace while a write-lock is in force. Lock status is controlled by the Codemgr_wsdata/locks file in each workspace.</p> <p>Remedy: If the system is running normally, wait until the command that is locking the workspace releases its lock. If the workspace is stuck in a locked state (for example, the system crashed while a command had a lock in force), use the GUI Props ⇒ Workspace ⇒ Locks window, or the workspace locks command to remove the lock.</p>
2022	<p>Invalid subcommand - <i>command_name</i></p> <p>Meaning: An attempt was made to obtain help on a subcommand of the resolve, workspace or codemgr command and the name of a non-existent subcommand was specified.</p> <p>Remedy: For the list of valid subcommands for each command, type the command and specify the help subcommand.</p>
2023	Not used.

Table 13-2 Configuring Error Messages

2024	<p>File <i>file_name</i> has no deltas</p> <p>Meaning: The SCCS history file <i>file_name</i> contains no deltas, therefore it cannot be processed.</p> <p>Remedy: Perhaps the history file was mistakenly overwritten.</p>
2025	<p>Could not find the <i>command_name</i> command. Executable does not exist: <i>name</i> Also could not find the <i>name</i> command in PATH <i>PATH_contents</i></p> <p>Meaning: A Configuring command attempted to execute another program and was not able to find it.</p> <p>Remedy: Ensure that your installation is correct. Include the directory that contains the missing program.</p>
2026	<p>Unknown SCCS control character (<i>char</i>) in file <i>file_name</i> at line <i>line_number</i></p> <p>Meaning: A Configuring command expected <i>file_name</i> to be an SCCS history file; based on the character it encountered, it is either not a history file, or it has been corrupted.</p> <p>Refer to the Solaris SCCS documentation regarding SCCS history file format.</p>
2027	<p>Corrupted file - <i>file_name</i>, line <i>line_number</i></p> <p>Meaning: A Configuring command was unable to read a workspace metadata file (a file in the Codemgr_wsdata directory). Illegal characters were found in line <i>line_number</i>.</p> <p>Remedy: Check and repair the file. All Configuring metadata files are ASCII text files and can be edited. See the <i>file_name</i>(5) manual page or Chapter 7, "TeamWare Configuring Workspace" for more information on its format.</p>
2028	<p>Could not find the <i>command_name</i> command in PATH <i>path_name</i></p> <p>Meaning: A Configuring command attempted to execute another program and was not able to find it.</p> <p>Remedy: Ensure that your installation is correct. Include the directory that contains the missing program.</p>
2029	<p>The file has unresolved conflicts. Run 'edit m' and search for ^&lt;&lt;&lt;&lt;&lt;&lt;</p> <p>Meaning: This error is issued by the <i>resolve</i> command. An attempt was made to save the file while it still contained unresolved conflicts.</p> <p>Remedy: Use the <i>edit m</i> subcommand (edit the merged result) to resolve the conflicts and then save the file. Conflicts are marked with ^&lt;&lt;&lt;&lt;&lt;&lt;.</p>
2030	<p>No file with number <i>file_number</i></p> <p>Meaning: The <i>resolve</i> command creates a numbered list of files that contain conflicts. The <i>file_number</i> chosen does not exist in this list.</p> <p>Remedy: Use the <i>list</i> subcommand to list the files and determine the correct number of the file you wish to specify.</p>

Table 13-2 Configuring Error Messages

---

2031	<p>Can't find home directory so can't write to file <i>file_name</i></p> <p>Meaning: A Configuring command was unable to find the user's home directory and cannot locate file <i>file_name</i>. This usually indicates a problem with NIS maps.</p> <p>Remedy: Check NIS server and appropriate NIS maps.</p>
2032	<p>Can't parse line in file <i>file_name: line</i></p> <p>Meaning: Upon startup, the <code>resolve</code> command reads the <code>~/ .codemgr_resrc</code> file to obtain user defined properties. The line <i>line</i> could not be interpreted correctly by the program.</p> <p>Remedy: Correct the file <code>~/ .codemgr_resrc</code> file so that it includes only valid entries. For information regarding these entries, see the <code>resolve(1)</code> manual page.</p>
2033	<p>Must specify a directory list either as arguments or via the <code>CODEMGR_WSPATH</code> variable</p> <p>Meaning: This message is reported by the <code>workspace list</code> command when a directory (or list of directories) was not specified in a way in which <code>workspace list</code> can search for workspaces to list. Directories can be specified as the standard argument to the command, or by defining the <code>CODEMGR_WSPATH</code> variable to contain the path name of a directory.</p> <p>Remedy: Re-execute the command specifying a directory, or set the <code>CODEMGR_WSPATH</code> directory to contain a directory path.</p>
2034	<p>internal error: Access control operation <i>operation_name</i> does not have a Ibuilt-in default</p> <p>Meaning: A Configuring command attempted to verify access permission for a workspace operation (for example: <code>bringover-from</code>, <code>putback-to</code>, <code>reparent-to</code>). An internal consistency check failed.</p> <p>Remedy: Contact your local service representative.</p>
2035	<p>Access control file does not exist</p> <p>Meaning: A Configuring command attempted to verify access permission for a workspace operation (for example: <code>bringover-from</code>, <code>putback-to</code>, <code>reparent-to</code>). The access control file (<code>Codemgr_wsdata/access_control</code>) in the affected workspace was not found.</p> <p>Remedy: If the access control file has been deleted from the workspace, copy a new one from another workspace and edit it so that the access permissions are correct. If no other workspaces are available, create a new workspace using the CLI <code>workspace create</code> or the GUI <code>File ⇒ Create Workspace</code> commands and copy the file from the newly created workspace. For more information refer to the <code>access_control(5)</code> manual page or "Controlling Access to Workspaces."</p>
2036	<p>Cannot specify common ancestor file; there is no common ancestor delta</p>

---

Table 13-2 Configuring Error Messages

	<p>Meaning: The ancestor (a) was specified as an argument to a <code>resolve</code> subcommand (<code>diff</code>, <code>edit</code>, <code>more</code>). The files that are being resolved do not have an ancestor in common. This occurs most commonly in cases where files with the same name are created concurrently in both the child and the parent. They have the same name but are not descended from a common ancestor. For more information about ancestors and their role in resolving conflicts, see Chapter 9, “Resolving Conflicts.”</p> <p>Remedy: Proceed with the conflict resolution process without specifying the ancestor (a) as an argument to the <code>diff</code>, <code>edit</code> and <code>more</code> subcommands.</p>
2037	<p>Invalid argument - <i>character</i></p> <p>Meaning: An invalid argument was specified to one of the <code>resolve</code> subcommands. The command expected one of the following characters: a (ancestor), c (child), p (parent), m (merged result).</p> <p>Remedy: Specify one of the valid arguments: a, c, p, m. See the <code>resolve(1)</code> manual page for more information.</p>
2038	<p>Parent workspace is an NSE environment. Use the <code>nseputback</code> command</p> <p>Meaning: The “parent” in a <code>putback</code> transaction is an NSE environment and not a Configuring workspace.</p> <p>Remedy: Use the <code>nseputback</code> command to <code>putback</code> changes from the workspace to the environment.</p>
2039	<p>File <i>file_name</i> is probably not an s-file on line <i>line_number</i> expected ^A, but got <i>char</i></p> <p>Meaning: A Configuring command expected <i>file_name</i> to be an SCCS history file; based on the format it is either not a history file, or it was corrupted.</p> <p>Remedy: Refer to Solaris SCCS documentation regarding SCCS history file format.</p>
2040	<p>File <i>file_name</i> has not been merged. Use the <code>twmerge</code> subcommand first or the <code>filemerge</code> subcommand</p> <p>Meaning: An attempt was made to commit (save) a file that had not yet been merged.</p> <p>Remedy: Merge the file using either the <code>twmerge</code> subcommand, or the <code>filemerge</code> subcommand (which executes the Merging GUI merge tool). For more information about the <code>resolve</code> command see the <code>resolve(1)</code> manual pages.</p>
2041	<p><i>path_name</i> is not a workspace or a directory</p> <p>Meaning: The string <i>path_name</i> specified in the Bringover Create transaction is not a Configuring workspace or a directory.</p> <p>Remedy: Specify a different workspace or directory name.</p>
2042	<p>Can't create ToolTalk message, error = <i>TT_error_code</i></p>



Table 13-2 Configuring Error Messages

---

	<p>Meaning: The <code>resolve</code> command communicates with the Merging program via the ToolTalk service. ToolTalk is an interapplication communication service distributed with the Solaris OpenWindows windowing system. In this case the <code>resolve</code> command called a ToolTalk routine to create a ToolTalk message for Merging. The ToolTalk routine could not create the message and passed back <code>TT_error_code</code>.</p> <p>Remedy: Refer to the OpenWindows ToolTalk documentation for information about the error.</p>
2043	<p>SCCS file <code>file_name</code> is corrupted</p> <p>Meaning: The SCCS <code>admin -h</code> command reports that the newly computed check-sum does not compare with the one stored in the first line of the file.</p> <p>Remedy: See the Solaris SCCS documentation for more information.</p>
2044	<p>Unable to create a temporary name from template <code>temp_file_name</code></p> <p>Meaning: A Configuring command was unable to create a temporary file for its use. This is a Configuring internal error.</p> <p>Remedy: Check for any system-level reasons why the command could not write this file (for example, file permission restrictions or incorrect command ownership).</p>
2045	<p>Fprintf of <code>file_name</code> failed</p> <p>Meaning: A command was unable to write to the file <code>file_name</code>.</p> <p>Remedy: Check file permissions and other such file system problems that would prohibit writing in the file system.</p>
2046	<p>Version mismatch in file <code>file_name</code>, expected version <code>expected_number</code>, but found <code>actual_number</code></p> <p>Meaning: Each Configuring metadata file (<code>Codemgr_wsdata/*</code>) contains a string that includes a version number (for release 1.0 the version number string is <code>VERSION 1</code>). As a consistency check, when Configuring commands read and write to these files, they check to determine whether the file contains the version that the command expects. In this case the command expected to find <code>expected_number</code> but found <code>actual_number</code> instead. This may indicate that old binaries are being used with new metadata files and could cause the file to be corrupted.</p> <p>Remedy: Make sure that the most current versions of the Configuring binaries are being accessed.</p>
2047	<p>Do not know how to convert file <code>file_name</code> from version <code>found_version_number</code> to <code>current_version_number</code></p> <p>Meaning: Each Configuring file (<code>Codemgr_wsdata/*</code>) contains a string that includes a version number (for release 1.0 the version number string is <code>VERSION 1</code>). As a consistency check, when Configuring commands read and write to these files, they check to determine whether the file contains the version that the command expects. It is anticipated that when new versions of Configuring binaries and metadata files are released, the formats of some of these files may change. Commands contain code to make this conversion. A command found a metadata file with a version number earlier than 1.</p>

---

Table 13-2 Configuring Error Messages

	<p>Remedy: Since this is the first release of Configuring, the version string in the metadata file must have been inadvertently changed during editing. Check the file and make sure that the first line reads "VERSION 1".</p>
2048	<p>Must specify at least one file, directory or -f argument to a bringover that creates a child workspace</p> <p>Meaning: The command-line for a Bringover transaction was not constructed properly. An argument that specifies at least one file, directory or FLP must be included. If this argument is omitted, Configuring attempts to take the arguments from the workspace's Codemgr_wsdata/args file.</p> <p>Remedy: Reenter the command and ensure that you've included the correct number of arguments.</p>
2049	<p>Could not determine where <i>file_name</i> is mounted from</p> <p>Meaning: Configuring commands convert path names of NFS mounted directories to the <i>machine_name:path_name</i> format to do much of their work. This message indicates that the mount entry that contains <i>file_name</i> in /etc/mntab (Solaris 1.x) or /etc/mntab (Solaris 2.x) is no longer present.</p> <p>Remedy: Remount the file system that contains <i>file_name</i>.</p>
2050	<p>Could not determine the absolute pathname for <i>file_name</i></p> <p>Meaning: A Configuring command was unable to read a directory. This indicates some corruption in the file system; for example, incorrect directory permissions.</p> <p>Remedy: Check the file system, especially directory and file permissions in the path of <i>file_name</i>.</p>
2051	<p>Can't rename to <i>file_name</i>; it exists</p> <p>Meaning: During a Bringover, Putback or Undo transaction a file was found that was renamed in the source workspace to a name already in use in the destination workspace.</p> <p>Remedy: Change the name in one of the directories.</p>
2052	<p>Corrupted file - <i>file_name</i>, text after BEGIN, line <i>number</i>. Can't send notification</p> <p>Meaning: A Configuring command encountered an error when reading the workspace notification file Codemgr_wsdata/notification. The BEGIN statement that delimits the list of files/directories for which notification is requested must be the only text on the line, other text was encountered. The Configuring command cannot correctly parse the request; if the file contains a notification request, it cannot be sent.</p> <p>Remedy: Edit the notification file and enter the appropriate BEGIN statement. See the notification(5) man page or "How to Notify Users of Changes to Workspaces" for more information on its format.</p>
2053	<p>Corrupted file - <i>file_name</i>, text after END, line <i>number</i>. Can't send notification</p> <p>Meaning: A Configuring command encountered an error when reading the workspace notification file Codemgr_wsdata/notification. The END statement that delimits the list of files/directories for which notification is requested must be the only text on the line, other text was encountered. The Configuring command cannot correctly parse the request; if the file contains a notification request, it cannot be sent.</p>

Table 13-2 Configuring Error Messages

	Remedy: Edit the notification file and enter the appropriate END statement. See the <code>notification(5)</code> manual page or “How to Notify Users of Changes to Workspaces” for more information on its format.
2054	<p>Corrupted file - <i>file_name</i>, missing BEGIN, line <i>number</i>. Can't send notification</p> <p>Meaning: A Configuring command encountered an error when reading the workspace notification file <code>Codemgr_wsdata/notification</code>. The BEGIN statement that delimits the list of files/directories for which notification is requested, is missing. The Configuring command cannot correctly parse the request; if the file contains a notification request, it cannot be sent.</p> <p>Remedy: Edit the notification file and enter the appropriate BEGIN statement. See the <code>notification(5)</code> man page or Section , “How to Notify Users of Changes to Workspaces” for more information on its format.</p>
2055	<p>File <i>file_name</i> has incomplete delta table</p> <p>Meaning: The delta table in the SCCS history file <i>file_name</i> is incomplete. This indicates that the file was corrupted.</p> <p>Remedy: Fix the file, or copy in a new version.</p>
2056	<p>Badly formatted line in <i>file_name</i>: <i>line_number</i></p> <p>Meaning: A Configuring command was reading a temporary log file left over from an aborted Bringover or Putback operation and encountered a malformed line. This indicates that the file was corrupted.</p> <p>Remedy: Execute the <code>workspace updatenames</code> command to rebuild the nametable and then re-execute the command.</p>
2057	<p>Zero-length SCCS file, <i>file_name</i></p> <p>Meaning: An SCCS history file was encountered that contained no data.</p> <p>Remedy: Remove the SCCS history file.</p>
2058	<p>Can't get a version of the child file until it is checked in</p> <p>Meaning: During a Resolve transaction a file was encountered that is not checked in to SCCS. Files must be checked in before conflicts can be resolved.</p> <p>Remedy: Check the file in and re-start the transaction.</p>
2059	<p>Name history serial number <i>number</i> out of order in file <i>file_name</i></p> <p>Meaning: Rename information in the SCCS history file <i>file_name</i> is corrupted. The name history records in this SCCS file are not in numerically descending order.</p> <p>Remedy: Reorder the name history records, or copy in a new version of the file using the Bringover or Putback transaction.</p>
2060	<p>Delta serial number <i>number</i> out of order in file <i>file_name</i></p>

Table 13-2 Configuring Error Messages

	<p>Meaning: Delta numbers are not in numerically descending order in the SCCS history file <i>file_name</i>. This indicates that the file is corrupted.</p> <p>Remedy: Reorder the delta numbers, or copy in a new version of the file using the Bringover or Putback transaction.</p>
2061	<p>Must have DISPLAY environment variable set to invoke twmerge or filemerge</p> <p>Meaning: The DISPLAY variable is automatically set by OpenWindows when it begins execution. Your machine must be running OpenWindows to use the Merging program.</p> <p>Remedy: Ensure that OpenWindows is executing properly; if it is, reset the DISPLAY variable.</p>
2062	<p>Can't resolve file <i>file_name</i> because it is writable</p> <p>Meaning: The file <i>file_name</i> is not checked out from SCCS but its file permissions indicate that it is writable. Resolving this conflict will result in writing to a file that is not checked out.</p> <p>Remedy: Reconcile the file permissions (for example, check the file out and then check it back in) and then re-execute the Resolve transaction.</p>
2063	<p>Cannot create workspace <i>name</i> because it would be nested within workspace <i>name</i></p> <p>Meaning: An attempt was made to create a workspace hierarchically beneath an existing workspace.</p> <p>Remedy: Create the new workspace hierarchically outside of any existing workspaces.</p>
2064	<p>Cannot delete a workspace that is a symbolic link. Run workspace delete <i>workspace_name</i></p> <p>Meaning: Configuring commands will not delete directories or files that are symbolic links. You must delete the physical copy of the file. The appropriate command line is provided.</p> <p>Remedy: Use the workspace delete command to delete <i>workspace_name</i>.</p>
2065	<p>This error message may be issued in any of the following form:</p> <p>User <i>user_name</i> does not have access to bringover from workspace <i>workspace_name</i>          User <i>user_name</i> does not have access to bringover to workspace <i>workspace_name</i>          User <i>user_name</i> does not have access to putback from workspace <i>workspace_name</i>          User <i>user_name</i> does not have access to putback to workspace <i>workspace_name</i>          User <i>user_name</i> does not have access to undo workspace <i>workspace_name</i>          User <i>user_name</i> does not have access to delete workspace <i>workspace_name</i>          User <i>user_name</i> does not have access to move workspace <i>workspace_name</i>          User <i>user_name</i> does not have access to change the parent of workspace <i>workspace_name</i>          User <i>user_name</i> does not have access to change the parent to workspace <i>workspace_name</i></p> <p>Meaning: The user <i>user_name</i> attempted an operation that affected the workspace <i>workspace_name</i>; access permissions in <i>workspace_name</i> do not permit <i>user_name</i> access to execute that operation.</p>

Table 13-2 Configuring Error Messages

	<p>Remedy: The file <i>workspace_name</i>/Codemgr_wsdata/access_control is a text file that specifies access permissions for various workspace operations. The owner of the workspace must change the permissions to include <i>user_name</i> in order for the operation to proceed. Permissions can be changed using the Workspace item in the GUI Props menu or by editing the access_control file directly. See the access_control(5) man page or Chapter , “TeamWare Configuring Workspace” of this manual for more information.</p>
2066	<p>Corrupted file - <i>file_name</i>, whitespace in pathname, line <i>line_number</i>. Can't send notification</p> <p>Meaning: A Configuring command encountered an error when reading the workspace notification file Codemgr_wsdata/notification. A whitespace character was encountered in a line where a single path name was expected.</p> <p>Remedy: Edit the Codemgr_wsdata/notification file to remove the whitespace characters from the line. See the notification(5) man page or “How to Notify Users of Changes to Workspaces” for more information on its format.</p>
2067	<p>Corrupted file - <i>file_name</i>, missing notification event, line <i>line_number</i>. Can't send notification</p> <p>Meaning: A Configuring command encountered an error when reading the workspace notification file Codemgr_wsdata/notification. The Configuring event (for example, bringover-to) was not specified.</p> <p>Remedy: Edit the Codemgr_wsdata/notification file to add the correct event. See “How to Notify Users of Changes to Workspaces” for a list of valid events.</p>
2068	Not used
2069	Not used
2070	Not used
2071	Not used
2072	Not used

Table 13-2 Configuring Error Messages

2073	Not used
2074	<p>Workspace <i>workspace_name</i> has no locks</p> <p>Meaning: An attempt was made to remove locks from a workspace that had no active locks.</p> <p>Not applicable</p>
2075	<p>Lock <i>lock_name</i> does not exist for workspace <i>workspace_name</i></p> <p>Meaning: While using the workspace locks <code>-r</code> command, a lock number was specified that is out of range of the lock list.</p> <p>Remedy: Check the lock numbers for the workspace using the <code>workspace locks</code> command and enter a valid number.</p>
2076	<p>Internal error: Cannot find the directory in which command <i>command_name</i> is located because <code>avo_find_dir_init()</code> has not been called</p> <p>Meaning: This is an internal error.</p> <p>Remedy: Please contact your local service representative.</p>
2077	<p><i>number</i> is not a valid number</p> <p>Meaning: While using the <code>resolve</code> command, a number was referenced that is outside of the listed values.</p> <p>Remedy: List the values to determine the valid number for your selection.</p>
2078	<p>Cannot access workspace <i>workspace_name</i></p> <p>Meaning: File permissions for <i>workspace_name</i> prohibit access by the <code>Configuring</code> command.</p> <p>Remedy: Default permissions for workspace directories are <code>777</code>.</p>
2079	<p>Could not parse name history for file <i>file_name</i>, contains: <i>text</i></p> <p>Meaning: There is a format error in the name history record in the SCCS history file <i>file_name</i>. The troublesome text is displayed.</p> <p>Remedy: If possible, fix the record; otherwise copy a new version of the file using the <code>Bringover</code> or <code>Putback</code> transaction.</p>
2080	<p>Could not remove or rename backup directory <i>directory_name</i></p>

Table 13-2 Configuring Error Messages

	<p>Meaning: Configuring attempted to clear the backup area <i>directory_name</i> so that it could backup a new transaction. Configuring was not able to delete or rename the directory out of the way. The most likely cause is that file permissions have been changed for the directory.</p> <p>Remedy: Check directory permissions for <i>directory_name</i>. Default Configuring permissions for this directory are 777.</p>
2081	<p>build_workspace_list: <i>path_name</i> does not start with a /</p> <p>The Configuring GUI program was expecting a fully qualified path name to be returned from a subprocess. This is an internal error.</p> <p>Please contact your local service representative.</p>
2082	<p>Workspace <i>workspace_name</i>'s parent does not exist in the filesystem</p> <p>Meaning: The parent workspace is not mounted or visible on this machine.</p> <p>Remedy: Mount the parent workspace on the executing machine.</p>
2083	<p>Workspace <i>workspace_name</i>'s child does not exist in filesystem</p> <p>Meaning: The child workspace is not mounted or visible on this machine.</p> <p>Remedy: Mount the child workspace on the executing machine.</p>
2084	<p>codemgrtool: internal error in args_strlist_from_wsname(): NULL args_list</p> <p>Meaning: Internal error.</p> <p>Remedy: Contact your local service representative.</p>
2085	<p>codemgrtool: internal error in undo_strlist_from_wsname() : NULL undo_list</p> <p>Meaning: Internal error.</p> <p>Remedy: Contact your local service representative.</p>
2086	<p>codemgrtool: <i>path_name</i> doesn't start with a /</p> <p>Meaning: Internal error.</p> <p>Remedy: Please contact your local service representative.</p>
2087	Not used.

Table 13-2 Configuring Error Messages

2088	<p>Nametable in workspace <i>workspace_name</i> cannot be read because the following SCCS files have identical root deltas</p> <p><i>file_name</i> <i>file_name</i></p> <p>Run the following command and then re-execute the <i>command_name</i> command:</p> <p><i>path_name</i>/workspace updatenames <i>workspace_name</i></p>
	<p>Meaning: An SCCS history file was copied within a workspace using the <code>cp</code> command. As a result, the two files contain the identical root delta. Configuring uses the root delta to distinguish between files. The workspace updatenames command enables Configuring to distinguish between the files.</p> <p>Remedy: Execute the workspace updatenames command and then re-execute the command that spawned the error.</p>
2089	<p>Cannot move workspace <i>workspace_name</i> Because it is a symlink to <i>directory_name</i>. Use a workspace name that is not a symlink.</p>
	<p>Meaning: Configuring commands will not move directories or files that are symbolic links.</p> <p>Remedy: Move the workspace to a name that is not a symlink.</p>
2090	<p>Nametable in workspace <i>workspace_name</i> not written because the following SCCS files have identical root deltas</p> <p><i>file_name</i> <i>file_name</i></p> <p>Run the following command and then re-execute the <i>command_name</i> command:</p> <p><i>path_name</i>/workspace updatenames <i>workspace_name</i></p>
	<p>Meaning: An SCCS history file was copied within a workspace using the <code>cp</code> command. As a result the two files contain the identical root delta. Configuring uses the root delta to distinguish between files. The workspace updatenames command enables Configuring to distinguish between the files.</p> <p>Remedy: Execute the workspace updatenames command and then re-execute the command that spawned the error.</p>
2091	<p>Internal error: hash table missing entry</p>
	<p>Meaning: Internal error.</p> <p>Remedy: Contact your local service representative.</p>
2092	<p>An SCCS file (A) was copied (to file B). The original SCCS file (A) cannot be found. Run the following command and then re-execute the <i>command_name</i> command:</p> <p><i>path_name</i>/workspace updatenames <i>workspace_name</i></p>



Table 13-2 Configuring Error Messages

---

	<p>Meaning: An SCCS history file was copied within a workspace using the <code>cp</code> command. The original file (A) was subsequently renamed or removed from the workspace. Configuring is unable to determine whether the files has been renamed (and to what name) or removed from the workspace. The <code>workspace updatenames</code> command interactively displays the possible names to which the file could have been renamed, and asks you to determine the file's current state: its new name, or its absence from the workspace. Configuring can then correctly propagate the changes throughout the workspace hierarchy.</p> <p>Remedy: Execute the <code>workspace updatenames</code> command and then re-execute the command that spawned the error.</p>
2093	<p>Internal error: SmIDs not equivalent</p> <hr/> <p>Meaning: Internal error.</p> <p>Remedy: Contact your local service representative.</p>
2094	<p>Internal error: SmID not found</p> <hr/> <p>Meaning: Internal error.</p> <p>Remedy: Contact your local service representative.</p>
2095 - 2499	<p>Not used</p> <hr/>
2500 - 2600	<p>Internal errors</p> <hr/> <p>Meaning: Error numbers 2500 through 2600 Configuring programs internal errors. These errors indicate problems that users cannot correct. If you encounter these errors, contact your local service representative.</p> <hr/>

## Warnings Messages

Table 13-3 Configuring Warning Messages

---

2601	<p>Could not remove backup directory <i>old_dir_name</i>, so it was renamed to <i>new_dir_name</i></p> <p>Meaning: Configuring attempted to clear the backup area <i>old_dir_name</i> so that it could backup a new transaction. Configuring was not able to clear the backup directory by deleting it, but it was able to rename it out of the way to the name <i>new_dir_name</i>. The most likely cause is that file permissions were changed for the directory.</p> <p>Remedy: Check directory permissions for <i>old_dir_name</i>. Default Configuring permissions for this directory are 777. Delete the contents of <i>new_dir_name</i>.</p> <hr/>
------	---

Table 13-3 Configuring Warning Messages

2602	<p>File <i>file_name</i> is not under SCCS in either workspace - ignored</p> <p>Meaning: Configuring could not find an SCCS history file in either workspace for <i>file_name</i>.</p> <p>Remedy: The file name was probably entered incorrectly, re-execute the command.</p>
2603	<p>Zero length filename - ignored</p> <p>Meaning: A file name specified as an argument on the command-line (or in the Codemgr_wsdata/args file) contained no characters (“”).</p> <p>Remedy: Re-execute the command and re-specify the file name argument. If the problem persists, check the arguments listed in the args file.</p>
2604	<p>Filename <i>file_name</i> has whitespace characters in it - ignored</p> <p>Meaning: A file name specified as an argument on the command-line (or in the Codemgr_wsdata/args file) contained whitespace characters. Configuring commands do not accept file names that contain whitespace characters.</p> <p>Remedy: Re-execute the command and re-specify the file name argument. If the problem persists, check the arguments listed in the args file.</p>
2605	Not used
2606	<p>File <i>file_name</i> not brought over because it is a <i>file_type</i> in workspace <i>workspace_name</i> and a <i>file_type</i> in workspace <i>workspace_name</i></p> <p>Meaning: A file name has a different file type (regular file vs. directory vs. symbolic link) in the parent and child workspaces.</p> <p>Remedy: Take whatever action is appropriate to make the listed files the same type, or change one of the names.</p>
2607	Not used
2608	<p>Workspace <i>child_ws_name</i> is a child of <i>parent_ws_name</i>. Could not update its parent file</p> <p>Meaning: During a workspace delete or workspace move operation involving <i>child_ws_name</i>, the command found that the children file in the workspace's parent (<i>parent_ws_name</i>) did not contain an entry specifying <i>child_ws_name</i> as a child of that parent.</p> <p>Remedy: Advisory only: the command corrects the discrepancy, however, this could indicate that the parent's children file was corrupted.</p>

Table 13-3 Configuring Warning Messages

---

2609	Not used
2610	<p><i>directory_name</i> is not a workspace</p> <p>Meaning: The directory specified in the command is not a Configuring workspace. Configuring workspaces are distinguished by the presence of the <code>Codemgr_wsdata</code> directory in the top level directory.</p> <p>Remedy: Specify a different workspace name or use the CLI <code>workspace create</code> command or GUI File ⇒ Create Workspace command to convert the directory into a workspace.</p>
2611	<p><i>file_name</i> does not exist in either workspace - ignored</p> <p>Meaning: The file <i>file_name</i> was not found in either the parent or child workspace.</p> <p>Remedy: Check to be sure the name was specified correctly.</p>
2612	Not used
2613	<p>Filename <i>file_name</i> has too many “.” path components in it - ignored</p> <p>Meaning: Possible causes include that the Configuring command cannot resolve the path name into a:</p> <ul style="list-style-type: none"><li>• workspace-relative file name</li><li>• fully qualified workspace name</li></ul> <p>Remedy: Specify the path name with fewer (or no) “.” path name components</p>
2614	<p>Line <i>line_number</i> too long or unexpected end of file in <i>file_name</i></p> <p>Meaning: While reading the <code>Codemgr_wsdata/nametable</code> file, a line was encountered that contained too many characters for a Configuring command to buffer. The maximum line length is 1024 characters. This indicates that <code>nametable</code> has been corrupted.</p> <p>Remedy: Configuring automatically rebuilds the <code>nametable</code>. This takes some time.</p>
2615	<p>Line <i>line_number</i> has bad format in <i>file_name</i></p> <p>Meaning: This indicates that the <code>Codemgr_wsdata/nametable</code> file has been corrupted.</p> <p>Remedy: Configuring automatically rebuilds the <code>nametable</code>. This takes some time.</p>
2616	Not used

---

Table 13-3 Configuring Warning Messages

2617	<p>Unexpected name table editlog record type <i>type_number</i> - ignored</p> <p>Meaning: A Configuring command was reading a temporary log file left over from an aborted Bringover or Putback operation and encountered a malformed record. This indicates that the file has been corrupted.</p> <p>Remedy: Execute the workspace updatenames command to rebuild nametable and then re-execute the command.</p>
2618	<p>Can't open <i>file_name</i> - can't send mail notification</p> <p>Meaning: The Configuring notification facility failed to open the file <i>file_name</i>. As a result, notification mail is not sent for the current operation.</p> <p>Remedy: Check file permissions for <i>file_name</i>.</p>
2619	Not used
2620	<p>Can't fork process to send notification</p> <p>Meaning: Lack of system resources (memory, swap space) prevented the Configuring notification facility from sending notification mail.</p> <p>Remedy: Check system resources.</p>
2621	Not used
2622	<p>Filename <i>file_name</i> contains a comment character (#) - ignored</p> <p>Meaning: A file name specified as an argument to a command (or in the Codemgr_wsdata/args file) contains the # character. Configuring reserves this character to denote comments.</p> <p>Remedy: Change the name of the file so that its file name does not contain the # character. If the problem persists, check the arguments listed in the args file.</p>
2623	<p>Read-lock left in workspace or Write-lock left in workspace</p> <p>Meaning: A Configuring command was unable to remove locks in <i>workspace_name</i>. This may indicate that there is insufficient disk space, or that permissions on the file Codemgr_wsdata/locks were changed since the lock was originally written.</p> <p>Remedy: Remove the locks using the Configuring GUI Props workspace command or the CLI workspace locks command.</p>
2624	<p>File <i>file_name</i> is checked out in workspace <i>workspace_name</i>. The changes in the checked out file will not be brought over</p>

Table 13-3 Configuring Warning Messages

---

	<p>Meaning: The file <i>file_name</i> is checked out in the parent workspace. You are being advised that any changes in the <i>g</i>-file were not brought over as part of the Bringover transaction.</p> <p>Remedy: Not applicable.</p>
2625	<p>File <i>file_name</i> is not in conflict according to the SCCS file. Removing it from the conflict file</p> <p>Meaning: The information in the SCCS history file indicates that the file contains no unresolved conflicts, however, the <code>Codemgr_wsdata/conflicts</code> file in the workspace lists it as being in conflict. The command removed it from the <code>conflicts</code> file.</p> <p>Remedy: Not applicable.</p>
2626	<p>File <i>file_name</i> not brought over because it is unresolved in workspace <i>workspace_name</i></p> <p>Meaning: The file <i>file_name</i> was not brought over because it contains an unresolved conflict in <i>workspace_name</i>.</p> <p>Remedy: Use the GUI Resolve transaction or the CLI <code>resolve</code> command to resolve the conflict and then re-execute the Bringover transaction.</p>
2627	<p>Directory <i>directory_name</i> is mounted read-only.</p> <p>Meaning: Before beginning Bringover and Putback transactions, Configuring checks to determine whether the destination workspace root (top-level) directory is accessible for writing. This is not treated as an error condition because lower level directories within the workspace could be mounted from different areas and they may be accessible for writing. This warning is issued as an early warning that directory permissions might be set incorrectly.</p> <p>Remedy: If write access is not intentionally denied, change the root directory permissions.</p>
2628	<p>Not updating <i>g</i>-files because <code>get</code> command couldn't be found in <code>PATH search_path</code></p> <p>Meaning: The <i>g</i>-files could not be updated as part of a Bringover or Putback transaction because the SCCS <code>get</code> command could not be executed; it was not found in your search path.</p> <p>Remedy: If you want <i>g</i>-files to be updated as part of transactions, include the <code>get</code> command in your search path.</p>
2629	<p>Will not be able to run <code>twmerge</code> or <code>filemerge</code></p> <p>Meaning: The <code>resolve</code> command was not able to connect with the ToolTalk message service. The ToolTalk service is used by the <code>resolve</code> command to communicate with the Merging program.</p> <p>Remedy: The ToolTalk service is normally installed as part of OpenWindows version 3. Check the OpenWindows documentation to determine why the ToolTalk service is not present or responding.</p>
2630	<p>This workspace is being created over an existing directory</p>

---

Table 13-3 Configuring Warning Messages

---

Meaning: You are converting an already existing directory into a Configuring workspace. Creating a workspace from an existing directory hierarchy consists of creating the `Codemgr_wsdata` metadata directory in the top-level directory. Once the directory becomes a workspace, its contents can be deleted using the Configuring workspace delete command.

Remedy: Not applicable.

---

2631 File *file\_name* not brought over because it is checked out and not writable in workspace *workspace\_name*

Meaning: The file *file\_name* was not brought over as part of the Bringover transaction because it is checked out (p-file exists) and writable in the child workspace *workspace\_name*. The unusual state of this file indicates that it is safer not to process the file.

Remedy: Reconcile the write permissions with its SCCS status.

---


2632 Omitting contents change to file *file\_name* because of rename error

Meaning: An error was encountered while processing the name of *file\_name*. As a result, the change in the file from the source workspace could not be propagated to the destination workspace.

Remedy: Correct the rename problem (see the rename error text) and re-execute the Configuring transaction.

---

# *Performing Basic SCCS Functions with Versioning*

14 

This chapter shows you how to perform basic SCCS functions using Versioning. It is organized into the following sections:

<i>Typical Sessions</i>	<i>page 171</i>
<i>Commands: Manipulating Files</i>	<i>page 174</i>
<i>View Option: Changing Versioning Properties</i>	<i>page 177</i>

## *Typical Sessions*

This section gives an overview of the most common SCCS sessions using Versioning. It assumes that you are familiar with the SCCS. The following scenarios are covered:

- An initial session where files are not yet under SCCS control
- A session where the project is already under SCCS control

## Changing Names

The current release of TeamWare uses new command names, so Table 14-1 summarizes the correspondences for you. Note that the old commands still work, however this manual uses the new commands and GUI names.

Table 14-1 New and Old TeamWare Commands

Old Command	New Command	Old Tool Name	New GUI Name
codemgrtool	twconfig, teamware	CodeManager	Configuring
vertool	twversion	VersionTool	Versioning
filemerge	twmerge	FileMerge	Merging
maketool	twbuild	MakeTool	Building
freezeptool	twfreeze	FreezePoint	Freezepointing

## Putting a Project Under SCCS Control

There may be instances where a project is under development before a source code control system is put in place. This scenario assumes a project is already under way and the hierarchy of the project is established. It is assumed that the project is ready to be put under SCCS control. The following process shows you how to do so.

- 1. Bring up Versioning at the top level of the source hierarchy by going to the appropriate directory and entering one of the commands shown in the following two examples:**

```
demo% twversion &
```

```
demo% twversion dirname &
```

- 2. Double click on the project directory from the list displayed in the base window.**  
Versioning automatically changes (cd) to the selected directory. As there are no files yet under SCCS control, the display will only show directories.



**3. From the Commands menu, choose Check In New.**

The Check In New window displays a list of files not under SCCS control.

**4. Select the files you want to put under SCCS control and add necessary comments in the Initial Comment pane.****5. Choose the OK button at the bottom of the window to check in the selected files.**

Once the files are under SCCS control, they will be transferred to the base window file list display. The Reset button clears the comment pane.

Repeat this scenario in as many project directories as necessary. Then proceed to the scenario in the next section for working with files under SCCS control.

### *Working with a Project Under SCCS Control*

Once a project is under SCCS control, you can use Versioning to perform SCCS functions. This section provides a scenario of basic SCCS tasks and how they might be applied on a project. These steps are simplified to give an overview of the process. The remaining sections of this chapter cover in-depth instructions on performing tasks.

---

**Note** – This is representative of a hypothetical session. The steps will vary according to project needs and the tasks required to fulfill them.

---

**1. Bring up Versioning in the working directory.****2. Check out a file.****3. From the View menu, select File History to display the history graph of the file.****4. Select two deltas from the history graph and inspect the diffs.****5. Make changes to the file.****6. Add necessary comments.****7. Check the file in.**

These steps can be repeated and varied as required by the needs of your project. The following sections of this chapter provide in-depth information on how to perform these, and other, SCCS functions with Versioning.

### *Getting Help on the GUI or the CLI*

Versioning is implemented in both the GUI and CLI. To use Versioning from the GUI, see the Sun WorkShop TeamWare online help.

To Access the Online Help

- 1. Open any Sun WorkShop TeamWare GUI from the Sun WorkShop.**
- 2. Or, open any Sun WorkShop TeamWare from the command line. For example, to open the Versioning GUI, enter the following:**

```
demo% twversion &
```

- 3. Open the pull-down menu from the Help button.**
- 4. Click on Help Contents.**

### *To Access Help for the CLI*

- ♦ **To access the manual page for information on how to use Versioning from the CLI, enter the following. The manual page gives information on all command-line options, variables, and macros necessary to use DMake.**

```
demo% man twversion.1
```

To discover what each of the windows and associated buttons do, bring up the GUI as shown above, and access the Help Contents>TeamWare Windows.

### *Commands: Manipulating Files*

You can perform SCCS operations within the file list on a per-file basis or on a multiple-file basis. You can select a single file, or multiple files, on which to perform a function.

The Commands menu allows you to do the following:

- Check out a file
- Check in a file

- Edit a checked out file
- Check a new file under SCCS control
- Uncheckout a file

## *Checking Out and Checking In Files*

This section covers the process of checking out and checking in files that are already under SCCS control.

### *Checking Out Files*

Two methods for checking out files are as follows:

- ♦ **Select the file(s) you want to check out from the base window display. Then, choose Check Out from the Commands menu.**

A check mark is displayed in the file icon(s) of the selected file(s). This method is valuable when you want to check out several files at once.

- ♦ **Double click on a file icon in the base window (if set in the Options window).**

A check mark is displayed in the file icon and the file is checked out with you as the owner. This is *not* the default behavior. You must change the default behavior using the View menu Options.

### *Checking In Files*

There are two methods for checking in files:

- ♦ **Select the file(s) you want to check in from the base window display and choose Check In from the Commands button menu. Enter the appropriate comments in the Check In pop-up window before choosing Check In.**

The check mark(s) continue to be displayed on the file icon(s) until you choose Check In from the pop-up window. This method is valuable when you want to check in several files at once and the same comment can apply to each.

- ♦ **Double click on a checked out file icon in the base window and add the appropriate comments in the Check In window before choosing Check In (if set in the Options window).**

The check mark continues to be displayed on the file icon until you choose Check In from the pop-up window. This is *not* the default behavior. You must change the default behavior using the View menu Options.

### *Editing a Checked-Out File*

This section covers the process of checking out a file that is under SCCS control and displaying it in a window with an editor. The Edit menu item allows you to do this in either of the following ways:

To check out a file and display it in a window with an editor:

- ♦ **Choose the Check Out & Edit menu item from the Commands menu.**  
The default is for the file to be brought up in a cmdtool window running `vi`. For instructions on defining an editor, see “View Option: Changing Versioning Properties.”

### *Checking in a New File*

Files that are not under SCCS control are not displayed in the base window. To see what files are in a directory and are *not* under SCCS control, use the Check In New menu item.

Check In New displays a window that contains a list of files. You can define the list by specifying a pattern in the filter text field.

- 1. Display a list of files from a directory that are *not* under SCCS control.**  
Double click on the directory in the base window, or define the directory path in the Directory text field. Once in the directory, choose Check In New from the Commands button menu. The Check In New window is displayed with a list of files not yet under SCCS control.
- 2. Check in new files under SCCS control using the Check In New window.**  
Select the files from the Check In New window display. Enter the appropriate initial comments in the Initial Comments pane before choosing Check In. The checked-in files are removed from the Check In New display and now appear in the base window file list display.

---

## *Unchecking Out a File*

When you have mistakenly checked out a file and want to return the file to an unchecked out state, there is a simple way to do so without having to check in the file and add comments. This is done through the Uncheck Out option of the Commands button menu.

To uncheck out a file:

- ◆ **Select the checked out file and choose Uncheckout from the Commands menu.**

You will be prompted for confirmation. When confirmed, the file reverts to its previous unchecked-out status, and no comments are required. No record is kept of your owning the file.

## *View Option: Changing Versioning Properties*

The View menu displays a pop-up window with Versioning properties options. Selecting options from this window sets the Versioning properties for the remainder of the session.

---

**Note** – You must choose the Apply button before the property selections are activated. Use the Set Default button to save the changes for subsequent `twversion` sessions.

---

## *Changing the Main File List Display*

The Main File List category lets you specify the type of SCCS files displayed in the base window file list.

To specify the base window file list display:

- ◆ **Select one of the following from the Options window:**
  - List all files under SCCS control
  - List only files which are checked out
  - List only files which are checked in

## *Defining an Editor*

The Editor category lets you specify an editor that automatically comes up when you view the contents of a delta, or bring up a delta to edit. You have the following list of editors to choose from. The last option allows you to specify your own editor.

To specify an editor:

♦ **Select one of the following from the Options window:**

- textedit
- emacs
- emacsclient
- vi
- Other: \_\_

---

**Note** - With Other: \_\_ you must supply a command that will bring up your editor in a separate window. The file name is tacked on the end of the supplied command.

---

---

**Note** - If you set the EDITOR environment variable to one of the top four selections, Versioning brings up the editor automatically without setting it from the Properties window.

---

## *Changing the Double-Click Action*

The Double-Click Action category lets you specify what happens when you double click in Versioning.

To specify the results of the double-click action:

♦ **Select one of the following from the Options window:**

- Toggle SCCS State — checked in or checked out  
When you select this option, you can optionally check the Confirm Double Click Check Out option.
- Show File History — automatically brings up the History window

## *Changing the History Graph Display*

The History Graph category lets you define items for display on the history graph.

To specify the display of the history graph:

- ◆ **Select the desired options from the Options window.**  
These options are toggles that you can turn off or on.
- Show Removed Deltas - Removed deltas displayed with an “X” through them
- Show Branch Closure - Shows dashed lines that indicate changes included from other deltas

## *Changing the History Information Display*

The History Information category lets you specify the extensiveness of the information displayed when you select a delta on the history graph.


To specify the history information:

- ◆ **Select one of the following from the Option window:**
    - Show Per-SID Information
    - Show Entire File History
- When you select this option, you can also specify a command to gather the history.





## Starting and Loading Merging

15 

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

<i>Changing Names</i>	<i>page 182</i>
<i>Starting Merging from Sun WorkShop</i>	<i>page 182</i>
<i>Loading Files from the Merging Window</i>	<i>page 182</i>
<i>Starting Merging from the Command Line</i>	<i>page 183</i>
<i>Saving the Output File</i>	<i>page 188</i>

You can start Merging from Sun Workshop, or the Sun Workshop TeamWare GUI, or from a shell command line. Each method has its advantages. The command line provides flexible access to all Merging options such as loading files at startup time. However, the GUI provides the following additional capabilities:

- An easy-to-use visual interface for starting all tools
- The ability to set properties, such as a default working directory, for all tools
- A properties sheet specific to each tool that lets you specify command-line options and save the command line for use in a later session

You can also start Merging from Versioning and the Sun Workshop Resolving tool.

## Changing Names

The current release of TeamWare uses new command names, so Table 15-1 summarizes the correspondences for you. Note that the old commands still work, however this manual uses the new commands and GUI names

*Table 15-1* Correspondences Between New and Old TeamWare Commands

Old Command	New Command	Old Tool Name	New GUI Name
codemgrtool	twconfig, teamware	CodeManager	Configuring
vertool	twversion	VersionTool	Versioning
filemerge	twmerge	FileMerge	Merging
maketool	twbuild	MakeTool	Building
freezepttool	twfreeze	FreezePoint	Freezepointing

### Starting Merging from Sun WorkShop

To start Merging from Sun WorkShop:

- ◆ **Select the Merging icon in the Sun WorkShop window and click on it.**

When Merging starts successfully, the Merging window opens.

### Starting Merging from Sun WorkShop TeamWare

To start Merging from Sun WorkShop TeamWare:

- ◆ **Pull down the TeamWare menu from the Sun WorkShop TeamWare Configuring window and select Merging.**

When Merging starts successfully, the Merging window opens.

### Loading Files from the Merging Window

To load files in Merging:

- ◆ **Select Open from the File menu, or from the Open button.**

The resulting pop-up window provides text fields in which to enter file names. As an example, from the directory `/usr/src`, `matriarch` is entered as the ancestor file, `file_1` as the left file, and `file_2` as the right file.

**Working Directory**— This text field shows the current working directory whenever you start Merging from the Manager or from the command line with no arguments. You can edit this field. Merging interprets the file names you specify in the window as relative to the current working directory. Therefore, you can use such constructs as `subdir/filename` to specify a file in a subdirectory and `../filename` to specify a file in a parent directory. Any file name you specify that begins with a “/” character is interpreted as an absolute path name, not as relative to the current working directory

**Left File**—The file you specify in this text field appears in the left text pane, also considered the “child” pane.

**Right File**—The file you specify in this text field appears in the right text pane, also considered the “parent” pane.

**Ancestor File**—If you enter the name of an ancestor file, Merging will compare it to the files to be merged and identify lines in those files that differ from the ancestor. The automerged file will be based on the ancestor file, but the ancestor file itself is not displayed in any Merging window. If you do not enter an ancestor file name, Merging compares only the left and right files and derives the output file from them. Automerging is not possible without an ancestor file.

**Output File**—The merged output file takes the name you specify in this text field. Merging uses the name `filemerge.out` unless you specify a different name, and stores the file in the current working directory.

**Open Button**—Click on the Open button to load the files you have specified in the text fields.

In a loaded Merging window the names of the left file, right file, and output file are displayed above each text pane. The name of the ancestor file (for a three-way diff only) is displayed in the window header. The “...” buttons to the right of each text field bring up a file chooser.

## *Starting Merging from the Command Line*

Starting Merging from the command line enables you to:

- Determine which files (if any) are loaded into Merging at startup
- Specify the name of the merged output file at startup
- Specify whether or not Merging should consider leading white space (tabs and spaces) when it identifies differences
- Specify whether Merging should produce a writable merged output file or function in read-only mode
- Load lists of files from specified directories
- Specify input file display names, which are displayed over the left and right text panes in the Merging base window (This feature is especially useful when you merge entire directories of files from a list.)

## *Basic Startup Command*

To start Merging from the command line without loading any input files (assuming that the Merging executable is in your search path):

♦ **Type the following command at a command-line prompt:**

```
demo% twmerge &
```

The command starts Merging (in background) without loading any files.

## *Command-Line Synopsis*

The complete `twmerge` command is summarized below, with command options enclosed in square brackets.

```
twmerge [-b] [-r] [-a ancestor] [-f1 name1] [-f2 name2] [-l listfile] [ leftfile rightfile [outfile] ] [-v]
```

## *Options Defined*

-b

Causes Merging to ignore leading blanks and tabs when comparing lines.

-r

Starts Merging in read-only mode. When you specify this option, only the input file text panes are displayed, and the output text pane is absent.

**-a ancestor**

Specifies an existing ancestor file of the two files to be merged (called *descendants* of the ancestor file). The merged file is based on this ancestor file and the changes to it that have been made in the descendants.

When used with the **-l listfile** option, *ancestor* is a directory of files, which you can load in succession from the File menu.

**-f1 name1**

Sets the file name displayed for the first (left) file. This option is useful when a list of files is being loaded (with the **-l** option), and you want to display a name for reference only in the Merging window.

For example, if you are loading files from two directories that correspond to two different revisions of a product, you could use the **-f1** option to display the name *Rev1* above the left pane and the **-f2** option to display the name *Rev2* above the right pane.

**-f2 name2**

Sets the file name displayed for the second (right) file.

**-l listfile**

Specifies a file that contains a list of individual file names. This option is useful for merging entire project directories.

Merging uses the names in *listfile* to successively load files from directories you name with the *leftfile* and *rightfile* arguments, placing the output files in the directory you name with the *outfile* argument. The names in *listfile* must match file names in the *leftfile* and *rightfile* directories. When used with the **-a ancestor** option, the *ancestor* argument must be a directory: Merging will look in the *ancestor* directory for files that have the same names as those in *listfile* and use those with matching names as ancestor files for each merge.

If you specify the character “-” for *listfile*, Merging reads the list of files from standard input.

**leftfile**

The name of the left file to be loaded for comparison. When used with the **-l listfile** option, *leftfile* is a directory of files, which you can load in succession from the File menu.

## ***rightfile***

The name of the right file to be loaded for comparison. When used with the `-l listfile` option, *rightfile* is a directory of files, which you can load in succession from the File menu.

---

**Note** – If you use the `-l listfile` option, then all three input file names (*ancestor*, *leftfile*, and *rightfile*) must be directories. If you do *not* use the `-l listfile` option, then any two input file names can be directories, but one of the three must be a simple file name. In this case, Merging uses the file name to find a file with the same local name in each directory.

---

## ***outfile***

Specifies the name of the merged output file. If you do not specify an *outfile*, the output file is given the default name `filemerge.out`. If you want to specify a different name when you save the file, use Save As from the File menu.

When used with the `-l listfile` option, *outfile* names the directory to be used when each merged output file is saved. Individual file names in the *outfile* directory are the same as the names listed in *listfile*.

`-v`

Does not start up Merging, but displays the version number of the application.

## ***Loading Two Files at Startup***

To load two files at the time you start Merging, change to the directory in which the files are stored and specify the file names on the command line. To merge two files named `file_1` and `file_2`, use the following command:

```
demo% twmerge file_1 file_2 &
```

The first file listed appears in the left text pane; the second file appears in the right pane.

### *Loading Three Files at Startup*

To merge the same two files and at the same time compare them to a common ancestor named `ancestor_file`, change to the directory in which the files are stored and use the following command:

```
demo% twmerge -a ancestor_file file_1 file_2 &
```

The ancestor file is not displayed, but differences between the ancestor file and the two descendants are marked, and the merged output file is based on the ancestor file.

### *Loading Files from a List File*

You can sequentially load files from a list of file names. For example, suppose ancestor versions of a project's source files are stored in a directory named `/src`. You have been editing the files `file_1`, `file_2`, and `file_3` in your directory `/usr_1`, and another developer has been simultaneously editing the same files in the directory `/usr_2`. You have been given the responsibility of merging the changes to both sets of files, and you want to place the merged versions in a directory named `/new_src`.

To merge the `/src`, `/usr_1`, and `/usr_2` directories, you first create a list file that contains only the names of the three files to be merged with each name on a separate line, as follows:

```
file_1  
file_2  
file_3
```

Name the file `sourcelist` and place it in the working directory where you plan to start Merging. Change to that directory (with the `cd(1)` command) and start Merging with the following command:

```
demo% twmerge -a /src -1 sourcelist /usr_1/usr_2/new_src &
```

This command causes Merging to load `/usr_1/file_1` into the left text pane, `/usr_2/file_1` into the right text pane, and compare both files to the common ancestor `/src/file_1`. After you have resolved the differences between the two files, choose Save from the File menu to automatically save the output file as `/new_src/file_1`.

### *Saving the Output File*

Save the output file by clicking on the Save button or choosing Save from the File menu. The name of the output file is the name you specify in the Output File text field of the Load pop-up window.

The Load item on the File menu changes depending on whether or not you are loading files from a list; however, Save and Save As are available in either case.

To change the name of the output file while saving, choose Save As and fill in the new file and directory names in the resulting pop-up window. Saving an Output File Under Another Name



This chapter further explains some features and presents an example of how to use Merging. The chapter is organized into the following sections:

<i>Moving Between Differences</i>	<i>page 189</i>
<i>Resolving Differences</i>	<i>page 189</i>
<i>Automatic Merging</i>	<i>page 190</i>
<i>An Example</i>	<i>page 190</i>

### *Moving Between Differences*

You can navigate through differences using either the Next and Prev buttons or the up arrow and down arrows. Merging automatically moves to the next difference immediately after you resolve the current difference.

### *Resolving Differences*

A difference is represented by a blank line in the merged (output) file in the lower text pane. To resolve a difference, you edit the line displayed there by either:

- Accepting the line displayed and incorporate it into the merged file by choosing either the Accept or Accept & Next button over the side you wish to accept.

- Editing the line in the merged file by hand, and marking the difference as resolved. From the Edit menu, choose Mark Selected as Resolved.

### *Automatic Merging*

If you have loaded a common ancestor file, Merging is often able to resolve differences automatically, based on the following rules:

- If a line has not been changed in either descendant (it is identical in all three files), it is placed in the merged file.
- If a line has been changed in only one of the descendants, the changed line is placed in the merged file. A change could be the addition or removal of an entire line, or an alteration to some part of a line.
- If identical changes have been made to a line in both descendants, the changed line is placed in the merged file.
- If a line has been changed differently in both descendant files so that it is different in all three files, Merging places no line in the merged file. You must then decide how to resolve the difference—either by using a line from the right or left file, or by editing the merged file by hand.

### *An Example*

This example merges two files that have a common ancestor. The files are `file_1` and `file_2`, and the ancestor file is named `matriarch`. The descendant files `file_1` and `file_2` were derived from `matriarch` by editing. The edits show all varieties of changes that could occur in the descendants: deleting lines, adding new lines, and changing lines.

The content of each line in the example helps to identify whether or not it was changed, and how. The ancestor file contains only twelve lines and is shown in Code Example 16-1.

Merging does not number lines in the files it loads; the numbers are part of the example text and were placed there for clarity.

*Code Example 16-1* Ancestor File (matriarch)

```
1 This line is deleted in file_1
2 This line is in all three files
3 This line is deleted in file_2
4 This line is in all three files
5 This line is in all three files
6 This line is changed in descendants
7 This line is in all three files
8 This line is changed in descendants
9 This line is in all three files
10 This line is changed in file_2
11 This line is in all three files
12 This line is in all three files
```

Code Example 16-2 shows the contents of `file_1`. This file is identical to `matriarch` with the following exceptions:

- The line numbered 1 in the `matriarch` file was deleted in `file_1`.
- A new line was added following the line numbered 4.
- The line numbered 6 was changed (a different change was made to this line in `file_2`).
- The line numbered 8 in the `matriarch` file was changed (an identical change was made to this line in `file_2`).

*Code Example 16-2* Descendant File (`file_1`)

```
2 This line is in all three files
3 This line is deleted in file_2
4 This line is in all three files
  &&& Added to file_1 &&&
5 This line is in all three files
6 This line is modified in file_1 from matriarch
7 This line is in all three files
8 ##&# Changed in file_1 and file_2 #&#
9 This line is in all three files
10 This line is changed in file_2
11 This line is in all three files
12 This line is in all three files
```

Code Example 16-3 shows the contents of `file_2`. This file is identical to `matriarch` with the following exceptions:

- The line numbered 3 in the `matriarch` file was deleted.
- The line numbered 6 was changed (a different change was made to this line in `file_1`).
- The line numbered 8 was changed (an identical change was made to this line in `file_1`).
- The line numbered 10 was changed (no change was made to this line in `file_1`).
- A new line was added following the line numbered 11.

*Code Example 16-3* Descendant File (`file_2`)

```
1 This line is deleted in file_1
2 This line is in all three files
4 This line is in all three files
5 This line is in all three files
6 This line is altered in file_2 from matriarch
7 This line is in all three files
8 ### Changed in file_1 and file_2 ###
9 This line is in all three files
10 ### Changed in file_2 ###
11 This line is in all three files
    ### Added to file_2 ###
12 This line is in all three files
```

To simplify the example, we place all three files in one directory, and use this directory as the working directory where we will start Merging.

## *Starting Merging*

- ♦ **Go to the directory in which `matriarch`, `file_1`, and `file_2` are stored. Type the following to start Merging in background mode and load the three files:**

```
demo% filemerge -a matriarch file_1 file_2 &
```

Merging starts up with the Auto Merge feature turned on by default. The window that appears displays an automerged output file.

In the upper left of the window, Merging has reported finding seven differences, of which only one remains unresolved—six differences were resolved by automerging, and are marked by glyphs in outline font.

The meaning of the glyphs is as follows: a vertical bar means a change in the marked line, a plus sign signifies a line added, a minus sign means a line was deleted. Unresolved states are marked by solid glyphs, unresolved by outline. These glyphs are highlighted in color except when the color map is full. The default significance is: red indicates a change, green indicates a deletion, yellow shows an addition.

The unresolved difference (line 6) is marked by a vertical bar.

## *Examining Differences*

Merging highlights the unresolved difference, which it identifies as the line numbered 6 in `file_1` and `file_2`. When differences are being resolved with Merging, the resulting Merging window (`filemerge.out`) shows the current state of the file with automatic merging.

Proceed to the next difference by choosing the down arrow above the appropriate file, or `Navigate: Next Difference`. The next difference becomes the current difference.

Proceed through the differences by clicking on the down arrow.

Auto Merge preserves a change that was made to one file if no change was made in the other file.

When a difference has not been resolved by Auto Merge, as indicated by the solid highlighted glyph next to the lines involved in the difference, you need to resolve the difference by making a choice. The vertical line indicates that the line has been changed (as opposed to added or deleted). In this case, Auto Merge failed because the same line was changed differently in the two files, and Merging could not decide which change was more valid.

You could resolve this difference in one of the following ways:

- Click on the `Accept`, `Accept & Next` button on the left to place the line from `file_1` into the output file.

- Click on the the Accept, Accept & Next button on the right to place the line from `file_2` into the output file.
- Edit the output file by hand.

### ***Editing the Output File***

To edit the output file, move the pointer into the output file's text pane and place it in the line you want to change. In this example, the following line was typed in:

```
>>> This line edited by hand <<<
```

Choose the Edit:Mark Selected as Resolved. This menu item marks the difference as resolved. In this example there are no more unresolved differences, so the next difference remains the current one.

Note that the message in the upper left part of the window now indicates that all differences have been resolved. Nevertheless, proceed to verify the automerged differences.

Continue through the differences by clicking on the down arrow.

The final difference results from a line that was added only to `file_2`. Merging places the new line in the output file just as it did when a new line was added to `file_1`, which resulted in the third difference.

### ***Saving Output File***

Now that all differences have been resolved and the automerged differences verified, you can save the output file. The output file takes the name shown in the Load Files pop-up window, which by default is `filemerge.out`. To write the file, choose Save from the File menu, or the Save button. To save the file under another name, use Save As, from the File menu.







This chapter discusses how FreezePointing can help you preserve snapshots of your work, and the process of recreating those key points.

<i>How FreezePointing Works</i>	<i>page 198</i>
<i>Starting FreezePointing</i>	<i>page 200</i>
<i>Creating a Freezepoint File</i>	<i>page 201</i>
<i>Recreating (Extracting) a Source Hierarchy</i>	<i>page 202</i>

## Changing Names

The current release of TeamWare uses new command names, so Table 17-1 summarizes the correspondences for you. Note that the old commands still work, however this manual uses the new commands and GUI names.

*Table 17-1* Matching New and Old TeamWare Commands

<b>Old Command</b>	<b>New Command</b>	<b>Old Tool Name</b>	<b>New GUI Name</b>
codemgrtool	twconfig, teamware	CodeManager	Configuring
vertool	twversion	VersionTool	Versioning
filemerge	twmerge	FileMerge	Merging
maketool	twbuild	MakeTool	Building
freezptool	twfreeze	FreezePoint	Freezepointing

During the software development process it is often useful to create “freezepoints” of your work at key points. Those freezepoints serve as snapshots of a project that enable you to later recreate the state of the project at key development points.

One way to preserve the state of the project is to make a copy of the project hierarchy using the `tar` or `cpio` utilities. This method is very effective, but it requires a large amount of storage resources and time.

With FreezePointing, you preserve freezepoints quickly and simply, using a small amount of storage resource.

You can use FreezePointing through two functionally equivalent user interfaces. You can access the user interfaces with the following commands

- `twfreeze`—for the GUI
- `freezept`—for the CLI

---

**Note** – FreezePointing is a companion tool to the TeamWare product. Therefore, FreezePointing assumes that you are creating freezepoints of Configuring workspace hierarchies. You can also use FreezePointing to preserve nonworkspace directories that contain SCCS files. If you specify a directory that is not a workspace, a cautionary warning is issued.

---

This chapter refers primarily to the GUI. For information about the CLI, see the `freezept(1)` man page. The GUI is documented online. You can access the online help from any TeamWare GUI, by opening the pull-down menu from the Help button, and clicking on Help Contents.

## *How FreezePointing Works*

FreezePointing enables you to create freezepoint files from Configuring workspaces.

---

**Note** – Nonworkspace directory hierarchies that contain SCCS history files can also be preserved using FreezePointing. FreezePointing issues a warning if the directory is not a workspace.

---

At a later time you can use the freezepoint files to recreate the directory hierarchies contained in the workspaces.

---

**Note** – The recreated hierarchy will not contain the original SCCS history files; only the g-files represented by the default deltas from the original hierarchy are recreated. The default delta is the delta that would be

---

The freeze point file that FreezePointing creates is a text file that lists the default deltas in SCCS history files in the hierarchy. When you later recreate the hierarchy, FreezePointing uses those entries as pointers back to the original history files and to the delta that was the default at the time the freeze point file was created.

When you create a freeze point file, you specify directories and files to FreezePointing in the Directories and Files pane. FreezePointing recursively descends the directory hierarchies and identifies the most recently checked-in deltas in each SCCS history file. FreezePointing then creates a freeze point file that consists of a list of those files and unique numerical identifiers for each delta.

You can later use FreezePointing to recreate the source hierarchy. You specify the name of the freeze point file, the path name of the directory hierarchy from which the deltas are to be extracted (if different from the hierarchy from which it was derived), and the directory where you want the source hierarchy recreated.

## ***Terminology***

### ***Freeze point File***

A freeze point file is a list of the default deltas from the SCCS history files contained in the workspace hierarchy being preserved. The freeze point file also contains the following information:

- The login name of the user who created the freeze point
- The date and time that the file was created
- The path name of the workspace from which the list of deltas was created
- An optional user-supplied comment

See “Details about the Freeze point File” on page 204 for more information.

### *Extract*

The extract operation consists of creating a new directory hierarchy based on the information contained in the freeze point file. The new hierarchy is comprised of g-files defined by the default deltas in the original SCCS history files; *the history files themselves are not recreated*. Deltas are extracted from SCCS history files located in the original source workspace.

### *Source Workspace*

The source workspace is the directory hierarchy that contains the SCCS history files from which the freeze point file is created. Usually, the source workspace is also the directory hierarchy from which g-files are later extracted to recreate the hierarchy.

---

**Note** – You can specify an alternate source directory at the time you perform the extract operation.

---

### *Destination Directory*

The destination directory is the top-level directory into which the files listed in the freeze point file are extracted. You specify the path name of this directory in the Extract pane of the FreezePointing base window.

## *Starting FreezePointing*

- ◆ **To start the FreezePointing GUI, type the following:**  
After a moment, the FreezePointing GUI appears.

```
demo% twfreeze &  
demo%
```

## Creating a Freezepoint File

**1. To create a Freezepoint file, use the Category menu to choose the Create pane.**

The pane below the Control area is used for both creating and extracting freezepoints. You switch between the Create pane and the Extract pane by choosing the appropriate item from the Category menu. The Create pane is the default and is displayed when you start FreezePointing.

**2. Enter the name of a freezepoint file.**

- When FreezePointing initially appears, the FreezePointing File text field is automatically set to contain the file `freezepoint.out` appended to the path name of the directory from which freezepoint.
- Delete `freezepoint.out` and type the path name of your freezepoint file in the FreezePointing File text field.

Note that path names that are not absolute are assumed to be relative to the directory in which FreezePointing is started.

**3. Enter the name of the source workspace.**

When you start FreezePointing, the Workspace text field is automatically set to be the workspace you have specified through the `CODEMGR_WS` environment variable. If the variable is not set, and the directory from which FreezePointing is started is hierarchically within a workspace, the Workspace field is initialized with the path name of that workspace.

**4. In the Directories and Files text window, compose a list of directories and/or files that you wish to preserve.**

The list of directories and files that you create in the Directories and Files text window are those that will be preserved in the freezepoint file.

You add directory and file entries to the Directories and Files window using the two items in the File menu:

- Load Entire Directory
- Add File to List

The Load Entire Directory inserts the “./” characters into the Directories and Files window; this indicates that the entire workspace hierarchy be recursively preserved.

The Add Files to List item activates a point-and-click chooser window with which you can search for and select files and directories to add to the list.

- Click SELECT on a directory icon to select it, and then select the chooser's Add to List button to add the choice to the list.
- Double click SELECT on a directory icon to descend in the file system hierarchy, double-click SELECT on a previous icon to ascend. (Alternatively, you can select a directory icon and click on the Load Directory button to hierarchically descend.)

---

**Note** – You can also type the path name of a directory or file into the chooser Directory field and then click SELECT on the Add to List button.

---

**5. Enter an optional comment in the Comments text pane.**

The comment is stored in the freezeport file for future reference.

**6. Select the Create button to create the freezeport file.**

A counter on the bottom right corner of the base window footer displays the progress of the freezeport operation.

## *Viewing or Modifying a Freezeport File*

Freezeport files are text files. You can view and edit their contents using standard text editors.

## *Recreating (Extracting) a Source Hierarchy*

To extract a new source hierarchy described by a freezeport file, follow these basic steps:

**1. Use the Category menu to choose the Extract pane**

The pane below the Control area is used for both creating and extracting freezeports. You switch between the Create pane and the Extract pane by choosing the appropriate item from the Category menu. Choose the Extract item to display the Extract pane.

**2. Type the name of an existing freezeport file.**

Type the path name of your freezeport file in the FreezePointing File text field.

Note that path names that are not absolute are assumed to be relative to the directory in which FreezePointing is started.

---

**3. Use the Extract From menu to choose how you will specify the source workspace.**

By default, FreezePointing extracts files from the source workspace path name stored in the freeze point file when it was created. (You can edit the freeze point file and change the path name of the source workspace.) By default, FreezePointing uses this path name as the source workspace from which to extract files. If you choose the Show Default menu item from the Extract From menu, FreezePointing displays the path name of the source workspace in the Workspace text field.

---

**Note** – Show Default displays the default source workspace in the Workspace text field

---

If you wish to specify a source workspace hierarchy other than the one contained in the freeze point file, choose the You Specify item from the Extract From menu and enter the path name of the alternate source workspace in the Workspace text field.

**4. Specify the Destination Directory.**

Enter in the Destination Directory text field the path name of the directory in which you want the new (extracted) hierarchy to be located.

Note that path names that are not absolute are assumed to be relative to the directory in which FreezePointing is started. The destination directory that you specify must be new or empty.

**5. Select the Extract button to begin the extraction.**

Selecting the Extract button causes a series of `scs get` operations to be performed on the source files listed in the freeze point file. The version of each file extracted is the version specified by the SMID in the freeze point file. The extracted g-files are written to destination directory.

A counter on the bottom right corner of the base window footer displays the progress of the extract operation.

### *Notes about Using FreezePointing*

- Use the Edit menu on the Create pane to delete selections from the Directories and Files text window. Select and deselect files using the SELECT mouse button and then use the Delete item from the Edit menu to delete selected directories/files. Use the Select All, Deselect All, Delete All items to edit large numbers of directories/files.
- Helpful status messages are displayed in the main window footer.
- If during an extraction, FreezePointing cannot locate a file that has been renamed or deleted, the extraction is aborted and the offending entry is named. You must edit the freezeptoint file to remove the entry. Refer to the freezeptointfile(5) man page for information that enables you to determine the new name of a renamed file.
- You can use the Tools menu to launch other TeamWare tools directly from FreezePointing.

### *Details about the Freezepoint File*

A freezepoint file contains:

- A list of source files
- A group of hex digits that identifies the most recent SCCS deltas found in each file's corresponding SCCS history file
- A group of hex digits that identifies the root delta in each file's corresponding SCCS history file

```
filemerge.1 (previously 1.5) 92/03/19 14:09:08 jon a6f4fe81 89b4632b 418e7950 5510740e cf9ab4e1 95627c33 2287acc3 b9e0877e  
putback.1 (previously 1.40) 92/06/02 16:36:16 george 5b791c60 2b827cfd f0cc9a73 46ac975 24d9b3ec f87d1975 9ea59e0d 72ce2a4d  
resolve.1 (previously 1.19) 92/06/10 16:38:07 paul f21fa6e6 668bf818 e4964f36 240d825c f1d3f57 8cc4c31c 9f53029f 8aaf3db1
```

*Figure 17-1* Three Entries From a Freezepoint File

The deltas are *not* identified as you might imagine, by their standard SCCS delta ID (SID). Instead, a new means of identification called an SCCS Mergeable ID (SMID) is used. Use of the SMID enables FreezePointing to work properly with files in which SIDS have been renumbered as part of a Configuring Bringover Update transaction. For more information see Section , "Why are SMIDs Necessary?" .



---

## *What is a SMID?*

The use of SMIDs ensures that every delta is uniquely identifiable, even if its SID is changed. A SMID is a number generated using the Xerox Secure Hash Function. When you use FreezePointing to create a freezept file, it calculates the SMID for both the current delta and the root delta in the SCCS history file. Using both of these values, FreezePointing can identify a delta in a file even if its SID has been changed.

## *Why are SMIDs Necessary?*

---

**Note** – This section briefly discusses how Configuring merges SCCS history files. For more information, see Chapter 11, “How the Configuring Program Merges SCCS Files.”

---

When Configuring encounters a file conflict during a Bringover Update transaction (file is changed in both the parent and child workspaces), it merges the new deltas from the parent workspace into the SCCS history file in the child. When this merge occurs, the deltas that were created in the child are moved to an SCCS branch off of the delta that both deltas have in common (common ancestor).

When Configuring relocates the child deltas to a branch, it changes their SID. If SIDs were used in freezept files to identify deltas, this relocation would invalidate the information contained in the freezept file. For that reason, SIDs cannot be used to identify deltas after conflicting SCCS histories have been merged.

## *SMID/SID Translation*

In release 2.0 of TeamWare, SMID/SID translation is available only through the FreezePointing CLI.

The `freezept` command `sid` and `smid` subcommands enable you to translate specified SIDs into SMIDs, and to translate specified SMIDs into SIDs. The ability to make these translations is useful if you wish to write your own scripts or programs to track deltas.

## Translating SIDs to SMIDs

Use the `freezept smid` command to translate SIDs to SMIDs. The syntax is:

```
freezept smid [-w workspace] [-r SID] [-a] file
```

- Use the `-r` option to specify the SID (in file *file*) for which you wish to calculate a SMID.
- Use the `-a` option to calculate a SMID for all of the SIDS in *file*.
- For convenience you can use the `-s` option to specify a directory from which *file* is relative.

### Examples

```
example% freezept smid -r 1.38 module.c
SID 1.38 = SMID "f5b67794 705f0768 a89b1f4 588de104"
```

```
example% freezept smid -a bringover.1
SID 1.1 = SMID "b05b0a2f 1db5246e 1a466014 707e38f5"
SID 1.2 = SMID "d6a5c61f 5634f0ef 9847a080 d0d7b212"
SID 1.2 = SMID "e31acdd5 6c1232e2 9e81c287 1edb2f41"
SID 1.3 = SMID "c34c91b4 a818622a 2457356a 489b2728"
SID 1.4 = SMID "98c0fd8d 889563fb cf722c2b 6afc9636"
SID 1.5 = SMID "b1e24be3 752fec3e df2d2717 a9b3f1fa"
SID 1.6 = SMID "2b93d39 1ea2f6ba 9814320c bc609acb"
SID 1.7 = SMID "1db7d640 42b0f009 35c60d7b b230bd85"
SID 1.8 = SMID "906dfe9a ca7e2d6c a64da5be 4baef254"
```

## Translating SMIDS to SIDS

Use the `freezept sid` command to translate SMIDs to SIDs. The syntax is:

```
freezept sid [-w workspace] [-m "SMID"] [-a] file
```

- Use the `-m` option to specify the SMID (in file *file*) for which you wish to calculate a SID.
- Use the `-a` option to calculate a SID for all of the deltas in *file*.

- For convenience you can use the `-s` option to specify a directory from which *file* is relative.

---

**Note** – Because the SMID contains white space, you must enclose it within quotation marks.

---


### Examples

```
example% freezept sid -m "64fdd0df de9d7dd de75812 23da96aa"  
module.c  
SMID "64fdd0df de9d7dd de75812 23da96aa" = SID 1.36
```

```
example% freezept sid -a bringover.1  
SMID "b05b0a2f 1db5246e 1a466014 707e38f5" = SID 1.1  
SMID "d6a5c61f 5634f0ef 9847a080 d0d7b212" = SID 1.2  
SMID "e31acdd5 6c1232e2 9e81c287 ledb2f41" = SID 1.2  
SMID "c34c91b4 a818622a 2457356a 489b2728" = SID 1.3  
SMID "98c0fd8d 889563fb cf722c2b 6afc9636" = SID 1.4  
SMID "ble24be3 752fec3e df2d2717 a9b3f1fa" = SID 1.5  
SMID "2b93d39 1ea2f6ba 9814320c bc609acb" = SID 1.6  
SMID "1db7d640 42b0f009 35c60d7b b230bd85" = SID 1.7  
SMID "906dfe9a ca7e2d6c a64da5be 4baef254" = SID 1.8  
SMID "77481e8a 61542339 cc28f532 e5fc6389" = SID 1.9  
SMID "cb97c9a6 d0342cf6 19b7b743 2436ca1c" = SID 1.10  
SMID "46de4131 b95b9973 93958a07 b960074c" = SID 1.11
```



# Troubleshooting Versioning and FreezePointing

18 

This chapter describes some of the most common problems in Versioning and FreezePointing. It indicates where to look for information on how to overcome the problem. It is organized into the following sections:

<i>Troubleshooting Checklist</i>	<i>page 209</i>
<i>Reporting Problems</i>	<i>page 210</i>
<i>Error Messages</i>	<i>page 210</i>

## Troubleshooting Checklist

If you are having problems using Versioning or FreezePointing, use the following checklist to rule out some of the most common reasons for the problem:

- Is the tool installed correctly?  
If not, contact your system administrator. You can also read *Sun Workshop Installation and Licensing Guide*.
- Is `/opt/bin` in your PATH?  
If not, see *Sun Workshop Installation and Licensing Guide* for information on how to add `/opt/bin` to your PATH.
- Is `/usr/lang` in your PATH?  
If not, see *Sun Workshop Installation and Licensing Guide* for information on how to add `/usr/lang` to your PATH.

- ❑ Is the `HELPPATH` environment variable set?  
Versioning relies on finding the `vertool.info` file in or near the directory that contains `vertool`. FreezePointing relies on finding the `freezepoint.info` file in or near the directory that contains `freezepoint`. On-line help is available for each control, window, pane, and error message displayed on the screen. See “Error Messages” for a list of the Versioning and FreezePointing error messages and instructions on what to do next.
- ❑ Do you have enough swap space?  
If you receive a message stating “Request for xxx bytes of memory failed,” you have run out of swap space. Use the `mkfile(8)` and `swapon(8)` commands to create more swap space or abort some existing processes (windows) to free up swap space. To determine which processes occupy significant swap space, use the `ps uagx` command and look in the `SZ` column. To determine how much swap space you have, use the `psstat -s` command.
- ❑ Does your window system have enough resources?  
If Versioning or FreezePointing cannot activate a pop-up window, your window system may be running out of resources. Contact your system administrator for help.

## *Reporting Problems*

If you have gone through the checklist and are still having problems, call your local service office. Have the version number of the tool ready to give to the dispatcher. To display the version number of any TeamWare component, type `toolname -V` at the prompt. For example:

```
twversion -V
```

## *Error Messages*

Versioning and FreezePointing display messages to provide you with information or tell you about an error.

## Display Problems

If you experience difficulties in color display in any of the services, you can copy the pertinent part of the resource files in the `app-defaults` directory. These files adjust color, fonts, mnemonics, and several other attributes for each TeamWare component.

These files are under the directory in which you have TeamWare installed, according to the language you are using. If you use CDE the files are in: `TW2.0/lib/locale/<lang>/app-defaults/CDE`. The resource files are listed according to component, preceded by `X`. For example:

```
...TW2.0/lib/locale/C/app-defaults/CDE/XCodeManager
```

If you use OpenWindows or any other non-CDE windowing system, the files are in `...TW2.0/lib/locale/C/app-defaults/non-CDE`.

To make changes to your display:

- 1. Look at the resource file you want to change, and copy the appropriate lines.**

For example, if you want to change the foreground color of the Configuring window, copy the line:

```
XCodeManager*foreground: black
```

- 2. Append the pertinent lines to your own `~/.Xdefaults` file.**

Change it, for example, to:

```
XCodeManager*foreground: green
```

- 3. Type `xrdb < ~/.Xdefaults`**


- 4. Restart the application.**

The window foreground color will be set.





# *Building Programs in Sun Workshop TeamWare*

19 

Sun WorkShop TeamWare provides you with the ability to run one build job at a time or you can run several build jobs concurrently. This chapter shows you how to quickly build a single application and how to fix build errors using the Building window and the WorkShop editor of your choice.

This chapter is organized into the following sections:

<i>Building a TeamWare Target</i>	<i>page 213</i>
<i>Building With Default Values</i>	<i>page 214</i>
<i>Building With NonDefault Values</i>	<i>page 214</i>
<i>Modifying a TeamWare Target</i>	<i>page 215</i>
<i>Fixing Build Errors</i>	<i>page 215</i>

## *Building a TeamWare Target*

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

- build directory
- build command
- makefile
- make target

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

## *Building With Default Values*

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

- 1. From the TeamWare menu bar choose Building, then choose Build ► New Target to open the Define New Target dialog box.**

The Define New Target dialog box contains the value `Default` in the Makefile and Target text fields. If you do not specify a particular makefile or make target, WorkShop looks for a makefile in the build directory named `makefile` or `Makefile` and uses the first make target in that makefile. The build command provided by default is `dmake`. It is set to run in distributed mode so you can run multiple jobs at the same time on the local host. For information on building in distributed mode, see “Running a Distributed Build” in the online help or Chapter 20, “Using DistributedMake.”

- 2. Type the build path in the Directory text field.**

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

- 3. Click Build at the bottom of the dialog box and watch the build output in the Build Output display pane in the Building window.**

The Building window displays the build output when you click the Build button.

Click on the Stop Build button in the Building window to stop the build process.

## *Building With NonDefault Values*

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

- 1. Type the name of the directory you want to build in and click Apply to apply the change.**

You can also select another directory from the Set Build Directory dialog box by clicking the button next to the Directory text field.

**2. Type the name of the makefile you want in the Makefile text field.**

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

**3. Type the name of the make target you want in the Target text field.**

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

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

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

---

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

---

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

## *Modifying a TeamWare Target*

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

## *Fixing Build Errors*

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

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

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

---

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

---

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

The message area of the text editor displays the error message.

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

**1. Click on a highlighted error in the Build Output display.**

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

**2. In the text editor, make sure the source file can be edited.**

From the vi editor, choose Version ► Checkout. From the XEmacs editor, choose Tools ► VC ► Check out File *file*. From the GNU Emacs editor, choose Tools ► Version Control ► Check Out.

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

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

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

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

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

---

**7. Click the Build button in the text editor's tool bar to rebuild.**

You can also build by clicking the Build button in the Building window's tool bar.

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

For detailed information on building in TeamWare, see the online help. You can access the online help by choosing Help ► Building from the Building window or by choosing Help ► Help Contents ► Editing and Building Your Source from the main TeamWare Configuring window.



This chapter describes the way DistributedMake (DMake) distributes builds over several hosts to build programs concurrently over a number of workstations or multiple CPUs.

<i>What You Should Know About DMake Before You Use It</i>	<i>page 223</i>
<i>DMake's Impact on Makefiles</i>	<i>page 223</i>
<i>How to Use DMake</i>	<i>page 229</i>
<i>Controlling DistributedMake Jobs</i>	<i>page 230</i>

### *Basic Concept of Distributed Make*

DistributedMake (DMake) allows you to concurrently distribute the process of building large projects, consisting of many programs, over a number of workstations and, in the case of multiprocessor systems, over multiple CPUs. DMake parses your makefiles and:

- Determines which targets can be built concurrently
- Distributes the build of those targets over a number of hosts designated by you

DMake is a superset of the `make` utility.

To understand DMake, you should know about the following:

- Configuration files
  - Runtime

- Build server
- The DMake host
- The build server

## *Configuration Files*

DMake consults two files to determine to which build servers jobs are distributed and how many jobs can be distributed to each.

### *Runtime Configuration File*

DMake searches for a runtime configuration file on the DMake host to know where to distribute jobs. Generally, this file is in your home directory on the DMake host and is named `.dmakerc`. It consists of a list of build servers and the number of jobs to be distributed to each build server. See “The DMake Host” on page 220 for more information.

### *Build Server Configuration File*

The `/etc/opt/SPROdmake/dmake.conf` file is in the file system of build servers. It is used to specify the maximum total number of DMake jobs that can be distributed to it by all DMake users. See “The Build Server” on page 223 for more information.

## *The DMake Host*

DMake searches for a runtime configuration file to know where to distribute jobs. Generally, this file must be in your home directory on the DMake host and is named `.dmakerc`. DMake searches for the runtime configuration file in these locations and in the following order:

1. The path name you specify on the command line using the `-c` option
2. The path name you specify using the `DMAKE_RCFILE` makefile macro
3. The path name you specify using the `DMAKE_RCFILE` environment variable
4. `$(HOME)/.dmakerc`



---

If a runtime configuration file is not found, DMake distributes two jobs to the DMake host. You edit the runtime configuration file so that it consists of a list of build servers and the number of jobs you want distributed to each build server. The following is an example of a `.dmakerc` file:

```
# My machine. This entry causes dmake to distribute to it.
falcon { jobs = 1 }
hawk
eagle { jobs = 3 }
# Manager's machine. She's usually at meetings
heron { jobs = 4 }
avocet
```

- The entries: falcon, hawk, eagle, heron, and avocet are listed build servers.
- You can specify the number of jobs you want distributed to each build server. The default number of jobs is two.
- Any line that begins with the “#” character is interpreted as a comment.

---

**Note** – This list of build servers includes falcon which is also the DMake host. The DMake host can also be specified as a build server. If you do not include it in the runtime configuration file, no DMake jobs are distributed to it.

---

You can also construct groups of build servers in the runtime configuration file. This provides you with the flexibility of easily switching between different groups of build servers as circumstances warrant. For instance you may define a different group of build servers for builds under different operating systems, or on groups of build servers that have special software installed on them.

The following is an example of a runtime configuration file that contains groups of build servers:

```

earth  { jobs = 2 }
mars   { jobs = 3 }

group lab1 {
    host falcon{ jobs = 3 }
    host hawk
    host eagle { jobs = 3 }
}

group lab2 {
    host heron
    host avocet{ jobs = 3 }
    host stilt { jobs = 2 }
}

group labs {
    group lab1
    group lab2
}

group sunos5.x {
    group labs
    host jupiter
    host venus{ jobs = 2 }
    host pluto { jobs = 3 }
}

```

- Formal groups are specified by the “group” directive and lists of their members are delimited by braces ({}).
- Build servers that are members of groups are specified by the optional “host” directive.
- Groups can be members of other groups.
- Individual build servers can be listed in runtime configuration files that also contain groups of build servers; in this case DMake treats these build servers as members of the *unnamed* group.

In order of precedence, DMake distributes jobs to:

1. The formal group specified on the command-line as an argument to the `-g` option
2. The formal group specified by the `DMAKE_GROUP` makefile macro
3. The formal group specified by the `DMAKE_GROUP` environment variable
4. The first group specified in the runtime configuration file.

### *The Build Server*

The `/etc/opt/SPROdmake/dmake.conf` file is in the file system of build servers. Use this file to limit the maximum total number of DMake jobs (from all users) that can run concurrently on a build server. The following is an example of an `/etc/opt/SPROdmake.conf` file. This file sets the maximum number of DMake jobs permitted to run on a build server (from all DMake users) to be eight.

```
jobs: 8
```

---

**Note** – If the `/etc/opt/SPROdmake.conf` file does not exist on a build server, no DMake jobs will be allowed to run on that server.

---

### *What You Should Know About DMake Before You Use It*

To use DMake, you use the executable file (`dmake`) in place of the standard `make` utility. You should understand the Solaris `make` utility before you use DMake. If you need to read more about the `make` utility see the *Programming Utilities Guide* in the *Solaris 2.5 Software Developer AnswerBook* documentation set. If you use the `make` utility, the transition to DMake require little if any alteration.

### *DMake's Impact on Makefiles*

The methods and examples shown in this section present the kinds of problems that lend themselves to solution with DMake. This section does not suggest that any one approach or example is the best. Compromises between clarity and functionality were made in many of the examples.

As procedures become more complicated, so do the makefiles that implement them. You must know which approach will yield a reasonable makefile that works. The examples in this section illustrate common code-development predicaments and some straightforward methods to simplify them using DMake.

### *Using Makefile Templates*

If you use a makefile template from the outset of your project, custom makefiles that evolve from the makefile templates will be:

- More familiar
- Easier to understand
- Easier to integrate
- Easier to maintain
- Easier to reuse

The less time you spend editing makefiles, the more time you have to develop your program or project.

### *Building Targets Concurrently*

Large software projects typically consist of multiple independent modules that can be built concurrently. DMake supports concurrent processing of targets on a multiple machines over a network. This concurrency can markedly reduce the time required to build a large project.

When given a target to build, DMake checks the dependencies associated with that target, and builds those that are out of date. Building those dependencies may, in turn, entail building some of their dependencies. When distributing jobs, DMake starts every target that it can. As these targets complete, DMake starts other targets. Nested invocations of DMake are not run concurrently by default, but this can be changed (see “Restricting Parallelism” on page 228 for more information).

Since DMake builds multiple targets concurrently, the output of each build is produced simultaneously. To avoid intermixing the output of various commands, DMake collects output from each build separately. DMake displays the commands before they are executed. If an executed command generates

any output, warnings, or errors, DMake displays the entire output for that command. Since commands started later may finish earlier, this output may be displayed in an unexpected order.

### *Limitations on Makefiles*

Concurrent building of multiple targets places some restrictions on makefiles. Makefiles that depend on the implicit ordering of dependencies may fail when built concurrently. Targets in makefiles that modify the same files may fail if those files are modified concurrently by two different targets. Some examples of possible problems are discussed in this section.

### *Dependency Lists*

When building targets concurrently, it is important that dependency lists be accurate. For example, if two executables use the same object file but only one specifies the dependency, then the build may cause errors when done concurrently. For example, consider the following makefile fragment:

```
all: prog1 prog2
prog1: prog1.o aux.o
    $(LINK.c) prog1.o aux.o -o prog1
prog2: prog2.o
    $(LINK.c) prog2.o aux.o -o prog2
```

When built serially, the target `aux.o` is built as a dependent of `prog1` and is up-to-date for the build of `prog2`. If built in parallel, the link of `prog2` may begin before `aux.o` is built, and is therefore incorrect. The `.KEEP_STATE` feature of `make` detects some dependencies, but not the one shown above.

### *Explicit Ordering of Dependency Lists*

Other examples of implicit ordering dependencies are more difficult to fix. For example, if all of the headers for a system must be constructed before anything else is built, then everything must be dependent on this construction. This causes the makefile to be more complex and increases the potential for error when new targets are added to the makefile. The user can specify the special target `.WAIT` in a makefile to indicate this implicit ordering of dependents. When DMake encounters the `.WAIT` target in a dependency list, it finishes processing all prior dependents before proceeding with the following

dependents. More than one `.WAIT` target can be used in a dependency list. The following example shows how to use `.WAIT` to indicate that the headers must be constructed before anything else.

```
all: hdrs .WAIT libs functions
```

You can add an empty rule for the `.WAIT` target to the makefile so that the makefile is backward-compatible.

### ***Concurrent File Modification***

You must make sure that targets built concurrently do not attempt to modify the same files at the same time. This can happen in a variety of ways. If a new suffix rule is defined that must use a temporary file, the temporary file name must be different for each target. You can accomplish this by using the dynamic macros `$$` or `$$*`. For example, a `.c.o` rule which performs some modification of the `.c` file before compiling it might be defined as:

```
.c.o:
    awk -f modify.awk $*.c > $*.mod.c
    $(COMPILE.c) $*.mod.c -o $*.o
    $(RM) $*.mod.c
```

### ***Concurrent Library Update***

Another potential concurrency problem is the default rule for creating libraries that also modifies a fixed file, that is, the library. The inappropriate `.c.a` rule causes DMake to build each object file and then archive that object file. When DMake archives two object files in parallel, the concurrent updates will corrupt the archive file.

```
.c.a:
    $(COMPILE.c) -o $$ $<
    $(AR) $(ARFLAGS) $@ $$
    $(RM) $$
```

A better method is to build each object file and then archive all the object files after completion of the builds. An appropriate suffix rule and the corresponding library rule are:

```
.c.a:
    $(COMPILE.c) -o $% $<

lib.a: lib.a($(OBJECTS))
    $(AR) $(ARFLAGS) $(OBJECTS)
    $(RM) $(OBJECTS)
```

### ***Multiple Targets***

Another form of concurrent file update occurs when the same rule is defined for multiple targets. An example is a `yacc(1)` program that builds both a program and a header for use with `lex(1)`. When a rule builds several target files, it is important to specify them as a group using the `+` notation. This is especially so in the case of a parallel build.

```
y.tab.c y.tab.h: parser.y
    $(YACC.y) parser.y
```

This rule is actually equivalent to the two rules:

```
y.tab.c: parser.y
    $(YACC.y) parser.y
y.tab.h: parser.y
    $(YACC.y) parser.y
```

The serial version of `make` builds the first rule to produce `y.tab.c` and then determines that `y.tab.h` is up-to-date and need not be built. When building in parallel, `DMake` checks `y.tab.h` before `yacc` has finished building `y.tab.c` and notices that it *does* need to be built, it then starts another `yacc` in parallel with the first one. Since both `yacc` invocations are writing to the same files (`y.tab.c` and `y.tab.h`), these files are apt to be corrupted and incorrect. The correct rule uses the `+` construct to indicate that both targets are built simultaneously by the same rule. For example:

```
y.tab.c + y.tab.h: parser.y
    $(YACC.y) parser.y
```

### *Restricting Parallelism*

Sometimes file collisions cannot be avoided in a makefile. An example is `xstr(1)`, which extracts strings from a C program to implement shared strings. The `xstr` command writes the modified C program to the fixed file `x.c` and appends the strings to the fixed file `strings`. Since `xstr` must be run over each C file, the following new `.c.o` rule is commonly defined:

```
.c.o:
    $(CC) $(CPPFLAGS) -E $*.c | xstr -c -
    $(CC) $(CFLAGS) $(TARGET_ARCH) -c x.c
    mv x.o $*.o
```

DMake cannot concurrently build targets using this rule since the build of each target writes to the same `x.c` and `strings` files, nor is it possible to change the files used. You can use the special target `.NO_PARALLEL:` to tell DMake not to build these targets in concurrently. For example, if the objects being built using the `.c.o` rule were defined by the `OBJECTS` macro, the following entry would force DMake to build those targets serially:

```
.NO_PARALLEL: $(OBJECTS)
```

If most of the objects must be built serially, it is easier and safer to force all objects to default to serial processing by including the `.NO_PARALLEL:` target without any dependents. Any targets that can be built in parallel can be listed as dependencies of the `.PARALLEL:` target:

```
.NO_PARALLEL:
.PARALLEL: $(LIB_OBJECT)
```

### *Nested Invocations of DistributedMake*

When DMake encounters a target that invokes another DMake command, it builds that target serially, rather than concurrently. This prevents problems where two different DMake invocations attempt to build the same targets in the same directory. Such a problem might occur when two different programs are built concurrently, and each must access the same library. The only way for each DMake invocation to be sure that the library is up-to-date is for each to invoke DMake recursively to build that library. DMake only recognizes a nested invocation when the `$(MAKE)` macro is used in the command line.



If you nest commands that you know will not collide, you can force them to be done in parallel by using the `.PARALLEL:` construct.

When a makefile contains many nested commands that run concurrently, the load-balancing algorithm may force too many builds to be assigned to the local machine. This may cause high loads and possibly other problems, such as running out of swap space. If such problems occur, allow the nested commands to run serially.

## How to Use DMake

You execute `dmake` on a *DMake host* and distribute jobs to *build servers*. You can also distribute jobs to the DMake host, in which case it is also considered to be a build server. DMake distributes jobs based on makefile targets that DMake determines (based on your makefiles) can be built concurrently. You can use any machine as a build server that meets the following requirements:

- From the DMake host (the machine you are using) you must be able to use `rsh`, without being prompted for a password, to remotely execute commands on the build server. See `man rsh(1)` or the system AnswerBook for more information about the `rsh` command. For example:

```
demo% rsh build_server which dmake
/opt/SUNWspro/bin/dmake
```

- The `bin` directory in which the DMake software is installed must be accessible from the build server. See the `share (1M)` and `mount (1M)` man pages or the system AnswerBook for more information.
- The `bin` directory in which the DMake software is installed must be in your execution path when you `rsh` to the build server. Be sure this directory is added to the `PATH` variable in your `.cshrc` file (or equivalent), *not* in your `.login` file. You can verify this as follows:

```
demo% rsh build_server which dmake
/opt/SUNWspro/bin/dmake
```

- The source hierarchy you are building must be:
  - accessible from the build server

- mounted under the same name

From the DMake host you can control which build servers are used and how many DMake jobs are allotted to each build server. The number of DMake jobs that can run on a given build server can also be limited on that server.

### *Notes*

- If you specify the `-m` option with the “parallel” argument, or set the `DMAKE_MODE` variable or macro to the value “parallel,” DMake does not scan your runtime configuration file. Therefore, you must specify the number of jobs using the `-j` option or the `DMAKE_MAX_JOBS` variable/macro. If you do not specify a value this way, a default of two jobs is used.
- If you modify the maximum number of jobs using the `-j` option, or the `DMAKE_MAX_JOBS` variable/macro when using DMake in distributed mode (DMake default, or specified either by option, variable or macro), the value you specify overrides the values listed in the runtime configuration file. The value you specify is used as the total number of jobs that can be distributed to all build servers.

### *Controlling DistributedMake Jobs*

The distribution of DMake jobs is controlled in two ways:

1. A DMake user on a DMake host can specify the machines they want to use as build servers and the number of jobs they want to distribute to each build server.
2. The “owner” on a build server can control the maximum total number of DMake jobs that can be distributed to that build server. The owner is a user that can alter the `/etc/opt/SPROdmake/dmake.conf` file.

---

**Note** – If you access DMake from the GUI (Building) use the online help to know how to specify your build servers and jobs. If you access DMake from the CLI see the DMake man page (`dmake.1`).

---

## *Getting Help on the GUI or the CLI*

DMake is fully implemented in both the GUI and CLI. To use DMake from the GUI, see the Sun WorkShop TeamWare online help.

### *To Access the Online Help*

- 1. Open any Sun WorkShop TeamWare GUI from the Sun WorkShop.**
- 2. Or, open any Sun WorkShop TeamWare from the command line. For example, to open the Configuring GUI, enter the following:**

```
demo% twconfig &
```

- 3. Open the pull-down menu from the Help button.**
- 4. Click on Help Contents.**

### *To Access Help for the CLI*

- ◆ To access the manual page for information on how to use DMake from the CLI, enter the following. The manual page gives information on all command-line options, variables, and macros necessary to use DMake.**

```
demo% man dmake.1
```



This chapter explores the post-build process and indicates where further help can be found in handling release and integration matters.

<i>Performing Master Builds</i>	<i>page 233</i>
<i>Performing Releases</i>	<i>page 234</i>

Before you begin, make sure you have read Chapter 5, “Starting a Project” —“Product Release Considerations” on page 47

### *Integrating Your Changes*

The online Help gives a brief overview of this subject. For more background information on how a specific environment handles the question, refer to: *Sun WorkShop TeamWare: Solutions Guide*.

For questions about resolving differences between files see Chapter 9, “Resolving Conflicts.” You can reverse a bringover or putback with Undo when conflicts occur, and review a file history to determine at which point any degradation in functionality may have occurred.

### *Performing Master Builds*

A Master Build is created subsequent to all contributing developers having putback clean files into the common integration workspace.

### *Establishing Nightly Builds*

If you are the build master, the person responsible for running nightly and master builds of the source, you determine if you will run your builds in the main integration area or in a child workspace. Refer to *Sun WorkShop TeamWare: Solutions Guide* for a description of how a specific environment handled these questions.

### *Performing Releases*

The release process demands control of source code and orderly hierarchical cutbacks. You can organize a release to follow the “train” process as explained in the next section.

### *Organizing a Release*

When a series of overlapping software releases is needed, the “train” theory can organize the projects. In principal, the release train leaves the station at designated intervals, with whatever features are ready at the time.

Developers working in their private workspaces put back code to the integration workspace, never to the train directly. Gate-keepers ensure the putbacks to the train from the integration area are without error. Versioning helps keep control of the changes as the release proceeds.

For an example of how the release process works in a specific environment see *Sun WorkShop TeamWare: Solutions Guide*.

### *How the Release Process Works*

When you reach a milestone in your source code development project, you can use FreezePointing to create a snapshot, or freezepoint of your project. You can later use FreezePointing to recreate the source hierarchy. Refer to Chapter 17, “Introduction to FreezePointing” for a full explanation of how FreezePointing can help in the release cycle.

## Glossary

---



### **Access control**

The Configuring facility by which users can control access to workspaces by Configuring commands.

### **Branch (SCCS)**

A delta or series of deltas that are placed off of the main line of deltas in an SCCS history file.

### **Bringover Create**

The transaction used to copy groups of files from a parent workspace to a nonexistent child workspace. The new child workspace is created as a result of the transaction. All Configuring transfer transactions are performed from the perspective of the child workspace; hence the Bringover Create transaction “brings over” files to the child from the parent workspace. See also *Bringover Update*, *Workspace* and *Putback*.

### **Bringover Update**

The transaction used to update an existing child workspace with respect to files contained in its parent workspace. All Configuring transfer transactions are performed from the perspective of the child workspace; the Bringover Update transaction “brings over” files to the child from the parent workspace. See also *Bringover Create*, *Workspace*, and *Putback*.



---

### **Child workspace**

A workspace that has a parent workspace listed in its `Codemgr_wsdata/parents` file. Development work is typically done in child workspaces and put back to parent workspaces after it has been tested. The Configuring transfer transactions are viewed from the child workspace perspective, and all conflicts are resolved in the child workspace.

### **Codemgr\_wsdata directory**

Every TeamWare workspace contains a “metadata” directory in its root directory named `Codemgr_wsdata`. Configuring stores data about the workspace in `Codemgr_wsdata`. The presence of this directory is the sole factor that defines it as a TeamWare workspace (as opposed to a normal directory). Configuring commands use the presence or absence of this directory to determine whether a directory is a workspace. All data stored in the `Codemgr_wsdata` directory is contained in flat ASCII text files that can be edited by users. See Section , “The Workspace Metadata Directory,” on page 62 for more information.

### **Conflict**

The condition that exists when a file has changed in both the child and parent workspace. Conflicts are identified by the Bringover Update transaction and are resolved by using the Resolve transaction.

### **Copy-Modify-Merge**

The concurrent development model upon which Configuring is based. Using this model, multiple developers concurrently *copy* sources from a common area, *modify* the source in isolation, and then *merge* those changes with changes made by other developers.

### **Create**

Used in Configuring transaction output. Files are said to be created if they exist in the source workspace and not in the destination workspace, and are copied into the destination workspace as part of a Bringover or Putback transaction.

### **Default line of work**

The branch in an SCCS history file upon which the next delta will be added.

### **def.dir.flp**

The default FLP shipped with Configuring is `def.dir.flp`; this FLP recursively descends directory hierarchies and lists all files for which SCCS history files exist. See *FLP*.





---

**Delta**

The set of differences between two versions of a file checked into SCCS. When you check in a file, SCCS records only the line-by-line differences between the text you check in and the previous *version* of the file. This set of differences is known as a *delta*. The file version that you initially checked out was constructed from a set of accumulated deltas. The terms *delta* and *version* are often used synonymously; however, their meanings are not the same. It is possible to retrieve a version that omits selected deltas. See *Version*.

**Merging**

The TeamWare utility used to merge deltas during Resolve transactions. See Chapter 9, “Resolving Conflicts,” and the section on Merging.

**FLP**

An FLP or *File List Program* is a program or script that generates a list of files to `stdout` that Configuring then processes during Bringover and Putback transactions. See *def.dir.flp*.

**FreezePointing**

The TeamWare utility used to make snapshots of workspaces (or portions of them) at important junctures or “freezepoints.”

**g-file (SCCS)**

The working copy of a file retrieved from an SCCS history file by the `sccs-get` command.

**History Files**

When you initially put a file under SCCS control, a history file is created for the new SCCS file. The initial version of the history file uses the complete text of the source file. The initial history file is the file that further deltas are compared to. Owing to its prefix (`s.`), the history file is often referred to as the *s.file* (*s-dot-file*).

**Integration workspace**

A workspace to which multiple developers put back (merge) their work.

**Lock**

To assure consistency, the Configuring file transfer transactions Bringover and Putback lock workspaces while they are working in them. Locks are recorded in the `Codemgr_wsdata/lock` file in each workspace; the Configuring commands consult that file before acting in a workspace. See *read-lock*, *write-lock*.



---

**Merge**

To produce a single version of a file from two conflicting files (deltas). Usually accomplished with the assistance of the Merging program.

**Notification**

A Configuring facility that mails notice of events, such as changes to files or directories, to users.

**Parent workspace**

A workspace that has a child workspace(s) listed in its `Codemgr_wsdata/children` file. Parent workspaces are typically used as integration areas, since development, testing, and conflict resolution occur in child workspaces.

**ParallelMake**

The program distributed as part of TeamWare that enables program builds to be parallelized over multiple processes and CPUs. See the section on ParallelMake.

**Putback**

The transaction used to update a parent workspace with respect to files contained in its child workspace. All Configuring transfer transactions are performed from the perspective of the child workspace; the Putback transaction “puts back” files to the parent from the child workspace. See also *Bringover Create*, *Bringover Update*, and *Workspace*.

**Read-lock**

A lock that is obtained by a Configuring command while it examines the contents of a workspace. A read-lock assures that the workspace does not change while the command is examining files in a workspace. Read-locks may be obtained concurrently by a number of commands; no Configuring command may write to the workspace while a read-lock is in force. See *lock*, *write-lock*.

**Reparent**

To change the parent of a child workspace.

**Resolve**

To produce a new delta of a file from two conflicting deltas. See *merged*, *conflict*.



---

<b>Root directory</b>	The top-level directory of a Configuring workspace. This directory's path name is the name by which the workspace is referred.
<b>SCCS Delta ID (SID)</b>	A SID is the number used to represent a specific delta. This is a two-part number, with the parts separated by a dot (.). The SID of the initial delta is 1.1 by default. The first part of the SID is referred to as the <i>release</i> number, and the second, the <i>level</i> number. When you check in a delta, the level number is incremented automatically.
<b>SCCS history file</b>	The file that contains a given file's delta history; also referred to as an "s-dot-file." All SCCS history files must be located in a directory named SCCS, which is located in the same directory as the <i>g-file</i> . See <i>g-file</i> .
<b>SID</b>	SCCS delta ID—The number used to represent a specific SCCS delta.
<b>Undo</b>	To return a workspace to the state it was in before the most recent Bringover or Putback transaction, thereby "undoing" the action of the transaction.
<b>Update</b>	Files are said to be updated during a Bringover or Putback transaction if they exist in both the source workspace and in the destination workspace, and have changed in the source workspace. The SCCS history file in the destination workspace is updated with new deltas from the source workspace.
<b>Version</b>	When you check in a file, SCCS records only the line-by-line differences between the text you check in and the previous <i>version</i> of the file. This set of differences is known as a <i>delta</i> . The file version that you initially checked out was constructed from a set of accumulated deltas. The terms <i>delta</i> and <i>version</i> are often used synonymously; however, their meanings are not the same. It is possible to retrieve a version that omits selected deltas. See <i>Delta</i> .
<b>Versioning</b>	The TeamWare program that provides a graphical interface to SCCS. See the section on Versioning.



## **Workspace**

A workspace is a specially designated (but standard) directory and its subdirectory hierarchy. Usually each developer on a project works in their own isolated workspace concurrently with other developers programming in other workspaces. Configuring provides utilities to “intelligently” copy files from workspace to workspace.

## **Workspace hierarchy**

A hierarchy of parent and child workspaces in which programmers and release engineers can develop, test, share, and release software products.

## **Write-lock**

A lock that is obtained by a Configuring command that changes data in a workspace. Only one write-lock may be obtained for a workspace at any time. When a write-lock is in force, only the Configuring command that owns the lock may write to the workspace; other commands cannot obtain read-locks from the workspace. See *lock*, *read-lock*.

# Index

---

## Symbols

`.NO_PARALLEL`: special target, 228  
`.PARALLEL`: special target, 228  
`.WAIT` special target, 225  
`~/ .codemgr_resrc`, 57  
`~/ .codemgrtoolrc`, 58

## A

access control, workspace, 71  
`access_control` file, 62, 71  
adjusting color, 211  
Ancestor File, 183  
Ancestor file, 191  
ancestor file, 35  
    loading at startup, 187  
archiving libraries, 226  
`args` file, 62, 87  
Auto Advance, 189  
Auto Bringover option, 102  
automatic merging, 190  
automerging algorithm, 38

## B

backup directory, 62, 106  
branch delta, 49

branches, 49  
branching, SCCS, 136  
Bringover Create transaction, 64, **90 to 145**  
    effect of checked out files, 92  
    file system accessibility, 93  
    Force Conflicts option, 92, 96  
    path name specification, 93  
    Preview option, 91, 96, 101  
    Quiet option, 92, 96, 102  
    search path, 94  
    Verbose option, 91, 96, 101  
    workspace locks, 93  
Bringover Transaction, **89 to 99**  
Bringover Update transaction, **94 to 99**  
    action summary table, 99  
    conflict detection during, 116  
    effect of checked-out files, 96  
    file system accessibility, 97  
    path names, 97  
    workspace locks, 98  
Bringover/Putback transaction  
    introduction, 27  
Build button, 217  
building with default values, 214  
building with nondefault values, 214

---

## C

- checked-out files, 96, 102
- children file, 62
- chooser, 88
- CLI, command-line interface
  - umbrella command, 52
- CODEMGR\_WS variable, 82
- Codemgr\_wsdata, 25, 62
  - access\_control file, 62
  - args file, 62, 87
  - backup directory, 62, 106
  - children file, 62
  - history file, 63
  - locks file, 63
  - nametable file, 63
  - notification file, 63, 75
  - parent file, 63
- CODEMGR\_WSPATH variable, 82
- color display, to adjust, 211
- command-line interface
  - umbrella command, 52
- command-line options, 184
- Commands button
  - checking in a new file, 176
  - checking in files, 175
  - checking out files, 175
  - displaying the differences between deltas, 177
  - editing checked out files, 176
  - unchecking out a file, 177
- comment
  - Putback transaction, 102
- common ancestor delta, 117
- concurrent file modification, 226
- Configuring, 165
  - base window, 55
  - Chooser, 88
  - control area, 57
  - customization, 57
  - menus, 57
  - moving an existing project, 123
  - properties, 57
  - starting execution, 53

- transaction model, 83
- Workspace Graph pane, 55
- conflict
  - detection during Bringover, 116
  - merging files in conflict, 133
- copy-modify-merge
  - example, 18
  - model, 16
- creating a workspace, 64
- Current difference
  - defined, 36
- current difference (Merging), 120
- Current Working Directory, 183

## D

- def.dir.flp, 86
- default build values, 214
- default list
  - loading, 87
  - saving, 87
- defaults files, 57
  - ~/.codemgr\_resrc, 57
  - ~/.codemgrtoolrc, 58
- Define New Target dialog box, 214
- Delete, 64
  - Codemgr\_wsdata Directory only, 65
  - Sources and Codemgr\_wsdata Directory, 65
- delta, 237, 239
- delta ID, 239
- dependency lists, 225
  - explicit ordering, 225
  - implicit ordering, 225
- descendant, 35
- Descendant file, 192
- Difference
  - current, 36
  - defined, 36
  - next, 36
  - previous, 36
  - remaining, 37
  - resolved, 36

---

- resolving, 189
- difference (Merging)
  - current, 120
  - defined, 120
  - next, 120
  - previous, 120
  - resolved, 120
- distributed make, explanation of, 219
- Dmake, basic concept, 219
- double-click action
  - Workspace Graph pane, 57

## E

- editing WorkShop targets, 215
- environment variables, 81
- example, 190
- examples
  - Bringover/Putback/Resolve cycle, 143 to 147
  - merging SCCS history files, 134
  - reparenting, 68

## F

- file
  - collision, 228
  - concurrent modification, 226
  - loading, 182
  - loading from list, 187
  - loading three files at startup, 187
  - loading two files at startup, 186
- file chooser, 88
- File List pane
  - changing contents of, 87
  - constructing directory and file lists, 86
  - selecting files, 87
- File List Program, see FLP
- file lists
  - initial state, 86
  - transactions, 86
- File menu, 57
- files

- merging, 133
  - relationships between files in parent and child workspaces, 27 to ??
  - specifying for transactions, 85
- fixing build errors, 215
- FLP, 86
  - default (`def.dir.flp`), 86
- fonts, changing, 211
- Force Conflicts option, 92, 96
- FreezePointing
  - creating a freeze point file, 201
  - extract a source directory, 202
- FreezePointing terms, 199
  - extract, 200
  - freeze point file, 199

## G

- glyphs, meaning of, 37, 193
- graphical user interface (GUI),
  - overview, 53
- grouping files

## H

- HELPPATH environment variable, 210
- hierarchy, workspace, ?? to 48, 124 to 129
- history file, 237
- history file, 63, 78

## I

- icons
  - drag and drop, 56

## L

- Left File, 183
- library update, concurrent, 226
- limitations on makefiles, 225
- list file, 187
- Load

---

- workspaces into Workspace Graph pane, 55
- Load button, 183
- loading
  - three files at startup, 187
  - two files at startup, 186
- loading files, 182
- locking workspaces, 80
- locks
  - removing workspace locks, 81
  - viewing workspace locks (GUI), 81
- locks file, 63

## M

- macro
  - dynamic, 226
- makefiles, limitations, 225
- manual pages, SunOS, xviii
- matriarch, 191
- menu buttons
  - File, 57
- menus
  - Configuring, 57
- merging files
  - not in conflict, 132
- Merging program, 119 to 121
- merging SCCS history files, 32
  - example, **134**
  - in conflict, **133**
- Merging window, 37
- merging, starting, 192
- metadata directory, 25
- minus sign, 38
- mkfile command, 210
- moving to the next build error, 216
- multiple targets, 227

## N

- name fields, workspace, 55
- nametable file, 63
- Next difference

- defined, 36
- next difference (Merging), 120
- Next Error button, 216
- nodes, 49
- notification, 75
- notification file, 63

## O

- Online Help, 53
- Output File, 183
- output file, 194
- output, transaction, 85

## P

- parallelism
  - restricting, 228
- Parent, 67
- parent file, 63
- parent/child introduction, 22 to 25
- plus sign, 38
- Preview option, 91, 96, 101
- Previous difference
  - defined, 36
- previous difference (Merging), 120
- project
  - moving an existing project, 41, 123
- Properties window, 57
- Props button
  - changing double click action, 178
  - changing file list display, 177
  - changing history graph display, 179
  - changing history information display, 179
  - defining an editor, 178
- Putback, 104
- Putback transaction, **99 to 104**
  - access control, 104
  - action summary table, 104
  - Auto Bringover option, 102
  - comment, 102
  - effect of checked-out files, 102



---

- file system accessibility, 103
- path names, 103
- workspace locks, 103

Putback/Bringover transactions,  
introduction, 27

## Q

Quiet option, 92, 96, 102

## R

Remaining difference  
defined, 37

Rename, 65

reparenting a workspace, 25, **66 to 70**  
example, 68

Resolve transaction, 118  
introduction, 33  
merging SCCS history files, 137  
preparing files for conflict  
resolution, 117

Resolved difference  
defined, 36

resolving differences, 189

restricting parallelism, 228

restrictions on makefiles, 225

Right File, 183

## S

s.file, 237

Save As, 188

SCCS, xiv, 48

- branches, 49
- delta, 237, 239
- delta ID, 239
- history file, 237
- nodes, 49
- version, 237, 239

SCCS history files, 19, 26, 85, 100, 131

- branching, 136
- common ancestor delta, 117
- merging, 32, **131**

- resolving, 117
- s-dot-file, 237
- selecting files, transactions, 87
- Set Default button, 57
- SID, 239
- Source Code Control System, xiv, 48
- specifying a build command, 214
- specifying a build directory, 214
- specifying a make target, 214
- specifying a makefile, 214
- spot help, see Online Help
- starting a project
  - default FLP, 44, 124
  - SCCS file location, 123
- starting Merging, 192
  - from Sun WorkShop, 182
  - from Sun WorkShop TeamWare, 182
  - from the command line, 184
- Sun WorkShop, 182
- Sun WorkShop TeamWare window, 182
- swap space, 210

## T

targets

- .NO\_PARALLEL:, 228
- .PARALLEL:, 228
- .WAIT, 225
- multiple, 227

TeamWare

- moving an existing project, 41

Tools menu, 204

transaction model, 83

Transaction Output window, 85

transactions

- file lists, 86
- file specification, 85

tutorial, 190

twconfig, 53

twmerge command, 184

---

## U

- Undo transaction, **105 to 108**
  - implementation, 106
  - workspace locks, 106
- unlocking workspaces, 80

## V

- variables, environment, 81
  - CODEMGR\_WS, 82
  - CODEMGR\_WSPATH, 82
- Verbose option, 91, 96, 101
- version, 237, 239
- version number, displaying, 210
- Versioning, 49
- vertical bar, 37, 38
- viewing the source of a build error, 216

## W

- window
  - Merging at startup, 37
  - showing three input files loaded, 38
  - showing two input files loaded, 37
  - Sun WorkShop TeamWare, 182
- workspace
  - access control, 71, 94, 98
  - command history log, 78
  - create, 64
  - create using Bringover Create, 64
  - delete, 64
  - event notification, 75
  - hierarchies, 23
  - hierarchy configuration, 48, 124, 129
  - locking, 80
  - metadata directory (Codemgr\_  
wsdata), 62
  - moving, 65
  - name fields, 55
  - removing locks, 81
  - renaming, 65
  - reparenting, **66 to 70**
  - viewing locks from GUI, 81
- Workspace Create, 64

- Workspace Graph pane, 55
  - double-click action, 57
  - loading workspaces, 55
- workspace introduction, 20 to 22

## X

- Xdefaults file, 211



Copyright 1996 Sun Microsystems Inc., 2550 Garcia Avenue, Mountain View, Californie 94043-1100, U.S.A. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou de sa documentation associée ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a.

Des parties de ce produit pourront être dérivées du système UNIX® licencié par Novell, Inc. et du système Berkeley 4.3 BSD licencié par l'Université de Californie. UNIX est une marque enregistrée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company Ltd. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Sun, Sun Microsystems, le logo Sun, SunOS, Sun WorkShop, [et Sun WorkShop TeamWare](#), sont des marques déposées ou enregistrées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC, utilisées sous licence, sont des marques déposées ou enregistrées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

Les interfaces d'utilisation graphique OPEN LOOK® et Sun™ ont été développées par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant aussi les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

Le système X Window est un produit de X Consortium, Inc.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" SANS GARANTIE D'AUCUNE SORTE, NI EXPRESSE NI IMPLICITE, Y COMPRIS, ET SANS QUE CETTE LISTE NE SOIT LIMITATIVE, DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DES PRODUITS A RÉPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ILS NE SOIENT PAS CONTREFAISANTS DE PRODUITS DE TIERS.