

*SPARCworks/TeamWare
ProWorks/TeamWare
Users Guide*

 *SunSoft*
A Sun Microsystems, Inc. Business
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A

© 1995 Sun Microsystems, Inc.
2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A.

All rights reserved. This product and related documentation are protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or related documentation 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[®] and Berkeley 4.3 BSD systems, licensed from UNIX System Laboratories, Inc., a wholly owned subsidiary of Novell, Inc., and the University of California, respectively. Third-party font software in this product is protected by copyright and licensed from Sun's font suppliers.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the United States Government is subject to the restrictions set forth in DFARS 252.227-7013 (c)(1)(ii) and FAR 52.227-19.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

TRADEMARKS

Sun, the Sun logo, Sun Microsystems, Sun Microsystems Computer Corporation, Solaris, are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and certain other countries. UNIX is a registered trademark of Novell, Inc., in the United States and other countries; X/Open Company, Ltd., is the exclusive licensor of such trademark. OPEN LOOK[®] is a registered trademark of Novell, Inc. PostScript and Display PostScript are trademarks of Adobe Systems, Inc. . All other product names mentioned herein are the trademarks of their respective owners.

All SPARC trademarks, including the SCD Compliant Logo, are trademarks or registered trademarks of SPARC International, Inc. SPARCstation, SPARCserver, SPARCengine, SPARCstorage, SPARCware, SPARCcenter, SPARCclassic, SPARCcluster, SPARCdesign, SPARC811, SPARCprinter, UltraSPARC, microSPARC, SPARCworks, ProWorks, and SPARCcompiler are licensed exclusively to Sun Microsystems, Inc. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun[™] 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 product of the Massachusetts Institute of Technology.

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.

THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS PUBLICATION AT ANY TIME.



Contents

Preface.....	xiii
<i>Part 1 —Quickstart Guide</i>	
1. QuickStart Guide	1
CodeManager.....	2
Some CodeManager Concepts	2
Parent and Child Workspaces	3
Source Code Control System	3
VersionTool.....	9
FreezePoint.....	12
ParallelMake.....	14
<i>Part 2 —CodeManager</i>	
2. Introduction to CodeManager	17
Coordinating the Work of Software Developers	17
Copy-Modify-Merge Model	18
Default CodeManager.....	21

Workspace	21
Copying Files between Workspaces	26
Resolving Conflicts	34
3. CodeManager User Interfaces.	37
CodeManager Command-Line Interface	38
CodeManager Graphical User Interface	39
About This Section.	39
Starting Up CodeManager	40
CodeManager Windows	40
CodeManager File and Directory Choosers	41
CodeManager Window	42
Transactions Window	51
Customizing CodeManager Using Properties	57
Footer Messages.	62
Accelerators	63
4. CodeManager Workspace	65
The Workspace Metadata Directory	65
Creating a Workspace	68
Using Workspace Create	68
Using the Bringover Create Transaction	69
Deleting a Workspace	69
Moving and Renaming a Workspace	70
A Note About Moving Workspaces.	70
Reparenting a Workspace	71

Two Ways to Reparent Workspaces	71
Reasons to Change a Workspace's Parent.	73
Controlling Access to Workspaces.	77
Viewing and Changing Access Control Values	80
How to Notify Users of Changes to Workspaces.	83
Viewing and Changing Notification Entries	84
Notes About Registering Notification Events	87
Viewing Workspace Command History	87
Ensuring Consistency through Workspace Locking	90
CodeManager Environment Variables	92
The CODEMGR_WS Variable	92
The CODEMGR_WSPATH Variable	93
5. Copying Files between Workspaces	95
CodeManager Transaction Model	95
General File Copying Information.	97
SCCS History Files.	97
Viewing Transaction Output	97
Specifying Directories and Files for Transactions	98
Copying Files from a Parent to a Child Workspace (Bringover)	104
Creating a New Child Workspace (Bringover Create)	104
Notes about the Bringover Create Transaction	107
Updating an Existing Child Workspace (Bringover Update)	110
Notes about the Bringover Update Transaction.	113
Bringover Action Summary	116

Copying Files from a Child to a Parent Workspace (Putback)	117
Updating a Parent Workspace Using Putback	117
Notes about the Putback Transaction	120
Putback Action Summary	123
Reversing Bringover and Putback Transactions with Undo	124
Notes about the Undo Transaction	125
How the Undo Transaction Works	125
Renaming, Moving, or Deleting Files	128
Renaming Files	128
Deleting Files	132
Notes about Renaming Files	133
6. Resolving Conflicts	135
Detecting Conflicts	137
Detecting Conflicts during Bringover Update Transactions	137
Preparing Files for Conflict Resolution	138
Resolving Conflicts	139
Resolve Transaction	139
7. CodeManger Administration	145
Starting a Project with CodeManager	145
Moving an Existing Project	145
Starting a New Project	146
Configuring Your Workspace Hierarchy	146
File Transfer Considerations	148
Product Release Considerations	150

8. How CodeManager Merges SCCS Files	153
Merging Files That Do Not Conflict	154
Merging Files That Conflict	155
A Merge Example	156
9. CodeManager Example	167
Creating Workspaces	168
Putting Back Changes	173
Updating a Workspace	176
Resolving Conflicts	180
10. CodeManager Messages	183
CodeManager Error Messages	183
CodeManager Warnings	206
 <i>Part 3 — Version Tool</i>	
11. Introduction to VersionTool	215
Terminology	216
Branches	216
Deltas and Versions	217
History Files	217
SCCS Delta ID (SID)	217
Graphical Tour	218
Base Window	218
Base Window Pop-up Menu	219
File Button	220

Load Button	221
View Button	222
History Window.	223
Commands Button.	228
Check In New Window.	229
Props Button	231
Properties Window	231
12. Performing Basic SCCS Functions with VersionTool.	233
Typical Tool Sessions	233
Putting a Project Under SCCS Control	234
Working with a Project Under SCCS Control.	235
File Button: Loading and Unloading a Directory.	235
Loading a Directory.	236
Unloading a Directory.	236
Load Button: Reloading Previous Directories	237
View Button: Viewing File Information	238
Viewing the History Graph of a Selected File	239
Viewing SCCS Command Output	241
Viewing SCCS File Status	241
Commands Button: Manipulating Files	242
Checking Out and Checking In Files.	243
Editing a Checked-Out File	246
Checking in a New File.	246
Unchecking Out a File.	247

Displaying the Differences Between Two Deltas	247
Props Button: Changing VersionTool Properties	247
Changing the Main File List Display	248
Defining an Editor	248
Changing the Double-Click Action	249
Changing the History Graph Display	249
Changing the History Information Display	249

Part 4 —FreezePoint

13. Introduction to FreezePoint	253
How FreezePoint Works	254
Terminology	256
Starting FreezePoint.	257
Creating a Freezepoint File	257
Viewing or Modifying a Freezepoint File	260
Recreating (Extracting) a Source Hierarchy	260
Notes about Using FreezePoint	262
Details about the Freezepoint File	263
What is a SMID?	263
Why are SMIDs Necessary?	264
SMID/SID Translation	264
Translating SIDs to SMIDs	264
Translating SMIDS to SIDS	265
14. Troubleshooting VersionTool and FreezePoint	267

Troubleshooting Checklist	267
Reporting Problems.....	268
Error Messages.....	268
<i>Part 5 —ParallelMake</i>	
15. Introduction to ParallelMake	271
Parallel Builds.....	271
New Options.....	272
Special-Purpose Targets	272
16. Using ParallelMake	273
A Note About Makefiles.....	273
Building Targets in Parallel.....	274
The <code>.make.machines</code> File	274
How Parallelism is Achieved.....	275
Collected Output	275
Limitations on Makefiles.....	276
Dependency Lists.....	276
Explicit Ordering of Dependency Lists.....	276
Concurrent File Modification	277
Concurrent Library Update	277
Multiple Targets.....	278
Restricting Parallelism.....	278
Nested Invocations of ParallelMake	279
Error Messages.....	280
Glossary	281

Index	287
-------------	-----

Preface

The *TeamWare Users Guide* (*TeamWare Users Guide*) describes how to use the *SPARCworks/TeamWare PorWorks/TeamWare* (*TeamWare*) code management tools. The concepts and information discussed apply to both command line and graphical user interfaces.

Who Should Use This Book

You, the software developer, 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 *CodeManager* section of this guide is primarily addressed to you. It also addresses the needs of integrators, administrators, and release engineers.

The *VersionTool* and *FreezePoint* section of this guide explains how to use *VersionTool* for controlling files and monitoring changes on concurrent software development projects. *VersionTool* is a graphical user interface (GUI) to the source code control system (SCCS). It also explains how to use *FreezePoint*, a tool that allows you to create snapshots of a project at various key junctures. These snapshots, or freeze points, 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 sections assumes that you are familiar with programming constructs and processes. You need not have previous experience with SCCS.

The *ParallelMake* section of this guide is a supplement to the standard `make` documentation. It describes how to use *ParallelMake* to parallelize the process of building programs. Use this section if you maintain programs using the `make` utility and wish to speed up the build process. This section also assumes 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 SunSoft Developer Products on Solaris* manual for information on how to install the *TeamWare* software.

How This Book Is Organized

Part 1—TeamWare Code Management Tools Quick-Start

Chapter 1, “QuickStart Guide” provides instructions ffor quickly getting starting using the *TeamWare Code Management Tools*.

Part 2—Code Manager

Chapter 2, “Introduction to CodeManager” presents a conceptual overview of CodeManager. Basic concepts are discussed that are vital to understanding CodeManager and the remainder of this manual.

Chapter 3, “CodeManager User Interfaces” describes the CodeManager user interfaces.

Chapter 4, “CodeManager Workspace” describes the CodeManager workspace and the commands used to work with workspaces.

Chapter 5, “Copying Files between Workspaces” describes the CodeManager transactions used to transfer files between workspaces.

Chapter 6, “Resolving Conflicts” explains how you resolve conflicts between files in parent and child workspaces.

Chapter 7, “CodeManger Administration” discusses issues related to starting to use CodeManager with a source hierarchy.

Chapter 8, “How CodeManager Merges SCCS Files” describes the ways that CodeManager manipulates SCCS history files during file transfer transactions.

Chapter 9, “CodeManager Example” contains a simple example that demonstrates the CodeManager Bringover/Putback/Resolve transaction cycle.

Chapter 10, “CodeManager Messages” contains a list of CodeManager error messages and warnings. For each message, the meaning of the message and a possible remedy for the error are provided.

Part 3—VersionTool

Chapter 11, “Introduction to VersionTool presents VersionTool terminology and a conceptual overview of how VersionTool works. It also provides a graphical tour of the main VersionTool windows and menus.

Chapter 12, “Performing Basic SCCS Functions with VersionTool” presents the basic operations of VersionTool and a typical tool session. It covers the basic operational tasks associated with each menu button and walks you through step-by-step instructions.

Part 4—FreezePoint

Chapter 13, “Introduction to FreezePoint presents FreezePoint, 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 14, “Troubleshooting VersionTool and FreezePoint provides a problem checklist to consider before calling the Sun Support hot line. 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.

Part 5—ParallelMake

Chapter 15, “Introduction to ParallelMake” is an introduction to ParallelMake.

Chapter 16, “Using ParallelMake” describes how to use ParallelMake.

“**Glossary**” is a list of words and phrases found in this book and their definitions.

“**Index**”

Related Documentation

The documentation for *TeamWare* is available in hard copy and online.

Hard Copy Documentation

The following manuals are available in hard copy:

- *TeamWare Users Guide (this book)*
- *Installing SunSoft Developer Products on Solaris*
- *Managing the Toolset*
- *SPARCworks/TeamWare ProWorks/TeamWare Solutions Guide*
- *Merging Source Files*

Online Documentation

You can get online documentation in the following ways:

AnswerBook Product—An online documentation tool that displays this manual along with other SPARCworks tools manuals. You can read this manual online and take advantage of dynamically linked headings and cross-references. To start the AnswerBook product, type:

% **answerbook**

Magnify Help[™]—A standard help system of the OpenWindows software environment. It furnishes help messages in Magnify Help windows. To access Magnify Help messages, place the pointer on the window, menu, or menu button, and press the keyboard Help key.

Notices—A standard feature of the OPEN LOOK environment that serve two functions. Some notices are prompts that inquire about whether you want to continue with a particular action. Other notices are precautionary in that they provide information about the end result of a particular action. They appear only when the end result of an action is irreversible.

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

Table P-1 TeamWare Manual Pages

codemgr(1)	rcstosccs(1)	conflicts(4)
codemgrtool(1)	resolve(1)	history(4)
bringover(1)	sccsmerge(1)	locks(4)
def.dir.flp(1)	workspace(1)	nametable(4)
putback(1)	ws_undo(1)	notification(4)
rcstosccs(1)	access_control(4)	parent(4)
resolve(1)	args(4)	putback.cmt(4)
putback(1)	children(4)	freezept(1)
freezepttool(1)	vertool(1)	make(1)
freezeptfile(4)	filemerge(1)	

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
AaBbCc123	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. machine_name% You have mail.
AaBbCc123	What you type, contrasted with on-screen computer output	machine_name% su Password:
<i>AaBbCc123</i>	Command-line placeholder: replace with a real name or value	To delete a file, type <code>rm filename</code> .
<i>AaBbCc123</i>	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

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

Part 1 — Quickstart Guide

<i>CodeManager</i>	<i>page 2</i>
<i>VersionTool</i>	<i>page 9</i>
<i>FreezePoint</i>	<i>page 12</i>
<i>ParallelMake</i>	<i>page 14</i>

QuickStart Guide



Use this chapter to quickly get started using the *TeamWare* code management tools. For more detailed information on the use of the tools, see the other parts of this guide.

Note – The Magnify Help™ facility is available at all times through all of the *TeamWare* GUIs. For information about any object on the screen (for example, a button, menu or window), move the cursor over the object and press the Help key on the keyboard.

Use Table 1-1 to find about the *TeamWare* tools and to locate quickstart and detailed information about them:

Table 1-1 TeamWare Tools Definitions and Information Locations

Tool	Definition	Start here if you want to quickly begin using this tool	Start here for more detailed information about this tool
CodeManager	Enables you to manage concurrent code development. Also facilitates release integration and release management.	“CodeManager” on page 2	Part 2—Code Manager
VersionTool	A graphical front end for the UNIX source code control system (SCCS).	“VersionTool” on page 9	Part 3—VersionTool

Table 1-1 TeamWare Tools Definitions and Information Locations

Tool	Definition	Start here if you want to quickly begin using this tool	Start here for more detailed information about this tool
FreezePoint	Enables you to capture source file configurations for subsequent retrieval.	“FreezePoint” on page 12	Part 4—FreezePoint
FileMerge	Assists you in comparing and merging concurrent modifications to source files.	Chapter , “QuickStart Guide	See the <i>Merging Source Files</i> manual
Parallel Make	Accelerates project builds on multiprocessor machines.	“ParallelMake” on page 14	Part 5—Parallel Make

CodeManager



Some CodeManager Concepts

CodeManager is based on a concurrent development model called *Copy-Modify-Merge*. Isolated (per developer) workspaces¹ form the basis of the CodeManager model. With CodeManager, you (the developer) *copy* source 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.

Besides providing isolated workspaces, CodeManager enables you to easily and “intelligently” copy files between workspaces and then merge changes that exist between corresponding files. The CodeManager “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.

CodeManager 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,

1. A work space is a designated UNIX directory and and subdirectories.

CodeManager commands prevent you (or another developer) from copying over those changes; CodeManager 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 CodeManager 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 CodeManager 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 CodeManager 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 said to be in conflict. If this is the case, CodeManager 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

CodeManager acts only upon files under the source code control system (SCCS). When considering CodeManager 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, CodeManager acts upon (copies or merges) the file's SCCS history file (also

known as the “s-dot-file”). How CodeManager manipulates and merges the history files is described in Chapter 8, “How CodeManager Merges SCCS Files.”

▼ Getting Started

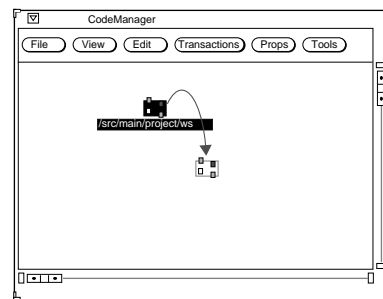
You can use CodeManager through either a graphical user interface (GUI) or command-line interface (CLI). The following flow diagram uses the GUI only; for information about the CLI, please refer to the `bringover(1)` and `putback(1)` man pages.

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

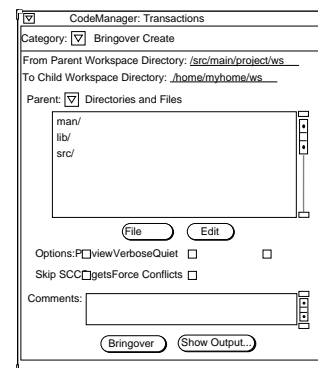
▼ Creating a New Workspace

- From a command prompt start the CodeManager GUI:

```
tutorial% codemgrtool &
```
- If the workspace from which you must obtain your files is not automatically loaded, you can load the workspace using the Load item from the File menu.
- Once you have loaded 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.
- In the Bringover Create Transactions window, enter the child workspace path name in the text field labeled: To Child Workspace Directory.
- In the Directories and Files text pane, create the list of directories and files you wish to bring over into your workspace from the parent workspace. Choose File ► Add Files to create the Directories and Files list.
- *Optionally:* Select the Preview option in order to view the results of the transaction prior to actually transferring any files.
- Click on the Bringover button at the bottom of the window to initiate the transaction.




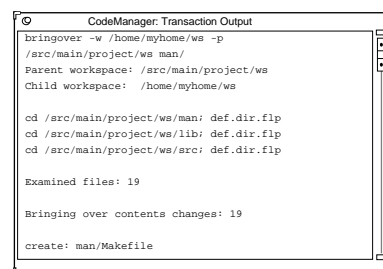
Drag-and-drop parent workspace icon to a vacant area



Bringover Create Transactions window

- View transaction output in the Transaction Output window.

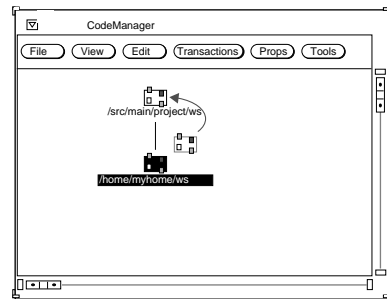
 For more information about the Bringover Create transaction, see “Creating a New Child Workspace (Bringover Create)” on page 104” .



Transaction Output

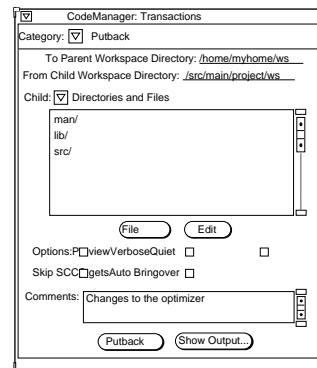
▼ Putting Back Changes to the Parent

- When you are ready, you will update the parent workspace with the changes you make. This CodeManager transaction is called Putback.
- You initiate the Putback transaction by dragging and dropping your child workspace icon onto the parent workspace icon. This activates the Putback version of the Transactions window.



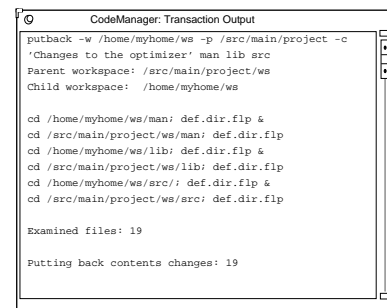
Drag-and-drop child workspace icon onto parent icon

- CodeManager 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.
- Type a comment in the Comments text window.
- *Optionally*. Select the Preview option in order to view the results of the transaction prior to actually transferring any files.
- Click on the Putback button at the bottom of the window to initiate the transaction.



Putback Transactions window

- View transaction output in the Transaction Output window.

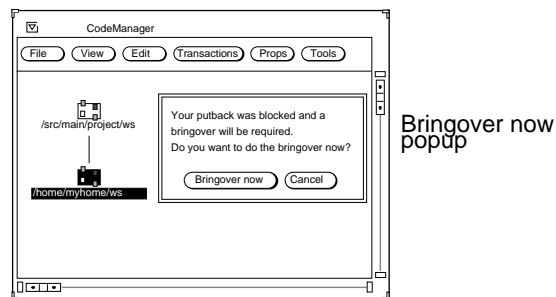


Transaction Output

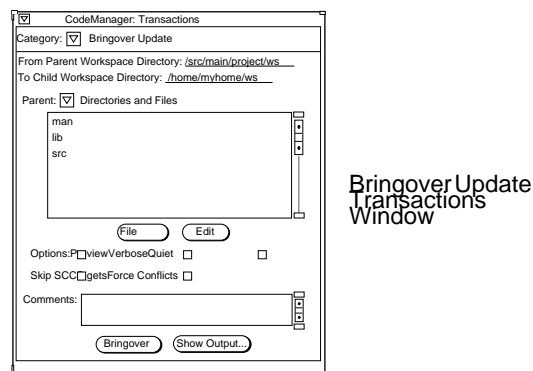
☞ For more information about the Putback transaction, see "Updating a Parent Workspace Using Putback" on page 117.

▼ Updating the Child Workspace


- The Putback transaction is blocked if any of the files have changed in the parent since you brought them over.
- If the Putback transaction is blocked, you must use the Bringover Update transaction to bring those changes into your child workspace, resolve any conflicts, test and then put them back to the parent. A popup window advises you that the transaction is blocked.
- You initiate the Bringover Update transaction by clicking on Bringover now in the popup window. This activates the Bringover Update version of the Transactions window.

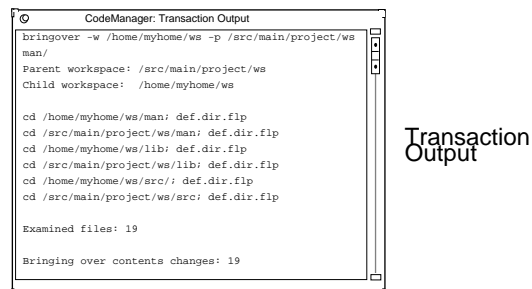


- In the Bringover Update Transactions window, CodeManager automatically fills in the names of the parent and child workspaces, and includes the same directories and files that you included when you created the child workspace.
- *Optionally*: Select the Preview option in order to view the results of the transaction prior to actually transferring any files.
- Click on the Bringover button at the bottom of the window to initiate the transaction.



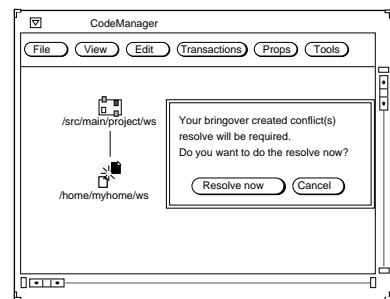
- View transaction output in the Transaction Output window.

 For more information about the Bringover Update transaction, see “Updating an Existing Child Workspace (Bringover Update)” on page 110”.



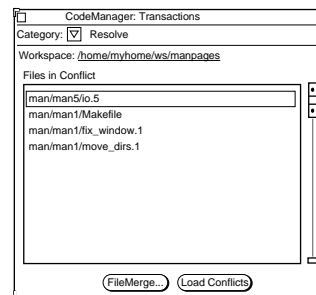
▼ Resolving any Conflicts

- If any of the files that you changed in your child workspace have *also* changed been changed in the parent workspace, they are said to be in *conflict*. If CodeManager discovers any conflicts during the Bringover Update transaction, it automatically activates a popup window advising you of this.
- You initiate the Resolve transaction by clicking on Resolve now in the popup window. This activates the Resolve version of the Transactions window. *Note:* CodeManager automatically alters the workspace icon to alert you that a workspace contains unresolved conflicts.



Resolve now popup

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



Resolve Transaction Window

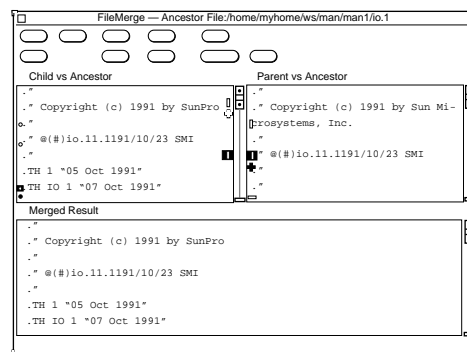
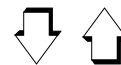
- FileMerge displays two text files (the versions of the file from the parent and child workspaces) for side-by-side comparison, each in a read-only subwindow. Each version is shown in comparison (using glyphs) to the version that existed before the changes were made. Beneath them, FileMerge displays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

FileMerge automatically merges the files for you in the bottom window. If you disagree with the choices made by the program, you can use the Left and Right buttons to accept the changes found in the left or right window.

- When you have merged the files, select the Save button to save the file. If there are more files in the Transactions window conflict list, CodeManager automatically loads the next file in the list into FileMerge.

❖ You can now successfully put back your files to the parent workspace.

👉 For more information about resolving conflicts and merging files, see Chapter 6, “Resolving Conflicts” .



FileMerge Program

VersionTool



VersionTool is a GUI to SCCS that enables you to manipulate files and perform SCCS functions without having to know SCCS commands. It provides an intuitive method for checking files in and out, as well as displaying a file's delta history and showing differences between deltas.

With VersionTool, 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 Filemerge
- Display the version log summarizing executed commands
- Create new SCCS files

▼ Starting VersionTool

To start VersionTool, at a shell command prompt type `vertool` followed by the ampersand symbol (&) as shown:

```
tutorial% vertool &  
tutorial%
```

Note – VersionTool can also be started directly from the CodeManager GUI by double-clicking on a workspace icon.

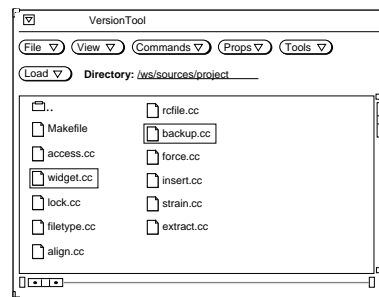
To use VersionTool, select a file (or group of files) in the File List pane and choose a menu item to operate on it. Commands are located in the:

- Commands menu
- View menu
- File List pane floating menu

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

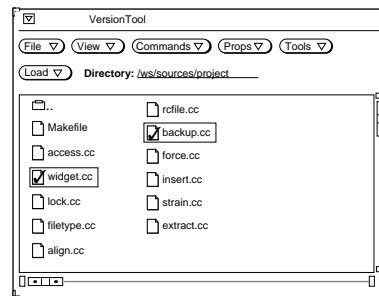
▼ Checking Files In and Out of SCCS

- From a command prompt start VersionTool:
tutorial% **vertool** &
- If the directory that contains your file is not automatically loaded, you can type the directory's path name (followed by Return), in the Directory text field.
- Click on a file icon to select a file; use the ADJUST mouse button to extend the selection.



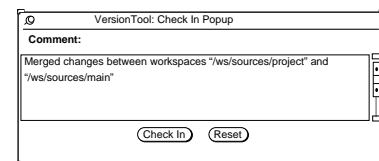
VersionTool main window with two files selected

- Choose either Checkout ► Default or Checkout ► Check Out, Edit from the Commands menu. As the files are checked out a check mark appears in their icons.




VersionTool main window with two files selected and checked out

- 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.
- Enter a comment in the text window that describes your changes and click on Check In to complete the check in process.
Note that the check mark is removed from the file icon as the files are checked in.



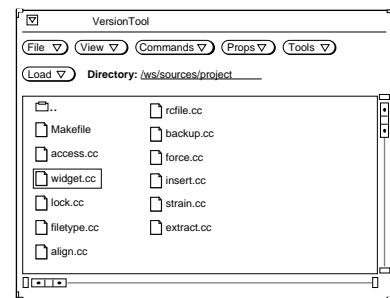
Check In popup

 For more information about VersionTool, see the section on VersionTool in this manual.

▼ Viewing and Comparing a File's Delta History

- 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.

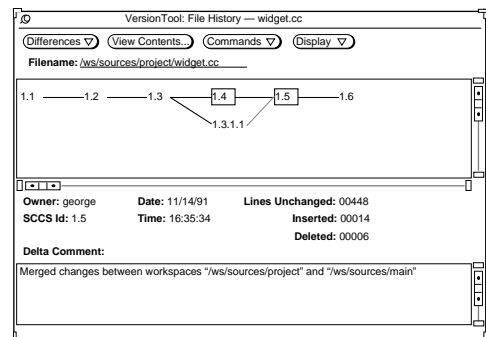
VersionTool main window with file selected



- Select two deltas in the graph and choose Use FileMerge from the Differences menu.

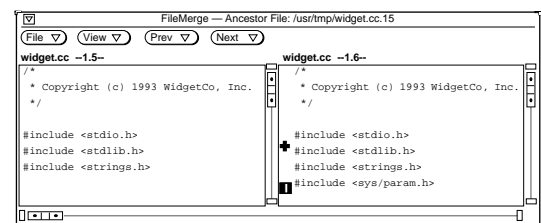
Note that the graph indicates that a branch (1.3.1.1) was created during a CodeManager Bringover Update operation (due to a conflict). The dotted line indicates that the conflict was subsequently resolved using CodeManager; this resulted in the creation of delta 1.5.

VersionTool File History window with deltas 1.5 and 1.6 selected



- FileMerge displays the two deltas side-by-side, marking differences with glyphs.

 For more information about VersionTool, see the VersionTool section in this manual.



FileMerge window with differences between deltas 1.5 and 1.6 displayed

FreezePoint



During software development it is often useful to create “freeze points” of your work at key junctures. These freeze points serve as “snapshots” of the project that enable you to recreate the state of the project at key development points.

With the FreezePoint program, you preserve these freeze points quickly and simply, using a very small amount of storage resource.

You can use FreezePoint through two functionally equivalent user interfaces:

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

The concepts discussed in this section generally apply to both the GUI and the CLI. Descriptions and examples are included for the GUI only. Information specific to the CLI can be obtained on line through the man pages: `freezept(1)`, `freezepttool(1)`, and `freezeptfile(5)`.

FreezePoint enables you to create freeze point files from CodeManager workspaces. Freeze point files are text files that list the default deltas in SCCS history files contained in the workspace. When you later recreate (extract) the files, FreezePoint 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 FreezePoint GUI

To start FreezePoint, at a shell command prompt type `freezepttool` followed by the ampersand symbol (&) as shown:

```
tutorial% freezepttool &
tutorial%
```

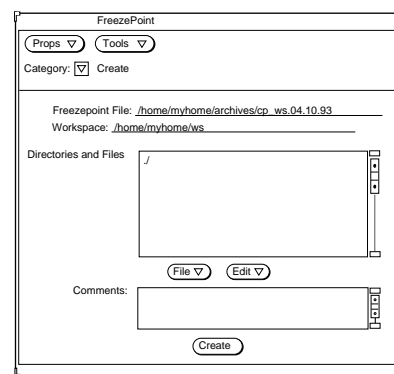

▼ Creating and Extracting Freezepoints

- From a command prompt start the FreezePoint GUI:

```
tutorial% freezeointtool &
```

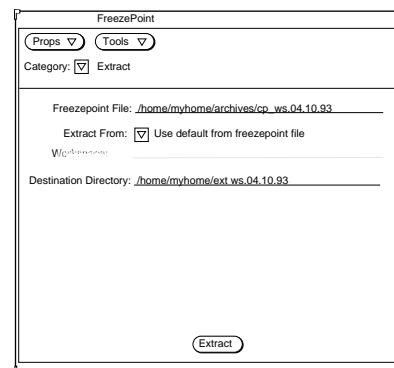
The pane below the Control area is used for both creating and extracting freezepoints. You switch between the Create and Extract panes by choosing the appropriate item from the Category menu. The Create pane is the default and is displayed when you start `freezeointtool`.

- Enter the path name of a freezepoint file. *Note:* FreezePoint automatically inserts the file name `freezeoint.out`. Delete it and replace it with a path name of your choosing.
- Enter the path name of the source workspace. This is the workspace that you are “freezing.”
- In the Directories and Files text window, compose a list of directories and/or files that you wish to freezepoint. Choose File ► Add Files to create the Directories and Files list
- Click Create to execute.




Freezepoint
Create
Window

- To extract a freezepoint, choose Extract from the Category menu. This changes the pane from Create to Extract..
- Type the path name of an existing freezepoint file.
- Specify the path name of the Destination Directory. This the directory into which the newly extracted files are placed.
- Click on Extract to execute.



Freezepoint
Extract
Window

 For more information about FreezePoint, see the chapter on FreezePoint in this manual.

ParallelMake

ParallelMake enables you to parallelize the process of building large programs over a number of processes and, in the case of multiprocessor systems, over multiple CPUs. ParallelMake reads your makefiles and automatically:

- Determines which targets can be built in parallel
- Parallelizes the build of those targets over a number of processes

As a developer, you shouldn't have to do anything to start using ParallelMake. Your project administrator may have to make adjustments to makefiles so that they work correctly with ParallelMake.



For more information about ParallelMake, see the ParallelMake section in this manual.

Part 2 — CodeManager



<i>Introduction to CodeManager</i>	<i>page 17</i>
<i>CodeManager User Interfaces</i>	<i>page 37</i>
<i>CodeManager Workspace</i>	<i>page 65</i>
<i>Copying Files between Workspaces</i>	<i>page 95</i>
<i>Resolving Conflicts</i>	<i>page 135</i>
<i>CodeManger Administration</i>	<i>page 145</i>
<i>How CodeManager Merges SCCS Files</i>	<i>page 153</i>
<i>CodeManager Example</i>	<i>page 167</i>
<i>CodeManager Messages</i>	<i>page 183</i>

Introduction to CodeManager



This chapter introduces basic CodeManager concepts. The practical implementation of these concepts is discussed in the remainder of *Part 2—CodeManager*. An example that demonstrates CodeManager use can be found in Chapter 9, “CodeManager Example.”

Coordinating the Work of Software Developers

Managing large programming projects involves the difficult and complex task of coordinating the work of many 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 effect the entire project. Preparing the code for a final build and release can be a daunting task.

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 very 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 innocently 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” (changes to source files) becomes very 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

CodeManager assists in the development and release of large software projects. CodeManager is based on a concurrent development model called *Copy-Modify-Merge*. Isolated (per developer) workspaces¹ form the basis of the CodeManager model. With CodeManager, 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.

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

1. A workspace is simply a specially designated SunOS™ directory and its subdirectories.

Besides providing isolated workspaces, CodeManager enables you to easily and “intelligently” copy files between workspaces and then merge changes that exist between corresponding files. CodeManager’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; CodeManager 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, CodeManager prevents you (or another developer) from copying over those changes. CodeManager then provides sophisticated window-based tools that help you to merge these differences.

Copy-Modify-Merge Example

Figure 2-1 illustrates the Copy-Modify-Merge concurrent development model employed by CodeManager. This example describes the common software development scenario where two developers are working simultaneously on the same or related parts of a project.

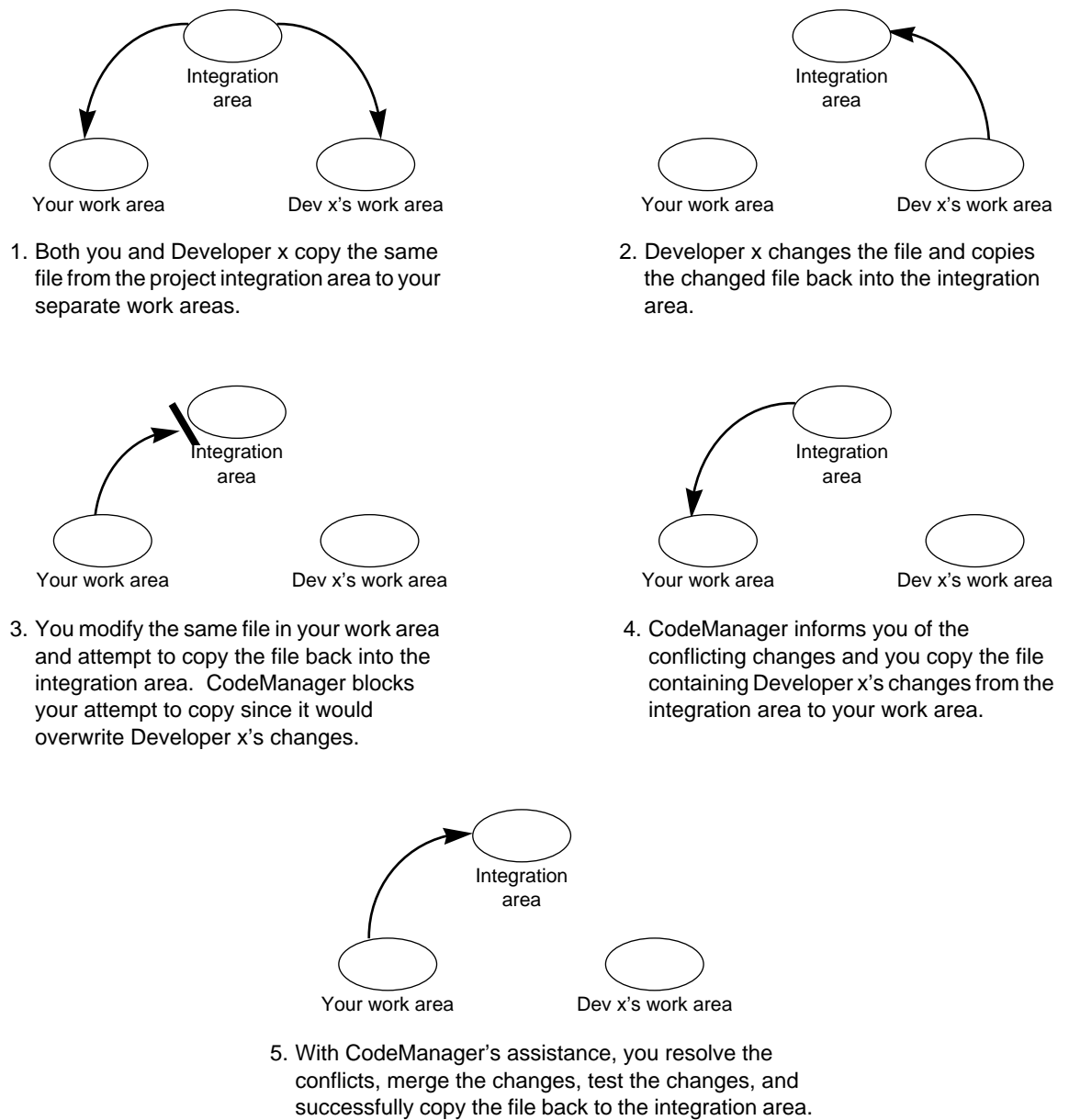


Figure 2-1 Copy-Modify-Merge Example

Default CodeManager

The following overview discusses the default CodeManager system. CodeManager can be customized in ways that modify its default behavior; many of those customizations are discussed in other sections of *Part 2—CodeManger*.

All source files in a CodeManager project are maintained under the UNIX SCCS. *CodeManager only copies files that are under SCCS*. Within your workspaces, you use SCCS in the normally. 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 CodeManager. When you copy files between workspaces and merge files that have changed, CodeManager manages SCCS history files for you, preserving all comments and deltas.

Workspace

The *workspace* forms the basis of the CodeManager 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 CodeManager commands.

The workspace is a directory and its subdirectory hierarchy. When the workspace is created, CodeManager creates a special subdirectory under the workspace, called `Codemgr_wsdata`, to store workspace information.

A CodeManager 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`.

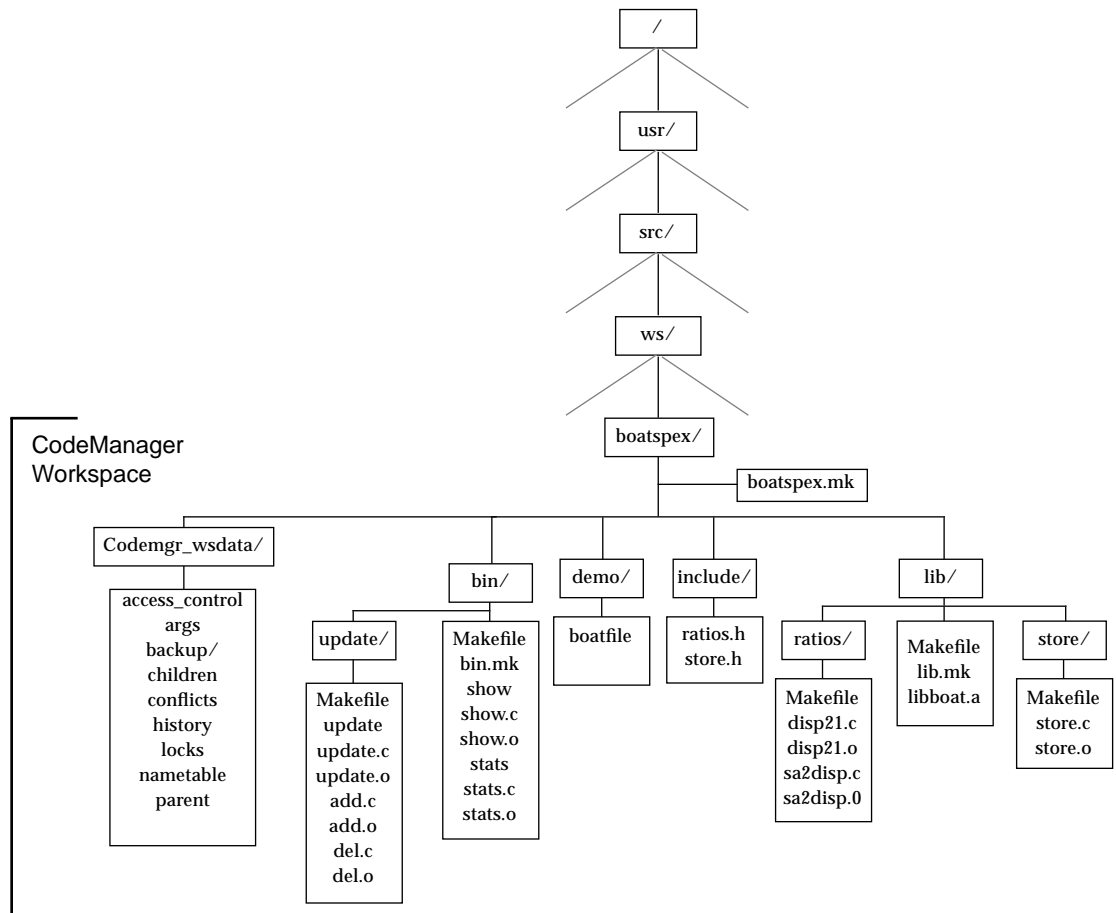


Figure 2-2 Project File System Hierarchy

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 2-3.

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

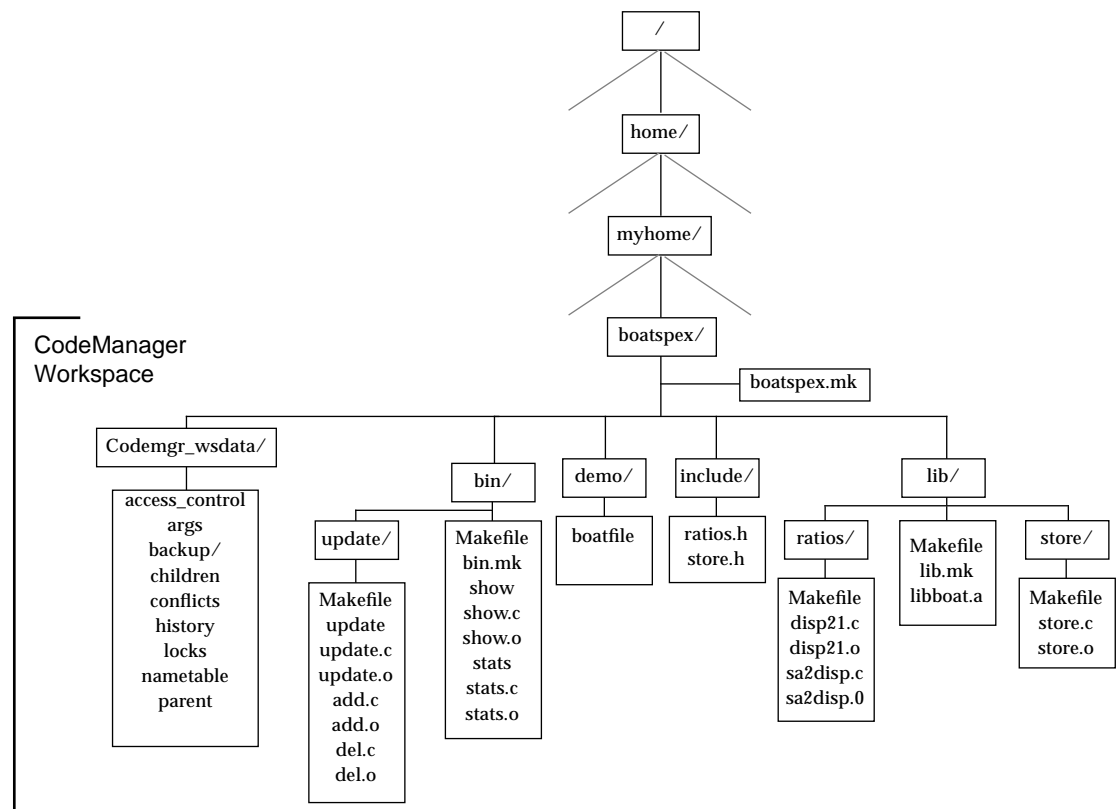


Figure 2-3 Your New Workspace

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, however, 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 CodeManager 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 CodeManager 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 workspace¹. 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 CodeManager transaction for copying files from a child workspace to a parent workspace is called *Putback*.

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 to create child workspaces from the integration workspace, which forms a three-tiered hierarchy of workspaces.

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

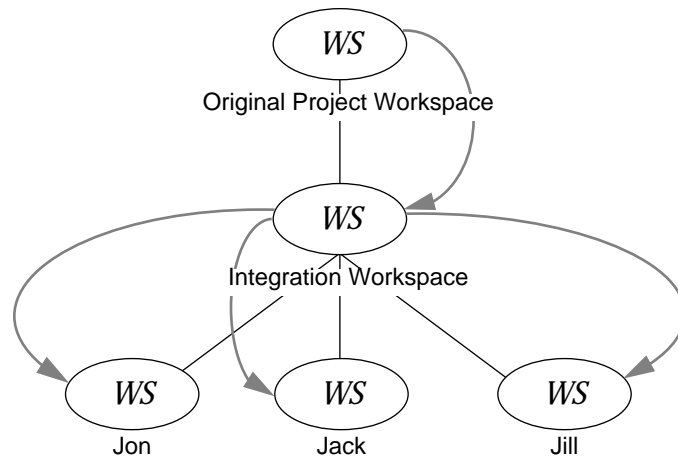


Figure 2-4 Using the Bringover Create Transaction to Create a Workspace Hierarchy

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).

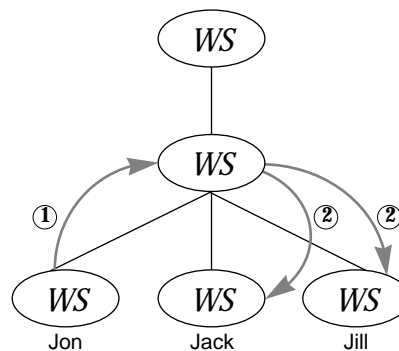


Figure 2-5 Copying Files between Workspaces

Reparenting

Parent and child relationships can be changed. CodeManager 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 Section , “Reparenting a Workspace,” on page 71 for more information.

Codemgr_wsdata Directory

Every CodeManager workspace contains a directory named `Codemgr_wsdata` that is a subdirectory of the workspace top-level (root) directory. This directory contains text files that CodeManager uses to log its actions, and store temporary and permanent data. You can view and alter these files using standard text utilities.

Refer to Section , “The Workspace Metadata Directory,” on page 65 for more information.

Modifying Files

Since CodeManager 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 CodeManager.

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. CodeManager 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 5, "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, CodeManager 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, CodeManager decides how to manipulate the SCCS history file used to derive the file. If the file does not exist in the target workspace, CodeManager 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, CodeManager 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. CodeManager 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, CodeManager 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, CodeManager blocks your attempt to put back the files into the parent and informs you of the conflicting change. (Additional information appears on page 33.)

Note – 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 in order 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 CodeManager’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. Figure 2-6 illustrates this process.

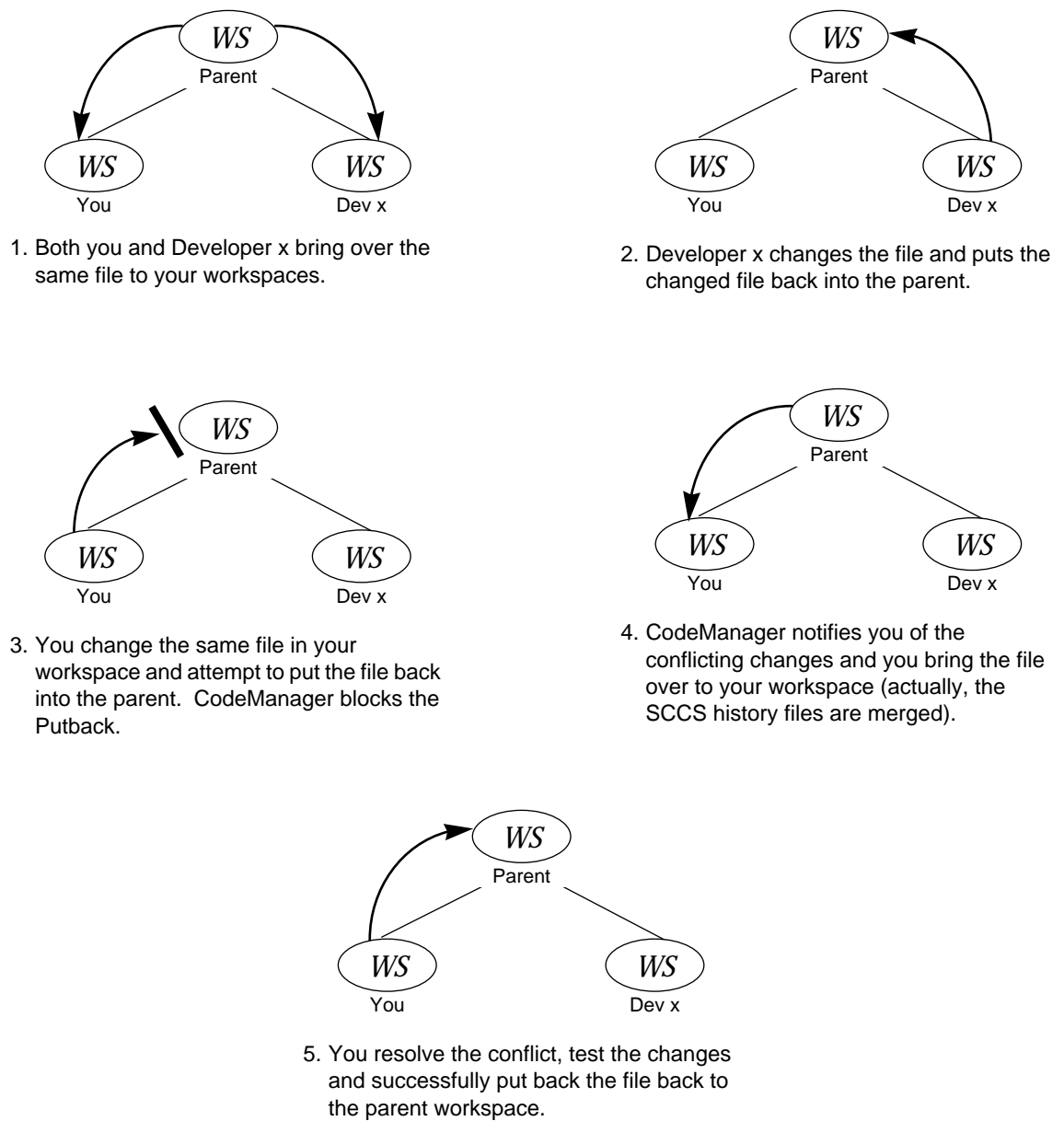
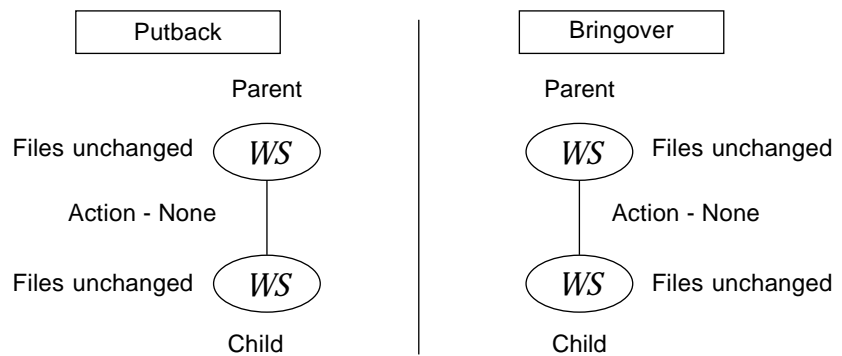


Figure 2-6 Conflict Example

Relationships between Files in Parent and Child Workspaces

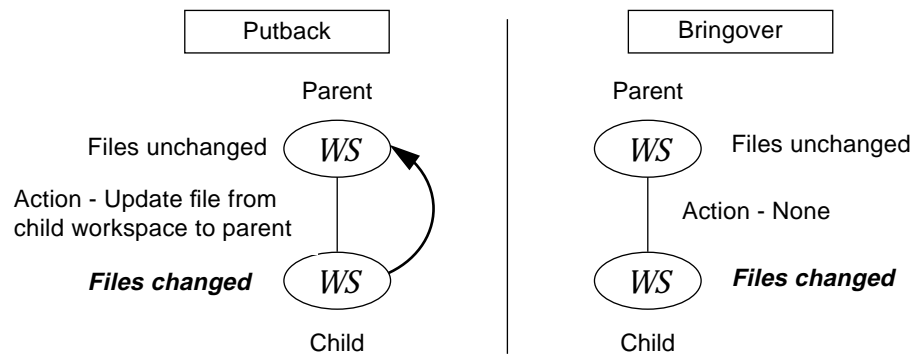
The previous example describes only one of four possible states of relationship that can exist between corresponding files in parent and child workspaces. The relationship between files in parent and child workspaces governs the way that CodeManager behaves when you attempt to copy files via Putback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

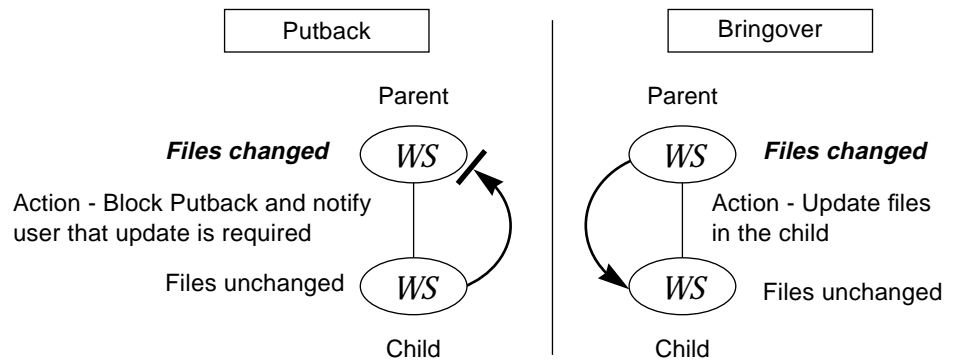
2. The specified files have *not* been modified in the parent since they were brought over from the parent into the child or put back from the child into the parent. However, the corresponding files *have* been modified in the child.



In this case when you use the Putback transaction to copy the file to the parent, the changed files are automatically updated from the child into the parent, replacing the corresponding files in the parent. This new data is now 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.

3. One or more files in the parent have been 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 have not been modified.

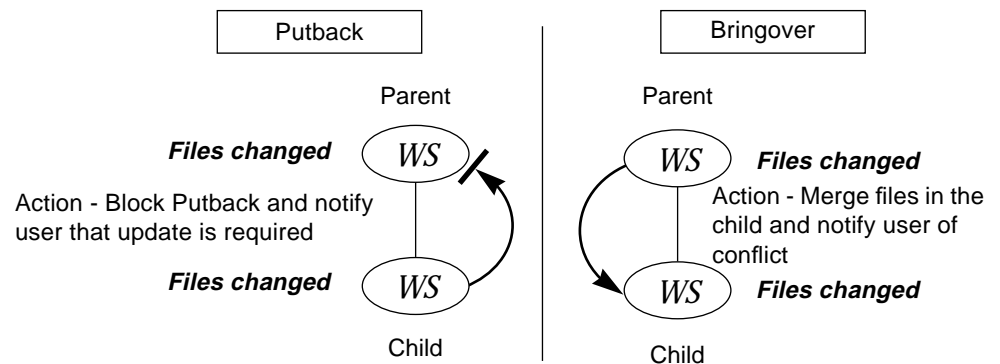


In this case the parent’s copy of the file being put back from the child has been modified (probably by one of its other children) since it was last brought over to the child; the corresponding file in the child has *not* been modified since it was last brought over into the child.

When CodeManager 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.¹

4. Corresponding files have been modified in both the parent and child workspaces.

1. You may optionally specify that the Putback transaction automatically update the child by invoking the Bringover Update transaction.



This is the most complicated of the four cases. CodeManager cannot allow the file to be put back from the child into the parent because the transaction will obscure modifications there. Likewise, CodeManager 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, CodeManager blocks the Putback transaction and notifies the user. When the user attempts to update the child workspace by means of the Bringover transaction, CodeManager 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 CodeManager merges the parent and child SCCS history files for the conflicting file in the child workspace.

CodeManager 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

Table 2-1 summarizes the action taken by CodeManager during a Putback transaction in each of the four cases described above. Table 2-2 does the same for the Bringover transaction.

Table 2-1 Summary of CodeManager Action during a Putback Transaction

Case	File in Parent	File in Child	Action by CodeManager
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 2-2 Summary of CodeManager Action during a Bringover Transaction

Case	File in Parent	File in Child	Action by CodeManager
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, CodeManager 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, CodeManager 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 CodeManager 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 CodeManager 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.

CodeManager provides tools that aid in resolving conflicts, however, the conflicts must be resolved by the user. Refer to Chapter 6, “Resolving Conflicts,” for a detailed discussion about conflict resolution.

CodeManager User Interfaces



You can work with CodeManager in two ways:

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

Both interfaces are included to provide flexibility and to accommodate different computing styles. Complete CodeManager 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 CodeManager functionality and command structure; the difference is an easy-to-use, graphical, point-and-click interface for the GUI.

The concepts discussed in this guide 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 `man` pages.

The remainder of this chapter serves as an overview and introduction to the CodeManager CLI and GUI.

CodeManager Command-Line Interface

The CodeManager command-line interface (CLI) is accessible from any SunOS shell. It is especially 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 named `codemgr` provides a unified method of execution that enables you to conveniently list CodeManager commands.

You can list CodeManager commands with their usage summaries by simply executing `codemgr` without specifying any arguments.¹

```
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
```

Since using the `codemgr` umbrella command requires extra typing, you may also execute the commands directly (without typing `codemgr`). For example:

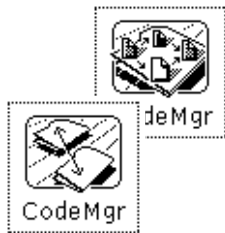
```
% bringover -w my_child -p their_parent /usr/ws/project
```

Note – The man pages for the CodeManager commands are referenced using only the individual command name.

1. You can achieve the same results by executing `codemgr` with the `help` subcommand.

CodeManager provides a number of ways for you to reduce typing long command-lines, including environment variables and argument files that store previously specified arguments. Refer to the respective man pages for details.

CodeManager Graphical User Interface



The CodeManager GUI (hereafter referred to simply as “CodeManager”) 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.
- Magnify Help to assist you at all levels, including explanation of error messages.

About This Section

CodeManager is extensively documented online using the OpenWindows Magnify Help feature. This enables you to conveniently and quickly obtain specific information regarding any object on the screen. For that reason, this overview does not discuss these objects (windows, menus, buttons) in great detail; rather, it serves as an orientation and guide to the workings of CodeManager. Specific, detailed information is available to you online as you require it.

Throughout the rest of this manual, CodeManager tasks such as Bringover/Putback transactions and conflict resolution are discussed in detail. As part of these discussions, examples are included that describe CodeManager being used to accomplish those tasks.

Starting Up CodeManager

To start up CodeManager, at a shell command prompt type `codemgrtool` followed by the ampersand symbol (&) as shown.

```
example% codemgrtool &
example%
```

After a moment, the CodeManager window appears.

CodeManager Windows

CodeManager consists of two base windows and a number of pop-up windows. The two base windows are:

- CodeManager window
- Transactions window

Most of your work is done in these two windows. Within each of these windows are control areas that contain menu buttons from which you can choose command items, menu items, and window items to help you accomplish your tasks.

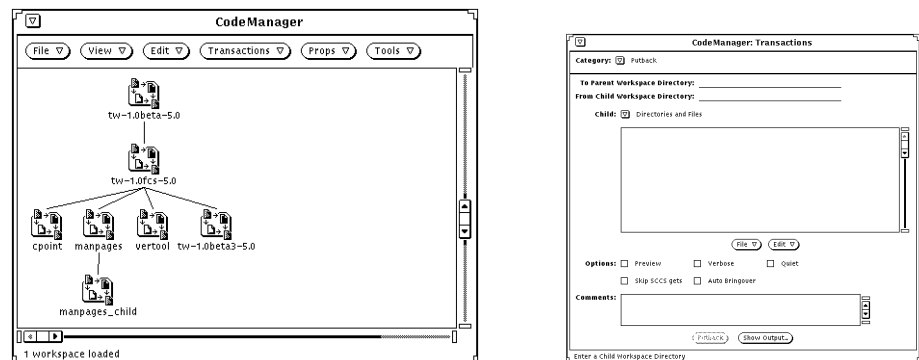


Figure 3-1 CodeManager Window and Transactions Window

CodeManager File and Directory Choosers


Wherever possible, CodeManager employs point-and-click chooser windows to help you conveniently navigate the file system and choose files and directories. Specifically, chooser windows are used to assist you in the following tasks:


- Choose workspaces to load into the Workspace Graph pane (Load ⇒ Workspaces item on the File menu).
- Choose files and directories for inclusion in interworkspace transactions (Add Files to List item on the Transactions window File menu).
- Choose FLPs (file list programs) for use during interworkspace transactions (Add FLPs item on the Transaction Window File menu).
- Choose directories and files about whose change of status you wish to be notified (Add Files to List button in the Properties Notification window).

You use the three choosers in the same manner. 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 have made 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 – A check mark in a file icon  indicates that the file is checked out from SCCS.

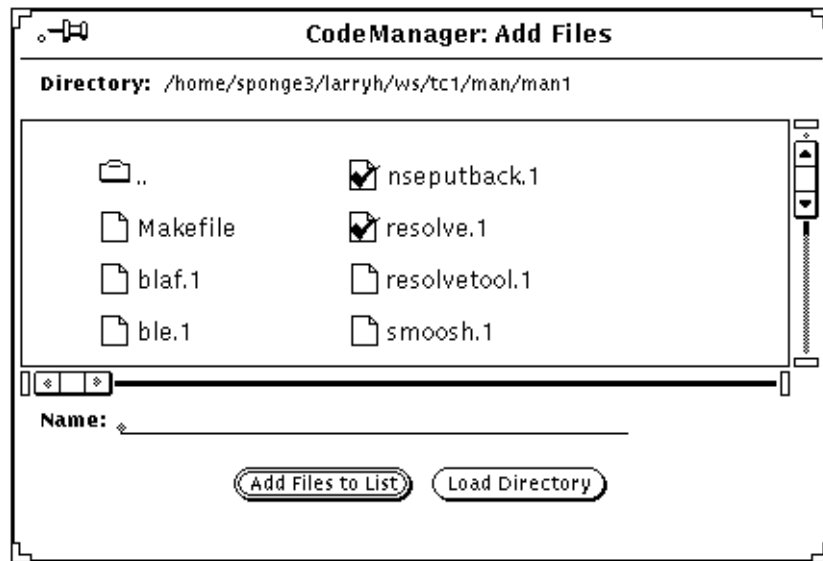


Figure 3-2 Add Files Chooser Window

CodeManager Window



When you start up CodeManager, you see its main base window.

When you work with CodeManager, you select workspace icons in the Workspace Graph pane and then choose commands from the control area that act upon the selected workspaces and the files they contain.

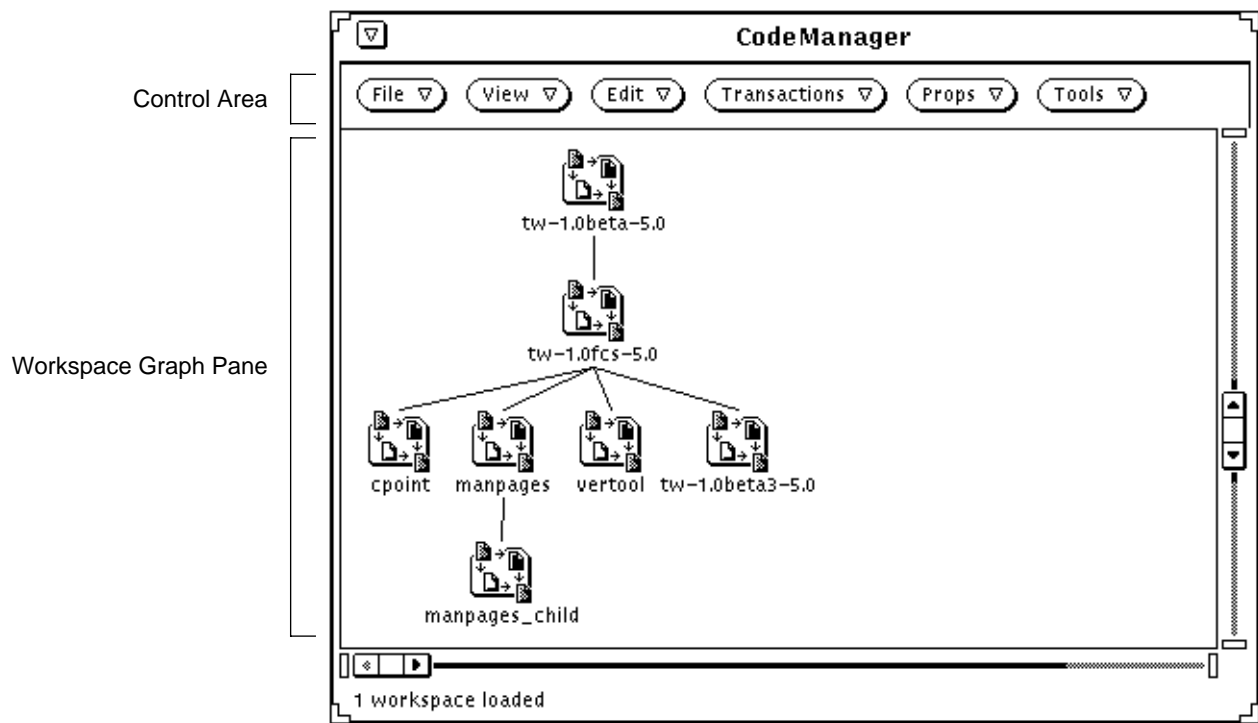


Figure 3-3 The CodeManager Base Window

Workspace Graph Pane

Workspace hierarchy graphs are displayed in the Workspace Graph pane. Each workspace is represented by a workspace icon; parent/child relationships are depicted by lines connecting workspaces. The path name of the workspace's top-level (root) directory is displayed beneath the icon.

Loading Workspaces into the Workspace Graph Pane

When CodeManager 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 this variable is not set, CodeManager attempts to load workspaces from the directory in which it is started. To load additional workspaces, use the Load Workspaces window from the File menu.

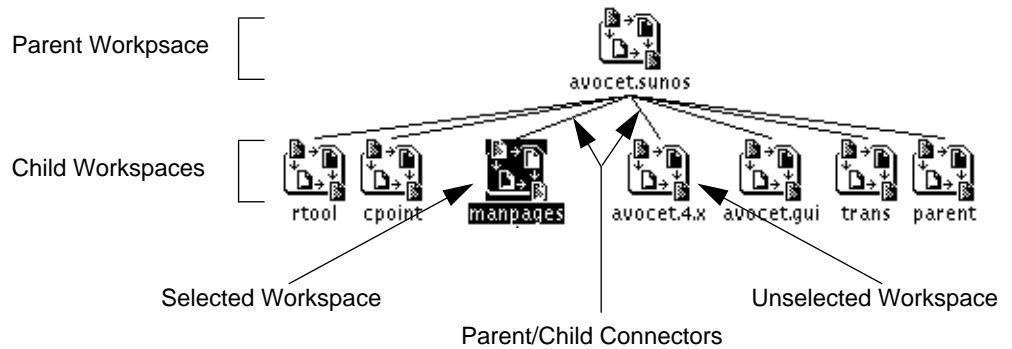


Figure 3-4 Graph of a Workspace Hierarchy

Layout

Workspace hierarchy graphs are automatically created in the Workspace Graph pane by CodeManager as you load workspaces (using the Load menu). Hierarchies are displayed either vertically or horizontally¹ starting from the upper left corner and distributed to the right as space permits. 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’s 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.

1. You can choose the orientation that you prefer using the Orientation item from the View menu. Vertical orientation is the default.

You can change the path name of a workspace by editing the name text field. Select the name field by moving the pointer over a portion of the text and click SELECT. This selects the text for editing. Use the standard OpenWindows text editing features to change the name; type Return to enter your changes. Click SELECT in an empty portion of the pane to deselect the text.

Workspace Selection

Select workspace icons by moving the pointer over the icon and clicking SELECT. You can extend your selection to any number of additional workspaces by moving the pointer over their icons and clicking ADJUST. You can also select groups of icons by pressing SELECT in an open area and dragging the pointer to surround a group of icons with a bounding box. All objects within the box are selected.

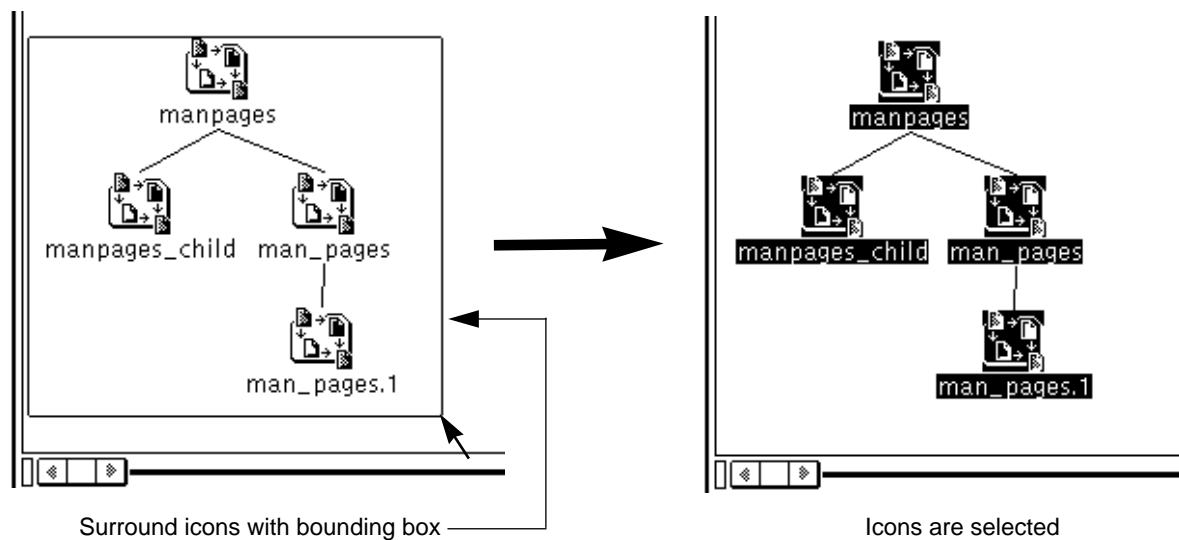


Figure 3-5 Group Selection of Workspaces

Once you have selected workspaces, you can choose CodeManager commands from the control area (see the following section) to act upon the workspaces.

Workspace Pop-up Menu

A pop-up window that contains the most frequently used commands is available by pressing the MENU mouse button while the pointer is in the Workspace Graph pane.

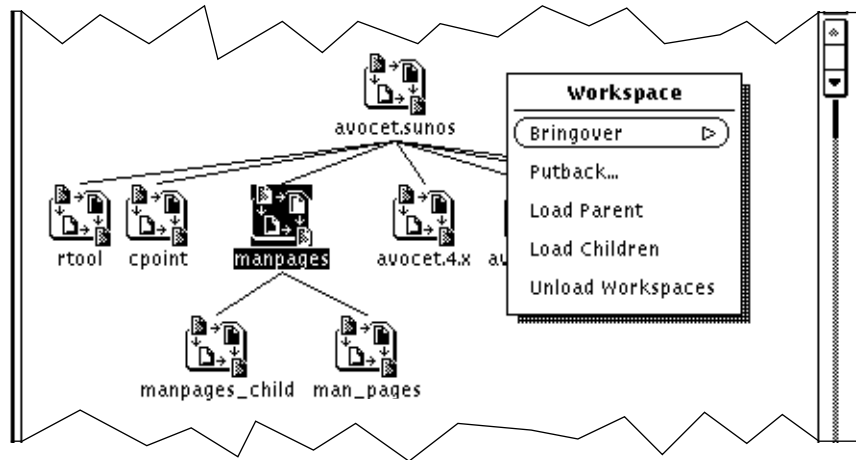


Figure 3-6 Workspace Pop-up Menu in the Workspace Graph Pane

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, CodeManager 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 3-1 summarizes these actions. For more information about interworkspace transactions, see Chapter 5, “Copying Files between Workspaces.”

Table 3-1 Workspace Drag and Drop Action

Drag:	To:	Action
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.¹ 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 Section , "Reparenting a Workspace," on page 71.

Double-Click Action

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

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

1. You are prompted to confirm the reparent operation.

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

CodeManager Window Control Area

Menu Buttons

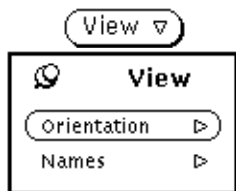
The CodeManager window control area contains six menu buttons.

Use items on the *File* menu to perform the following tasks:



- Load workspaces from specified directories into the Workspace Graph pane.
- Create a new workspace.
- Create a new child workspace.
- Unload workspaces from the Workspace Graph pane.

Use items on the *View* menu to perform the following tasks:



- Choose the orientation (vertical vs. horizontal) of the workspace hierarchy display in the Workspace Graph pane.
- Choose whether workspace names in the Workspace Graph pane are displayed in “full” or “short” format.

Use items on the *Edit* menu to perform the following tasks:



- Delete selected workspaces.
- Rename the selected workspace.
- Change the parent of the selected workspace.



Use the *Transactions* menu to perform the following tasks:

- Bring over files from a selected parent workspace to a new, or existing, child workspace.
- Put back files from a selected child workspace to its parent workspace.
- In the selected workspace, resolve conflicts that were created during a Bringover transaction.
- Undo (reverse) the action of the last Putback or Bringover transaction in the selected workspace.



Use the *Properties* menu to perform the following tasks:

- Customize one or more workspace's properties (Codemgr_wsdata).
- Customize aspects of CodeManager behavior.
- Display the version number of the *TeamWare* release.



Use the *Tools* menu to:

- Launch other TeamWare tools directly from CodeManager.

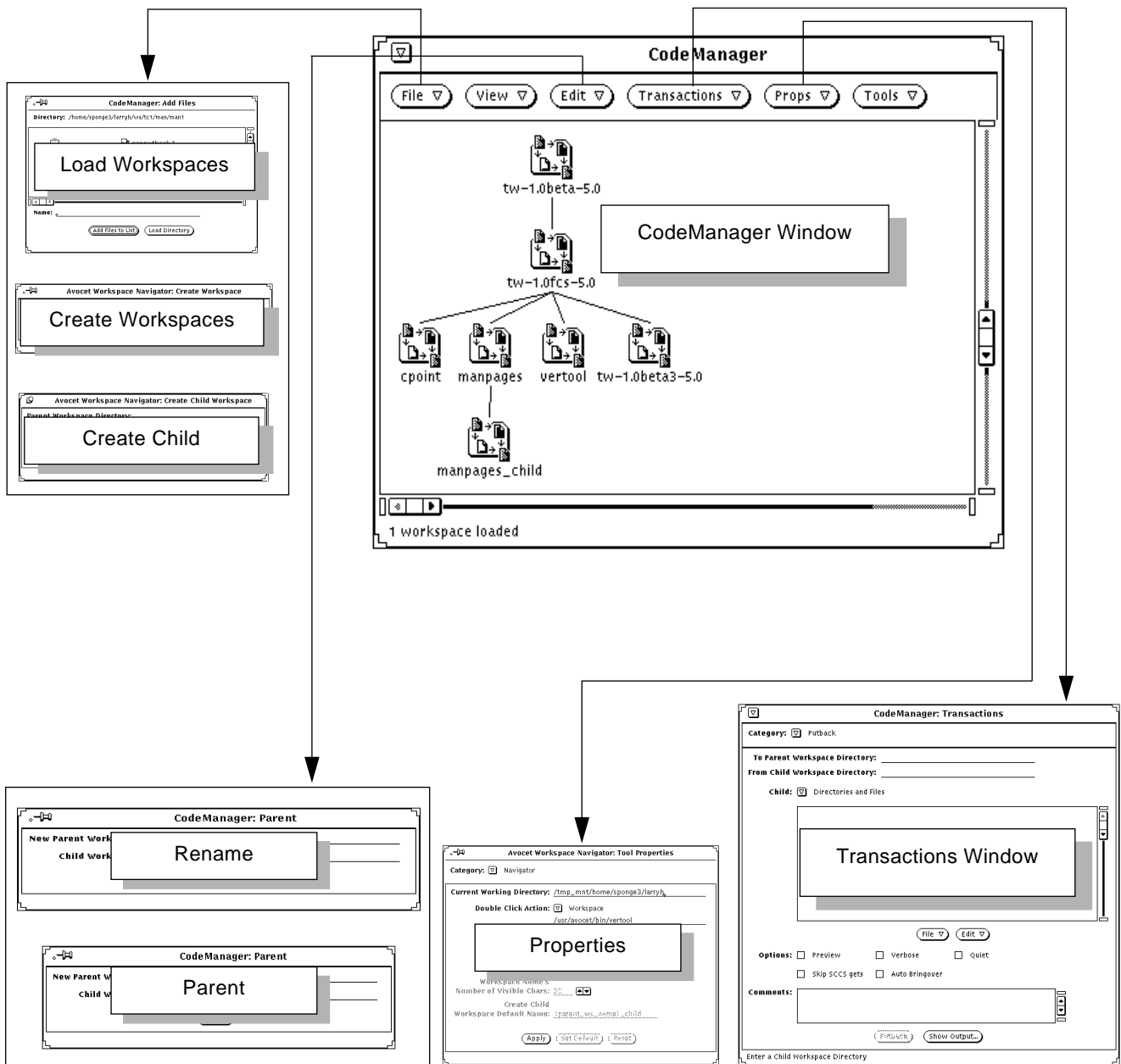


Figure 3-7 CodeManager Subwindows

Transactions Window



The Transactions window is activated when you choose any of the items on the CodeManager window Transactions menu, or the Bringover and Putback items on the pop-up window in the Workspace Graph pane.

The Transactions window is a base window that you use to initiate and complete interworkspace transactions. These transactions are:

- **Bringover Create** Bring over files into a previously nonexistent child workspace from the selected (or specified) parent workspace. Refer to Section , “Creating a New Child Workspace (Bringover Create),” on page 104 for information about the Bringover Create transaction.
- **Bringover Update** Update the contents of the selected (or specified) child workspace by bringing over files from its parent workspace. Refer to Section , “Updating an Existing Child Workspace (Bringover Update),” on page 110 for information about the Bringover Update transaction.
- **Putback** Put back files from the selected (or specified) child workspace into its parent workspace. Refer to Section , “Copying Files from a Child to a Parent Workspace (Putback),” on page 117 for information about the Putback transaction.
- **Resolve** Resolve conflicts in the selected (or specified) workspace. Refer to Chapter 6, “Resolving Conflicts” for information about resolving conflicts.
- **Undo** Undo the action of the last Bringover or Putback transaction in the selected (or specified) workspace. Refer to Section , “Reversing Bringover and Putback Transactions with Undo,” on page 124 for information about reversing the action of Bringover and Putback transactions.

The layout of the Transactions window changes when you select each of the five transactions listed above. When you use the Category menu button in the control area to select a different transaction, the layout of the entire window is changed to accommodate the new transaction.

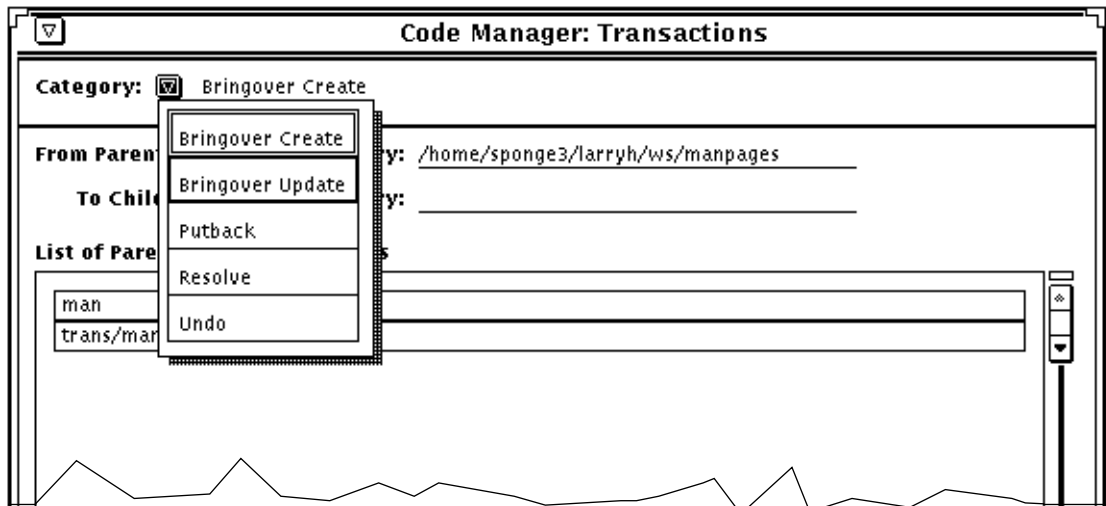


Figure 3-8 Transactions Window Category Menu

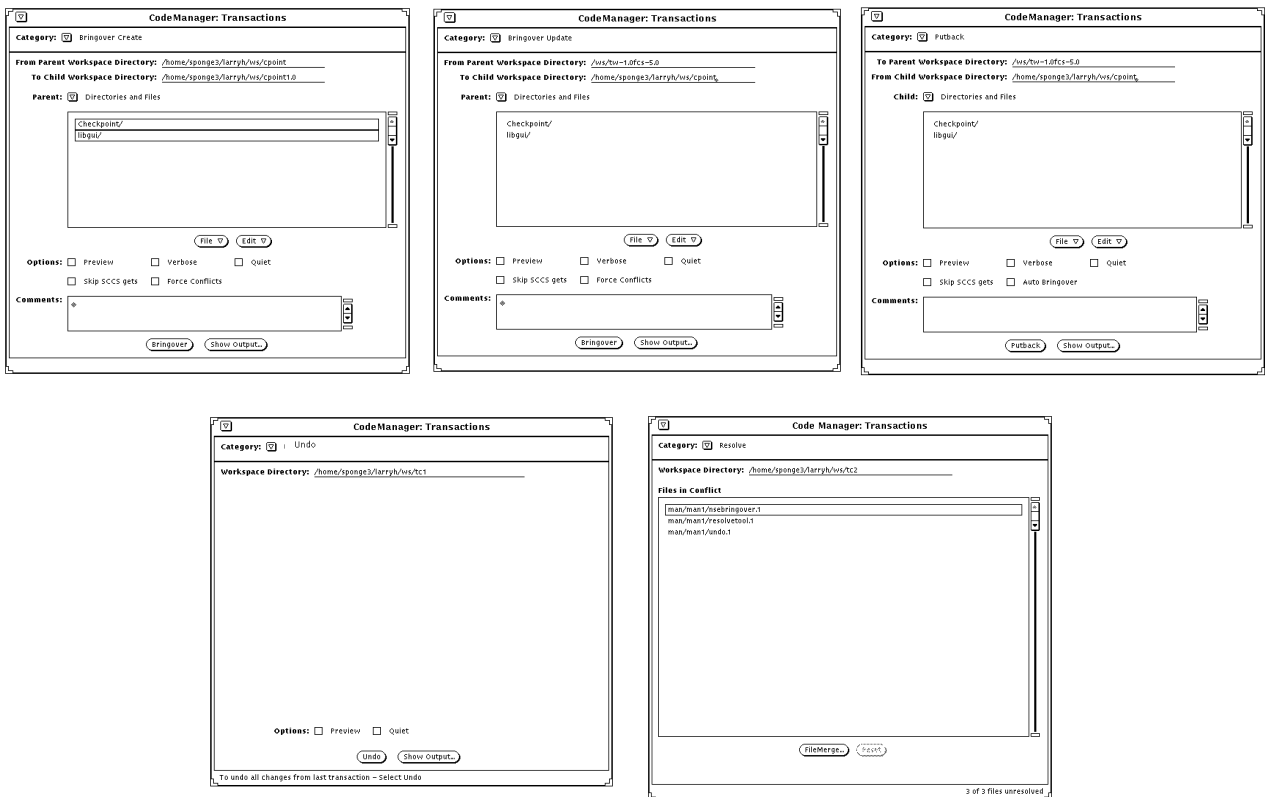


Figure 3-9 Bringover Create, Bringover Update, Putback, Undo, and Resolve Transactions Windows

File List Pane

The File List pane can be used for two functions:

- To construct the list of files and directories you want included in a transaction
- To specify file list programs (FLPs) that generate lists of files for Bringover and Putback transactions

Since the same pane is used for both functions, it has two modes that are controlled by the abbreviated menu directly above the File List pane. Use the CodeManager Chooser window to construct both lists.

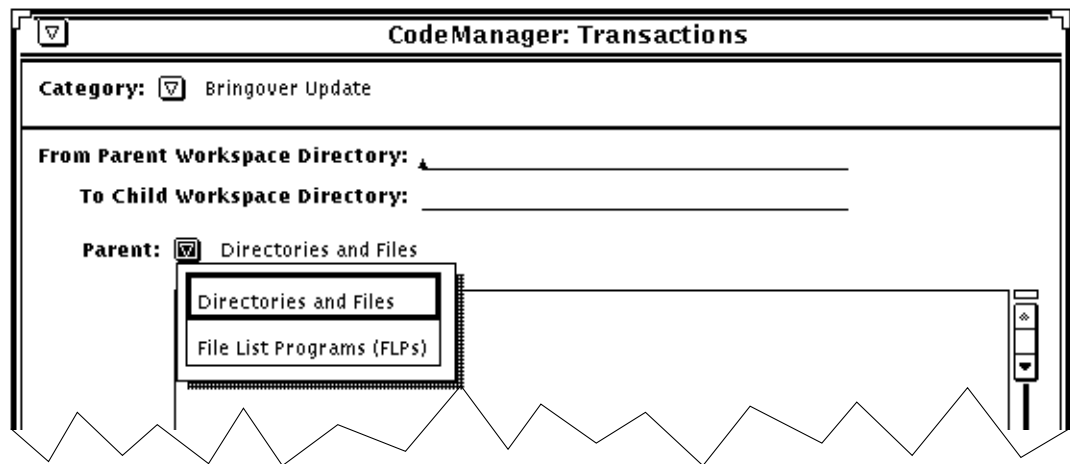


Figure 3-10 File List Pane Menu

Transaction Pop-up Windows

Two pop-up windows can be activated from the Transactions window:

- CodeManager Chooser window

Activate the CodeManager Chooser window by invoking the Add Files to List (or Add FLPs to List) item on the File menu below the File List pane. Use this window to do the following:

- Choose files you wish copied during a Bringover, Putback, or Undo transaction. See Section , “Specifying Directories and Files for Transactions,” on page 98 for more information.
- Choose FLPs you wish executed to generate a list of files to copy during Bringover and Putback transactions. See Section , “Grouping Files for Transfer Using File List Programs,” on page 99 for more information.

- Transaction Output window

CodeManager automatically activates the Transaction Output window when you execute Bringover and Putback transactions. Status messages about the transaction are displayed. You can also activate the Transaction Output window by invoking the Show Output button. See Section , “Viewing Transaction Output,” on page 97 for more information.

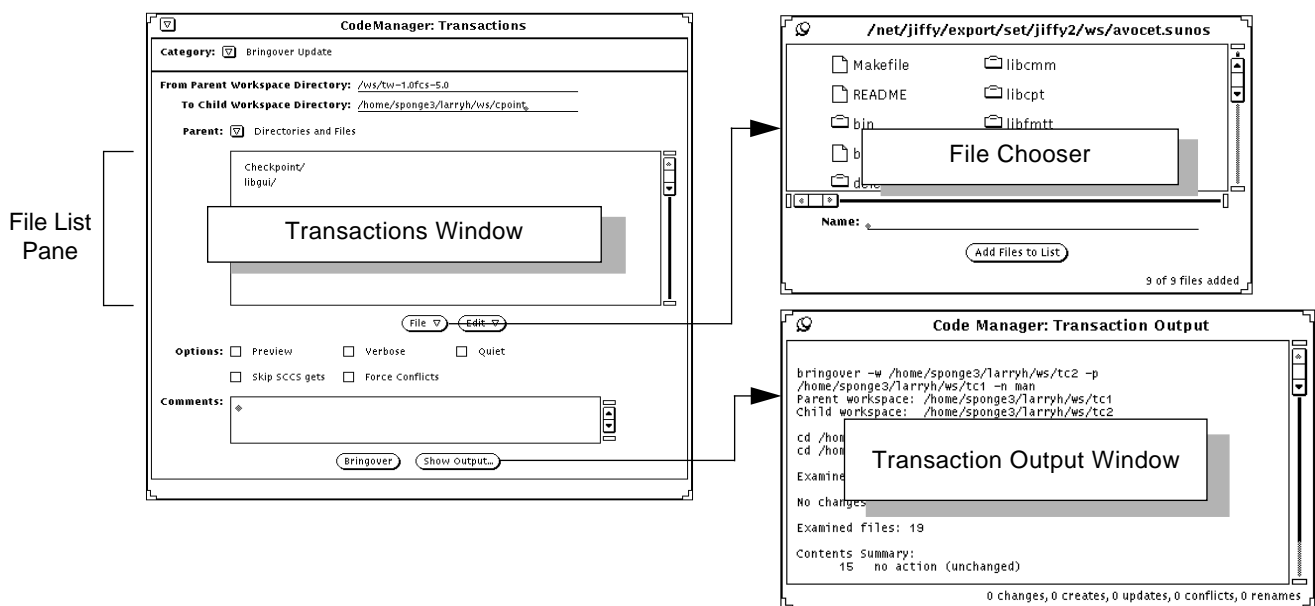


Figure 3-11 Transaction Subwindows

CodeManager Chooser Window

The File List pane is used to construct the list of directories and files that you wish included in a Bringover or Putback transaction. You use the CodeManager Chooser window to move about the file system and select files to add to the file list.

The File Chooser window is activated by selecting the Add Files to List item from the File menu on the Transactions window.

See Section , “Specifying Directories and Files for Transactions,” on page 98 for details on using the chooser to select files for copying.

Transaction Output Window

Output from CodeManager 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.

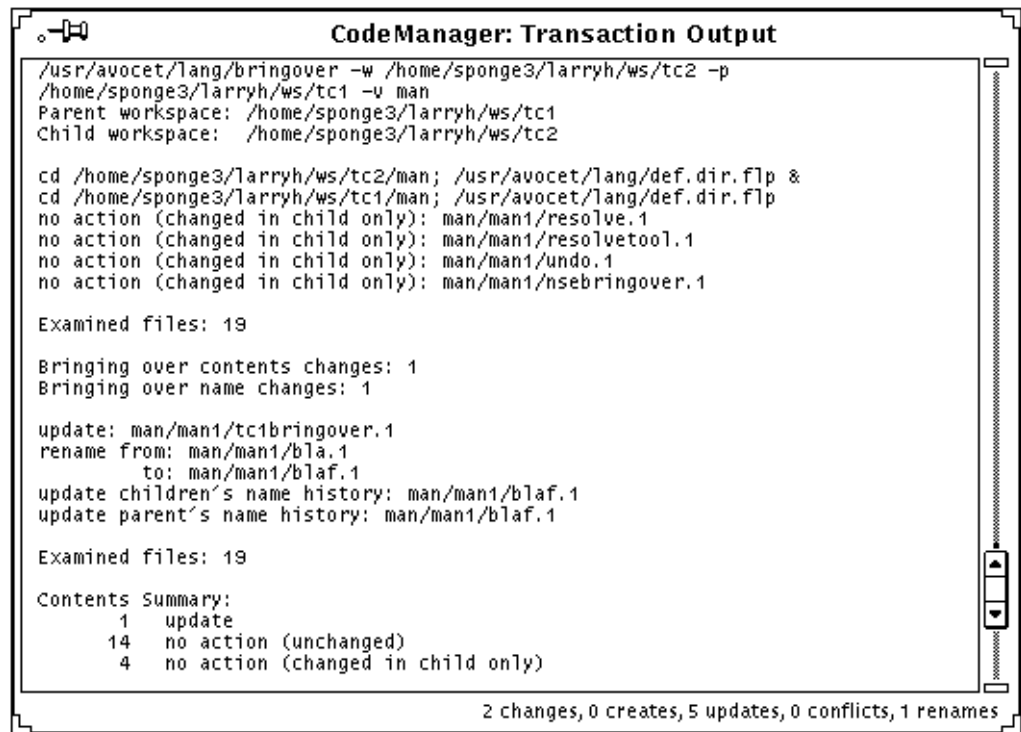


Figure 3-12 Transaction Output Window

Customizing CodeManager Using Properties

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

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

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

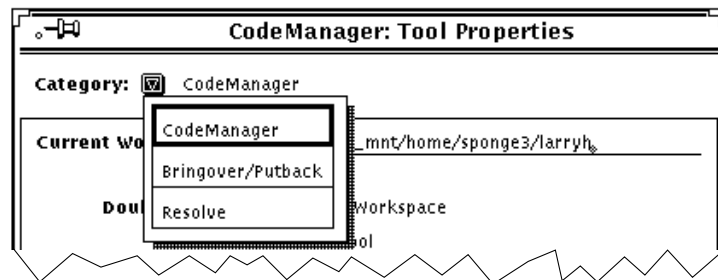


Figure 3-13 Tool Properties Category Menu

CodeManager Defaults Files

When you change CodeManager behavior using the Tool Properties window, you can use the Set Default button to preserve the changes in default files in your home directory. The default files are consulted by CodeManager 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 CodeManager and Bringover/Putback panes of the Tool Properties window are written to the file `~/.codemgrtoolrc`. This file is an OpenWindows XDefaults format file.

CodeManager Pane

The CodeManager pane of the Tool Properties window enables you to change the behavior of the CodeManager base window. The specific properties are described in Table 3-2.

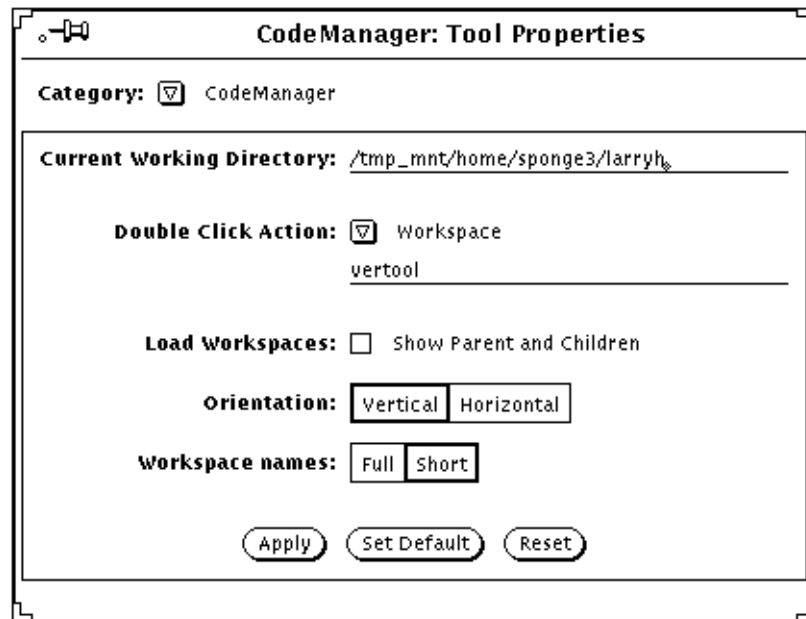


Figure 3-14 Coder Manager Pane of the Tool Properties Window

Table 3-2 CodeManager Tool Properties

Property	Description
Default Directory	Directory to which CodeManager 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 VersionTool (vertool); by default, the Resolve Transaction window (<resolve_pane>) 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 CodeManager 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 CodeManager 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 3-3.

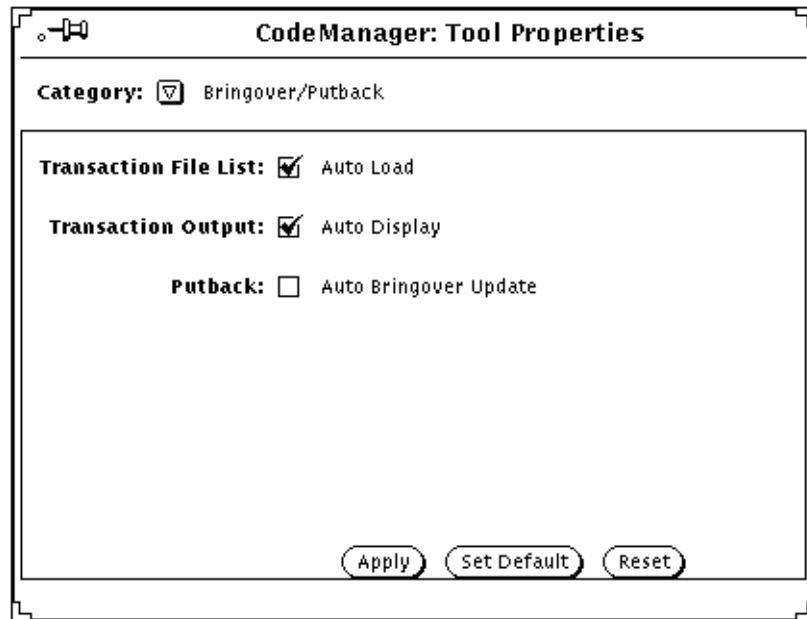


Figure 3-15 Bringover/Putback Pane of the Tool Properties Window

Table 3-3 Bringover/Putback Tool Properties

Property	Description
Auto Load	Causes CodeManager 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 3-4.

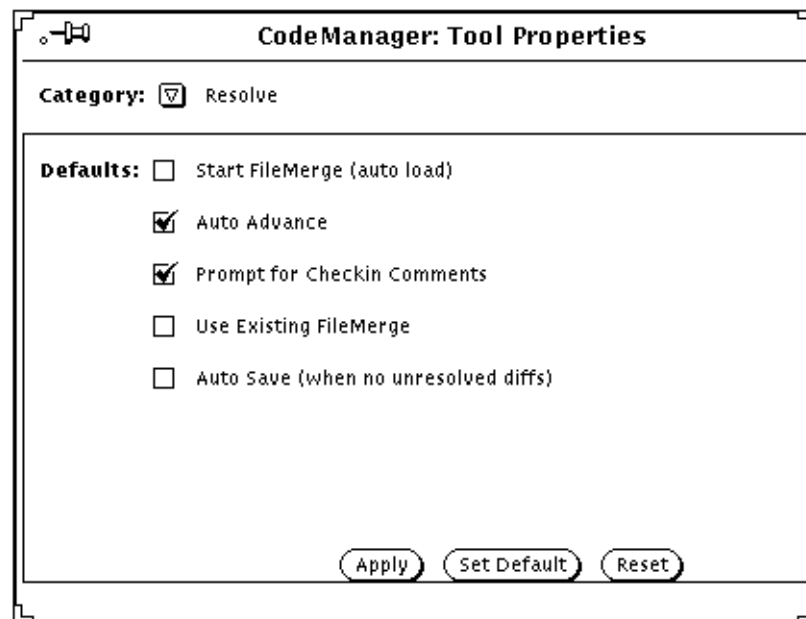


Figure 3-16 Resolve Pane of the Tool Properties Window

Table 3-4 Resolve Tool Properties

Property	Description
Start FileMerge (auto load)	Causes 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 FileMerge after the current file is resolved.
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 FileMerge	If this property is set, an already running FileMerge 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 FileMerge Save button.

Footer Messages

CodeManager provides helpful messages in the footers of both the CodeManager window and the Transactions window.

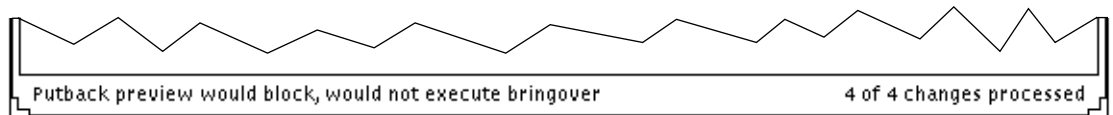


Figure 3-17 A Transaction Window Footer Message

Accelerators

Table 3-5 summarizes the various accelerators available for CodeManager functions.

Table 3-5 Summary of CodeManager Accelerators

Accelerator	Action	Where to Find More Information
Drag and drop workspace icon	Activate Bringover/Putback transaction window	“Dragging and Dropping Workspace Icons” on page 46
SHIFT + drag and drop workspace icon	Reparent workspace	“Dragging and Dropping Workspace Icons” on page 46
Click SELECT on workspace icon name field	Rename workspace	“Workspace Name Fields” on page 44
Double-click SELECT on workspace icon	Launch a tool. User configurable, VersionTool is the default	“Double-Click Action” on page 47
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” on page 47

CodeManager Workspace



As discussed in Chapter 2, “Introduction to CodeManager,” the *workspace* forms the basis of the CodeManager 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 CodeManager workspace, refer to “Workspace” on page 21 of this manual.

This chapter discusses specific aspects of workspaces and the CodeManager commands you use to configure, create, manipulate, and administer them.

The Workspace Metadata Directory

A CodeManager workspace is a directory hierarchy that contains a directory named `Codemgr_wsdata` in its root directory. CodeManager stores data (metadata) about that workspace in `Codemgr_wsdata`. CodeManager 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 4-1 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(5)` page for each file.

Table 4-1 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 CodeManager 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 77.
<code>args</code>	The <code>args</code> file is maintained by the CodeManager 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 CodeManager uses to “undo” a Bringover or Putback transaction. See Section , “Reversing Bringover and Putback Transactions with Undo,” on page 124.
<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. CodeManager consults this file to obtain the list of child workspaces. When you delete, move, or reparent a workspace, CodeManager updates the <code>children</code> file in its parent.
<code>conflicts</code>	The <code>conflicts</code> file contains a list of files in that workspace that are currently in conflict. See Chapter 6, “Resolving Conflicts,” for more information about conflicts and how to resolve them.
<code>history</code>	The <code>history</code> file is a historical log of transactions and updated files that affect a workspace. See Section , “Viewing Workspace Command History,” on page 87 for more information.

Table 4-1 Contents of the Codemgr_wsdata Metadata Directory

File/Dir Name	Description
locks	To assure consistency, CodeManager locks workspaces during Bringover, Putback and Undo transactions. Locks are recorded in the <code>locks</code> file in each workspace; CodeManager consults that file before acting in a workspace. See Section , “Ensuring Consistency through Workspace Locking,” on page 90.
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 CodeManager during Bringover and Putback to accelerate the processing of files that have been renamed. If this file is not available, CodeManager rebuilds it automatically during the next Putback or Bringover transaction. See Section , “Renaming, Moving, or Deleting Files,” on page 128.
notification	The <code>notification</code> file is edited by users to register notification requests. This facility permits CodeManager 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 83.
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). CodeManager consults this file to determine a workspace’s parent. When you delete, move, or reparent a workspace, CodeManager 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. CodeManager 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 CodeManager 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 CodeManager 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, CodeManager 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, CodeManager creates both the root directory and the `Codemgr_wsdata` directory.



Figure 4-1 Create Workspace Pop-up Window

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 Section , “Creating a New Child Workspace (Bringover Create),” on page 104 for details.

Deleting a Workspace

You delete workspaces by selecting their icons in the Workspace Graph Pane and then invoking the Delete ⇒ item from the CodeManager 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 CodeManager automatically updates records in parent and child workspaces to reflect the deletion of the workspace.

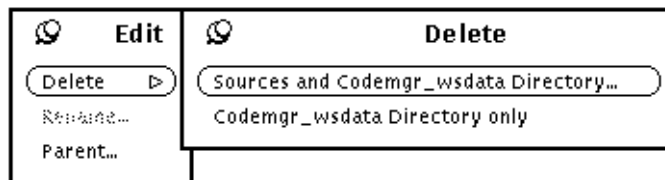


Figure 4-2 The Edit And Delete Pop-up Windows

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 the standard OpenWindows text editing features 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 CodeManager 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 65.

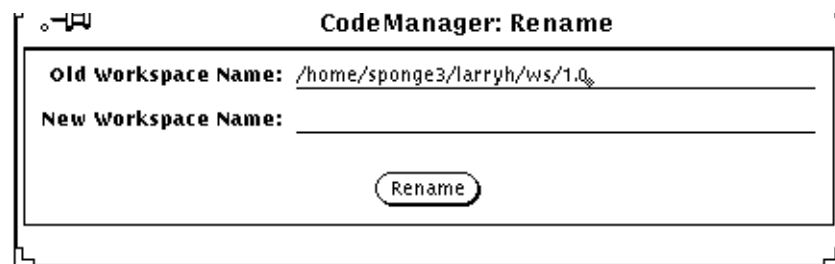


Figure 4-3 Rename Pop-up Window

A Note About Moving Workspaces

Do not use the SunOS mv command to rename or move workspaces. The CodeManager 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 CodeManager 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 2, “Introduction to CodeManager,” (page 23) the parent/child relationship is the thread that connects the workspace hierarchy. CodeManager 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 Chapter 5, “Copying Files between Workspaces,” 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.¹ The display is automatically adjusted to reflect the new relationship.

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.

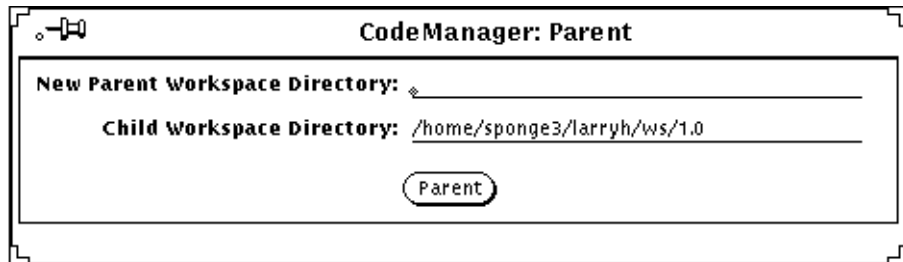


Figure 4-4 Parent Pop-up Window

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.

1. You are prompted to confirm the change.

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:

- a. Create a new (empty) Release 2 workspace by means of the Create Workspace command item.
- b. Use either of the two methods described above to make the Release 2 workspace the new parent of the Release 1 workspace.
- c. Use the Putback transaction to copy files to the Release 2 workspace.
- d. 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 CodeManager 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 CodeManager, 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 4-5)

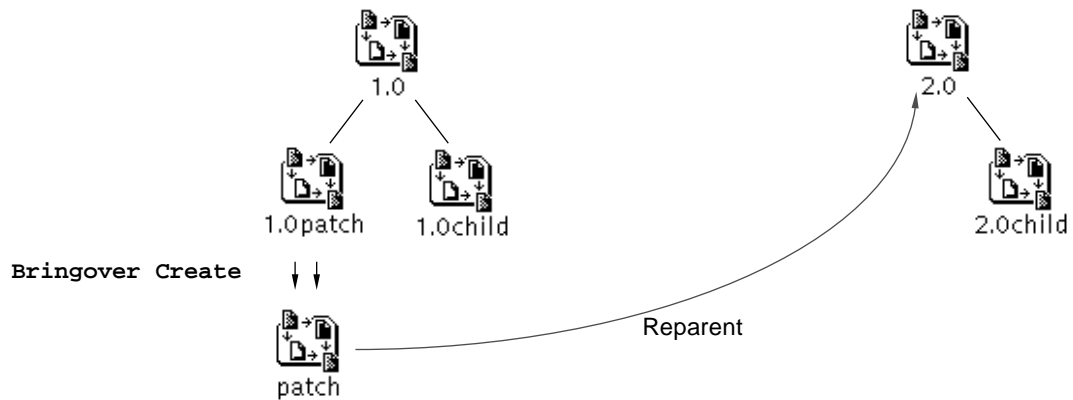


Figure 4-5 Patch Workspace Reparented to New Release

3. The workspace is then updated from its new parent, and any new work is brought over from 2.0. (Figure 4-6A)
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 4-6B)

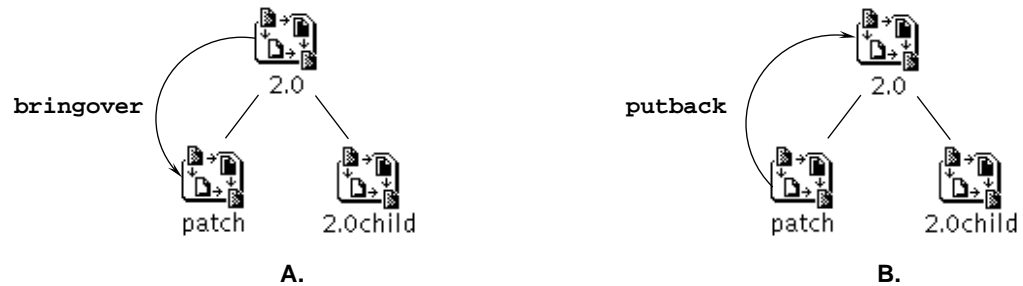


Figure 4-6 Files Brought Over, Merged, and Incorporated into the New Release

5. 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 4-7)



Figure 4-7 Patched Files Brought Over into 2.0child; patch Deleted

Controlling Access to Workspaces

CodeManager permits you to control the access that users have to your workspaces. Table 4-2 lists and describes the eight types of access over which you can exercise control.

Table 4-2 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, CodeManager 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.

Use the Workspace item in the Props menu to view and change access permissions. Use the Category menu to choose the Access Control pane of the Workspace Properties pop-up window (see “Viewing and Changing Access Control Values” on page 80).

Table 4-3 shows the default contents of `access_control` after you create a workspace:

Table 4-3 Default Access Control Permissions

Operation	Permissions
bringover-from	
bringover-to	creator ¹
putback-from	
putback-to	
undo	
workspace-delete	creator
workspace-move	creator
workspace-reparent	creator
workspace-reparent-to	

1. Creator's login name actually appears

You can express which users have or do not have access to a workspace in a number of ways. Figure 4-8 shows all of the possible types of values you can specify to control access to your workspaces. Table 4-4 describes what the entries mean.

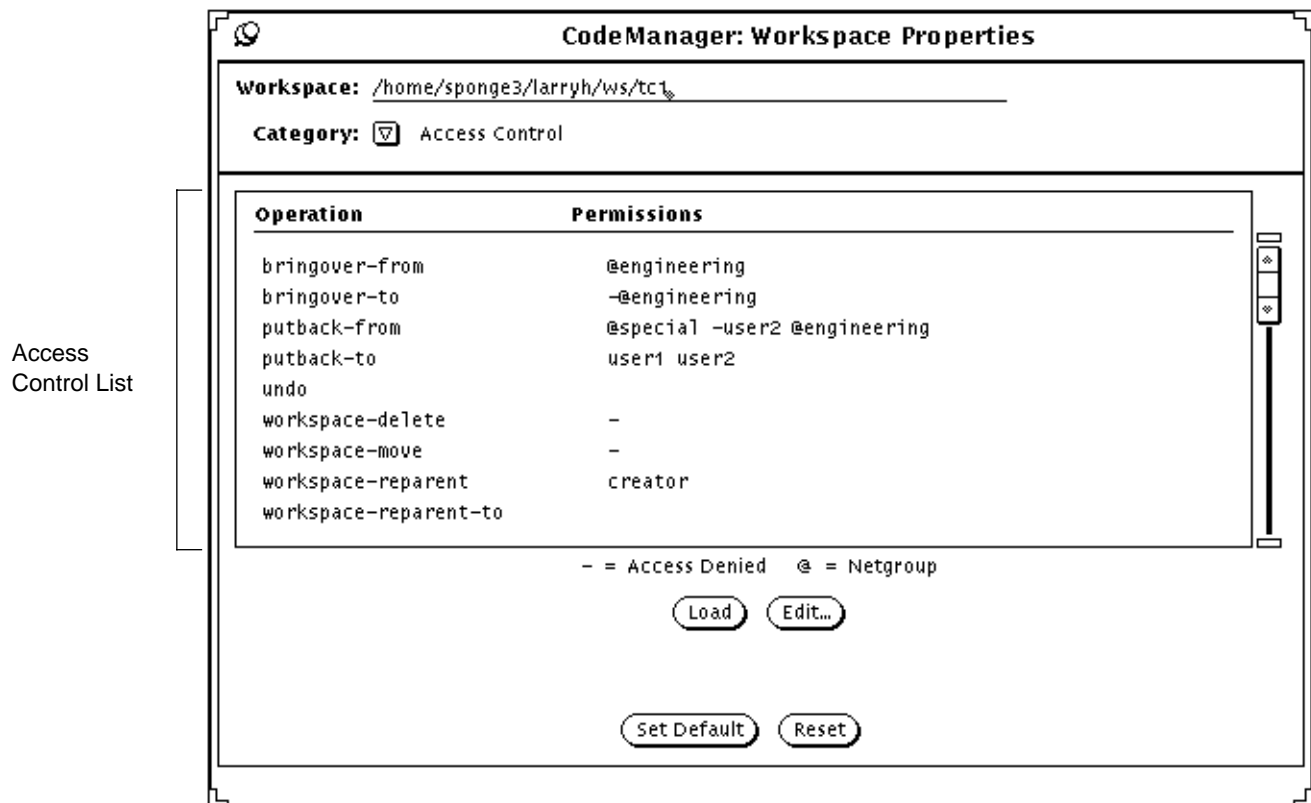


Figure 4-8 Example Access Control Values

Table 4-4 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 Props 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 Props 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.**

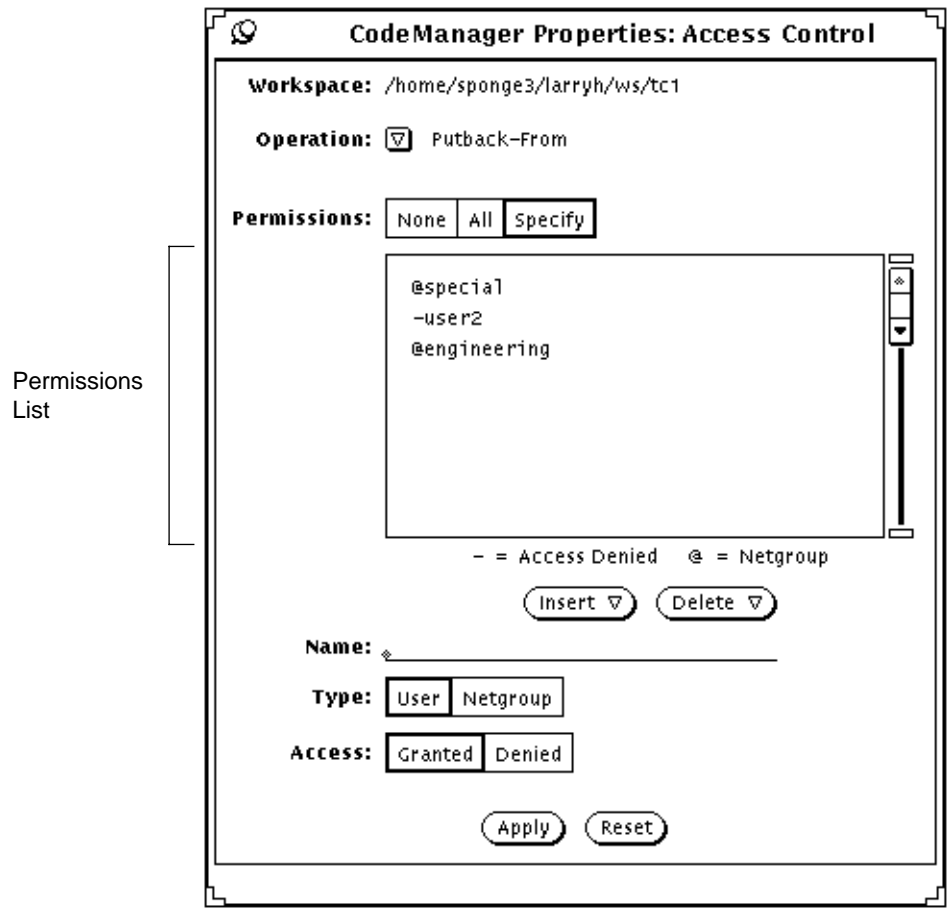


Figure 4-9 Access Control Edit Pop-up

How to Notify Users of Changes to Workspaces

You can request CodeManager to notify you (through an electronic mail message) when a variety of CodeManager 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 4-5 lists the events for which you can register notification requests:

Table 4-5 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 Props menu. Use the Category menu to choose the Notification pane.

Figure 4-10 shows how the requests contained in the `notification` file described on the previous page are displayed in the Workspace Properties pop-up.

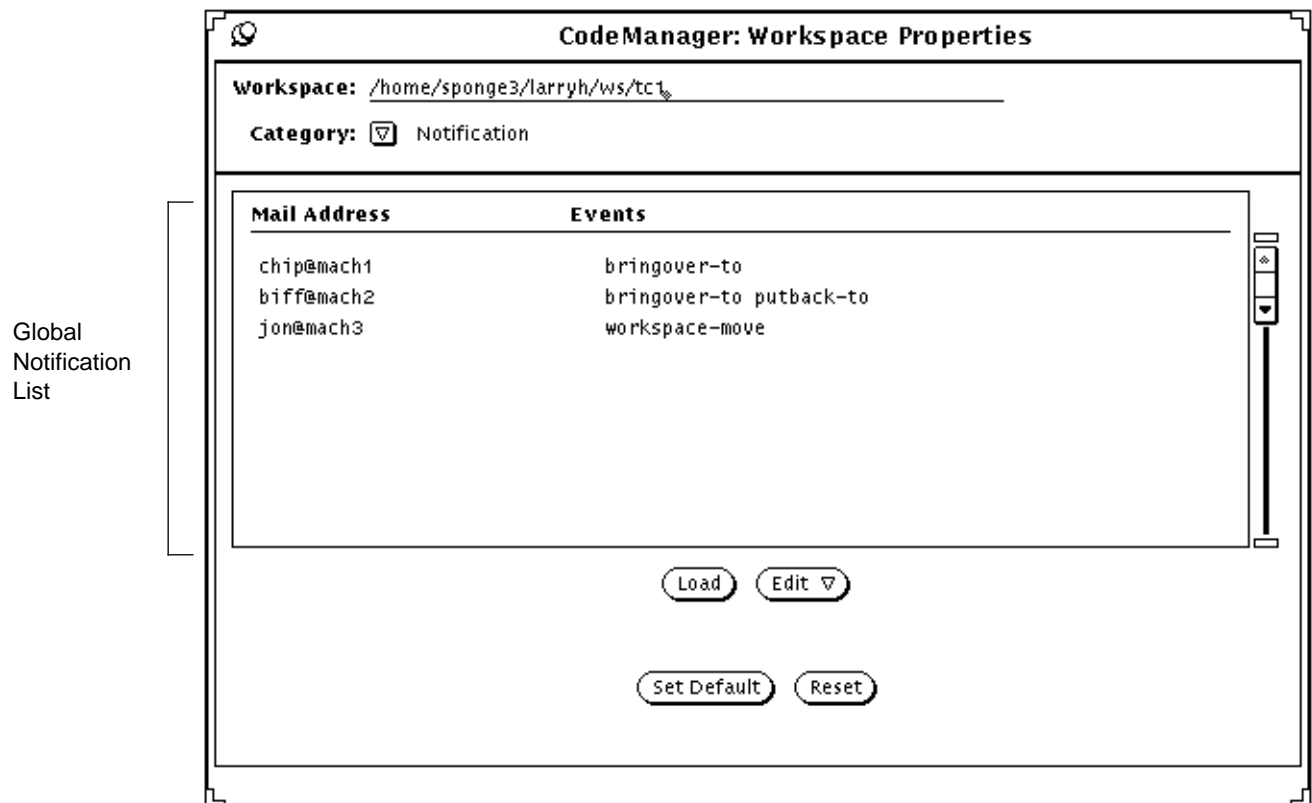


Figure 4-10 Example Notification Entries

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 (for information about CodeManager chooser windows see Section , “CodeManager File and Directory Choosers,” on page 41). Delete files from the list using the Delete button.

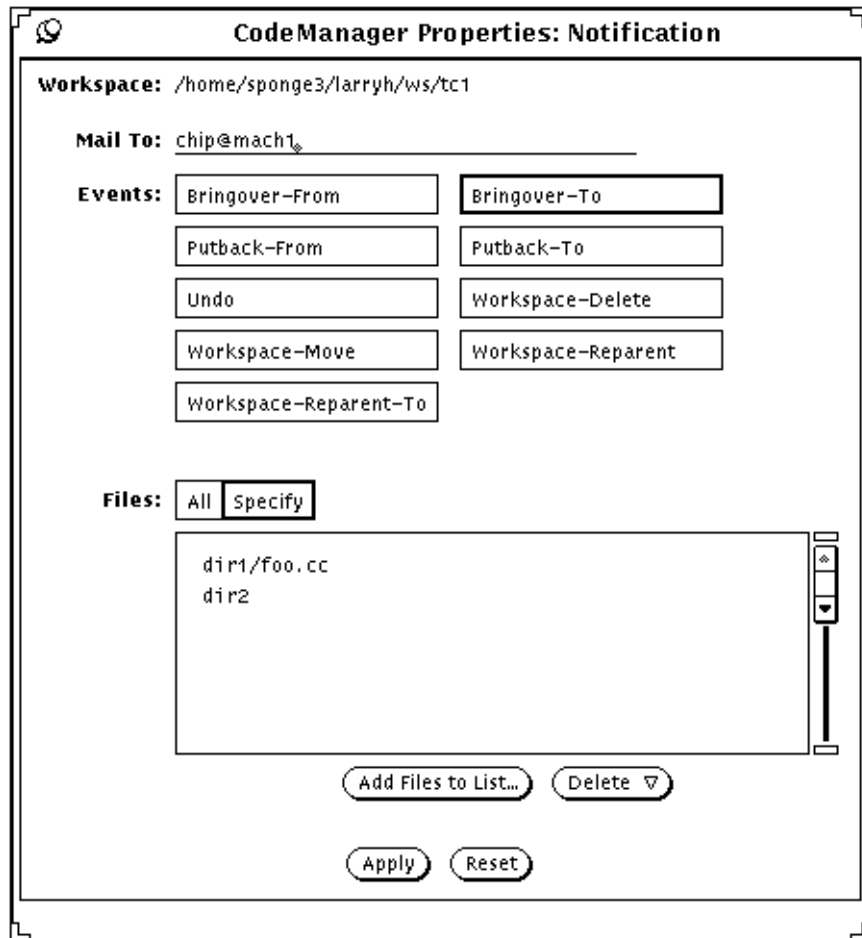


Figure 4-11 Notification Edit Pop-up

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

CodeManager 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.¹

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

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

have any questions about the meaning or syntax of a command, refer to its `man` page for details. Table 4-6 lists the GUI operations and the corresponding CLI command that is entered in the history log.

Table 4-6 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 4-7. 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 4-7 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 CodeManager transactions—Bringover, Undo, and Putback—lock workspaces while they are working in them. These locks only affect CodeManager transactions; other commands such as SCCS programs, are

not affected. Locks are recorded in the `Codemgr_wsdata/locks` file in each workspace; the CodeManager 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 CodeManager 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 CodeManager 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 CodeManager command is unable to remove its lock after completion (for example, the system crashes), you must remove the lock yourself before CodeManager commands will again be able to read and/or write in the workspace. You can use the CodeManager GUI to view and delete active locks for a workspace, or you can edit the file directly.

To view and delete locks using the CodeManager 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.

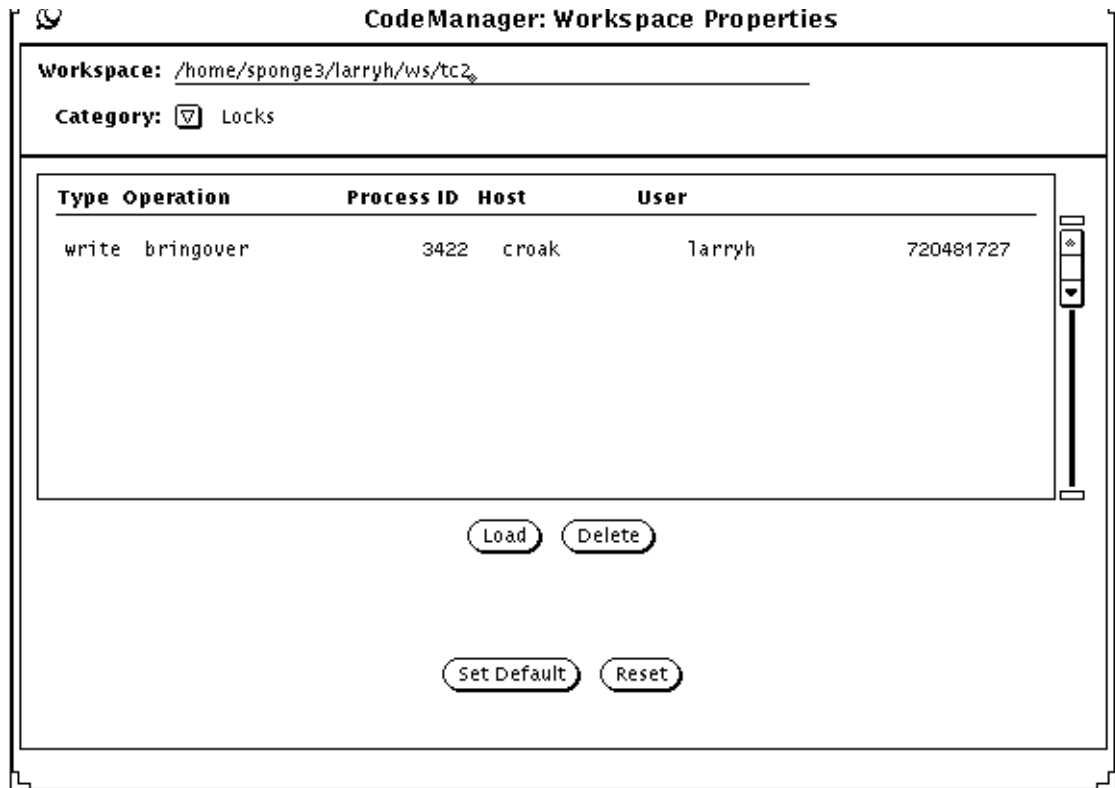


Figure 4-12 Example Lock Entry

CodeManager Environment Variables

CodeManager 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 CodeManager 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, CodeManager automatically loads workspaces from directory path names specified in the CODEMGR_WSPATH variable.

Copying Files between Workspaces



Chapter 2, “Introduction to CodeManager,” describes copying files up and down the parent/child hierarchy. This chapter describes how you use CodeManager to copy files.

The chapter covers the following topics:

- CodeManager transaction model
- Information common to all of the workspace transactions
- How you use the Bringover and Putback transactions to copy files
- How you can “undo” the actions of Bringover and Putback transactions
- How to correctly delete, move, and rename files and directories
- All of the transactions discussed in this chapter are initiated in the CodeManager Transactions window.

An example demonstrating these transactions can be found in Chapter 9, “CodeManager Example.”

CodeManager Transaction Model

CodeManager 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 5-1. The ways in which the transactions differ are described later in this chapter (the Resolve transaction is described in Chapter 6, “Resolving Conflicts”).

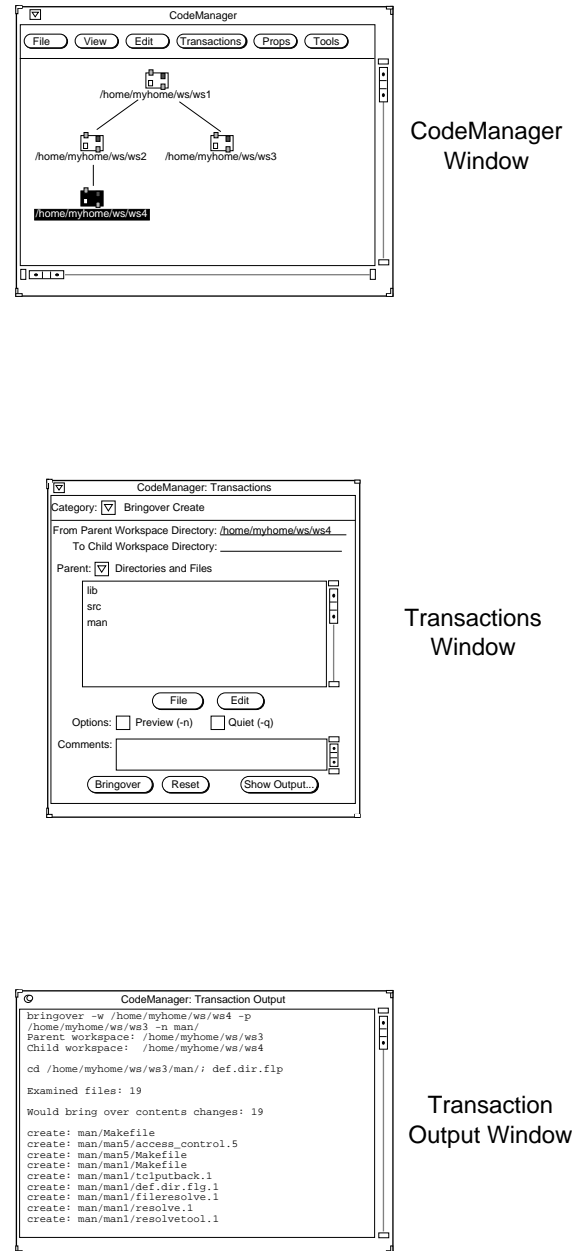
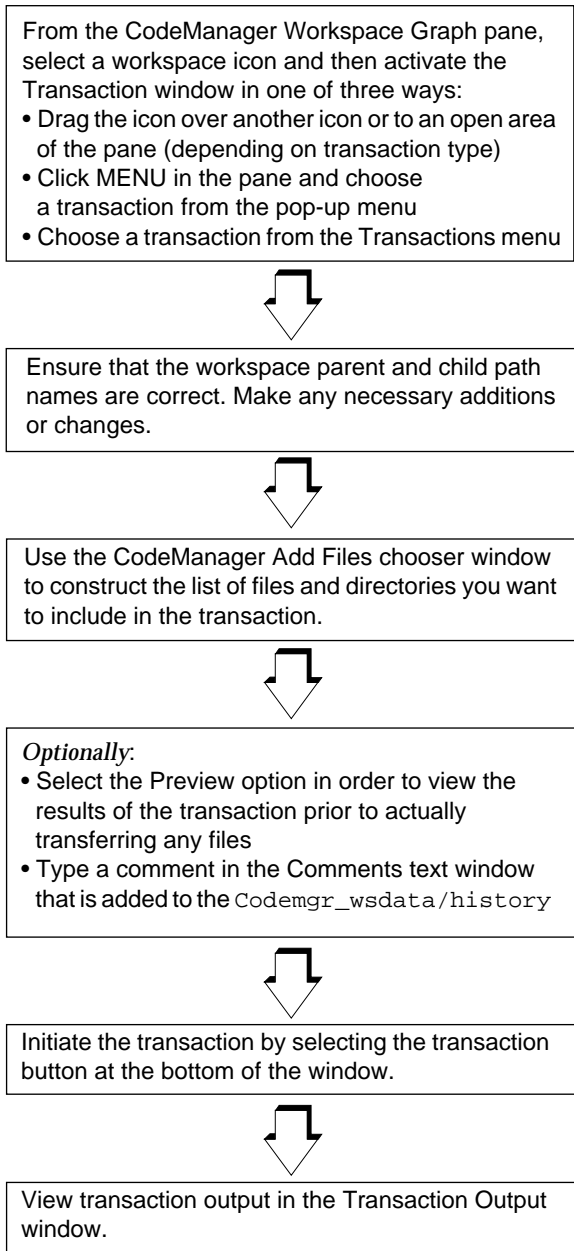


Figure 5-1 CodeManager Transaction Model

General File Copying Information

This section contains background information about copying files between workspaces.

SCCS History Files

When considering CodeManager 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, CodeManager actually acts upon (copies or merges) the file's SCCS history file (also known as the "s-dot-file").

The means by which CodeManager manipulates and merges the history files is described in detail in Chapter 8, "How CodeManager Merges SCCS Files."

Viewing Transaction Output

Output from CodeManager 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.

```

CodeManager: Transaction Output
/usr/avocet/lang/bringover -w /home/sponge3/larryh/ws/tc2 -p
/home/sponge3/larryh/ws/tc1 -v man
Parent workspace: /home/sponge3/larryh/ws/tc1
Child workspace: /home/sponge3/larryh/ws/tc2

cd /home/sponge3/larryh/ws/tc2/man; /usr/avocet/lang/def.dir.flp &
cd /home/sponge3/larryh/ws/tc1/man; /usr/avocet/lang/def.dir.flp
no action (changed in child only): man/man1/resolve.1
no action (changed in child only): man/man1/resolve.1
no action (changed in child only): man/man1/undo.1
no action (changed in child only): man/man1/nsebringover.1

Examined files: 19

Bringing over contents changes: 1
Bringing over name changes: 1

update: man/man1/tc1bringover.1
rename from: man/man1/bla.1
         to: man/man1/blaf.1
update children's name history: man/man1/blaf.1
update parent's name history: man/man1/blaf.1

Examined files: 19

Contents Summary:
  1 update
 14 no action (unchanged)
  4 no action (changed in child only)

2 changes, 0 creates, 5 updates, 0 conflicts, 1 renames
  
```

Figure 5-2 Transaction Output Window

Note – CodeManager 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.

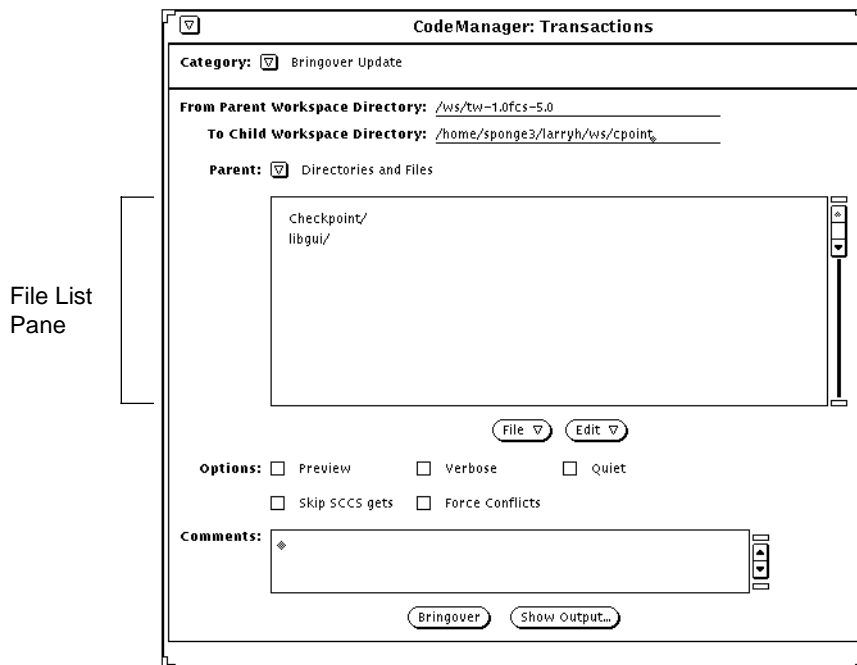


Figure 5-3 Transactions Window

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.

CodeManager 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 102 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

CodeManager 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:

- Bringover Create The initial list is empty.
- Bringover Update The initial list is retrieved from the `Codemgr_wsdata/args` file in the child workspace. This file contains a list of arguments specified during previous Bringover and Putback transactions.
- Putback The initial list is retrieved from the `Codemgr_wsdata/args` 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 CodeManager 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, CodeManager 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, CodeManager 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, CodeManager Chooser. See “Add Files Chooser” on page 102 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 CodeManager Chooser.

Add Files Chooser

You can use the Add Files chooser to conveniently add directories and files to the Transaction window File List pane.¹ The CodeManager 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.

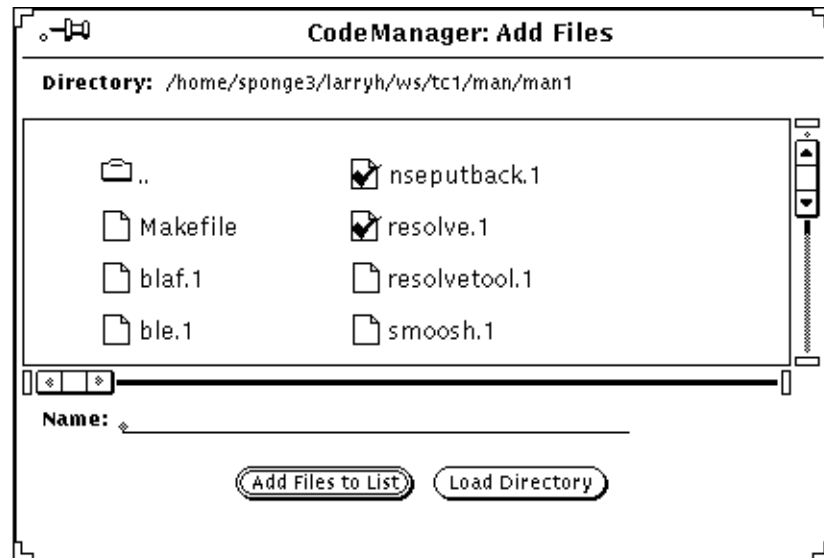


Figure 5-4 The Add Files to List Window

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.

1. The CodeManager 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.

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 CodeManager 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 CodeManager workspaces. You cannot Putback files to directories that are not workspaces.

Creating a New Child Workspace (Bringover Create)

You use the CodeManager 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.

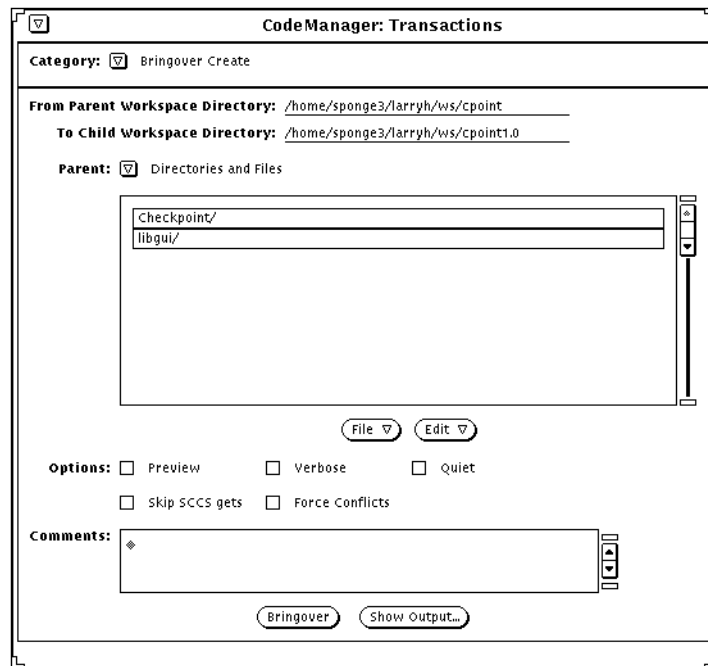


Figure 5-5 Transactions Window Bringover Create Layout

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 materialized through the SCCS `get` command.

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

1. Specify the parent workspace.¹

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.

1. 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.¹

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 Section , “Specifying Directories and Files for Transactions,” on page 98 for information about specifying directory and file arguments.

4. Select options.

- 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.

- 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.

- Quiet Select this option to suppress the output of status messages to the Transaction Output window (Show Output).

- Skip SCCS gets Select this option to inhibit the automatic invocation of the SCCS `get` program as part of the Bringover transaction. Normally g-files are extracted after they are brought over. This option

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

improves file transfer performance although it shifts the responsibility to the user to do the appropriate gets at a later time.

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, CodeManager 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 5-1 shows the different actions that CodeManager takes when it encounters checked-out files.

Table 5-1 Effects of Checked-out Files on Bringover Create Transactions

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

- 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 124 for details.
- While CodeManager 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 CodeManager 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 CodeManager 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 77 for more information.

-
- CodeManager 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 87 for further information regarding these files.
 - CodeManager 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 CodeManager commands are installed.

Updating an Existing Child Workspace (Bringover Update)

You use the CodeManager 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.

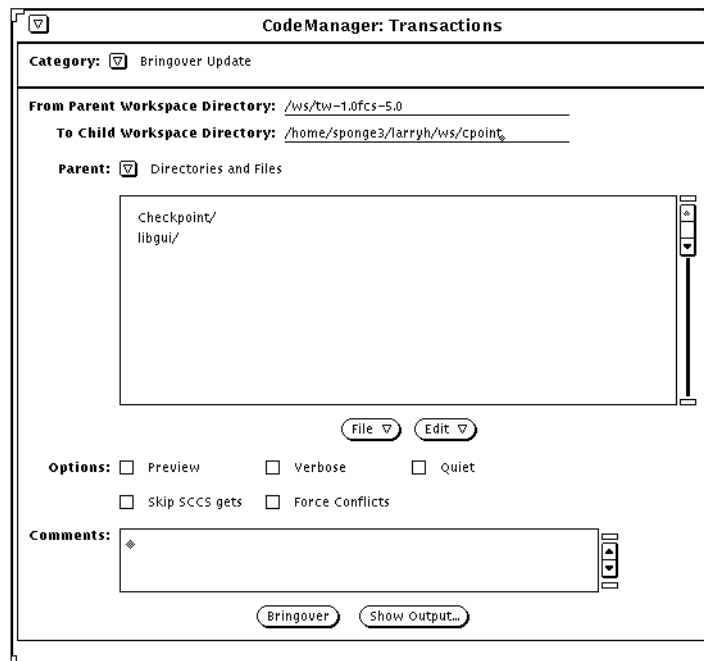


Figure 5-6 Transactions Window Bringover Update Layout

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 materialized through the SCCS `get` command. When a file exists in both workspaces and has changed only in the parent, CodeManager copies the new deltas from the parent to the child. When a file has changed in both workspaces, CodeManager 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.¹

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 CodeManager metadata file named `Codemgr_wsdata/parent`.

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 CodeManager window Edit menu or drag the child workspace icon over the new parent's icon. See Section , "Reparenting a Workspace," on page 71 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, CodeManager automatically updates the parent field if you rechoose the Bringover Update item in the Category menu.

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

3. Create a list of directory and file names in the File List Pane.¹

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 Section , “Specifying Directories and Files for Transactions,” on page 98 for information about specifying directory and file arguments.

4. Select options.

- | | |
|----------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <input checked="" type="checkbox"/> Preview | Select this option to preview the results of the transaction. If you invoke the Bringover Update 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. |

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

Force Conflicts

Select this option to cause all file updates to be treated as conflicts.

5. Invoke the Bringover button to initiate the transaction.

Notes about the Bringover Update Transaction

- Checked-out files

When, during a Bringover Update transaction, CodeManager 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 5-2 shows the different actions that CodeManager takes when it encounters checked-out files.

Table 5-2 Effects of Checked-out Files on Bringover Update Transactions

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

- 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 6, “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 Section , “Reversing Bringover and Putback Transactions with Undo,” on page 124 for details.

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

Read-locks may be obtained concurrently by multiple CodeManager 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 CodeManager 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 Section , “Controlling Access to Workspaces,” on page 77 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 87 for further information regarding these files.
- CodeManager 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 CodeManager commands are installed.

Bringover Action Summary

Table 5-3 summarizes the actions that CodeManager takes during Bringover transactions.

Table 5-3 Summary of CodeManager Action during a Bringover Transaction

File in Parent	File in Child	Action by CodeManager
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 CodeManager 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, CodeManager 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, CodeManager 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.

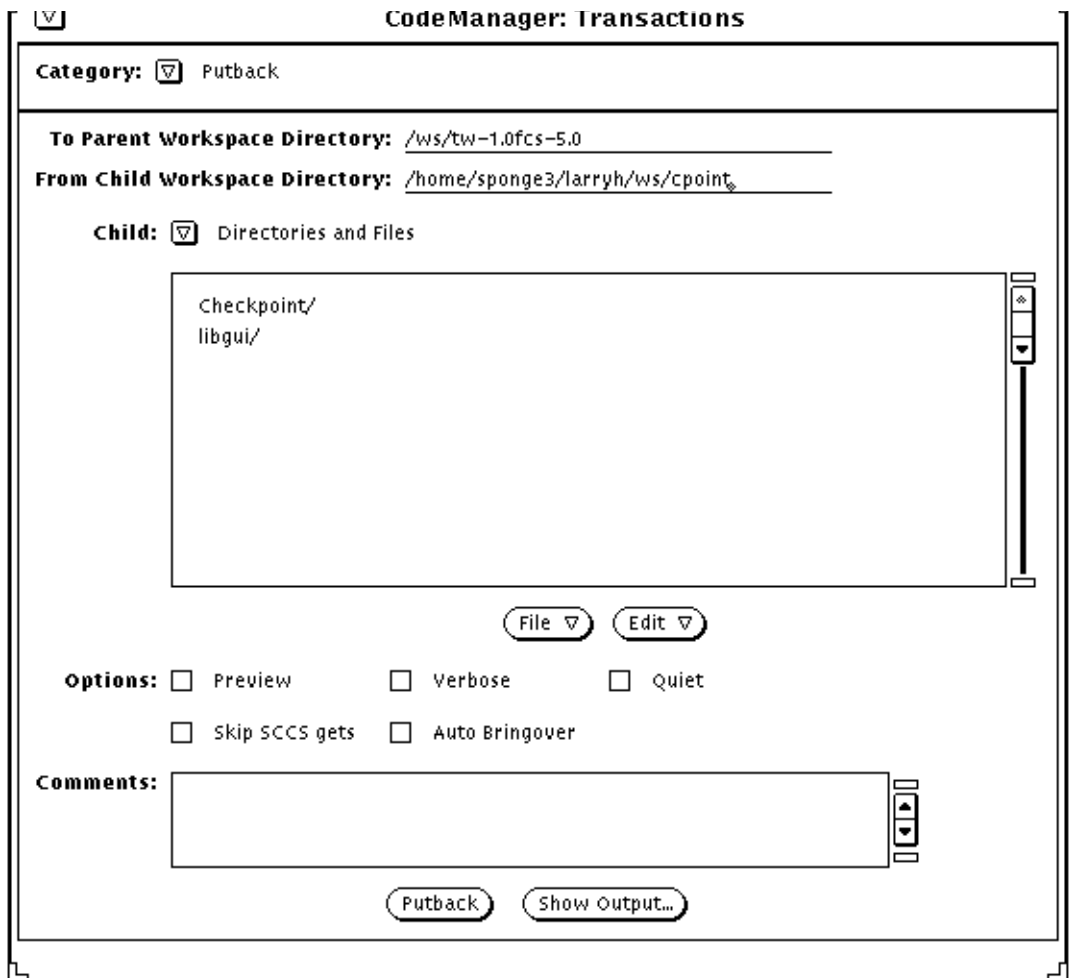


Figure 5-7 Putback Transactions Window Layout

To initiate a Putback transaction, follow these six basic 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 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 CodeManager 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 CodeManager window Edit menu or drag the child workspace icon over the new parent's icon. See Section , "Reparenting a Workspace," on page 71 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, CodeManager 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.¹

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 Section , "Specifying Directories and Files for Transactions," on page 98 for information about specifying directory and file arguments.

4. Select options.

Preview

Select this option to preview the results of the transaction. If you invoke the Putback transaction with this option, the transaction will proceed without actually transferring any files. You can

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

- monitor the output messages in the Transaction Output window (Show Output) as if the transaction were actually proceeding.
- 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.
 - Quiet** Select this option to suppress the output of status messages to the Transaction Output window (Show Output).
 - Skip SCCS gets** Select this option to inhibit the automatic invocation of the SCCS `get` program as part of the Bringover transaction. Normally `g`-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 `gets` at a later time.
 - Auto Bringover** Select this option to cause CodeManager 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 8Kbytes long.

6. Invoke the Putback button to initiate the transaction.

Notes about the Putback Transaction

- Checked-out files

When, during a Putback transaction, CodeManager 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 5-4 shows the different actions that CodeManager takes when it encounters checked-out files.

Table 5-4 Effects of Checked-out Files on Putback Transactions

File Checked-out in Parent	File Checked-out in Child	CodeManager 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

- 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 Putback transaction can be reversed using the Undo transaction. Refer to Section , “Reversing Bringover and Putback Transactions with Undo,” on page 124 for details.

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

Read-locks may be obtained concurrently by multiple CodeManager 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 CodeManager 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 Section , “Controlling Access to Workspaces,” on page 77 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 87 for further information regarding these files.
- CodeManager 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 CodeManager is installed.

Putback Action Summary

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

Table 5-5 Summary of CodeManager Action during a Putback Transaction

File in Parent	File in Child	Action by CodeManager
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 <code>g</code> -file that consists of the most recent delta.)
Changed	Unchanged	Block Putback, notify user.
Changed	Changed	Block Putback, notify user.
Checked out	Checked out ¹	Block Putback, notify user.
Unresolved conflict	Unresolved conflict ²	Block Putback, notify user.

1. If a file is checked out in *either* the parent or the child, the transaction is blocked. See Table 5-4 for more information about putting back files that are checked out.

2. 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.

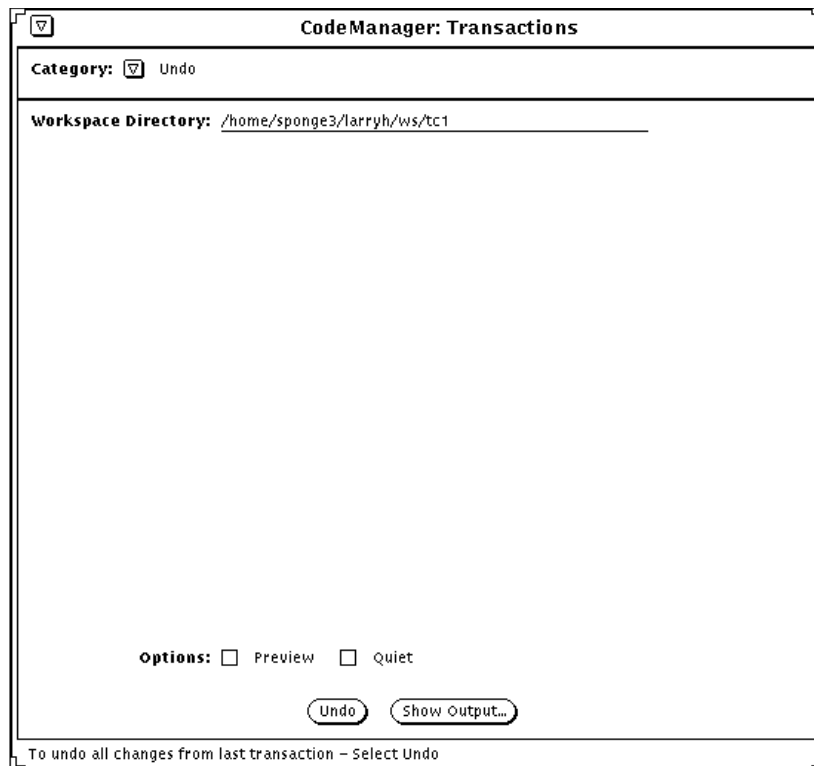


Figure 5-8 Undo Transactions Window Layout

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, CodeManager obtains a *write-lock* for the workspace. Only a single write-lock may be in force at any time; no CodeManager 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 CodeManager cannot obtain the lock, it will display an error message and abort.
- CodeManager 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 87 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`.

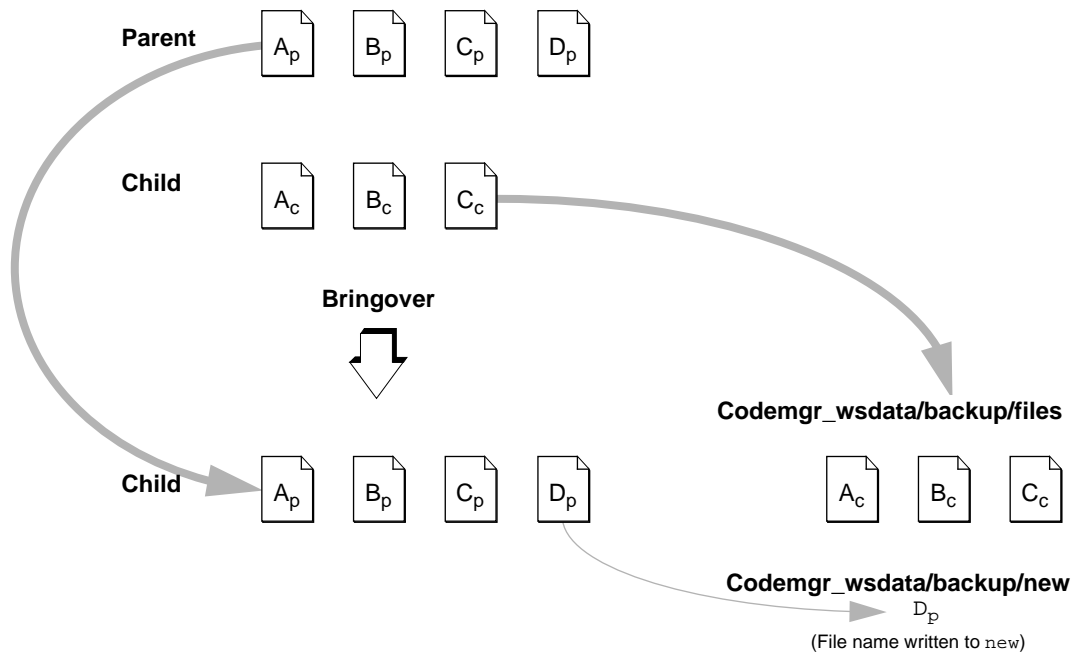


Figure 5-9 How Bringover Transactions Are Backed Up

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 CodeManager 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,¹ CodeManager 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.

1. CodeManager 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.

Renaming, Moving, or Deleting Files

When you rename, move, or “delete” files as described in this section, CodeManager tracks those changes so that it knows how to manage the altered files during Bringover and Putback transactions. Although CodeManager 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.”

This section describes the best ways to delete and rename files.

Renaming Files

When you bring over or put back files that you (or another user) have renamed, CodeManager 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 CodeManager 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.

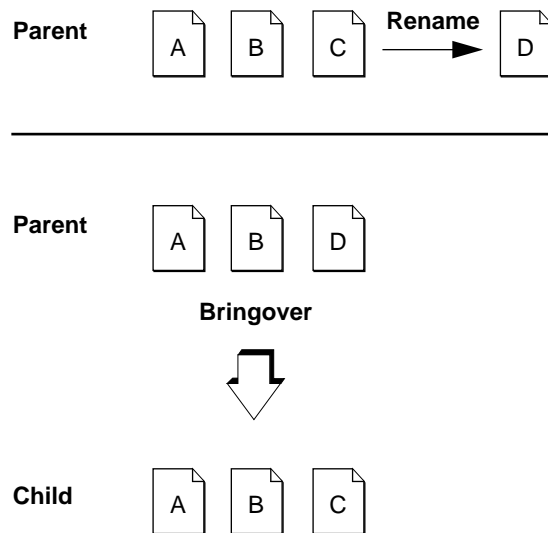


Figure 5-10 File “C” Renamed to “D”

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

CodeManager 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:
```

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

When CodeManager 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

As mentioned in the previous section, CodeManager 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 CodeManager to differentiate between files that have been renamed and those that are new. When you rename a file, CodeManager 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.

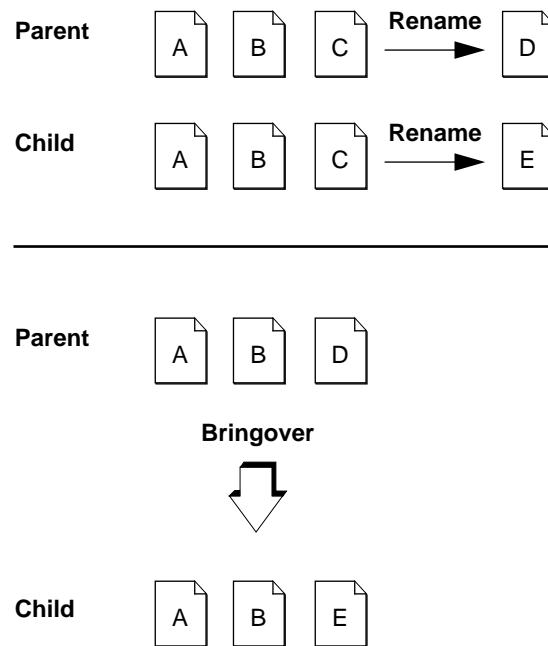


Figure 5-11 File “C” is Concurrently Renamed in both Parent and Child Workspaces

When this occurs, CodeManager 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:

- CodeManager reports the conflict using the name of the file in the child.
- CodeManager 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 CodeManager 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 CodeManager workspace is a little trickier than it first appears. Deleting a file from a workspace with the `rm` command causes CodeManager 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.

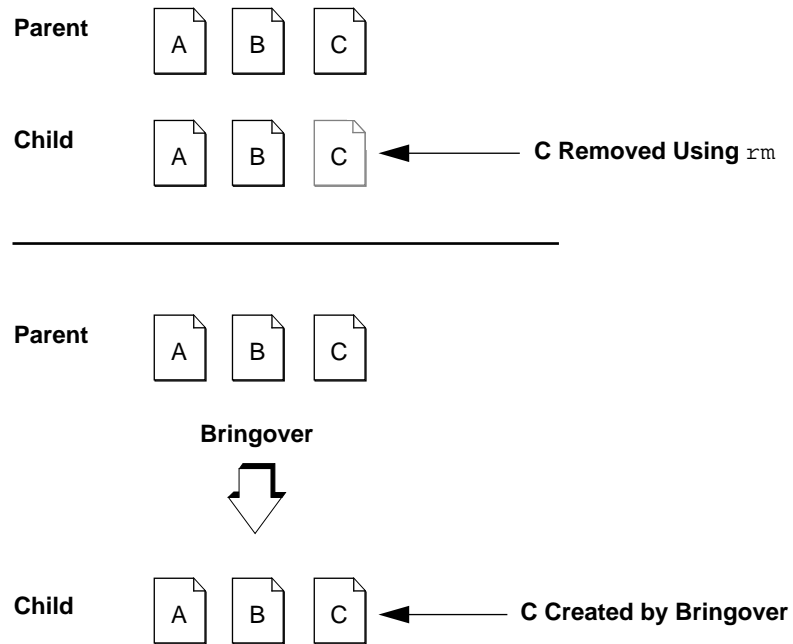


Figure 5-12 File "C" Is Removed From The Child Using the `rm` Command, Then Created Again by Bringover

CodeManager examines the two workspaces and determines that the file "C" exists in the parent and not in the child — following the usual CodeManager 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`.
- CodeManager does not recreate the file.
- CodeManager 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 *Part 3 Version Tool* and *Part 4 FreezePoint* 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, CodeManager 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, CodeManager propagates the change throughout the workspace hierarchy using the same rules used with file content updates and conflicts.

Resolving Conflicts



When files change concurrently in both a parent and child workspace, they are said to be 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. CodeManager assists you in resolving conflicts.

This chapter discusses the process by which CodeManager detects conflicts and then assists you in resolving these conflicts. Figure 6-1 outlines the CodeManager conflict resolution process.

Note – You can also resolve conflicts using the CLI. See the `resolve(1)` man page for more information.

You execute the Bringover Update transaction. CodeManager:

- Detects conflicts
- Merges deltas in the child workspace
- Enters file names in child `conflicts` file
- Notifies user of conflicts



You execute the Resolve transaction, selecting or specifying the name of the child workspace that contains the conflicts.



As part of the Resolve transaction, CodeManager:

1. Reads the `conflicts` file
2. Lists conflicted files in the File List Pane of the Resolve window
3. Extracts the parent, child and common ancestor deltas from the next file in its list



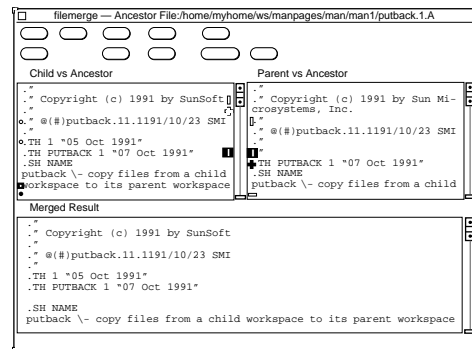
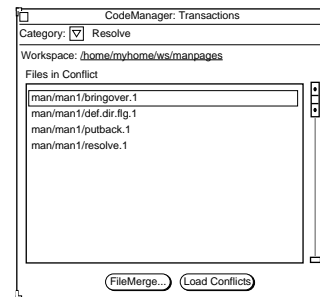
From the Resolve window, you invoke FileMerge to resolve differences between the parent and child deltas, creating a "Merged Result". CodeManager passes the names of the deltas to be compared to FileMerge.



After you save the Merged Result with FileMerge, CodeManager:

1. Creates a new delta in the SCCS history that contains the Merged Result you created with FileMerge
2. Extracts deltas from the next file in the list (if present) and passes them to FileMerge

Resolve Transactions Window



FileMerge Program

Figure 6-1 Conflict Resolution Process

Detecting Conflicts

Before you can resolve conflicts, CodeManager 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 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.¹ If, during the Bringover transaction, CodeManager 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.

The following is an example of output from a Bringover Update operation in which two conflicts were found.

1. If Putback is executed with the Auto Bringover option specified, then the Bringover transaction is initiated automatically by CodeManager.

```

CodeManager: Transaction Output

/usr/avocet/lang/bringover -w /home/sponge3/larryh/ws/tc2 -p
/home/sponge3/larryh/ws/tc1 man
Parent workspace: /home/sponge3/larryh/ws/tc1
Child workspace: /home/sponge3/larryh/ws/tc2

cd /home/sponge3/larryh/ws/tc2/man; /usr/avocet/lang/def.dir.flp &
cd /home/sponge3/larryh/ws/tc1/man; /usr/avocet/lang/def.dir.flp

Examined files: 19

Bringing over contents changes: 3

conflict: man/man1/resolve.1
conflict: man/man1/workspace.1
conflict: man/man1/undo.1

Examined files: 19

Contents Summary:
  3  conflict
 11  no action (unchanged)
  5  no action (changed in child only)

3 changes, 0 creates, 0 updates, 3 conflicts, 0 renames
  
```

Figure 6-2 Transaction Output with File Conflicts

Preparing Files for Conflict Resolution

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

CodeManager 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 – VersionTool 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 CodeManager Resolve transaction and FileMerge command to compare the parent and child deltas — both to their common ancestor, and to each other.

In addition to merging deltas, CodeManager 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 CodeManager Resolve transaction.

Resolving Conflicts

The two tools that you use to resolve conflicts are:

- CodeManager Resolve Transactions window
- FileMerge

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 — FileMerge.

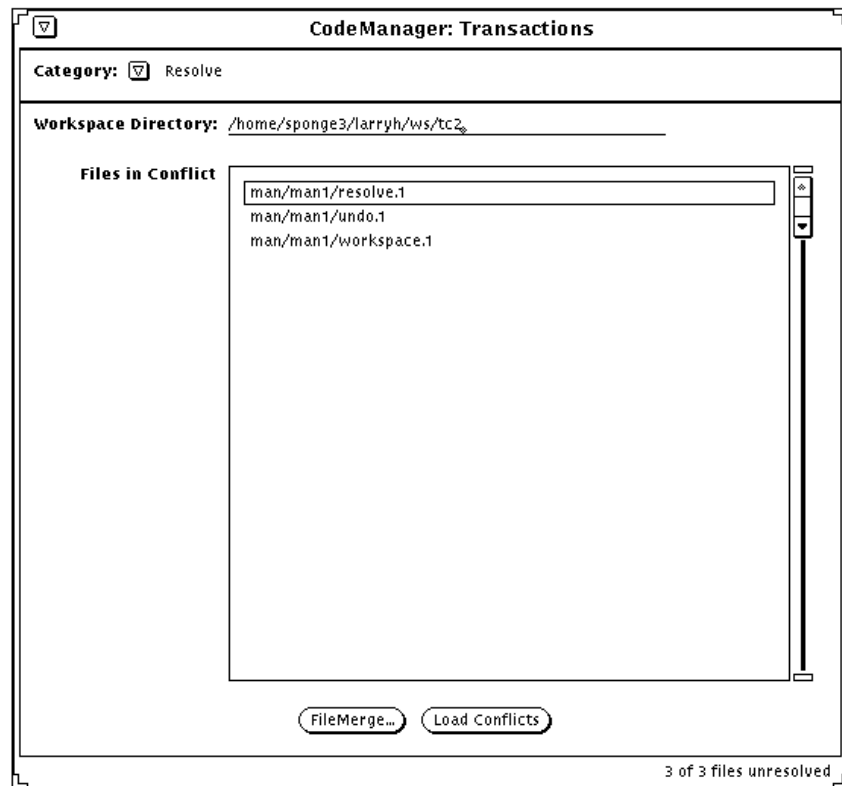


Figure 6-3 Resolve Transactions Window

As previously mentioned, when CodeManager 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_wdata/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 FileMerge selection button.

CodeManager starts the FileMerge program and begins to process the list of files from the File List Pane. For the next file in the list, CodeManager extracts the parent delta, the child delta, and the common ancestor from the SCCS history file and passes their path names to the FileMerge program.¹ The FileMerge window appears with the files loaded and ready for merging.

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

See Section , “The FileMerge Program,” on page 142 for more information.

4. Save the file in FileMerge.

After you use FileMerge to resolve differences between the parent and child versions of the file, CodeManager 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 FileMerge.

Notes about the Resolve Transaction

- By default, CodeManager automatically, sequentially processes the list of files from the File List Pane; after you resolve a conflict, CodeManager 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. In fact, 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

1. CodeManager and FileMerge 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 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 CodeManager creates the new delta in the child SCCS history file, it includes the following standard comment:

Merged changes between workspace x and y.

By default, CodeManager 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 FileMerge Program

Note – This section is intended to serve as a brief introduction to the FileMerge program as used with CodeManager. For a more detailed description, please refer to the FileMerge section in this manual.

FileMerge displays two text files (the parent and child deltas) for side-by-side comparison, each in a read-only subwindow. Beneath them, FileMerge 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.

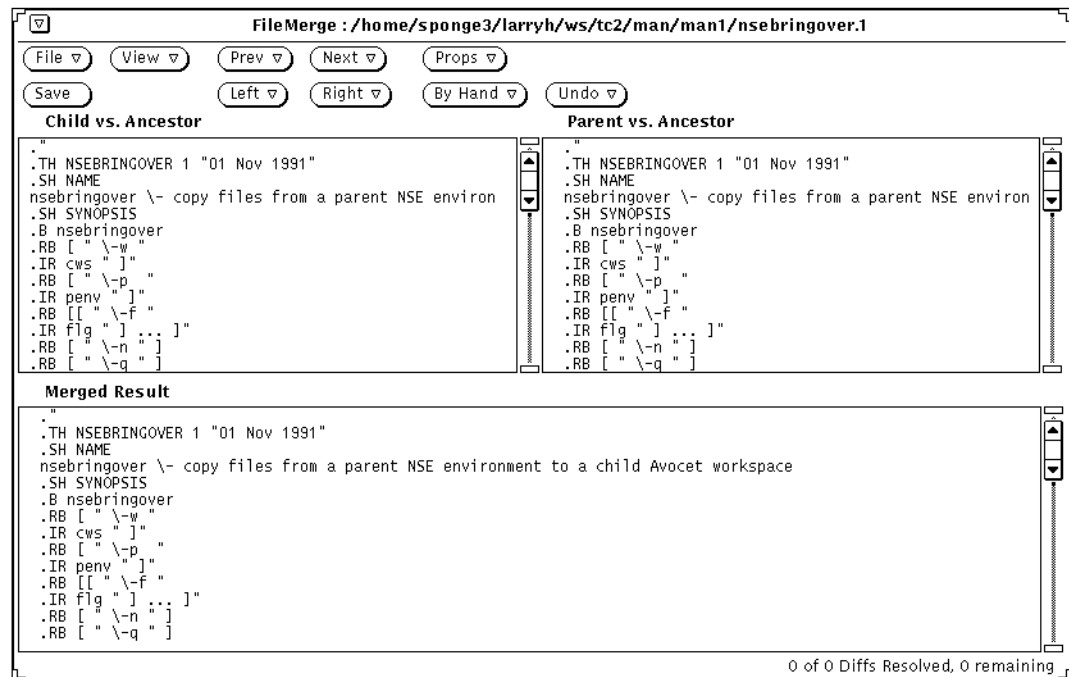


Figure 6-4 FileMerge Window with Loaded Files

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

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

As mentioned earlier, 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.

As mentioned previously, when FileMerge 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 FileMerge is focusing on a difference, it highlights the glyphs.

The difference on which FileMerge 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, FileMerge changes the glyphs that mark the difference to outline (hollow) font. FileMerge then automatically advances to the next difference (if the Auto Advance property is on), or moves to another difference of your choice.

As mentioned in the previous section, when used with CodeManager, FileMerge activity is coordinated by the Resolve transaction window. CodeManager and FileMerge programs communicate bidirectionally through the ToolTalk service. CodeManager extracts the parent, child, and common ancestor deltas and starts FileMerge, passing it the names of the files that contain the deltas to be merged. When you complete the merge process using the FileMerge Save button, CodeManager 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.

CodeManger Administration



CodeManager requires little administrative support. However, there are some things to consider when starting out. This chapter contains information about:

- Starting a project using CodeManager
- Configuring workspace hierarchies

Starting a Project with CodeManager

Getting started with CodeManager 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 CodeManager.

Moving an Existing Project

CodeManager works only with projects that use SCCS for version control. Moving an existing SCCS-based project to CodeManager 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 Section , “Configuring Your Workspace Hierarchy,” on page 146 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 CodeManager FLP. See Section , “Grouping Files for Transfer Using File List Programs,” on page 99 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).

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 Section , “Configuring Your Workspace Hierarchy,” for guidelines regarding workspace hierarchies.
- The default CodeManager 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 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 CodeManager’s workspace reparenting feature. See Section , “Reparenting a Workspace,” on page 71 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 7-1 depicts a “flat” (three-tiered) hierarchy and a “multitiered” (four-tiered) hierarchy.

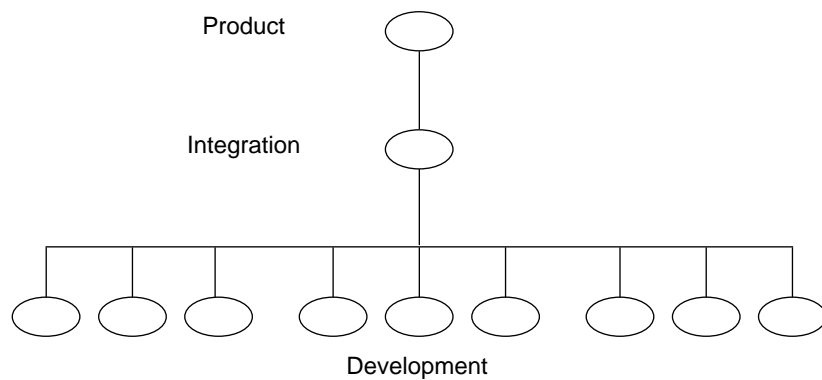


Figure 7-1 A “Flat” (Three-Tiered) Hierarchy

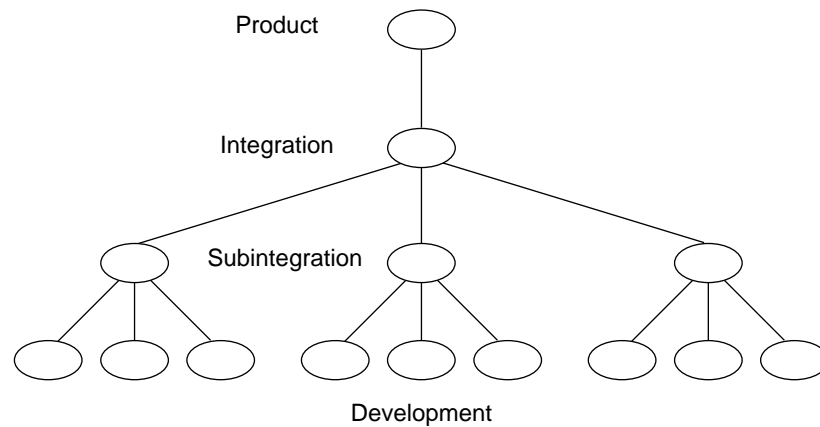


Figure 7-2 A “Multitiered” (Four-Tiered) Hierarchy

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

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, it is useful to 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.

Experience has shown that it is best to dedicate a workspace as a product release staging area for each release. It is generally 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 in this manner 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 7-3 shows a hierarchy that contains a product workspace and release workspaces for six different releases.

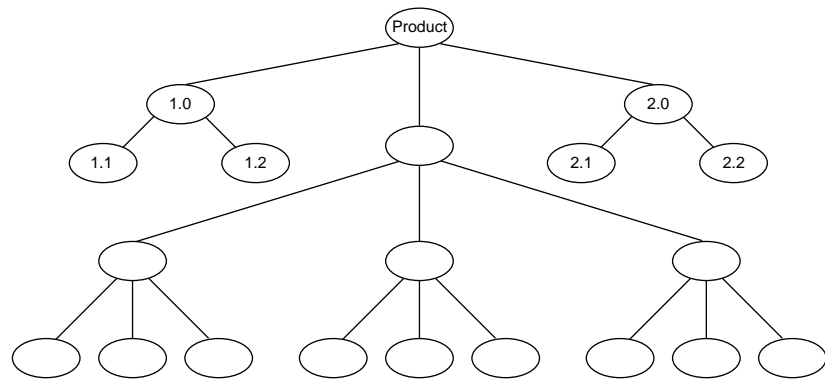


Figure 7-3 Product and Release Workspaces

Note – You can use the reparenting feature to transfer data between release workspaces directly. See “A Reparenting Example” on page 74 for details.

How CodeManager Merges SCCS Files



This chapter describes the ways CodeManager manipulates SCCS history files when you do the following:

- Copy files between workspaces
- Resolve conflicts

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

When considering Bringover and Putback transactions, it is important to recognize 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, CodeManager must actually 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.¹

Merging Files That Do Not Conflict

If the file in the destination workspace is simply being updated (the file has changed in the source of a Bringover or Putback transaction, and has not changed in the destination), the merging process is straightforward — the new deltas from the destination are added to the history file in the destination².

To accomplish the merger, CodeManager determines where the delta histories diverge and adds (to the destination workspace) only the deltas that have been created in the source workspace since they diverged. To determine where the histories diverge, the CodeManager 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 8-1 contains an example of a Putback transaction where CodeManager adds deltas 1.3 and 1.4 from the child workspace (denoted by □) to the SCCS history file in the parent (denoted by ○).

1. The exception is the case where the file does not exist in the destination workspace. In this case the entire history file is simply copied from the source workspace to the destination.

2. The reason that SCCS history files are merged at all in this case (rather than simply copying the source history file over the destination history file) is that administrative information (for example, flags and access lists) stored in the destination history file would be overwritten.

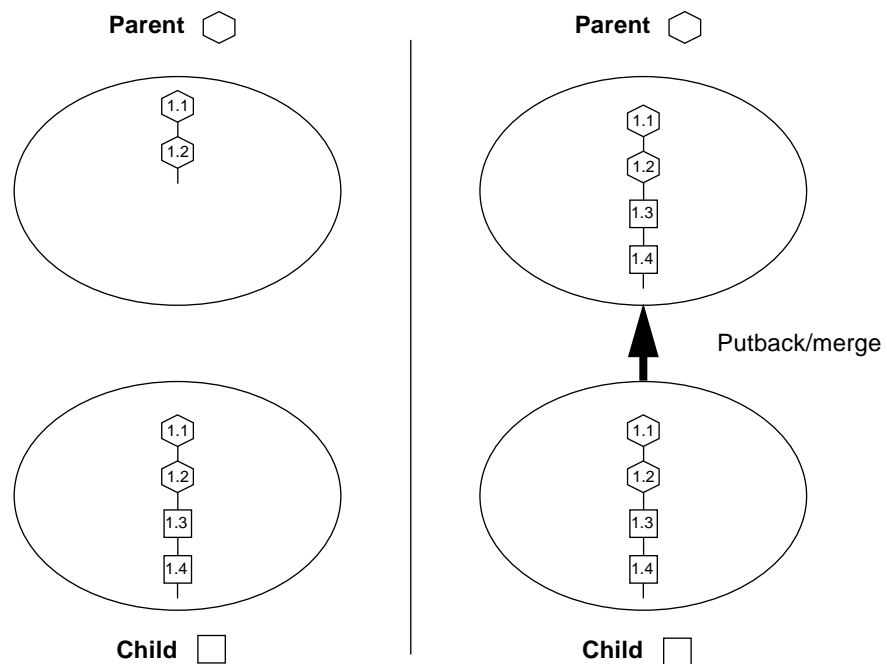


Figure 8-1 Updating a File in the Destination Workspace That Has Not Changed

Merging Files That Conflict




When you propagate files between parent and child workspaces, it is often the case that both the version of the file from the parent *and* the version in your child have changed since they were last updated. When that is the case, the parent and child versions of the file are said to be in *conflict*. In the case of conflicting files, the merge process is more complex.

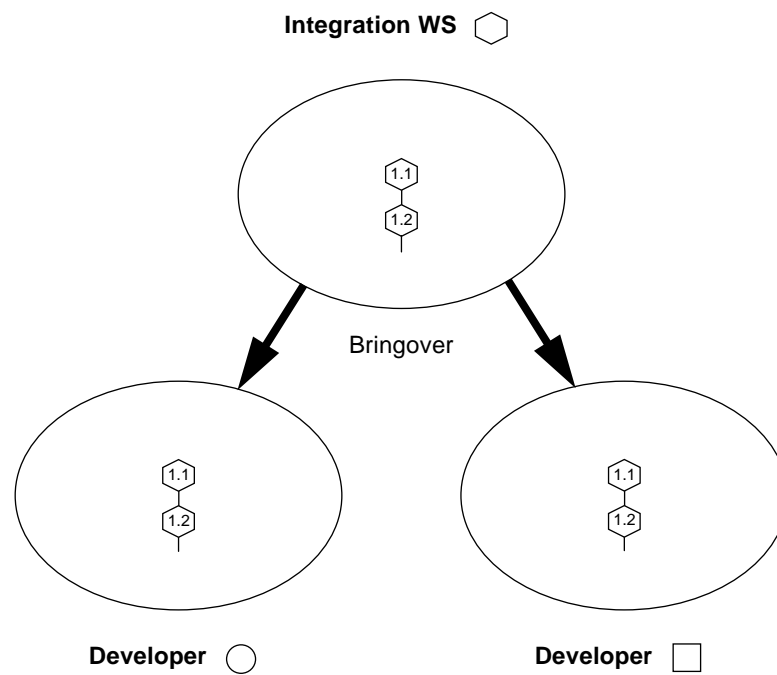
When the contents of files conflict, CodeManager's goal is to aid you in resolving the potentially conflicting changes that have been made to the file, as well as to preserve the file's delta, administrative, and comment history. To accomplish this, CodeManager merges the SCCS deltas from the parent into the history file in the child. CodeManager's Resolve transaction is then used to resolve the conflict in the child. See Chapter 6, "Resolving Conflicts," for details on resolving conflicts.




A Merge Example

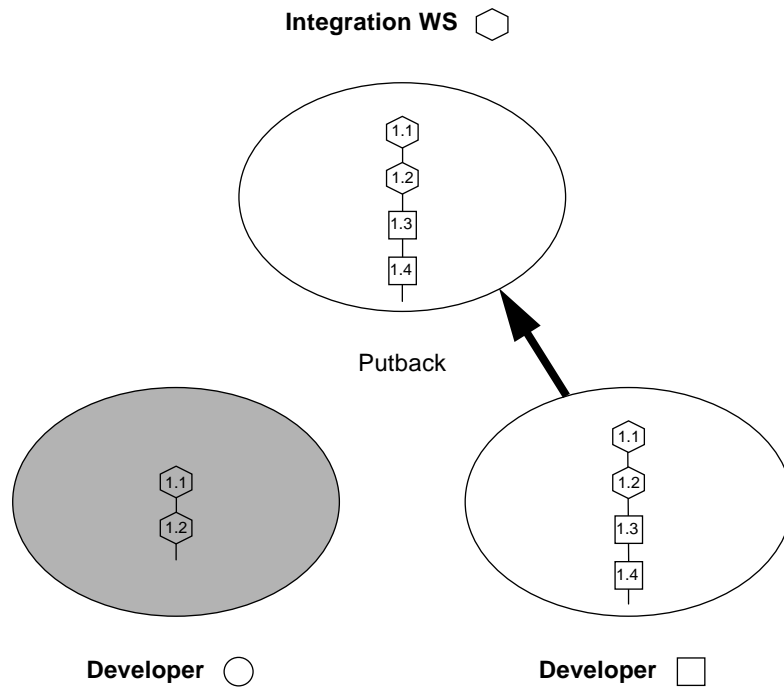
This section contains a series of figures designed to illustrate the CodeManager merging process. This scenario involves an integration workspace and two child workspaces owned by different developers. The developers bring over copies of the same file from the integration workspace, and independently change the file. The example illustrates how the SCCS history file is manipulated when conflicts occur and when they are resolved.

Some notes regarding the following examples:

- The geometric shapes  are used to identify the workspace from which the deltas originate.
- The default delta (the point at which the next delta will be added to the SCCS delta tree) is identified by an unattached descending line.
- VersionTool (part of the *TeamWare* product) can be used, to graphically display SCCS delta trees in much the same way they are depicted here.

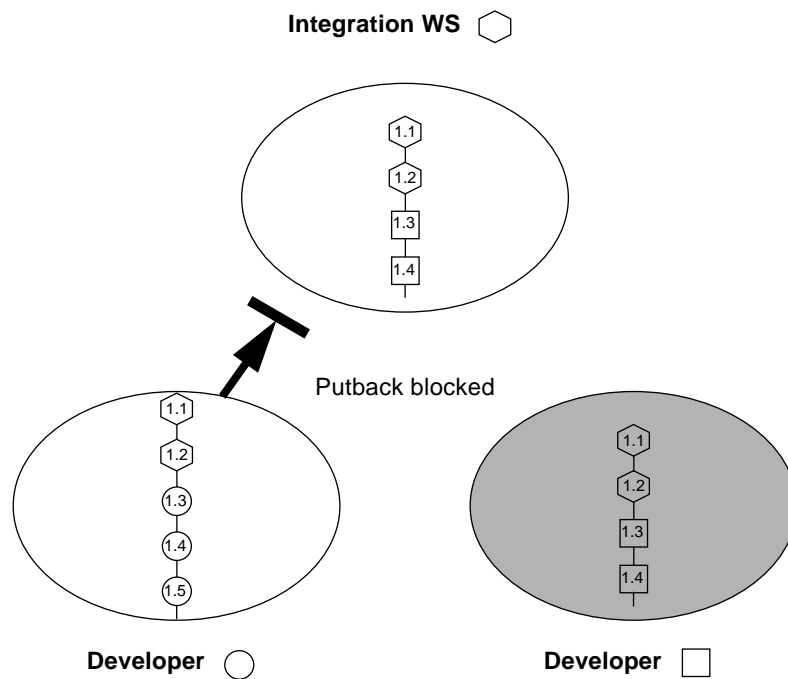


Both Developer  and Developer  copy the same file from the integration workspace  by means of the Bringover transaction. The file is new in both workspaces, so the SCCS history file is simply copied to both.



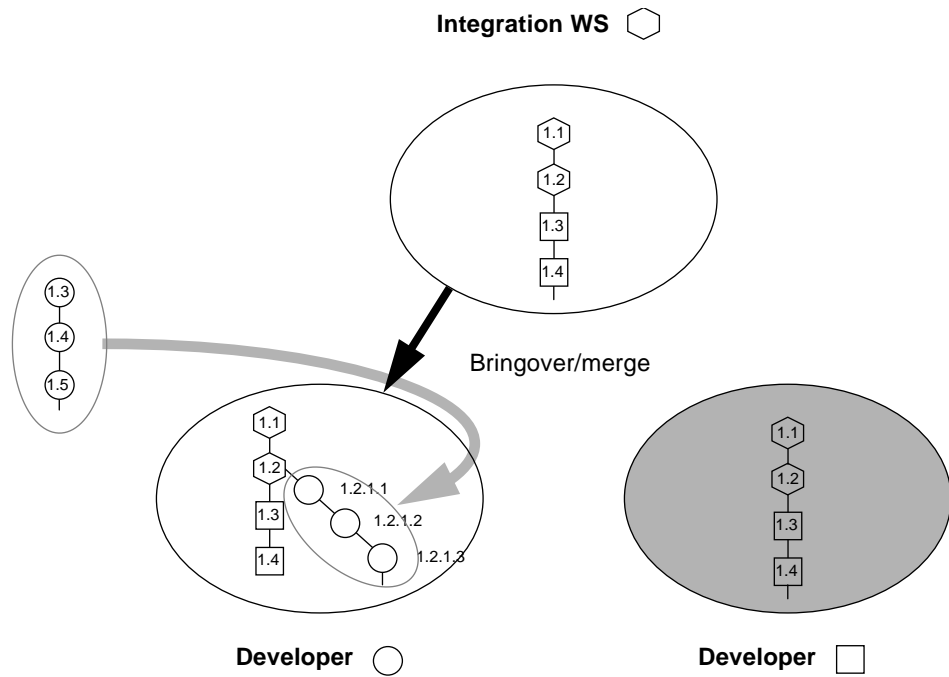
Developer makes changes to the file, creating two new deltas: 1.3 and 1.4, and then puts the file back into the integration workspace (by means of the Putback transaction). CodeManager appends the two new deltas to the parent SCCS delta tree.

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.



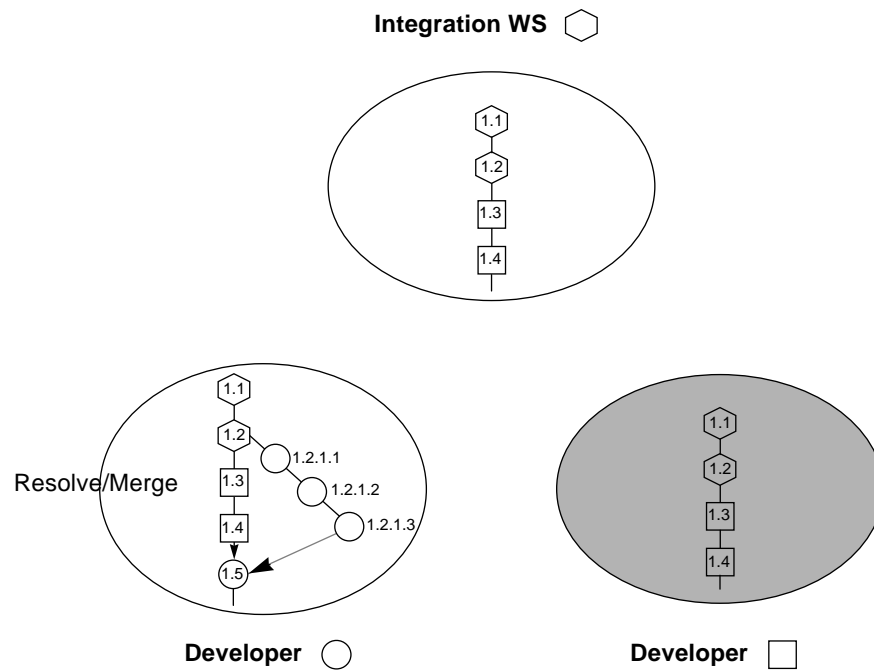
In the meantime, Developer ○ also changes the file (creating three new deltas: 1.3, 1.4, and 1.5) and now attempts to put back the file into the integration workspace.




CodeManager blocks the Putback of Developer ○ because the files are in conflict — the changes put back by Developer □ would be overwritten. Developer ○ must also incorporate the changes made by Developer □ into his work.






Developer ○ brings over the file that now contains the changes made by Developer □ into his workspace from the integration workspace. The deltas created by Developer □ are added into the child SCCS history file by CodeManager.

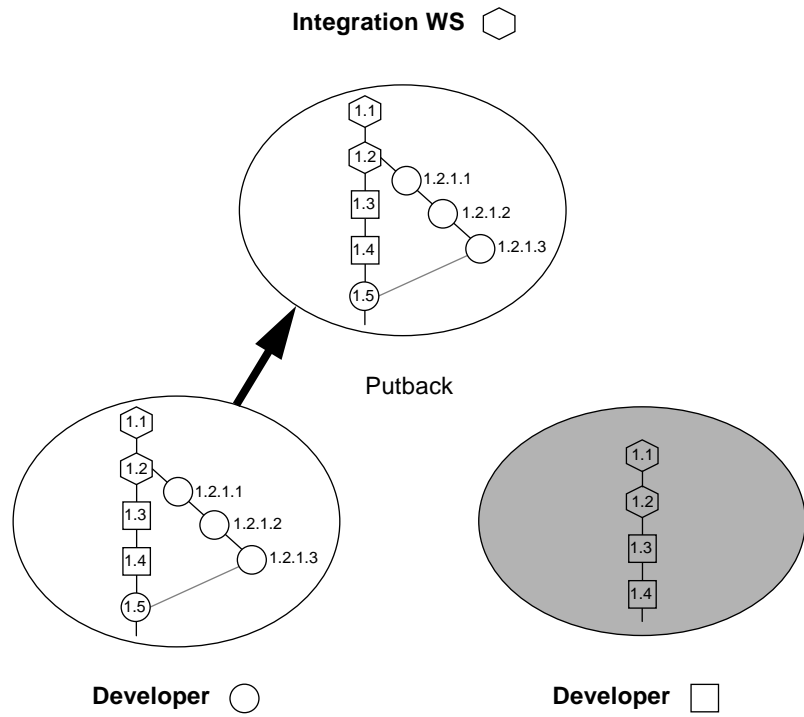
The delta tree brought down from the parent is unchanged in the child. The new deltas created in the child are attached as an SCCS branch to the last delta that the child and parent had in common; the deltas from the child are assigned new SIDs accordingly. The deltas are renumbered using the SCCS branch 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 ○ 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).



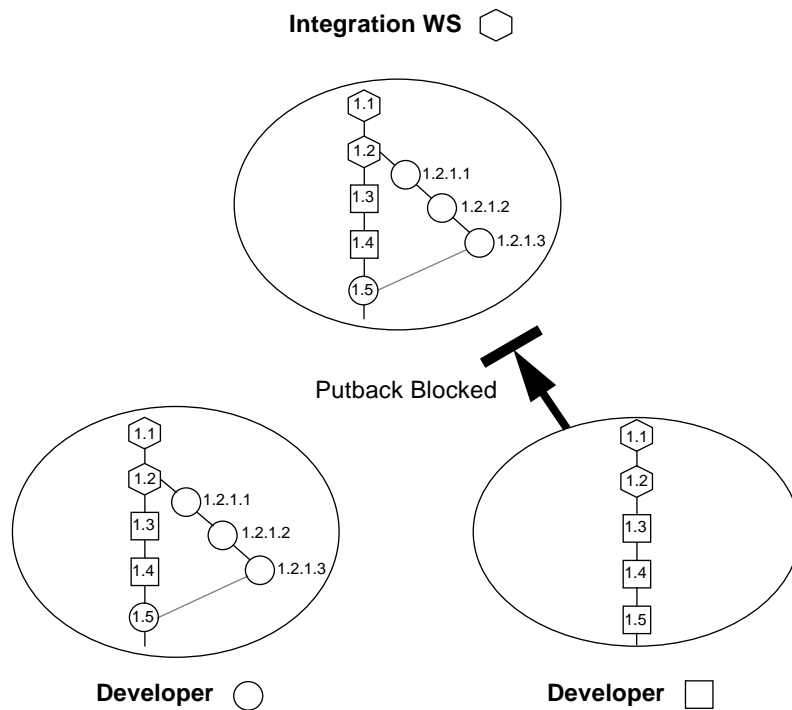
Developer  resolves the conflict in his workspace using the CodeManager Resolve transaction (see Chapter 6, “Resolving Conflicts,” for details regarding conflict resolution). Developer  uses the Resolve transaction to help him decide how to merge the versions of the file represented by SIDs 1.2.1.3 and 1.4. When he commits the changes, the Resolve transaction places the newly merged contents into a new delta— .

Notes:

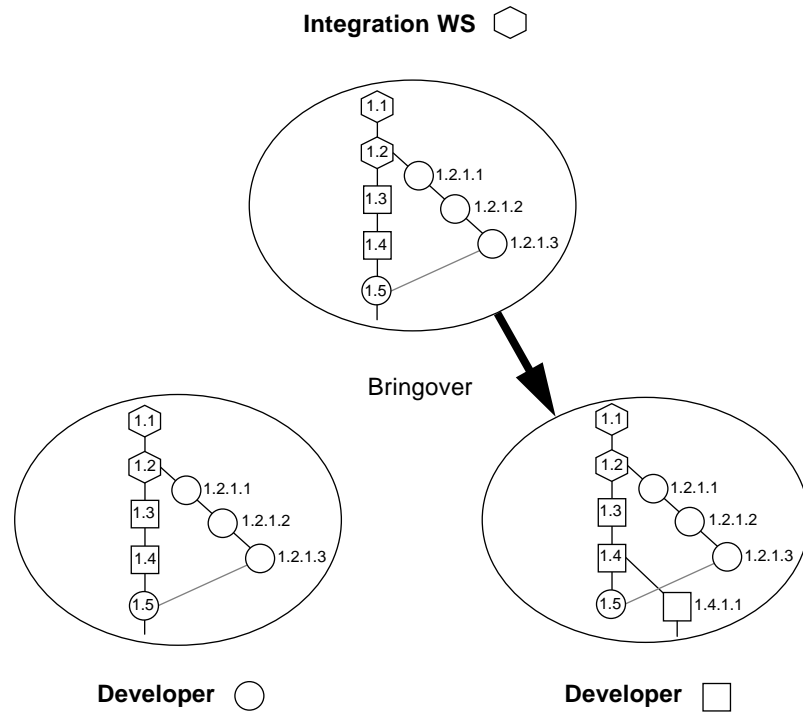
1. The new delta, , is contained in a circle because it is created by Developer .
2. The newly created delta is now the default location for any new work created by Developer .



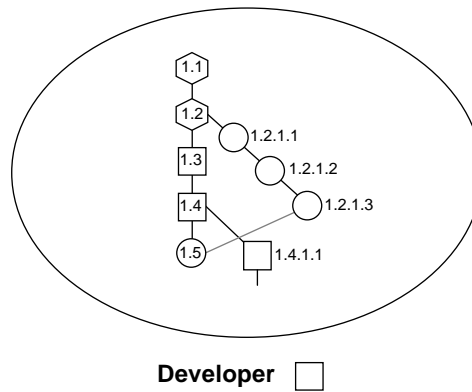
Now that the conflict has been resolved, Developer ○ successfully puts back the file into the integration workspace. The branch and the newly created delta are added to the SCCS history file in the integration workspace.



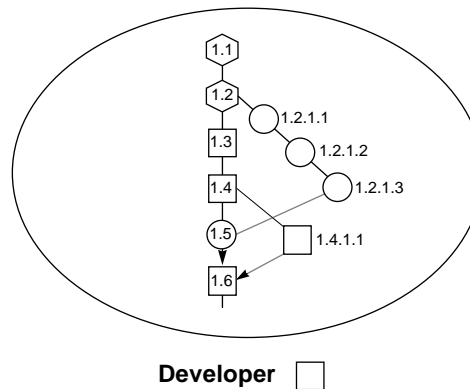
Developer makes another change to the file in her workspace, creating delta 1.5. She attempts to put back the new work to the integration workspace, but the Putback is blocked because it conflicts with the newly merged delta 1.5 that was put back by Developer .



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



As in the previous case, CodeManager appends the delta created by Developer □ to the last common delta on the delta tree trunk as a branch and rennumbers it appropriately—in this case 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.



Using the CodeManager Resolve transaction, Developer □ resolves the conflict merging the differences between 1.5 and 1.4.1.1 to create the new delta 1.6.

Notes:

1. The newly created merged contents are added as a new delta to the parent delta — 1.6.

2. The new delta is owned by the developer who owns the workspace.
3. The new delta becomes the default delta, therefore, new work in the child will now be added beneath it.

CodeManager Example



The example in this chapter illustrates the basic bringover, putback, resolve cycle. It 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 CodeManager 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 9-1 shows the file system hierarchy in the workspace `man_pages`.

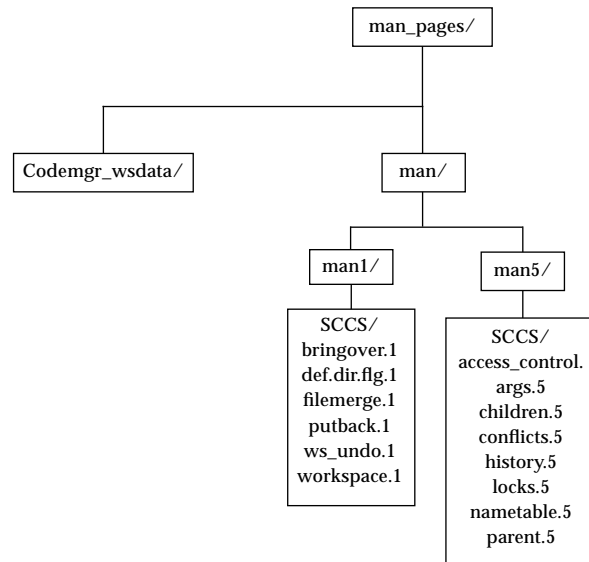


Figure 9-1 The man_pages Workspace

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 (Figure 9-2).¹

1. For accelerator options see Section , “Accelerators,” on page 63.

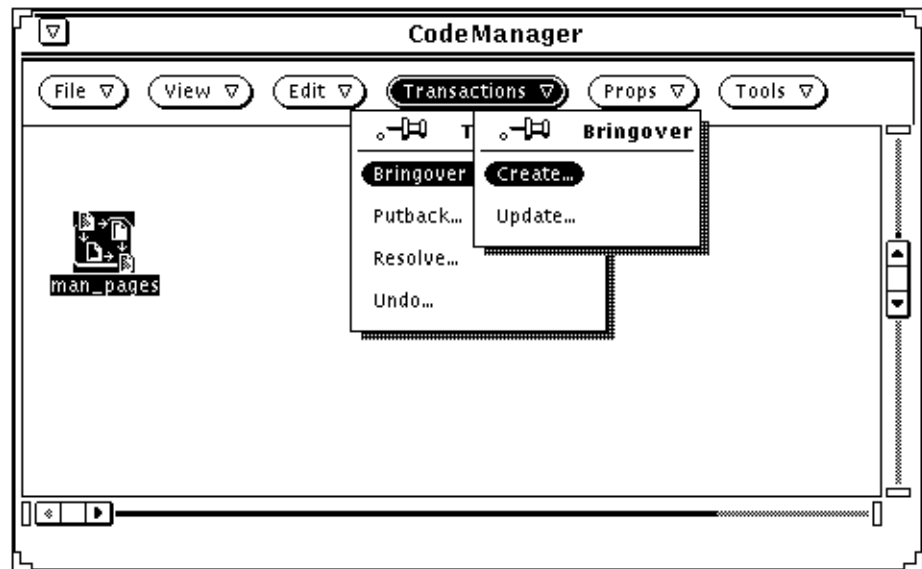


Figure 9-2 Activating the Bringover Create Transactions Window

2. The Bringover Create Transactions window is activated (Figure 9-4). 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 (Figure 9-3)

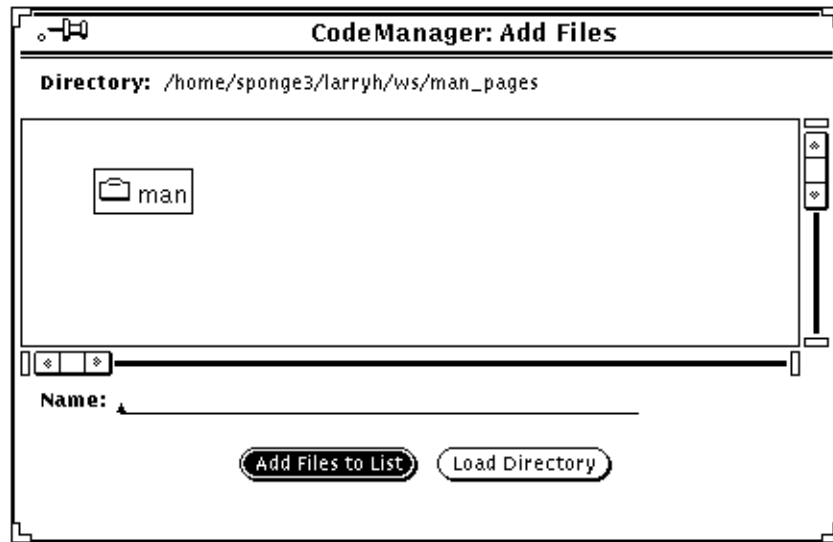


Figure 9-3 Add Files Point-And-Click Chooser Window

Figure 9-4 depicts 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`.)

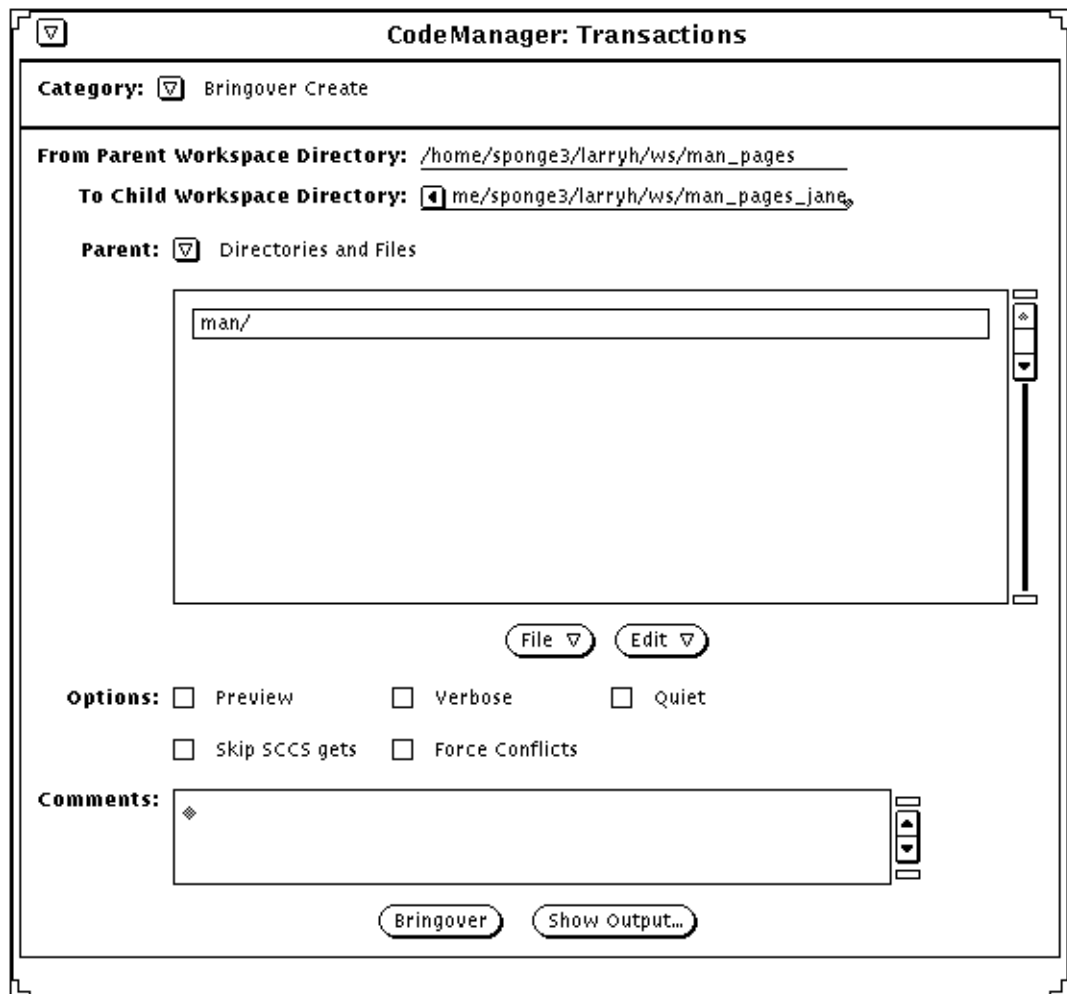


Figure 9-4 Transactions Window — Bringover Create of man_pages_jane

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.

Figure 9-5 shows the output produced during the transaction.

```

CodeManager: Transaction Output
/usr/avocet/lang/bringover -w /home/sponge3/larryh/ws/man_pages_jane -p
/home/sponge3/larryh/ws/man_pages man/
Parent workspace: /home/sponge3/larryh/ws/man_pages
Child workspace: /home/sponge3/larryh/ws/man_pages_jane
Updating names in child workspace's name table

cd /home/sponge3/larryh/ws/man_pages/man; /usr/avocet/lang/def.dir.flp

Examined files: 15

Bringing over contents changes: 15

create: man/Makefile
create: man/man5/access_control.5
create: man/man5/args.5
create: man/man5/children.5
create: man/man5/conflicts.5
create: man/man5/history.5
create: man/man5/locks.5
create: man/man5/parent.5
create: man/man5/nametable.5
create: man/man1/bringover.1
create: man/man1/putback.1
create: man/man1/def.dir.flp.1
create: man/man1/workspace.1
create: man/man1/filemerge.1
create: man/man1/ws_undo.1

Examined files: 15

Contents Summary:
  15  create

15 changes, 15 creates, 0 updates, 0 conflicts, 0 renames

```

Figure 9-5 Output from Bringover Create of man_pages_jane

Notes Regarding the transaction output in Figure 9-5:

- Updating names in child workspace's name table
 The name table is a file that assists CodeManager 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, CodeManager 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

CodeManager 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 (Figure 9-6).¹

1. For accelerator options see Section , “Accelerators,” on page 63.

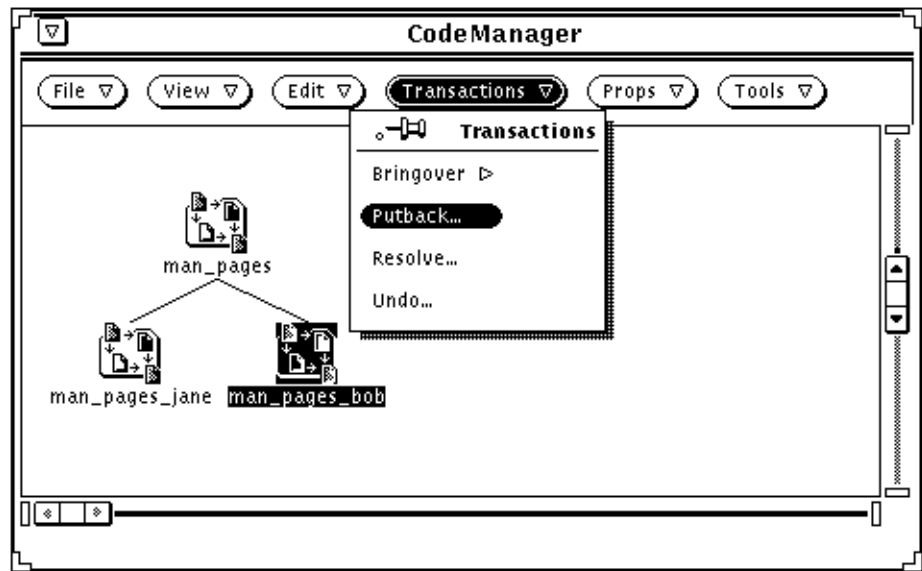


Figure 9-6 Activating the Putback Transaction Window

2. When the Putback window is activated (Figure 9-7), 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.

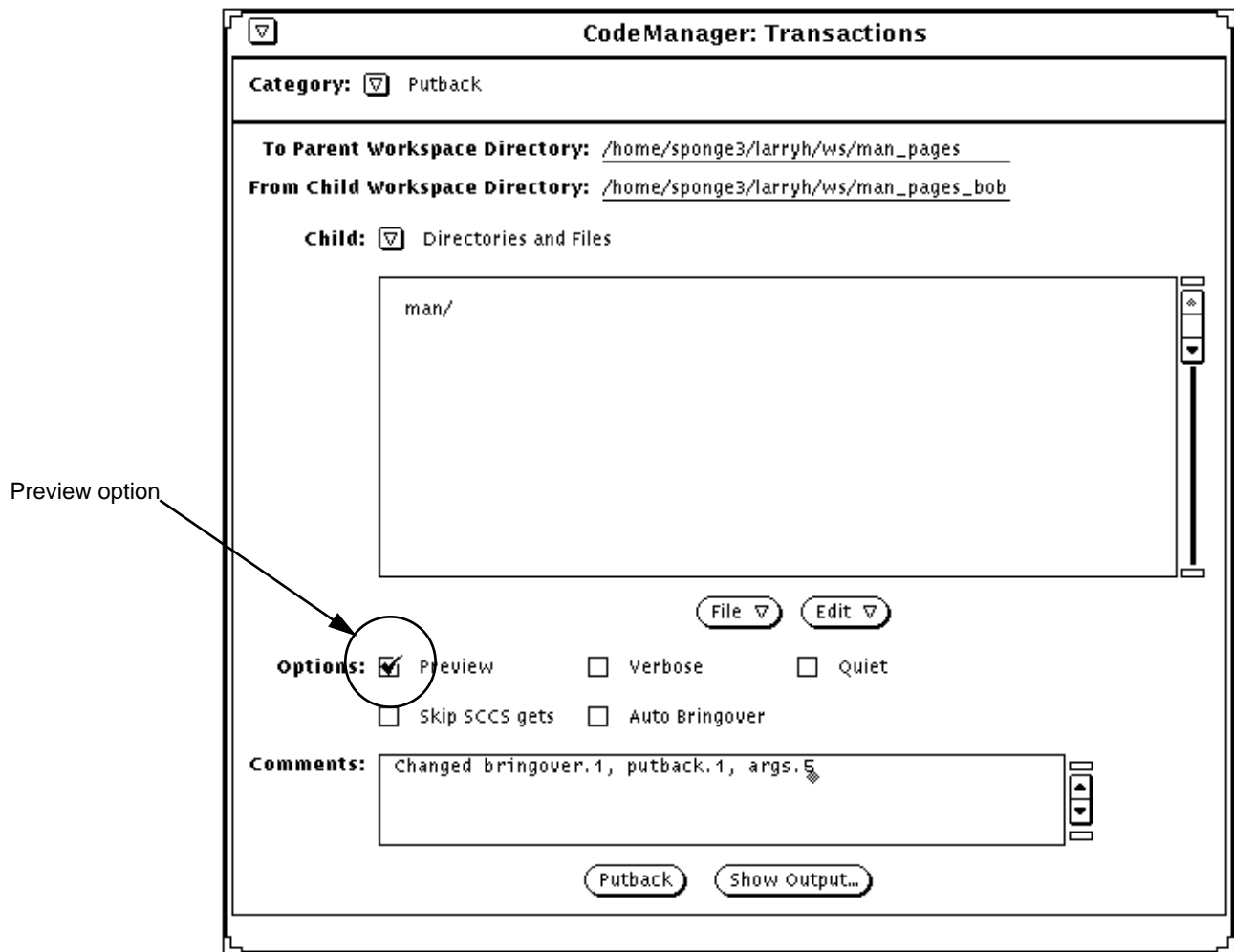


Figure 9-7 Transactions Window — Putback of man_pages_bob

Note that CodeManager 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's 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.

The output is shown in Figure 9-8.

```

CodeManager: Transaction Output

/usr/avocet/lang/putback -w /home/sponge3/larryh/ws/man_pages_bob -p
/home/sponge3/larryh/ws/man_pages -n man/
Parent workspace: /home/sponge3/larryh/ws/man_pages
Child workspace: /home/sponge3/larryh/ws/man_pages_bob

cd /home/sponge3/larryh/ws/man_pages/man; /usr/avocet/lang/def.dir.flp &
cd /home/sponge3/larryh/ws/man_pages_bob/man; /usr/avocet/lang/def.dir.flp

Examined files: 15

Would put back contents changes: 3

update: man/man5/args.5
update: man/man1/bringover.1
update: man/man1/putback.1

Examined files: 15

Contents Summary:
    3 update
   12 no action (unchanged)

No changes were put back

3 changes, 0 creates, 3 updates, 0 conflicts, 0 renames

```

Figure 9-8 Output from Putback of man_pages_bob

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 (Figure 9-9).¹

¹ For accelerator options see Section , “Accelerators,” on page 63.

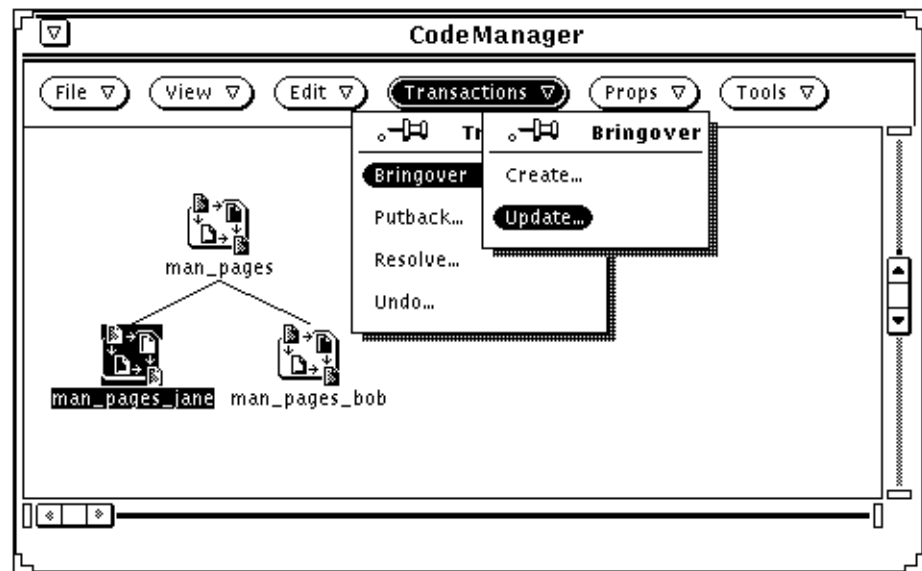


Figure 9-9 Activating the Bringover Update Transactions Window

2. When the Bringover Update window is activated (Figure 9-10), 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.

Preview option

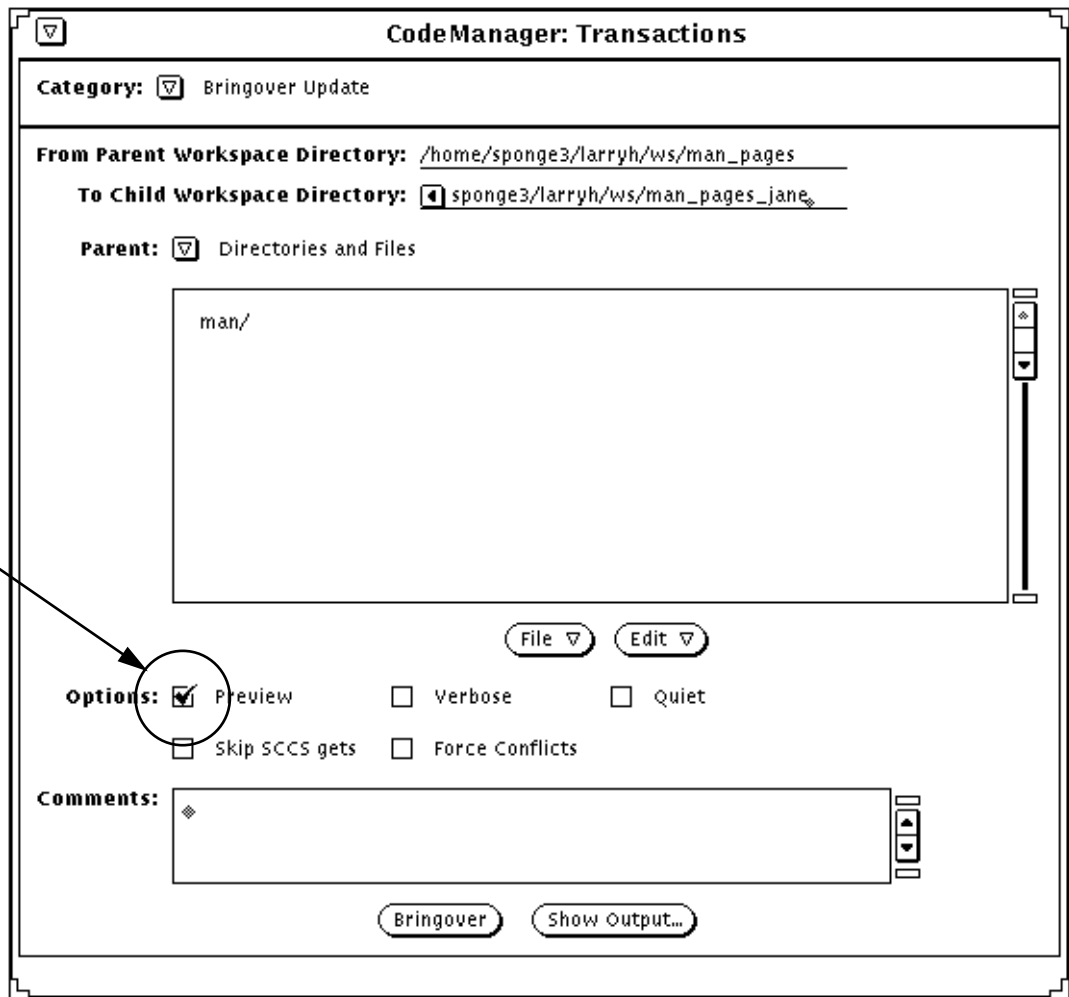
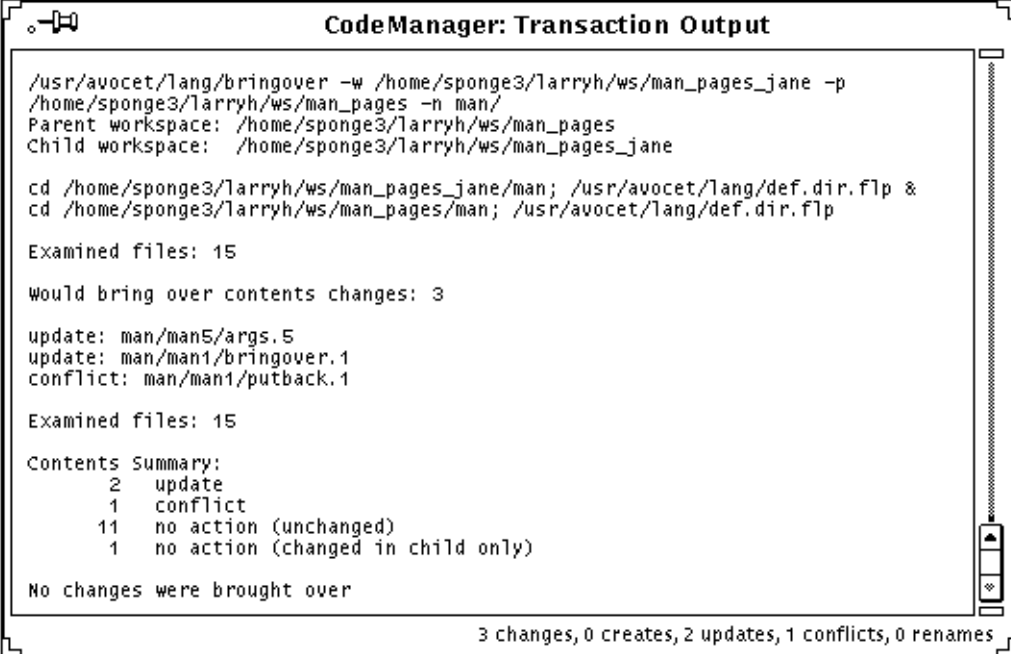


Figure 9-10 Transactions Window — Bringover Update of man_pages_jane

Figure 9-11 shows the output of the transaction.



```
CodeManager: Transaction Output

/usr/avocet/lang/bringover -w /home/sponge3/larryh/ws/man_pages_jane -p
/home/sponge3/larryh/ws/man_pages -n man/
Parent workspace: /home/sponge3/larryh/ws/man_pages
Child workspace: /home/sponge3/larryh/ws/man_pages_jane

cd /home/sponge3/larryh/ws/man_pages_jane/man; /usr/avocet/lang/def.dir.flp &
cd /home/sponge3/larryh/ws/man_pages/man; /usr/avocet/lang/def.dir.flp

Examined files: 15

Would bring over contents changes: 3

update: man/man5/args.5
update: man/man1/bringover.1
conflict: man/man1/putback.1

Examined files: 15

Contents Summary:
    2 update
    1 conflict
   11 no action (unchanged)
    1 no action (changed in child only)

No changes were brought over

3 changes, 0 creates, 2 updates, 1 conflicts, 0 renames
```

Figure 9-11 Output from Bringover Update of man_pages_jane

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. The output is shown in Figure 9-12.

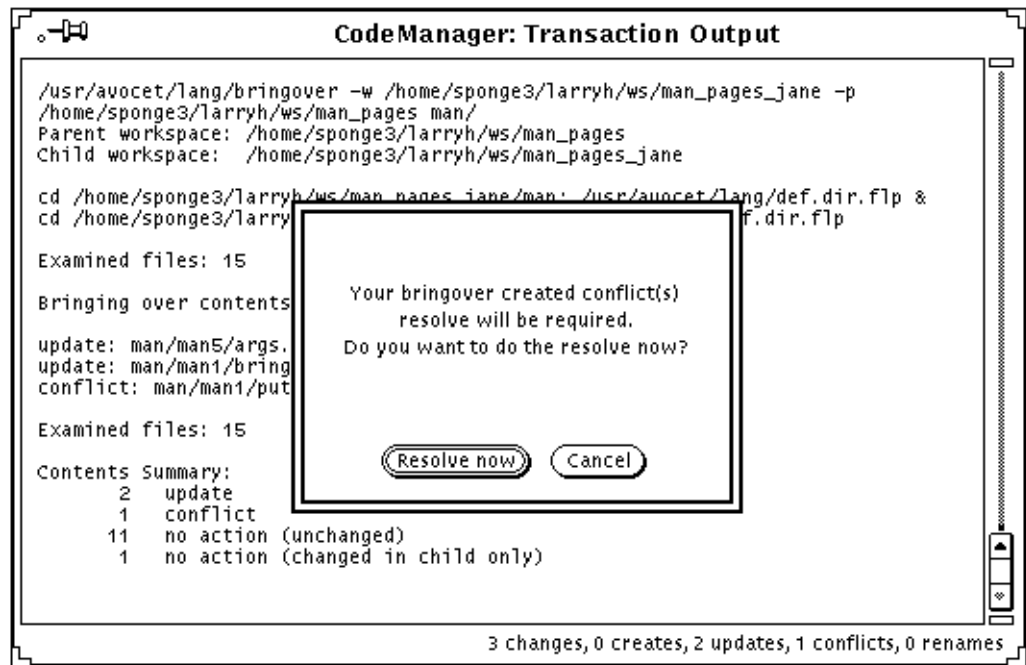


Figure 9-12 Resolve Confirmation Pop-up Window

After the transaction completes as expected, CodeManager 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.¹ The Resolve transaction window is activated.²

1. 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.

2. If the "auto load" property is set for the Resolve window, FileMerge begins execution automatically.

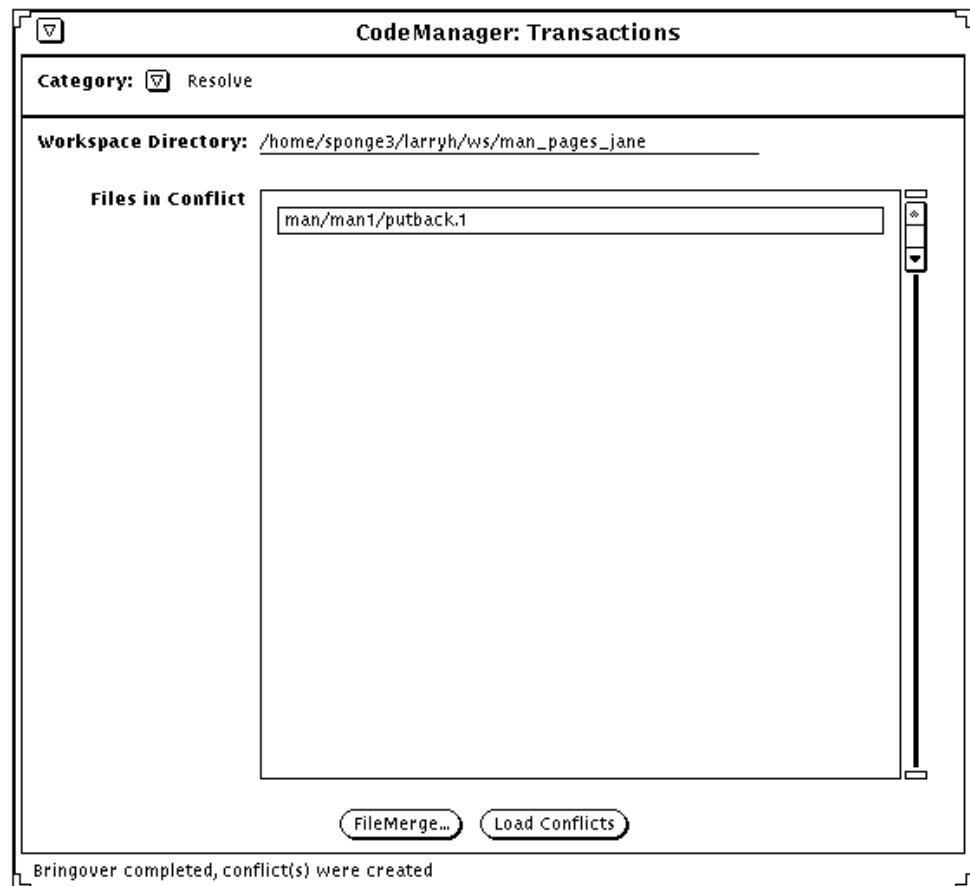


Figure 9-13 Transactions Window — Resolve Conflict on putback.1

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 FileMerge button.¹

1. 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), CodeManager automatically works its way down through the list of selected files.

CodeManager starts the FileMerge program and passes it the name of putback.1 (Figure 9-14).

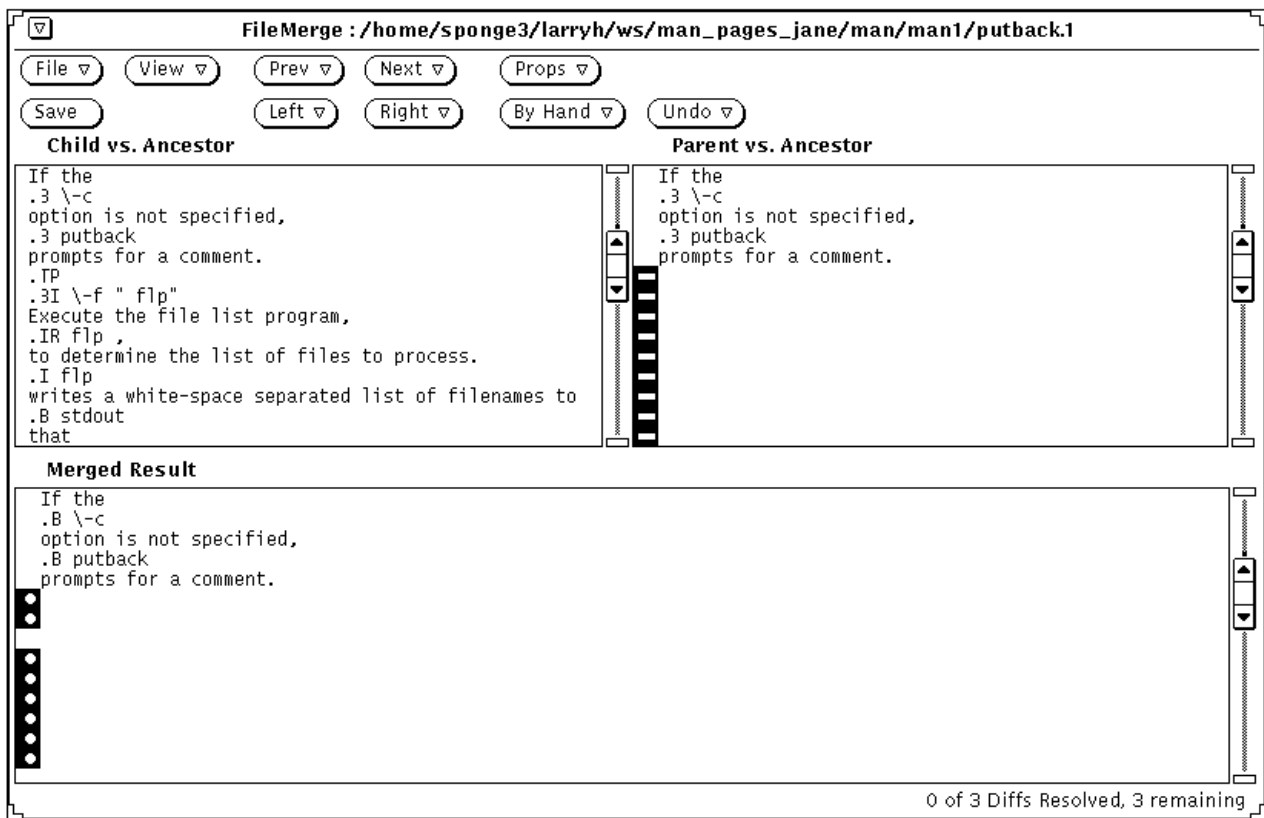


Figure 9-14 FileMerge Window—Merging putback.1

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 6, “Resolving Conflicts,” and *Merging Source Files* for more information about using FileMerge.

Jane can now put back her changes to the parent workspace (man_pages) following the same procedure that Bob used (Section , “Putting Back Changes,” on page 173).

CodeManager Messages

10 

This chapter describes CodeManager error messages and warning messages. Error messages are described in Section , “CodeManager Error Messages” and warning messages are described in Section , “CodeManager Warnings.” All CodeManager 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.

CodeManager Error Messages

1000 - 1999: System errors

Note – Error messages numbered between 1000 and 1999 report errors from operating system calls made by CodeManager commands. They consist of a short CodeManager 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 “*file_name*”

Meaning: While reading the *file_name*, a line was encountered that contained too many characters for a CodeManager 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]¹ workspace either with the `-w` option or via the `CODEMGR_WS` environment variable
- Meaning:** The CodeManager command could not determine the workspace on which to act. CodeManager commands attempt to acquire the workspace path name in the following order:
- As specified by the command's `-w` option
 - As specified by the value of the environment variable `CODEMGR_WS`
 - The current directory, if it is hierarchically within a workspace
- 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:** The `-p` option to the CLI `bringover` and `putback` commands cannot be used to reparent a workspace that has an NSE environment as a parent.
- Remedy:** Use the `workspace reparent` command to reparent a workspace whose parent is an NSE environment to a CodeManager workspace. *Note that you cannot reparent such a workspace to another NSE environment.*
- 2003:** "`directory_name`" is not a workspace
- Meaning:** The directory specified in the command is not a CodeManager workspace. CodeManager workspaces are distinguished by the presence of the `Codemgr_wsdata` directory in the top level directory.
- Remedy:** Specify a different workspace name or use the CLI `workspace create` command or GUI File ⇒ Create Workspace command to convert the directory into a workspace

1. When the error is reported by Bringover and Putback the word "child" is included, when reported by Undo and Resolve it is not included.

-
- 2004:** Workspace "*workspace_name*" doesn't have a parent workspace
- Meaning: A CodeManager command (Bringover or Putback) could not complete execution because a parent workspace could not be found for workspace *workspace_name*.
- Remedy: Use the CLI `workspace parent` command or the GUI Edit ⇒ Parent command to reparent the orphaned workspace.
- 2005:** Parent workspace "*workspace_name*" is not visible as it is not mounted on *machine_name*
- Meaning: The file system that contains the parent workspace is not currently mounted on machine *machine_name*.
- Remedy: Mount the file system that contains the parent workspace and re-issue the command.
- 2006:** Filename *file_name* has too many ".." path components in it
- Meaning: Relative file names specified to CodeManager 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.
- Remedy: Specify the path name with fewer (or no) ".." path name components
- 2007:** Could not get username for uid *uid_number*
- Meaning: The uid could not be found in the NIS maps or in `/etc/passwd`
- Remedy: Check NIS server and maps.

- 2008:** No version number in file "*file_name*"
- Meaning: When a CodeManager command accesses a metadata file (a file in the Codemgr_wsdata directory) it checks the version number written in the file when it was created (for example, VERSION 1). The metadata file *file_name* does not contain the version string.
- Remedy: Check the integrity of *file_name*. The version string may have been inadvertently removed when the file was edited. If it is missing and the file is not otherwise corrupted, use the `workspace create` 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 *file_name*.
- 2009:** Command "*command_name*" failed, /bin/sh killed by signal *signal*
- Meaning: A CodeManager command attempted to execute *command_name* and was unable to because the shell was killed by *signal*.
- Remedy: Re-execute the CodeManager command.
- 2010:** Command "*command_name*" failed, could not execute the shell, /bin/sh
- Meaning: A CodeManager command could not start a shell. This indicates that some system resource, such as swap space or memory was insufficient.
- Remedy: Check system resources.
- 2011:** Command "*command_name*" killed by signal *signal*
- Meaning: A command started by a CodeManager command received signal *signal*.
- Remedy: Re-execute the command. If the error re-occurs, refer to the Solaris documentation for information about the signal.

-
- 2012:** Command "*command_name*" exited with status *status*
- Meaning: CodeManager expects commands it executes to exit with a status of zero indicating successful completion. CodeManager considers it an error if a command exits with a non-zero status.
- Remedy: Refer to the documentation for *command_name* to determine the meaning of *status*.
- 2013:** FLP "*FLP_name*" does not exist in the parent or child workspace
- Meaning: The FLP (File List Program) *FLP_name* specified for the Bringover or Putback transaction, could not be found in either the parent or child workspace
- Remedy: Check the path name of the intended FLP and re-execute the transaction.
- 2014:** Could not execute "*program_name*"
- Meaning: A CodeManager command attempted to execute another program and was unable to do so.
- Remedy: Ensure that your installation is correct. Ensure that the program is in your search path and that its permissions are set correctly.
- 2015:** Workspace "*workspace_name*" already exists
- Meaning: An attempt was made to create a workspace that already exists.
- Remedy: Re-execute the command using a different workspace name.
- 2016:** Workspace "*name*" does not exist
- Meaning: The workspace *name* specified as an argument for a CodeManager command could not be found.
- Remedy: Check to ensure that the path name was specified correctly.

- 2017:** Can't open file "*file_name*" so can't get comments for check in
Meaning: CodeManager stored checkin comments in a temporary file and was unable to subsequently open that file to read the comments.
Remedy: Check file permissions and other such file system problems that would prohibit opening the file.
- 2018:** Can't re-parent a workspace to itself
Meaning: An attempt was made (either as part of a transaction, or by using an explicit re-parent command) to make a workspace its own new parent.
Remedy: Re-execute the command, specifying a different parent.
- 2019:** Internal error: unknown locktype *lock*
Meaning: Indicates that the workspace's lock file (Codemgr_wsdata/locks) is corrupted. A unknown lock value was found.
Remedy: Edit the lock file to repair the damage. For more information refer to the locks(5) man page or Section , "Ensuring Consistency through Workspace Locking."
- 2020:** You must specify a workspace name
Meaning: The CodeManager command could not determine the workspace on which to act. CodeManager commands attempt to acquire the workspace path name in the following order:
 - As specified by the command's *-w* option
 - As specified by the value of the environment variable CODEMGR_WS
 - The current directory, if it is hierarchically within a workspaceRemedy: Specify the workspace path name using one of the methods listed above

2021: Cannot obtain a *type* lock in workspace "*workspace_name*" because it has the following locks:
Command: *command* (*pid*), user: *user*, machine: *machine*, time: *time*

Meaning: In order to ensure consistency, CodeManager 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 CodeManager 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 CodeManager 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 CodeManager 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.

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.

2022: Invalid subcommand - *command_name*

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.

Remedy: For the list of valid subcommands for each command, type the command and specify the `help` subcommand.

2023: Not used

2024: File "*file_name*" has no deltas

Meaning: The SCCS history file *file_name* contains no deltas, therefore it cannot be processed.

Remedy: Perhaps the history file has been mistakenly overwritten.

- 2025:** Could not find the "*command_name*" command. Executable does not exist: "*name*"
Also could not find the "*name*" command in PATH "*PATH_contents*"
- Meaning: A CodeManager command attempted to execute another program and was not able to find it.
- Remedy: Ensure that your installation is correct. Include the directory that contains the missing program.
- 2026:** Unknown SCCS control character (*char*) in file "*file_name*" at line *line_number*
- Meaning: A CodeManager command expected *file_name* to be an SCCS history file; based on the character it encountered, it is either not a history file, or it has been corrupted.
- Remedy: Refer to Solaris SCCS documentation regarding SCCS history file format.
- 2027:** Corrupted file - "*file_name*", line "*line_number*"
- Meaning: A CodeManager command was unable to read a workspace metadata file (a file in the Codemgr_wsdata directory). Illegal characters were found in line *line_number*.
- Remedy: Check and repair the file. All CodeManager metadata files are ASCII text files and can be edited. See the *file_name*(5) man page or Chapter 4, "CodeManager Workspace" for more information on its format.
- 2028:** Could not find the "*command_name*" command in PATH "*path_name*"
- Meaning: A CodeManager command attempted to execute another program and was not able to find it.
- Remedy: Ensure that your installation is correct. Include the directory that contains the missing program.

- 2029:** The file has unresolved conflicts. Run `'edit m'` and search for `^<<<<<<<`
- Meaning: This error is issued by the `resolve` command. An attempt was made to save the file while it still contained unresolved conflicts.
- Remedy: Use the `edit m` subcommand (edit the “merged result”) to resolve the conflicts and then save the file. Conflicts are marked with the “`^<<<<<<<`” characters.
- 2030:** No file with number *file_number*
- Meaning: The `resolve` command creates a numbered list of files that contain conflicts. The *file_number* chosen does not exist in this list.
- Remedy: Use the `list` subcommand to list the files and determine the correct number of the file you wish to specify.
- 2031:** Can't find home directory so can't write to file "*file_name*"
- Meaning: A CodeManager command was unable to find the user's home directory and cannot locate file *file_name*. This usually indicates a problem with NIS maps.
- Remedy: Check NIS server and appropriate NIS maps.
- 2032:** Can't parse line in file "*file_name*" : *line*
- Meaning: Upon startup, the `resolve` command reads the `~/ .codemgr_resrc` file to obtain user defined properties. The line *line* could not be interpreted correctly by the program.
- Remedy: Correct the file `~/ .codemgr_resrc` file so that it includes only valid entries. For information regarding these entries, see the `resolve(1)` man page.
- 2033:** Must specify a directory list either as arguments or via the `CODEMGR_WSPATH` variable
- Meaning: This message is reported by the `workspace list` command when a directory (or list of directories) has not been specified in which it can search for workspaces to list. Directories can be specified as the standard argument to the command, or by defining the `CODEMGR_WSPATH` variable to contain the path name of a directory.
- Remedy: Re-execute the command specifying a directory, or set the `CODEMGR_WSPATH` directory to contain a directory path.

- 2034:** Internal error: Access control operation "*operation_name*" does not have a built-in default
- Meaning:** A CodeManager command attempted to verify access permission for a workspace operation (for example: bringover-from, putback-to, reparent-to). An internal consistency check failed.
- Remedy:** Please contact your local service representative.
- 2035:** Access control file does not exist
- Meaning:** A CodeManager command attempted to verify access permission for a workspace operation (for example: bringover-from, putback-to, reparent-to). The access control file (Codemgr_wsdata/access_control) in the affected workspace was not found.
- 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 workspace create or the GUI File ⇒ Create Workspace commands and copy the file from the newly created workspace. For more information refer to the access_control(5) man page or Section , "Controlling Access to Workspaces."
- 2036:** Cannot specify common ancestor file; there is no common ancestor delta
- Meaning:** The ancestor ("a") was specified as an argument to a resolve subcommand (diff, edit, more). 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 6, "Resolving Conflicts."
- Remedy:** Proceed with the conflict resolution process without specifying the ancestor ("a") as an argument to the diff, edit and more subcommands.

- 2037:** Invalid argument - "*character*"
- Meaning: An invalid argument was specified to one of the `resolve` subcommands. The command expected one of the following characters: a (ancestor), c (child), p (parent), m (merged result).
- Remedy: Specify one of the valid arguments: a, c, p, m. See the `resolve(1)` man page for more information.
- 2038:** Parent workspace is an NSE environment. Use the '`nseputback`' command
- Meaning: The "parent" in a putback transaction is an NSE environment and not a CodeManager workspace.
- Remedy: Use the "nseputback" command to putback changes from the workspace to the environment.
- 2039:** File "*file_name*" is probably not an s-file on line "*line_number*" expected ^A, but got '*char*'
- Meaning: A CodeManager command expected *file_name* to be an SCCS history file; based on the format it is either not a history file, or it has been corrupted.
- Remedy: Refer to Solaris SCCS documentation regarding SCCS history file format.
- 2040:** File "*file_name*" has not been merged.
Use the `merge` subcommand `first` or the `filemerge` subcommand
- Meaning: An attempt was made to commit (save) a file that had not yet been merged.
- Remedy: Merge the file using either the `merge` subcommand, or the `filemerge` subcommand (which executes the FileMerge GUI merge tool). For more information about the `resolve` command see the `resolve(1)` man pages. For information about the FileMerge program, see manual *Merging Source Files* (included with TeamWare).
- 2041:** "*path_name*" is not a workspace or a directory
- Meaning: The string *path_name* specified in the Bringover Create transaction is not a CodeManager workspace or a directory.
- Remedy: Specify a different workspace/directory name.

- 2042:** Can't create ToolTalk message, error = *TT_error_code*
- Meaning: The `resolve` command communicates with the FileMerge program via the ToolTalk service. The ToolTalk service is an interapplication communication service distributed with the Solaris OpenWindows windowing system.
- In this case the `resolve` command called a ToolTalk routine in order to create a ToolTalk message to FileMerge. The ToolTalk routine could not create the message and passed back *TT_error_code*.
- Remedy: Refer to the OpenWindows ToolTalk documentation for information about the error.
- 2043:** SCCS file "*file_name*" is corrupted
- Meaning: The SCCS `admin -h` command reports that the newly computed check-sum does not compare with the one stored in the first line of the file.
- Remedy: See the Solaris SCCS documentation for more information.
- 2044:** Unable to create a temporary name from template "*temp_file_name*"
- Meaning: A CodeManager command was unable to create a temporary file for its use. This is a CodeManager internal error.
- Remedy: Check for any system-level reasons why the command could not write this file (for example, file permission restrictions or incorrect command ownership).
- 2045:** Fprintf of "*file_name*" failed
- Meaning: A command was unable to write to the file *file_name*.
- Remedy: Check file permissions and other such file system problems that would prohibit writing in the file system.
- 2046:** Version mismatch in file "*file_name*", expected version *expected_number*, but found *actual_number*
- Meaning: Each CodeManager metadata file (`Codemgr_wsdata/*`) contains a string that includes a version number (for release 1.0 the version number string is `VERSION 1`). As a consistency check, when CodeManager 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

expected_number but found *actual_number* instead. This may indicate that old binaries are being used with new metadata files and could cause the file to be corrupted.

Remedy: Make sure that the most current versions of the CodeManager binaries are being accessed.

2047: Do not know how to convert file "*file_name*" from version *found_version_number* to *current_version_number*

Meaning: Each CodeManager metadata file (Codemgr_wsdata/*) contains a string that includes a version number (for release 1.0 the version number string is VERSION 1). As a consistency check, when CodeManager 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 CodeManager 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.

Remedy: Since this is the first release of CodeManager, 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".

2048: Must specify at least one file, directory or -f argument to a bringover that creates a child workspace

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, CodeManager attempts to take the arguments from the workspace's Codemgr_wsdata/args file.

Remedy: Reenter the command and ensure that you've included the correct number of arguments.

2049: Could not determine where "*file_name*" is mounted from

Meaning: CodeManager commands convert path names of NFS mounted directories to the *machine_name:path_name* format to do much of their work. This message indicates that the mount entry that contains *file_name* in /etc/mstab (Solaris 1.x) or /etc/mntab (Solaris 2.x) is no longer present.

Remedy: Remount the file system that contains *file_name*.

- 2050:** Could not determine the absolute pathname for "*file_name*"
- Meaning:** A CodeManager command was unable to read a directory. This indicates some corruption in the file system; for example, incorrect directory permissions.
- Remedy:** Check the file system, especially directory and file permissions in the path of *file_name*.
-
- 2051:** Can't rename to "*file_name*"; it exists
- 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.
- Remedy:** Change the name in one of the directories.
-
- 2052:** Corrupted file - "*file_name*", text after "BEGIN", line *number*.
Can't send notification
- Meaning:** A CodeManager 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 CodeManager command cannot correctly parse the request; if the file contains a notification request, it cannot be sent.
- Remedy:** Edit the notification file and enter the appropriate BEGIN statement. See the notification(5) man page or Section , "How to Notify Users of Changes to Workspaces" for more information on its format.
-
- 2053:** Corrupted file - "*file_name*", text after "END", line *number*. Can't send notification
- Meaning:** A CodeManager 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 CodeManager command cannot correctly parse the request; if the file contains a notification request, it cannot be sent.
- Remedy:** Edit the notification file and enter the appropriate END statement. See the notification(5) man page or Section , "How to Notify Users of Changes to Workspaces" for more information on its format.

- 2054:** Corrupted file - "*file_name*", missing BEGIN, line *number*. Can't send notification
- Meaning:** A CodeManager 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, is missing. The CodeManager command cannot correctly parse the request; if the file contains a notification request, it cannot be sent.
- Remedy:** Edit the notification file and enter the appropriate BEGIN statement. See the `notification(5)` man page or Section , "How to Notify Users of Changes to Workspaces" for more information on its format.
- 2055:** File "*file_name*" has incomplete delta table
- Meaning:** The delta table in the SCCS history file *file_name* is incomplete. This indicates that the file has been corrupted.
- Remedy:** Fix the file, or copy in a new version.
- 2056:** Badly formatted line in "*file_name*" :
line_number
- Meaning:** A CodeManager 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 has been corrupted.
- Remedy:** Execute the workspace `updatenames` command to rebuild the nametable and then re-execute the command.
- 2057:** Zero-length SCCS file, "*file_name*"
- Meaning:** An SCCS history file was encountered that contained no data.
- Remedy:** Remove the SCCS history file.
- 2058:** Can't get a version of the child file until it is checked in
- 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.
- Remedy:** Check the file in and re-start the transaction.

- 2059:** Name history serial number *number* out of order in file *file_name*
- Meaning: Rename information in the SCCS history file *file_name* is corrupted. The name history records in this SCCS file are not in numerically descending order.
- Remedy: Reorder the name history records, or copy in a new version of the file using the Bringover or Putback transaction.
-
- 2060:** Delta serial number *number* out of order in file "*file_name*"
- Meaning: Delta numbers are not in numerically descending order in the SCCS history file *file_name*. This indicates that the file is corrupted.
- Remedy: Reorder the delta numbers, or copy in a new version of the file using the Bringover or Putback transaction.
-
- 2061:** Must have DISPLAY environment variable set to invoke filemerge
- Meaning: The DISPLAY variable is automatically set by OpenWindows when it begins execution. Your machine must be running OpenWindows to use the FileMerge program.
- Remedy: Ensure that OpenWindows is executing properly; if it is, reset the DISPLAY variable.
-
- 2062:** Can't resolve file "*file_name*" because it is writable
- Meaning: The file *file_name* 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.
- Remedy: Reconcile the file permissions (for example, check the file out and then check it back in) and then re-execute the Resolve transaction.
-
- 2063:** Cannot create workspace "*name*" because it would be nested within workspace "*name*"
- Meaning: An attempt was made to create a workspace hierarchically beneath an existing workspace.
- Remedy: Create the new workspace hierarchically outside of any existing workspaces.

- 2064:** Cannot delete a workspace that is a symbolic link.
Run `"workspace delete workspace_name"`
- Meaning:** CodeManager 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.
- Remedy:** Use the workspace delete command to delete *workspace_name*.
- 2065:** This error message may be issued in any of the following forms:
- User *user_name* does not have access to bringover from workspace "*workspace_name*"
 - User *user_name* does not have access to bringover to workspace "*workspace_name*"
 - User *user_name* does not have access to putback from workspace "*workspace_name*"
 - User *user_name* does not have access to putback to workspace "*workspace_name*"
 - User *user_name* does not have access to undo workspace "*workspace_name*"
 - User *user_name* does not have access to delete workspace "*workspace_name*"
 - User *user_name* does not have access to move workspace "*workspace_name*"
 - User *user_name* does not have access to change the parent of workspace "*workspace_name*"
 - User *user_name* does not have access to change the parent to workspace "*workspace_name*"
- Meaning:** The user *user_name* attempted an operation that affected the workspace *workspace_name*; access permissions in *workspace_name* do not permit *user_name* access to execute that operation.
- Remedy:** The file *workspace_name*/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 *user_name* 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 , “CodeManager Workspace” of this manual for more information.

- 2066:** Corrupted file - "*file_name*", whitespace in pathname, line *line_number*. Can't send notification
- Meaning: A CodeManager 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.
- Remedy: Edit the `Codemgr_wsdata/notification` file to remove the whitespace characters from the line. See the `notification(5)` man page or Section , “How to Notify Users of Changes to Workspaces” for more information on its format.
- 2067:** Corrupted file - "*file_name*", missing notification event, line *line_number*. Can't send notification
- Meaning: A CodeManager command encountered an error when reading the workspace notification file `Codemgr_wsdata/notification`. The CodeManager event (for example, `bringover-to`) was not specified.
- Remedy: Edit the `Codemgr_wsdata/notification` file to add the correct event. See Section , “How to Notify Users of Changes to Workspaces,” on page 83 for a list of valid events.
- 2068:** Not used
- 2069:** Not used
- 2070:** Not used
- 2071:** Not used
- 2072:** Not used
- 2073:** Not used

-
- 2074:** Workspace "*workspace_name*" has no locks
Meaning: An attempt was made to remove locks from a workspace that had no active locks.
Remedy: N/A
- 2075:** Lock *lock_name* does not exist for workspace "*workspace_name*"
Meaning: While using the workspace locks `-r` command, a lock number was specified that is out of range of the lock list.
Remedy: Check the lock numbers for the workspace using the workspace locks command and enter a valid number.
- 2076:** Internal error: Cannot find the directory in which command "*command_name*" is located because `avo_find_dir_init()` has not been called
Meaning: This is an internal error.
Remedy: Please contact your local service representative.
- 2077:** *number* is not a valid number
Meaning: While using the `resolve` command, a number was referenced that is outside of the listed values.
Remedy: List the values to determine the valid number for your selection.
- 2078:** Cannot access workspace "*workspace_name*"
Meaning: File permissions for *workspace_name* prohibit access by the CodeManager command.
Remedy: Default permissions for workspace directories are "777".

- 2079:** Could not parse name history for file
"file_name" , contains: text
- Meaning: There is a format error in the name history record in the SCCS history file
file_name. The troublesome text is displayed.
- Remedy: If possible, fix the record; otherwise copy a new version of the file using the
Bringover or Putback transaction.
- 2080:** Could not remove or rename backup directory
"directory_name"
- Meaning: CodeManager attempted to clear the backup area *directory_name* so that it
could backup a new transaction. CodeManager 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.
- Remedy: Check directory permissions for *directory_name*. Default CodeManager
permissions for this directory are "777".
- 2081:** build_workspace_list: "path_name" does not start with a /
- Meaning: The CodeManager GUI program was expecting a fully qualified path name to
be returned from a subprocess. This is an internal error.
- Remedy: Please contact your local service representative.
- 2082:** Workspace *workspace_name*'s parent does not exist in the filesystem
- Meaning: The parent workspace is not mounted or visible on this machine.
- Remedy: Mount the parent workspace on the executing machine.
- 2083:** Workspace *workspace_name*'s child does not exist in filesystem
- Meaning: The child workspace is not mounted or visible on this machine.
- Remedy: Mount the child workspace on the executing machine.

- 2084:** `codemgrtool: internal error in args_strlist_from_wsname() :
NULL args_list`
- Meaning: Internal error.
- Remedy: Please contact your local service representative.
- 2085:** `codemgrtool: internal error in undo_strlist_from_wsname() :
NULL undo_list`
- Meaning: Internal error.
- Remedy: Please contact your local service representative.
- 2086:** `codemgrtool: "path_name" doesn't start with a /`
- Meaning: Internal error.
- Remedy: Please contact your local service representative.
- 2087:** Not used
- 2088:** Nametable in workspace "*workspace_name*" cannot be read because the following SCCS files have identical root deltas
file_name
file_name
Run the following command and then re-execute the "*command_name*"
command:
`path_name/workspace updatenames workspace_name`
- Meaning: An SCCS history file was copied within a workspace using the `cp` command. As a result, the two files contain the identical root delta. CodeManager uses the root delta to distinguish between files. The `workspace updatenames` command enables CodeManager to distinguish between the files.
- Remedy: Execute the `workspace updatenames` command and then reexecute the command that spawned the error.

- 2089:** Cannot move workspace "*workspace_name*"
Because it is a symlink to "*directory_name*".
Use a workspace name that is not a symlink.
- Meaning:** CodeManager commands will not move directories or files that are symbolic links.
- Remedy:** Move the workspace to a name that is not a symlink.
- 2090:** Nametable in workspace "*workspace_name*" not written because the following SCCS files have identical root deltas
file_name
file_name
Run the following command and then re-execute the "*command_name*" command:
path_name/workspace updatenames *workspace_name*
- Meaning:** An SCCS history file was copied within a workspace using the cp command. As a result the two files contain the identical root delta. CodeManager uses the root delta to distinguish between files. The workspace updatenames command enables CodeManager to distinguish between the files.
- Remedy:** Execute the workspace updatenames command and then reexecute the command that spawned the error.
- 2091:** Internal error: hash table missing entry
- Meaning:** Internal error.
- Remedy:** Please contact your local service representative.
- 2092:** 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 "*command_name*" command:
path_name/workspace updatenames *workspace_name*
- Meaning:** An SCCS history file was copied within a workspace using the cp command. The original file (A) was subsequently renamed or removed from the workspace. CodeManager is unable to determine whether the files has been renamed (and to what name) or removed from the workspace. The workspace updatenames 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. CodeManager can then correctly propagate the changes throughout the workspace hierarchy.

Remedy: Execute the `workspace updatenames` command and then reexecute the command that spawned the error.

2093: Internal error: SmIDs not equivalent

Meaning: Internal error.

Remedy: Please contact your local service representative.

2094: Internal error: SmID not found

Meaning: Internal error.

Remedy: Please contact your local service representative.

2095 - 2499: Not used

2500 - 2600: Internal errors

Note – Error numbers in the 2500 range are all errors internal to CodeManager programs. Occurrence of these errors indicates problems that users cannot correct. If you encounter errors numbered in this range, please contact your local service representative.

CodeManager Warnings

- 2601:** Could not remove backup directory
"old_dir_name" ,
so it was renamed to
"new_dir_name"
- Meaning:** CodeManager attempted to clear the backup area *old_dir_name* so that it could backup a new transaction. CodeManager 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 *new_dir_name*. The most likely cause is that file permissions have been changed for the directory.
- Remedy:** Check directory permissions for *old_dir_name*. Default CodeManager permissions for this directory are "777". Delete the contents of *new_dir_name*.
- 2602:** File "*file_name*" is not under SCCS in either workspace - ignored
- Meaning:** CodeManager could not find an SCCS history file in either workspace for *file_name*.
- Remedy:** The file name was probably entered incorrectly, re-execute the command.
- 2603:** Zero length filename - ignored
- Meaning:** A file name specified as an argument on the command-line (or in the Codemgr_wsdata/args file) contained no characters ("").
- Remedy:** Re-execute the command and re-specify the file name argument. If the problem persists, check the arguments listed in the args file.
- 2604:** Filename "*file_name*" has whitespace characters in it - ignored
- Meaning:** A file name specified as an argument on the command-line (or in the Codemgr_wsdata/args file) contained whitespace characters. CodeManager commands do not accept file names that contain whitespace characters.
- Remedy:** Re-execute the command and re-specify the file name argument. If the problem persists, check the arguments listed in the args file.

- 2605:** Not used
- 2606:** File "*file_name*" not brought over because it is a *file_type* in workspace "*workspace_name*" and a *file_type* in workspace "*workspace_name*"
- Meaning: A file name has a different file type (regular file vs. directory vs. symbolic link) in the parent and child workspaces.
- Remedy: Take whatever action is appropriate to make the listed files the same type, or change one of the names.
- 2607:** Not used
- 2608:** Workspace "*child_ws_name*" is a child of "*parent_ws_name*". Could not update its parent file
- Meaning: During a workspace delete or workspace move operation involving *child_ws_name*, the command found that the children file in the workspace's parent (*parent_ws_name*) did not contain an entry specifying *child_ws_name* as a child of that parent.
- Remedy: Advisory only. The command will correct the discrepancy, however, this could indicate that the parent's children file has been corrupted.
- 2609:** Not used
- 2610:** "*directory_name*" is not a workspace
- Meaning: The directory specified in the command is not a CodeManager workspace. CodeManager workspaces are distinguished by the presence of the *Codemgr_wsdata* directory in the top level directory.
- Remedy: Specify a different workspace name or use the CLI *workspace create* command or GUI File ⇒ Create Workspace command to convert the directory into a workspace.
- 2611:** "*file_name*" does not exist in either workspace - ignored
- Meaning: The file *file_name* was not found in either the parent or child workspace.
- Remedy: Check to be sure the name was specified correctly.

2612: Not used

2613: Filename "*file_name*" has too many ".." path components in it - ignored

Meaning: Possible causes include:

- The CodeManager command cannot resolve the path name into a workspace-relative file name
- The CodeManager command cannot resolve the path name into a fully qualified workspace name

Remedy: Specify the path name with fewer (or no) ".." path name components

2614: Line "*line_number*" too long or unexpected end of file in "*file_name*"

Meaning: While reading the `Codemgr_wsdata/nametable` file, a line was encountered that contained too many characters for a CodeManager command to buffer. The maximum line length is 1024 characters. This indicates that `nametable` has been corrupted.

Remedy: CodeManager automatically rebuilds the `nametable`. As you have probably noticed, this takes some time.

2615: Line "*line_number*" has bad format in "*file_name*"

Meaning: This indicates that the `Codemgr_wsdata/nametable` file has been corrupted.

Remedy: CodeManager automatically rebuilds the `nametable`. As you have probably noticed, this takes some time.

2616: Not used

2617: Unexpected name table editlog record type "*type_number*" - ignored

Meaning: A CodeManager 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.

Remedy: Execute the `workspace updatenames` command to rebuild `nametable` and then re-execute the command.

-
- 2618:** Can't open "*file_name*" - can't send mail notification
Meaning: The CodeManager notification facility failed to open the file *file_name*. As a result, notification mail will not be sent for the current operation.
Remedy: Check file permissions for *file_name*.
- 2619:** Not used
- 2620:** Can't fork process to send notification
Meaning: Lack of system resources (memory, swap space) prevented the CodeManager notification facility from sending notification mail.
Remedy: Check system resources.
- 2621:** Not used
- 2622:** Filename "*file_name*" contains a comment character (#) - ignored
Meaning: A file name specified as an argument to a command (or in the Codemgr_wsdata/args file) contains the "#" character. CodeManager reserves this character to denote comments.
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.
- 2623:** Read-lock left in workspace or Write-lock left in workspace
Meaning: A Codemanager command was unable to remove locks in "workspace_name. 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.
Remedy: Remove the lockes using the CodeManager GUI Props workspace command or the CLI workspace locks command.

- 2624:** File "*file_name*" is checked out in workspace "*workspace_name*". The changes in the checked out file will not be brought over
- Meaning: The file *file_name* is checked out in the parent workspace. You are being advised that any changes in the g-file have not been brought over as part of the Bringover transaction.
- Remedy: N/A
- 2625:** File "*file_name*" is not in conflict according to the SCCS file. Removing it from the conflict file
- Meaning: The information in the SCCS history file indicates that the file contains no unresolved conflicts, however, the `Codemgr_wsdata/conflicts` file in the workspace lists it as being in conflict. The command removed it from the `conflicts` file.
- Remedy: N/A
- 2626:** File "*file_name*" not brought over because it is unresolved in workspace "*workspace_name*"
- Meaning: The file *file_name* was not brought over because it contains an unresolved conflict in *workspace_name*.
- Remedy: Use the GUI Resolve transaction or the CLI `resolve` command to resolve the conflict and then re-execute the Bringover transaction.
- 2627:** Directory "*directory_name*" is mounted read-only.
- Meaning: Before beginning Bringover and Putback transactions, Codemanager 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.
- Remedy: If write access is not intentionally denied, change the root directory permissions.

- 2628:** Not updating g-files because "get" command couldn't be found in PATH "*search_path*"
- Meaning: The g-files could not be updated as part of a Bringover or Putback transaction because the SCCS `get` command could not be executed; it was not found in your search path.
- Remedy: If you want g-files to be updated as part of transactions, include the `get` command in your search path.
- 2629:** Will not be able to run `filemerge`
- Meaning: The `resolve` command was not able to connect with the ToolTalk message service. The ToolTalk service is used by the `resolve` command to communicate with the FileMerge program.
- 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.
- 2630:** This workspace is being created over an existing directory
- Meaning: This message advises you that you are converting an already existing directory into a CodeManager workspace. Creating a workspace from an existing directory hierarchy consists of creating the `Codemgr_wsdata` metadata directory in the top-level directory. Be aware that once the directory becomes a workspace, its contents can be deleted using the CodeManager `workspace delete` command.
- Remedy: N/A
- 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 CodeManager transaction.

Part 3 — Version Tool



<i>Introduction to VersionTool</i>	<i>page 215</i>
<i>Performing Basic SCCS Functions with VersionTool</i>	<i>page 233</i>

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.

VersionTool is a GUI to SCCS. Version Tool allows you to manipulate files and perform 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 VersionTool, 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

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

This chapter is organized into the following sections:

- “Terminology” on page 216
- “Graphical Tour” on page 218
- “Base Window” on page 218
- “File Button” on page 220
- “Load Button” on page 221
- “View Button” on page 222
- “Commands Button” on page 228
- “Props Button” on page 231

Terminology

This section covers the terminology and concepts of SCCS. Familiarity with these terms will help you when working with a project under SCCS control. This terminology applies mainly to SCCS and its ability to keep track of changes to files.

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.

Situations can arise, however, when it is necessary to create an alternative *branch* on the tree. Branches can be used, for instance, 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 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).

Deltas and Versions

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.

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*).

SCCS Delta ID (SID)

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 *release* number, and the second, the *level* number. When you check in a delta, the level number is incremented automatically.

Graphical Tour

This chapter is dedicated to provide an overview of the functionality of the VersionTool Base window and control buttons. Included in this discussion are the pop-up windows associated with their buttons. The overview is organized in the following sections:

- Base Window
- File Button
- Load Button
- View Button
- Commands Button
- Show Output Button
- Props Button

Base Window

The VersionTool base window displays the directories and SCCS files of the *loaded* directory; the directory path is shown in the *Directory* text field. Directories are shown as file folders; files are shown as pieces of paper. Files checked out from SCCS are marked with a check mark. Figure 11-1 is an example of a typical base window directory listing.

Directories are shown whether they contain files which are under SCCS control or not. A directory is a container for files and directories and can possibly contain SCCS files and directories further down the hierarchy. *Files are only shown in the Base window if they are under SCCS control.* To look at the non-SCCS files in a directory, use the Check in New window. For more information, see “Check In New Window” on page 229.

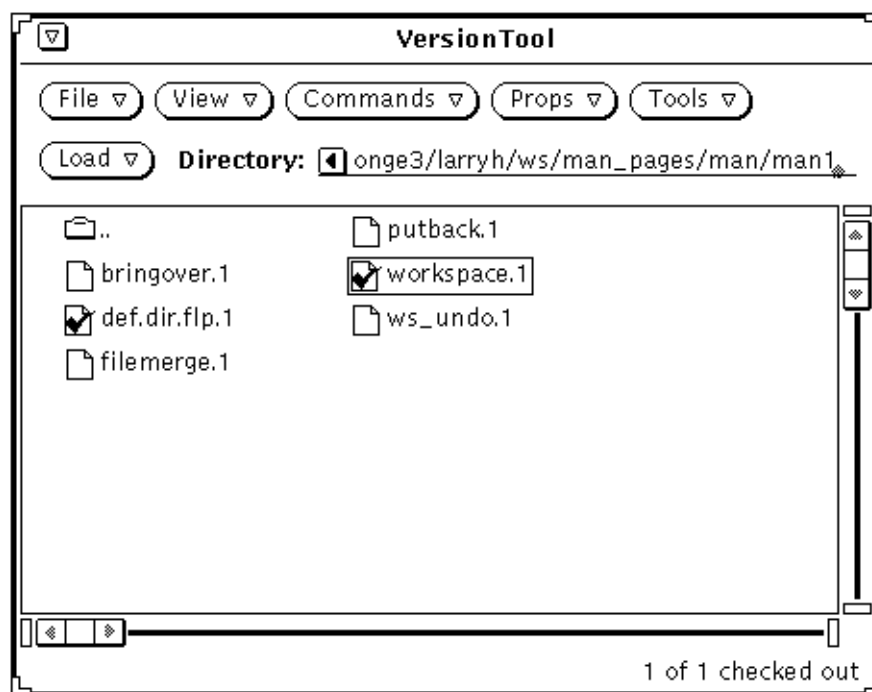


Figure 11-1 *VersionTool Base Window*

The control buttons at the top of the window provide menus and options for performing SCCS functions. The buttons are described starting with “File Button” on page 220.

The scroll bars at the side and bottom of the window allow you to scroll through, and across, an extensive listing.

Base Window Pop-up Menu

The Base window has a pop-up menu that offers the menu items shown in Figure 11-2. For more information on the base window pop-up menu options, see “Commands Button” on page 228.

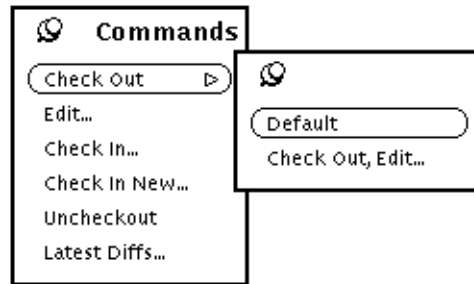
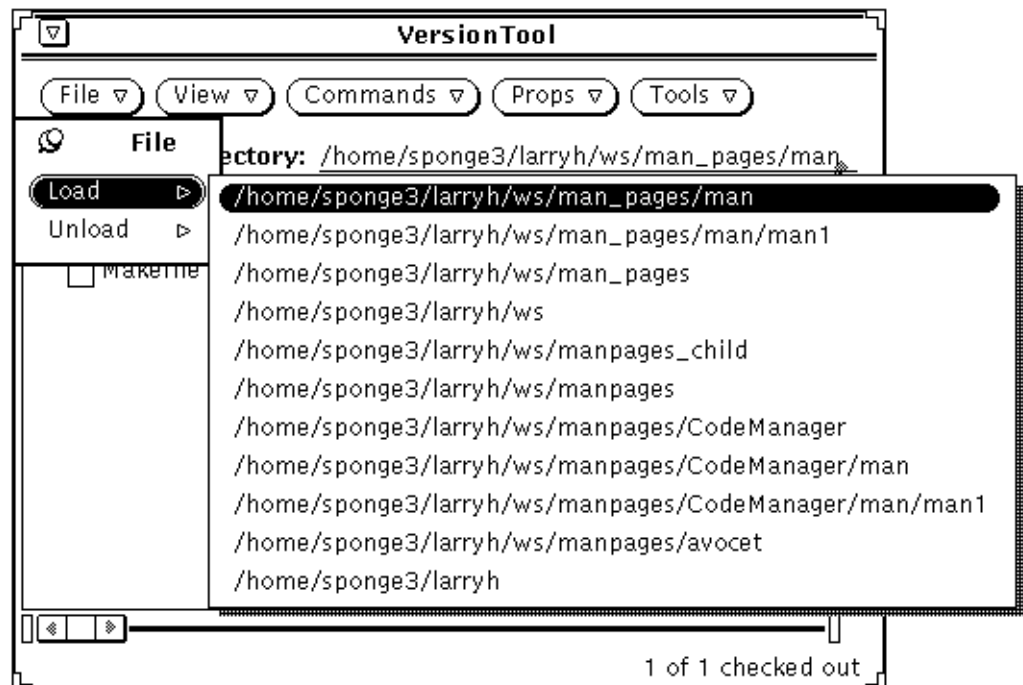


Figure 11-2 Base Window Pop-up Menu

File Button

The File button allows you to load and unload files into the base window file list display. The File button menu has the following menu items:



The **Load** menu item loads the directory name you type in the Directory text field. A list of previously loaded directories is saved. The Load directory list is the same as the Unload directory list.

The **Unload** menu item lets you unload a directory. The pop-up submenu displays a list of previously loaded directories. The directory displayed at the top of the list is the currently loaded directory. You unload a directory by choosing it from the display list. For more information on unloading directories, see Chapter 2, “VersionTool Basics.”

Load Button

The Load button displays a menu with a list of previously loaded directories. It is the same list displayed under Unload in the File menu. To reload a directory, choose it from the list. For more information on loading directories, see Chapter 2, “VersionTool Basics.”

View Button

The View button menu displays the following menu and pop-up submenu:

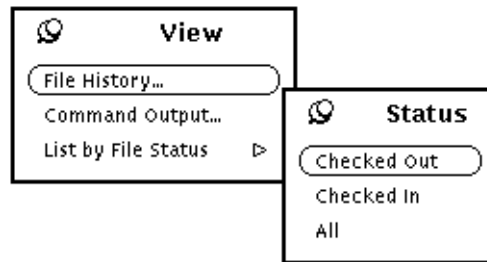


Figure 11-3 View Pop-up Window

The **File History** menu item brings up the History window and displays a history graph of a selected SCCS file. For information on the History window, see “History Window” on page 223.

The **Command Output** menu item brings up a pop-up window that allows you to selectively display the following information:

- Commands being executed
- Output of executed commands
- Errors from executed commands

You can display information types exclusively or in combinations by selecting the appropriate box in the pop-up window header. A check mark is displayed in the box once it is selected.

The Clear button allows you to clear the pane of displayed information. The scroll bar at the side of the output pane lets you peruse an extensive list of displayed information.

Figure 11-4 is an example of the Command Output pop-up window.

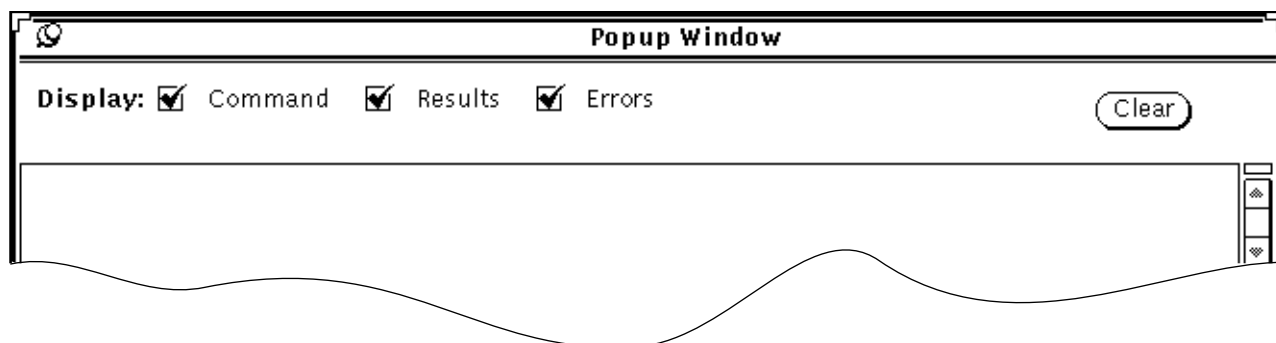


Figure 11-4 Command Output Pop-Up Window

The **List by File Status** menu item brings up a pop-up window that lets you filter the file list display according to status. You have the option of creating a list of:

- Checked-out files
- Checked-in files
- All files (default)

History Window

The History window displays an illustration of SCCS delta branches for a selected file. This *history graph* allows you to peruse the delta structure of a file and assess associations between versions. Dashed lines are shown by default and indicate that the delta to the right of the dashed line was created by including the changes from the delta on the left. Following the dashed line provides you with a time-ordering sequence. In Figure 11-5, delta 1.123 is comprised of changes from both 1.122 and 1.120.1.2.

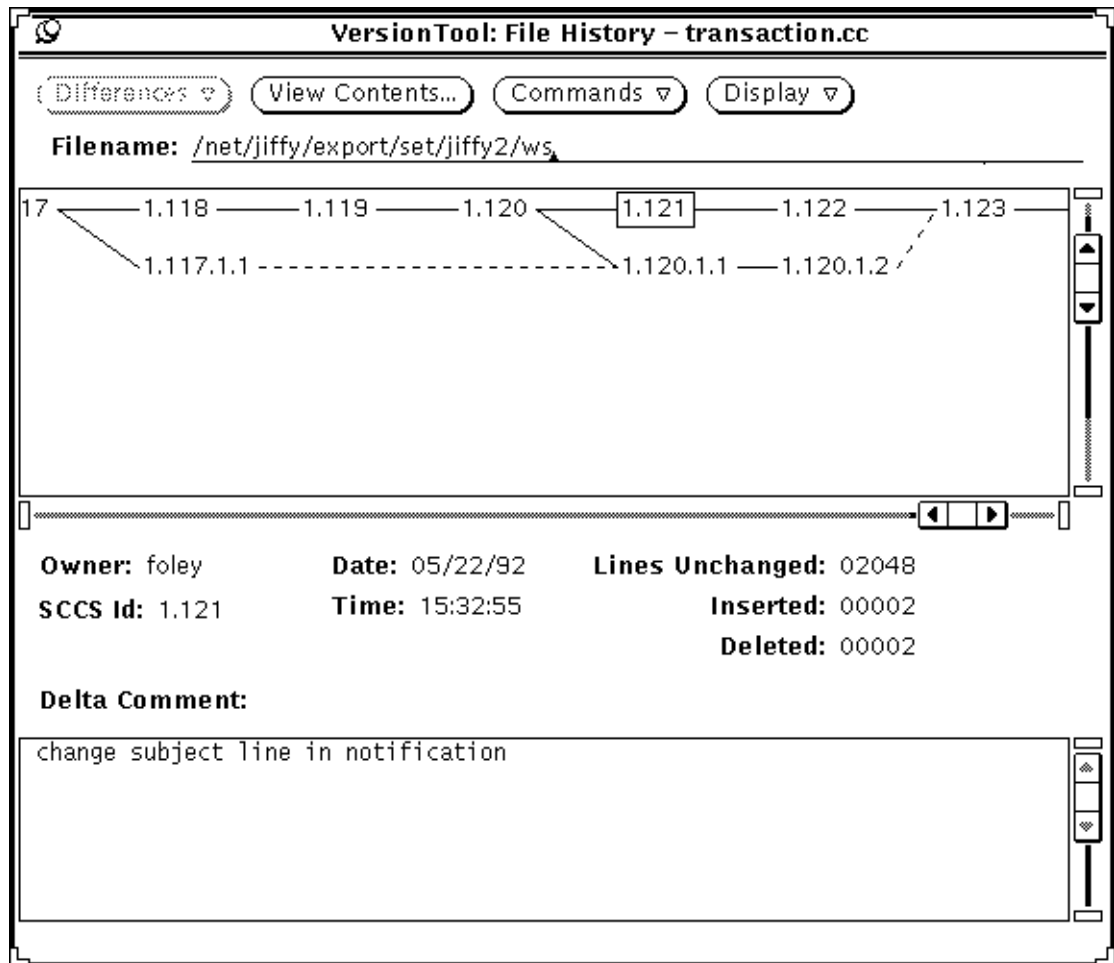
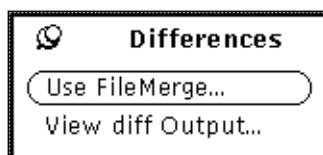


Figure 11-5 File History Window

The scroll bars at the right and bottom of the window allow you to scan a large history graph. If a delta on the displayed branch is checked out, a check mark is shown to the left of the SID.

The **Differences** button menu gives you the choice between bringing up a FileMerge window or a text editor displaying the textual differences between two selected deltas. The Differences button is only activated when two deltas are selected.



- The **Use FileMerge** menu item automatically brings up FileMerge when selected. The two selected deltas are displayed in the side-by-side panes. The name of the common ancestor is shown at the top of the window.
- The **View diff Output** menu item allows you to display the textual differences between two selected deltas. Figure 11-6 is an example.

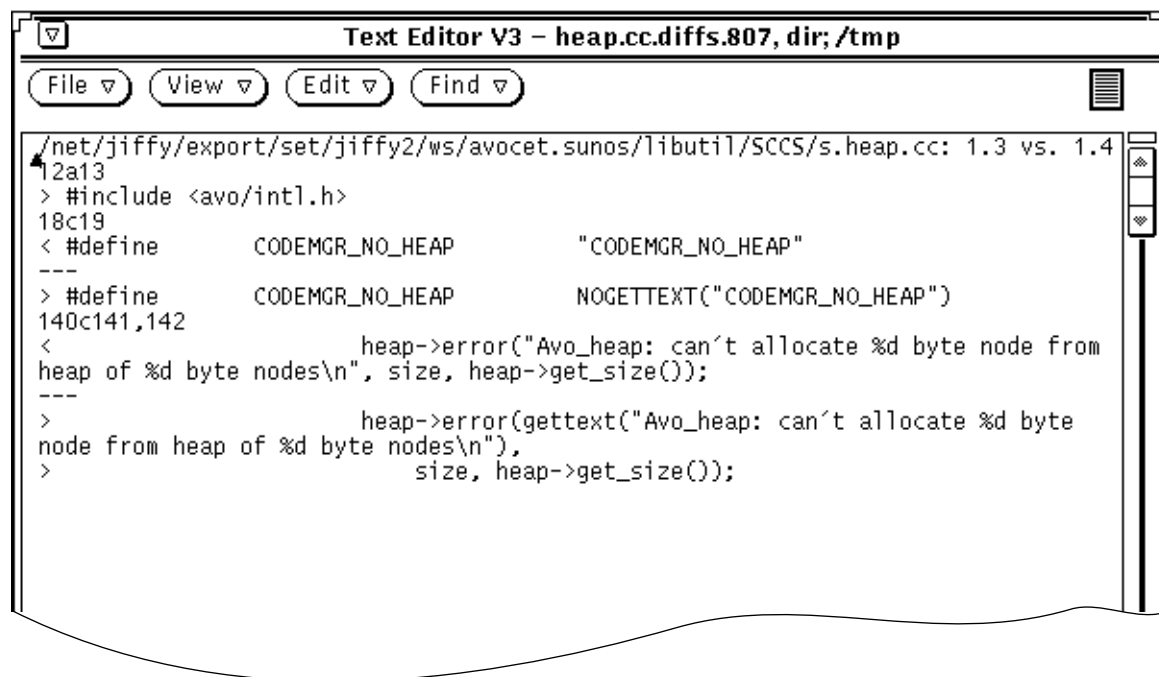


Figure 11-6 Text Editor Displaying Textual Output of Two Deltas

The **View Contents** button brings up the editor of your choice and displays the contents of the selected file. You can edit the file and create a temporary copy in your `/tmp` directory. You cannot write to the file itself. For information on how to define an editor, see Section , “Props Button,” on page 231.

The **Commands** button menu lets you check out a file, check in a file, view the contents of a delta, or the differences between two selected deltas. For more information on the Commands menu, see “File History Window Commands Button” on page 226.

The **Version Information** pane at the bottom of the History window displays information for the most recently selected delta. The following information is displayed in the Version Information pane:

- SCCS ID
- Owner
- Time
- Date
- Lines unchanged, inserted, deleted

The **Delta Comment** pane is a read-only display of the comments for the selected delta.

File History Window Commands Button

The Commands button menu on the File History window displays the following items:

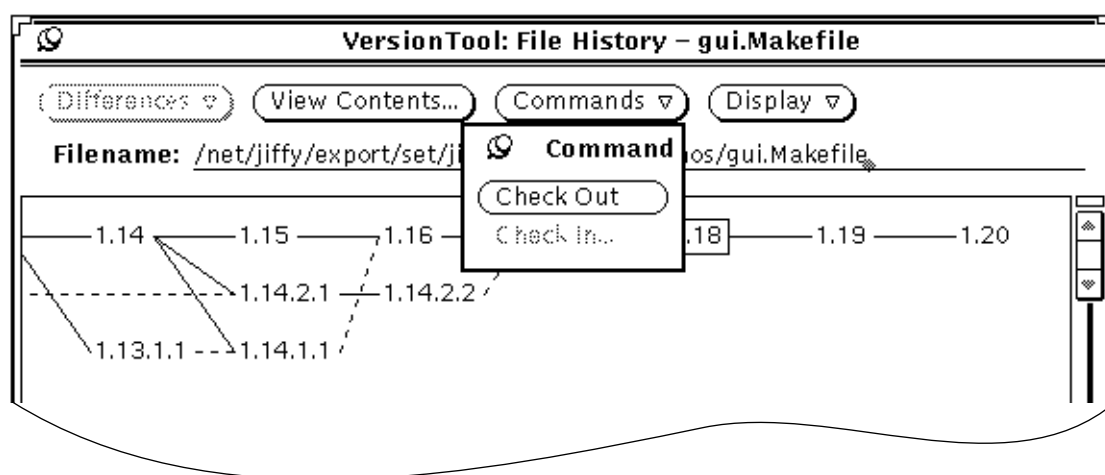


Figure 11-7 History Window Commands Button Menu

The **Check Out** menu item has a pop-up menu that lets you check out a selected delta, or check out a delta and bring it up in an editor. For information on how to define an editor, see Section , “Props Button,” on page 231. The Check Out menu item is *only* active when you select a checked in delta.

The **Check In** menu item brings up a pop-up window that lets you specify comments for the delta to be created. Once you add comments, you can check in the file by selecting the Check In button.

File History Window Pop-up Menu

The History window has a floating pop-up menu that, when displayed, provides the same menu items as the File History window Commands button menu. For more information on the File History Commands button menu, see “File History Window Commands Button” on page 226.

Commands Button

The Commands button menu on the base window displays the following items:

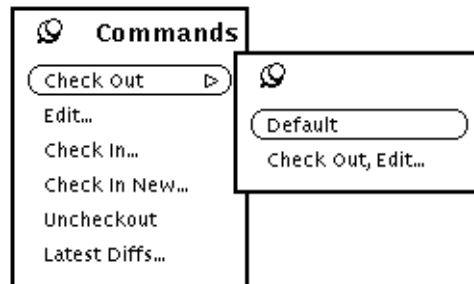


Figure 11-8 Commands Pop-up Menu

The **Check Out** menu item has a pop-up menu that lets you check out a selected delta, or check out a delta and bring it up in an editor. For information on how to define an editor, see Section , “Props Button,” on page 231.

The **Edit** menu item brings up the contents of a selected file into an editor. If you select multiple files, an editor is brought up for each file.

The **Check In** menu item brings up a pop-up window that lets you specify comments for the delta to be created. Once you add comments, you can check in the file using the Check In button on the Check In pop-up.

The **Check In New** menu item brings up the Check In New window. The Check In New window displays a list of files *not yet* under SCCS control. It allows you to select files, add comments, and check them in. For more information on the Check In New window, see “Check In New Window” on page 229.

The **Uncheckout** menu item takes a checked-out writable file, removes its write permissions, and reverts it back to the last fixed delta state for the file.

The **Latest Diffs** menu item brings up FileMerge with the latest clear copy of a selected file and the latest checked-in delta displayed in side-by-side panes. This is a three way diff with the differences between the files discerned from

the common ancestor. The common ancestor file name is displayed in the Filemerge window header. This is a read-only display for browsing the latest differences between versions.

Check In New Window

The Check In New window displays the directory files that are *not* under SCCS control. The scroll bars to the right and bottom of the window let you peruse an extensive file listing.

The **List** field gives you the option of displaying all the files that are not under SCCS control or showing a specified group of files as defined by the shell pattern text field. Figure 11-9 shows the List pop-up menu. Figure 11-10 shows the Check In New window with the by Pattern menu item activated.

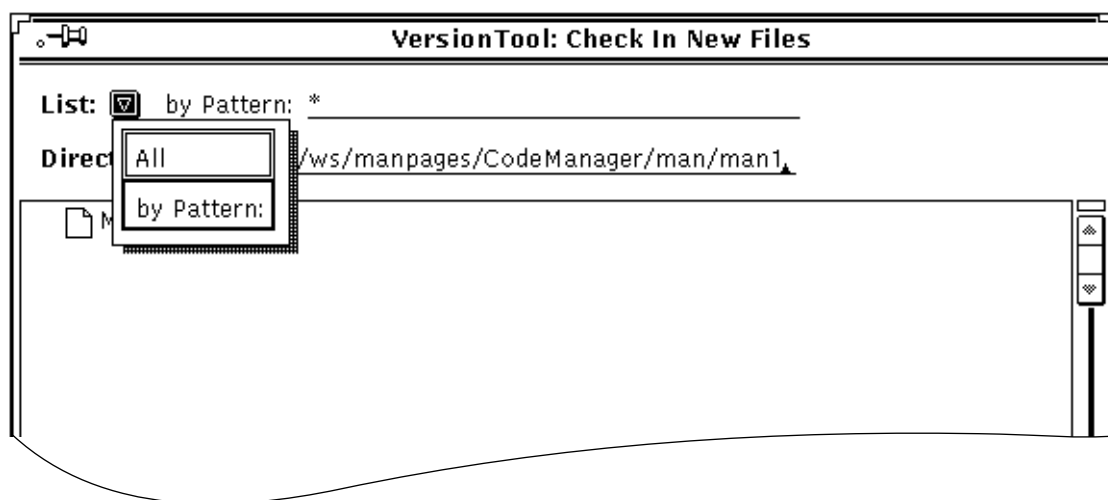


Figure 11-9 List Field Options

The **Directory** text field displays the directory path name for the files listed in the Check In New window. You can edit this field to define the path name of different directories.

The **Initial Comments** pane at the bottom of the window allows you to enter comments for a selected file that you want to check in under SCCS control. The scrollable comments pane supports a text-wrapping function that allows for extensive comments. When you select a file to check in, it is highlighted with a rectangular box.

The **Check In** button lets you check in the selected file or files.

The **Reset** button automatically clears the comment pane.

Once you check in a file under SCCS control, it is automatically removed from the Check In New display and transferred to the file list displayed in the Base window.

Figure 11-10 is an example of the Check In New window.

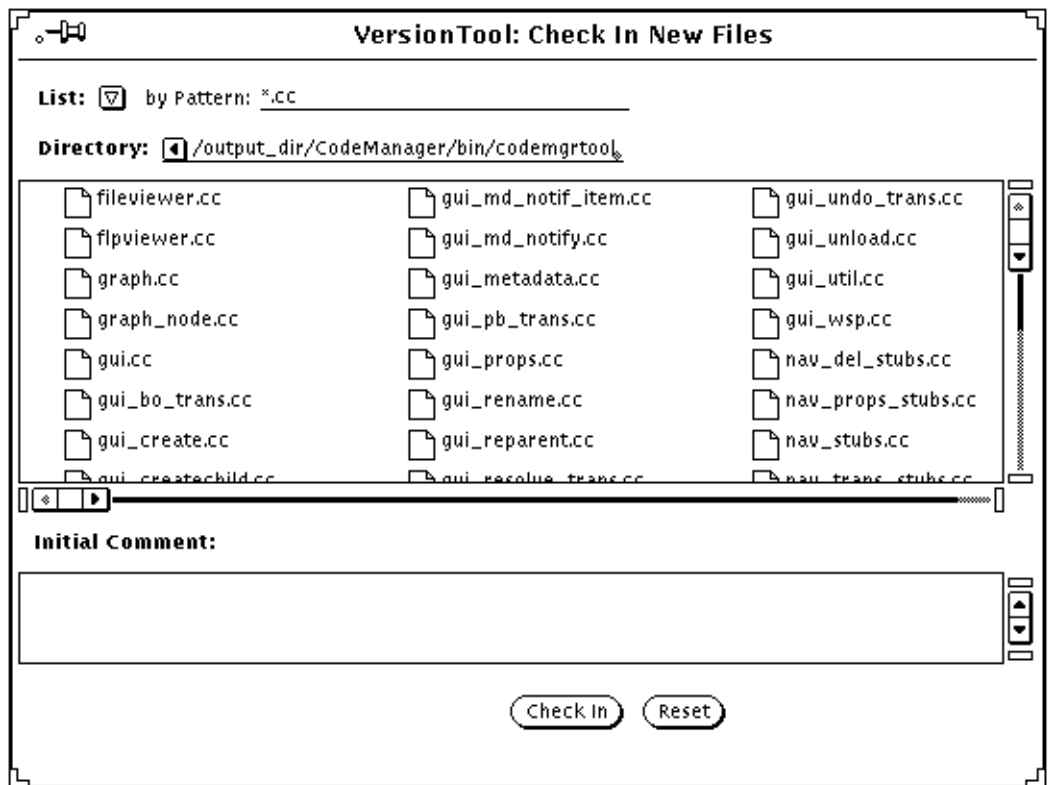
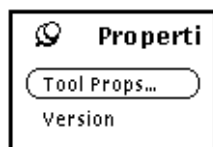


Figure 11-10 Check In New Window

Props Button

The Props button menu displays the following items:



The **VersionTool Props** menu item brings up a Properties window that allows you to specify VersionTool properties. The Properties window and its options are described in the following section.

The Version menu item displays the current version number of VersionTool in the message area of the base window. Figure 11-11 shows the message area where the version number is displayed.

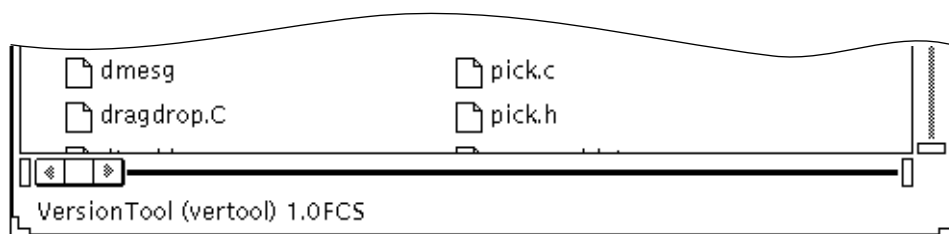


Figure 11-11 Version Display in Footer of Base Window

Properties Window

The Properties window allows you to set VersionTool properties in the following categories:

- Main File List
- Editor
- Double Click Action
- History Graph
- History Information
- SCCS File History Command

Figure 11-12 is an example of the Properties window. For more information on changing VersionTool properties, see Chapter 2, "VersionTool Basics."

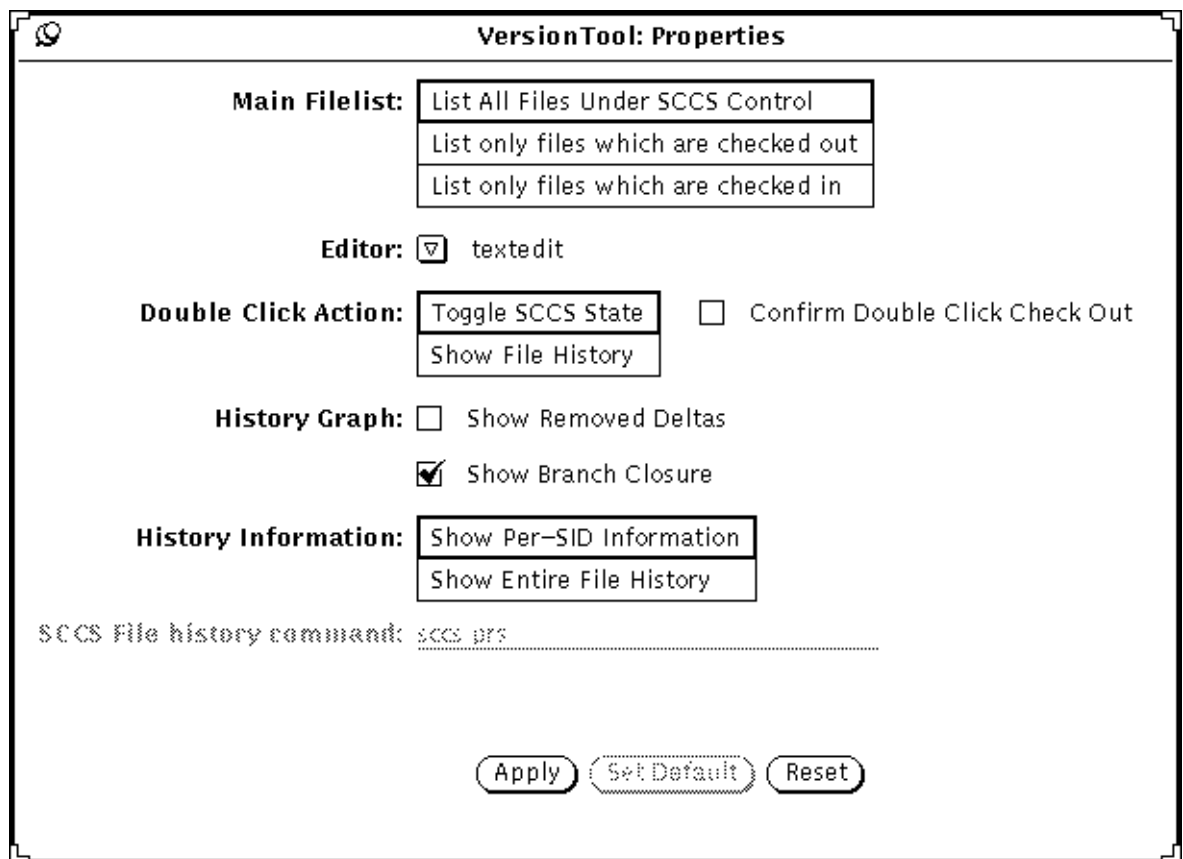


Figure 11-12 Properties Window

Performing Basic SCCS Functions with VersionTool

12 

This chapter shows you how to perform basic SCCS functions using VersionTool. It is organized into the following sections:

- “Typical Tool Sessions” on page 233
- “File Button: Loading and Unloading a Directory” on page 235
- “Load Button: Reloading Previous Directories” on page 237
- “View Button: Viewing File Information” on page 238
- “Commands Button: Manipulating Files” on page 242
- “Props Button: Changing VersionTool Properties” on page 247

Typical Tool Sessions

This section gives an overview of the most common types of tool sessions. 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

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 VersionTool 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% vertool &
```

```
demo% vertool dirname &
```

- 2. Double click on the project directory from the list displayed in the base window.**
VersionTool 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 button 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 Check In 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 VersionTool 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 VersionTool 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 VersionTool.

File Button: Loading and Unloading a Directory

The File button menu allows you to:

- Load a directory
- Display a list of loaded directories
- Unload a directory

There are several methods for loading and unloading the contents of a directory into the VersionTool file list. The Load and Unload options are on the File button menu as shown in Figure 12-1.

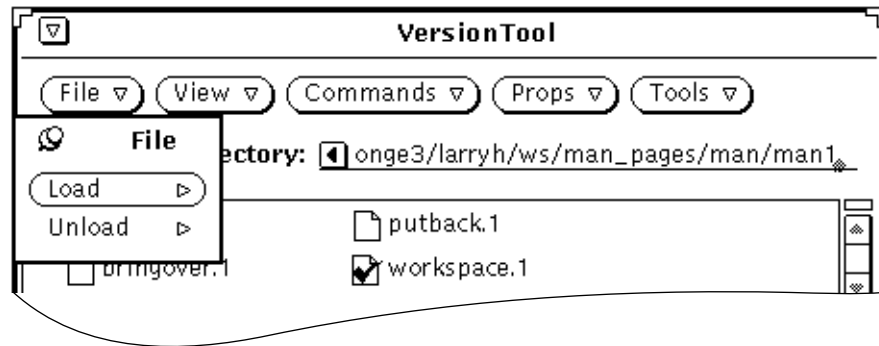


Figure 12-1 File Button and Menu

Loading a Directory

To load the contents of a directory into VersionTool:

1. **Type, or cut and paste, the directory name into the Directory text field.**
2. **Press Return, select the Load button, or choose Load from the File button menu.**

The files of the directory that are under SCCS control, as well as the directories they contain are displayed, replacing the previous contents of the base window.

3. **To load subdirectories, double click on the name or glyph of the directory you wish to load, or repeat Steps 1 and 2.**

The files of the directory which are under SCCS control and the contained directories are automatically loaded and displayed.

Unloading a Directory

The paths of loaded directories are stored in a list that is displayed when you select the Unload option of the File menu. This directories list is the same as the one displayed under the Load menu item. The current directory is placed at the top of the directories menu. As you load more directories, previously loaded directories are pushed down the list. Figure 12-2 shows an example of the directories list displayed under the Unload option.

To unload a directory from the directories list under the Unload menu item:

- ◆ **Choose the directory name from the Unload menu item display.**
The directory is automatically removed from the directories list.

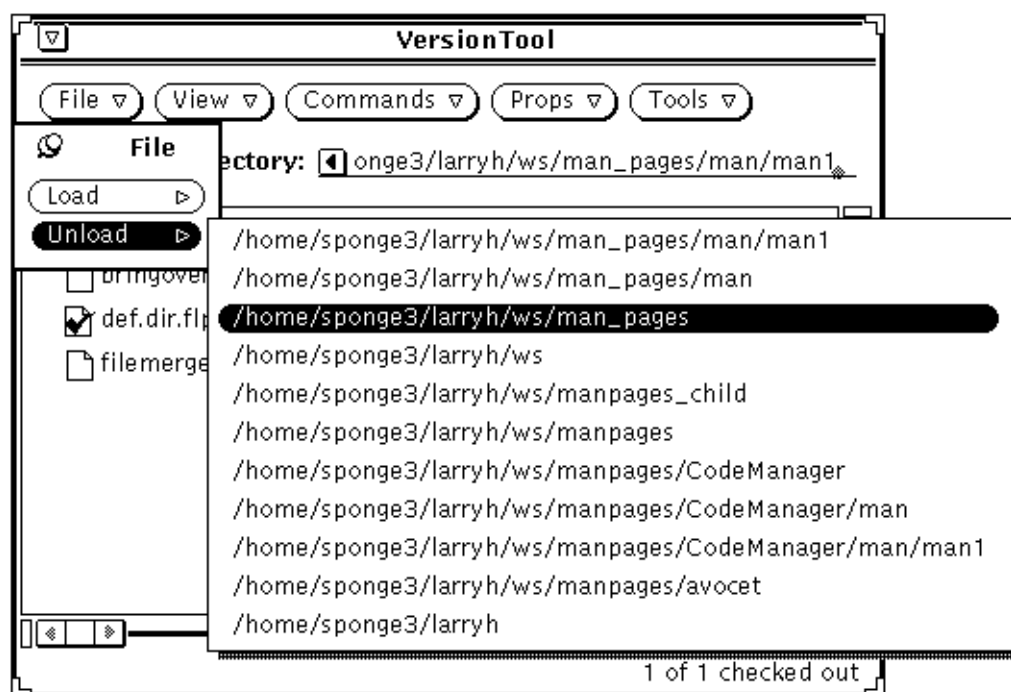


Figure 12-2 Unload Directories List

Load Button: Reloading Previous Directories

The Load button menu allows you to do the following:

- Display a list of loaded directories.
- Reload a previously loaded directory.

The paths of loaded directories are stored in a list that is displayed when you choose the Load button. The current directory is placed at the top of the directories menu. As you load more directories, previously loaded directories are pushed down the list. Figure 12-3 shows an example of the directories list displayed under the Load button.

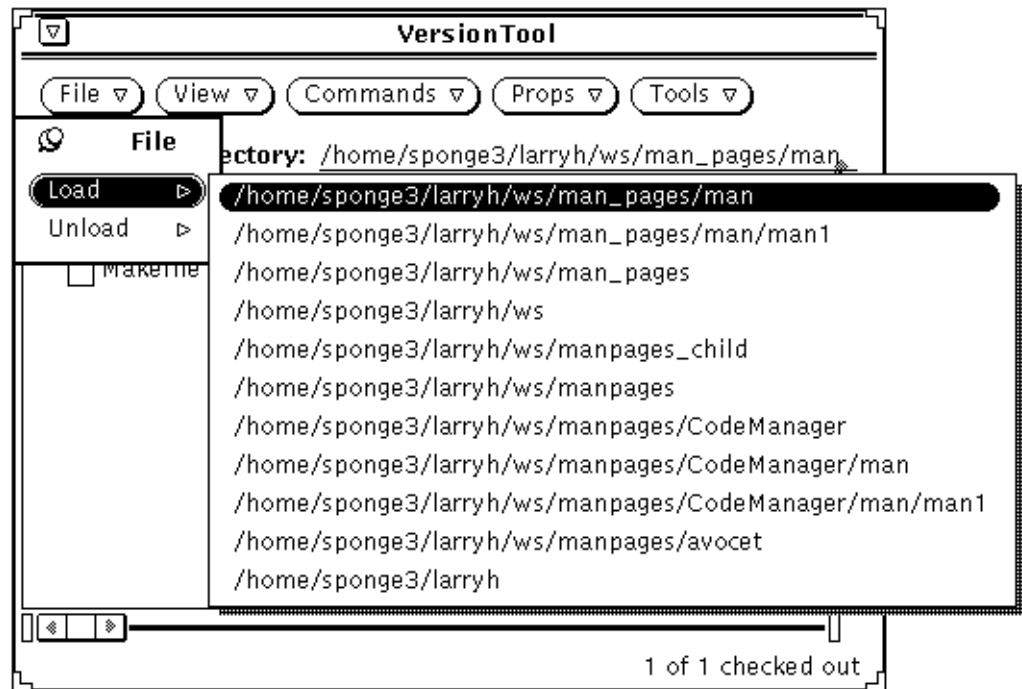


Figure 12-3 Loaded Directories List

To reload a previously loaded directory:

- ◆ **Choose the directory name from the Load menu item display.**
Its contents are loaded into the base window file list display, and the directory name is displayed in the Directory text field.

View Button: Viewing File Information

The View button menu allows you to do the following:

- View the history graph of a file.
- View SCCS command output.
- List files according to the SCCS state.

Viewing the History Graph of a Selected File

A history graph is a pictorial display of the delta structure of a selected file. The information contained in the File History window includes the history graph, as well as specific information on the selected deltas in the graph.

To display the history graph:

- ◆ **Select the file from the base window file list and choose File History from the View button menu.**

The History window displays the history graph of the file shown. When you select a delta version from the history graph, it is highlighted with a rectangular box, and the information for the delta is shown in the Version Information Pane.

Figure 12-4 shows a history graph with the Show Default Delta selected.

Note – In the case where several delta versions are highlighted simultaneously, the information of the most recently selected delta is displayed.

Using the File History Window

Once you have brought up the File History window with a history graph, you can:

- Select a delta from the history graph which will display information about the delta in the Version Information Pane.
- Select a delta from the history graph to check it in or out, depending on its current SCCS state.
- View the contents of a selected delta by choosing the View Contents button. This brings up an editor with the contents of the selected delta displayed. For information on how to define the editor, see “Props Button: Changing VersionTool Properties” on page 247.
- Select two deltas and choose the View Differences menu item from the Differences button menu. This displays a FileMerge window. The two selected deltas are displayed side by side for comparison. The common ancestor name is displayed at the top of the window.

- Select two deltas and choose the View diff Output menu item from the Differences button menu. A Text Edit window automatically comes up displaying the textual differences from the `scs diffs` command.

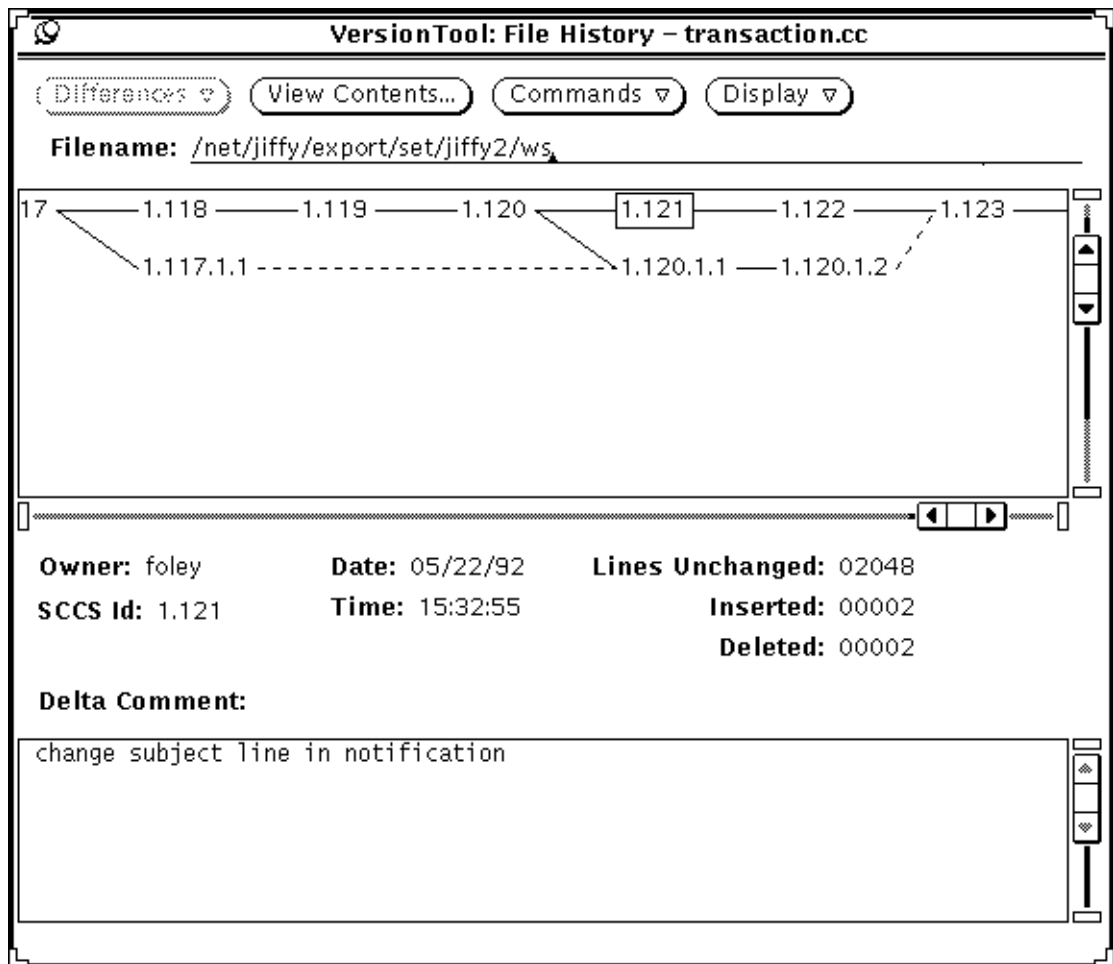


Figure 12-4 History Graph Showing Default Delta

Viewing SCCS Command Output

The Commands Output menu item displays a pop-up window that allows you to view:

- Executed commands
- Output of executed commands
- Errors from executed commands

You have the option of displaying the information types exclusively, or in combinations. You do this by selecting the appropriate box in the control area. Once selected, a check mark is displayed in the box. To undo a selection, reselect the box. The check mark goes away upon reselection.

Note – This filter only affects new output. For instance, errors from previously shown output are not removed if the Errors check mark is turned off.

Viewing SCCS File Status

The List by File Status menu item allows you to filter the file list according to SCCS status as follows:

- Checked-out files
- Checked-in files
- All files

Note – Checked-out files have a check mark on the file icon. Checked-in files have the regular file icon.

To display a selective list of files by status:

- ◆ **Choose the List by File Status menu item from the View button menu. Select the appropriate SCCS status from the pop-up submenu.**

The base window file list changes to display a list of files of the selected SCCS status. Figure 12-5 shows File Status pop-up menu.

Note – To show all the files after choosing the Checked In files or Checked Out files menu items, you must reselect the All Files menu item.

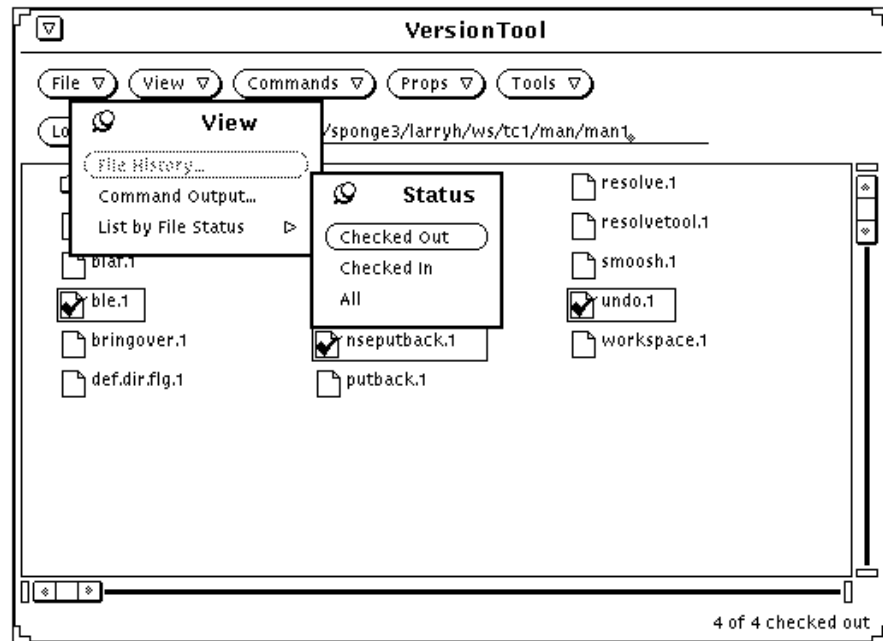


Figure 12-5 File Status

Commands Button: 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 button 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.
- Display the differences between two selected deltas.

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.

Figure 12-6 is an example of selecting a file from the base window and using the Check Out option of the Commands menu.

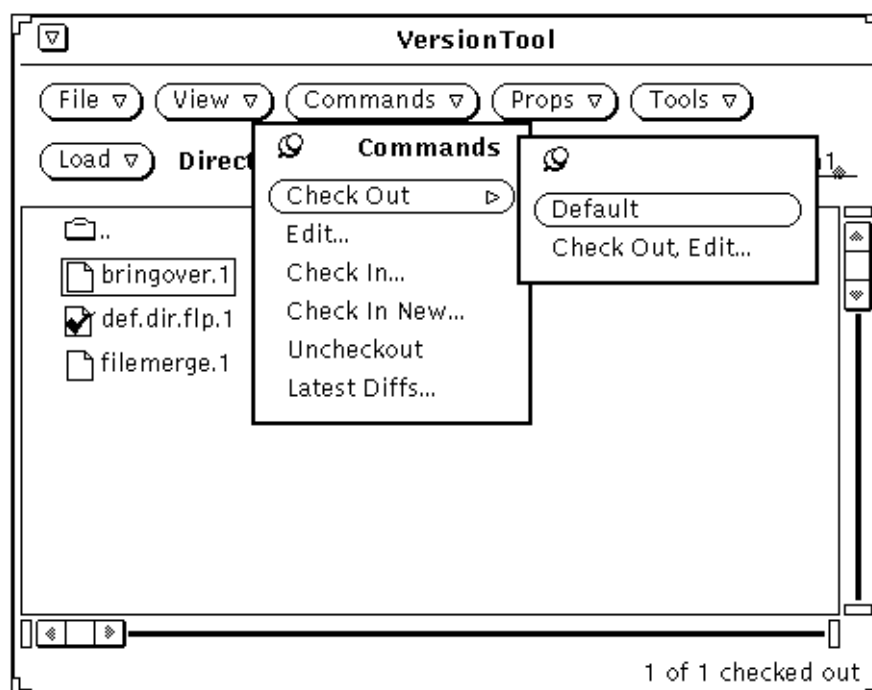


Figure 12-6 Check Out a File

- ◆ **Double click on a file icon in the base window (if set in the Properties 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 Props button functions. See, “Props Button: Changing VersionTool Properties” on page 247.

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.

Figure 12-7 is an example of selecting a file from the base window and using the Check In option of the Commands menu.

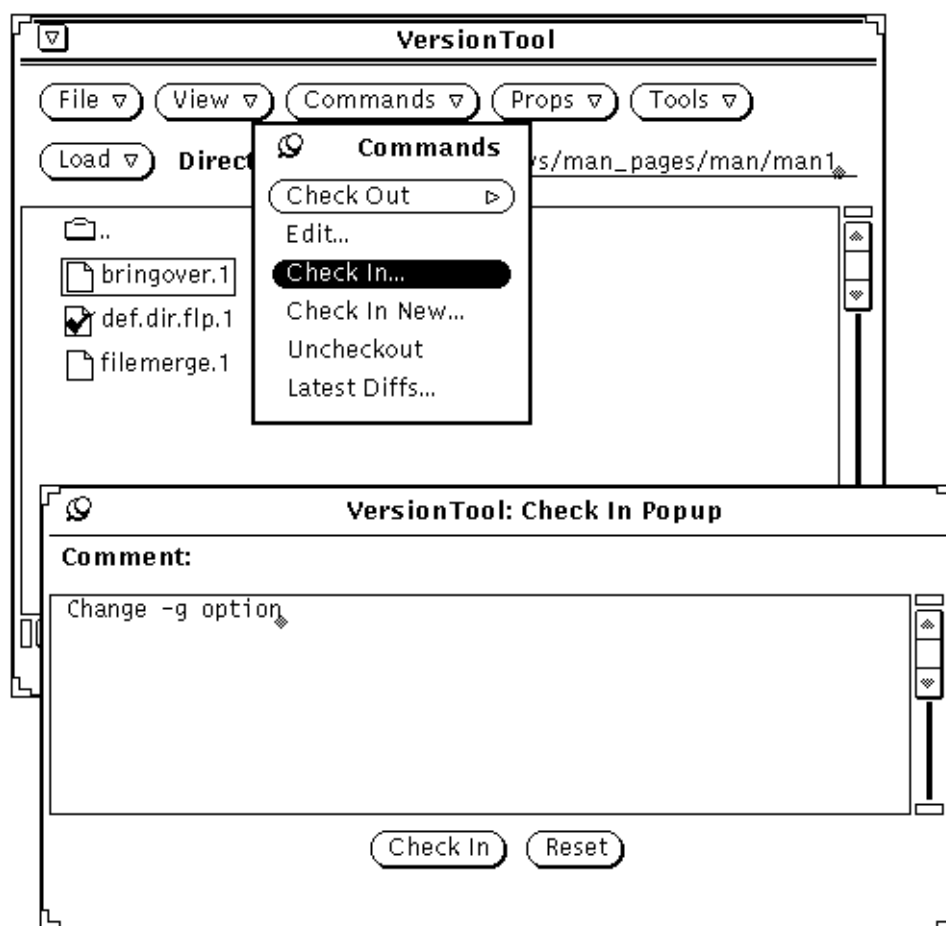


Figure 12-7 Check In a File

- ◆ **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 Properties 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 Props button functions. See, “Props Button: Changing VersionTool Properties” on page 247.

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 edit a file after checking it out:

- ◆ **Select the checked out file from the base window. Then choose Edit from the Commands button menu.**

The default is for the file to be brought up in a cmdtool window running `vi`. For instructions on defining an editor, see “Props Button: Changing VersionTool Properties” on page 247.

To check out a file and display it in a window with an editor:

- ◆ **Choose the Check Out, Edit menu item from the Check Out pop-up menu of the Commands button menu.**

The default is for the file to be brought up in a cmdtool window running `vi`. For instructions on defining an editor, see “Props Button: Changing VersionTool Properties” on page 247.

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 shell pattern in the Shell Pattern 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.

Displaying the Differences Between Two Deltas

It is possible to display the differences between two delta versions in side-by-side panes. The display is a three way diff with the differences between the files discerned from the common ancestor. The common ancestor name is displayed at the top of the FileMerge window. This is a *read only* display that lets you browse the latest differences between the file versions.

To display the differences between two delta versions:

- ◆ **Select the file from the base window list and choose Latest Diffs from the Commands menu.**

FileMerge comes up automatically displaying the latest clear copy of the selected file and the latest checked in delta.

Props Button: Changing VersionTool Properties

The Props button displays a pop-up window with VersionTool properties options. Selecting options from this window sets the VersionTool 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 `vertool` 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 options from the Properties 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 options from the Properties 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, VersionTool 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 VersionTool.

To specify the results of the double-click action:

♦ **Select one of the following options from the Properties 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 following options as desired from the Properties 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 options from the Properties window:**

- Show Per-SID Information
 - Show Entire File History
- When you select this option, you can also specify a command to gather the history.

Part 4 — FreezePoint



Introduction to FreezePoint

page 253

Troubleshooting VersionTool and FreezePoint

page 267

Introduction to FreezePoint

13 



During the software development process it is often useful to create “freezepts” of your work at key points. Those freezepts 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 FreezePoint, you preserve freezepts quickly and simply, using a small amount of storage resource.

You can use FreezePoint through two functionally equivalent user interfaces. You can access the user interfaces with the following commands

- `freezepttool`—for the GUI
- `freezept`—for the CLI

Note – FreezePoint is a companion tool to the CodeManager product. Therefore, FreezePoint assumes that you are creating freezepts of CodeManager workspace hierarchies. You can also use FreezePoint 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 information by using the OpenWindows Magnify Help feature.

How FreezePoint Works

FreezePoint enables you to create freeze point files from CodeManager workspaces.¹ At a later time you can use the freeze point files to recreate the directory hierarchies contained in the workspaces.²

The freeze point file that FreezePoint creates is a text file that lists the default deltas in SCCS history files in the hierarchy. When you later recreate the hierarchy, FreezePoint 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 FreezePoint in the Directories and Files pane. FreezePoint recursively descends the directory hierarchies and identifies the most recently checked-in deltas in each SCCS history file. FreezePoint 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 FreezePoint 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.

1. Nonworkspace directory hierarchies that contain SCCS history files can also be preserved using FreezePoint. FreezePoint issues a warning if the directory is not a workspace.

2. 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 retrieved using the SCCS `get` command with no options specified.

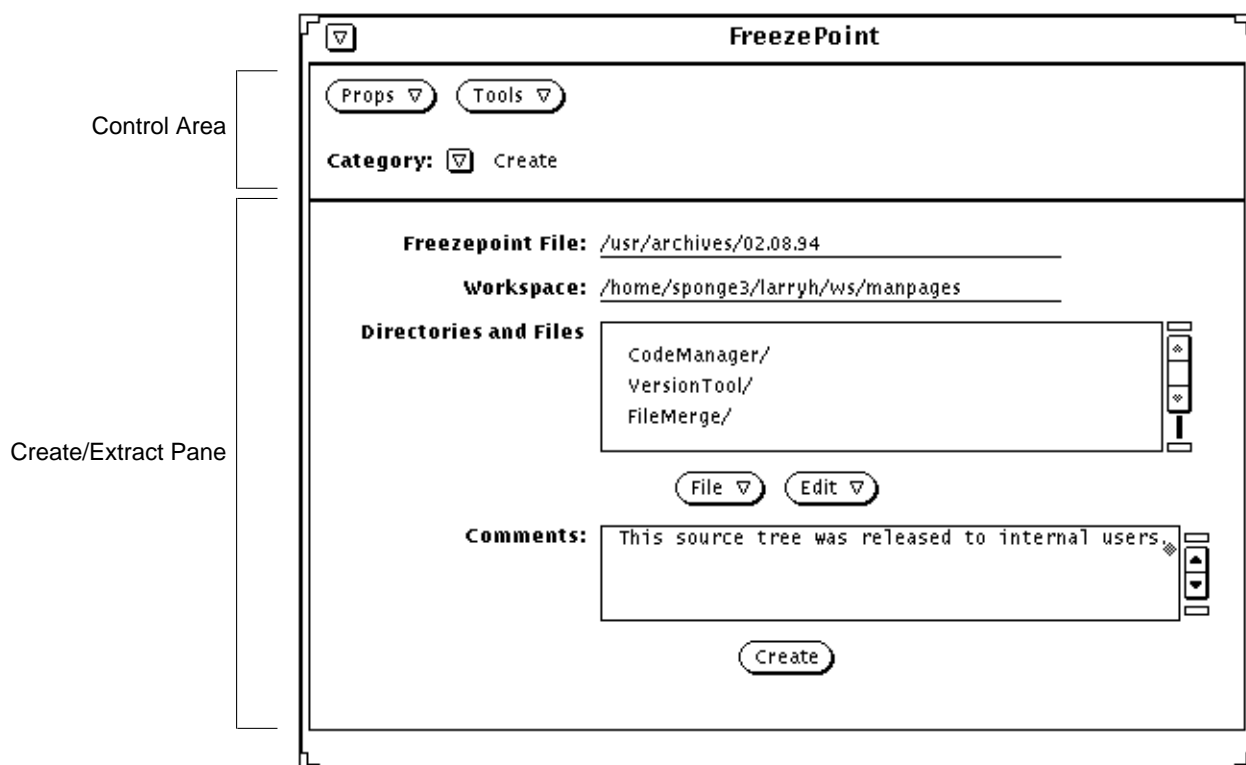


Figure 13-1 The FreezePoint Base Window

Terminology

FreezePoint File

A freezePoint file is a list of the default deltas from the SCCS history files contained in the workspace hierarchy being preserved. The freezePoint file also contains the following information:

- The login name of the user who created the freezePoint
- 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 Section , “Details about the FreezePoint File,” on page 263 for more information.

Extract

The extract operation consists of creating a new directory hierarchy based on the information contained in the freezePoint 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 freezePoint file is created. Usually, the source workspace is also the directory hierarchy from which g-files are later extracted to recreate the hierarchy.¹

Destination Directory

The destination directory is the top-level directory into which the files listed in the freezePoint file are extracted. You specify the path name of this directory in the Extract pane of the FreezePoint base window.

1. You can specify an alternate source directory at the time you perform the extract operation.

Starting FreezePoint

♦ **To start the FreezePoint GUI, type the following:**

```
example% freezepttool &  
example%
```

After a moment, the FreezePoint window will appear.

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 FreezePoint.

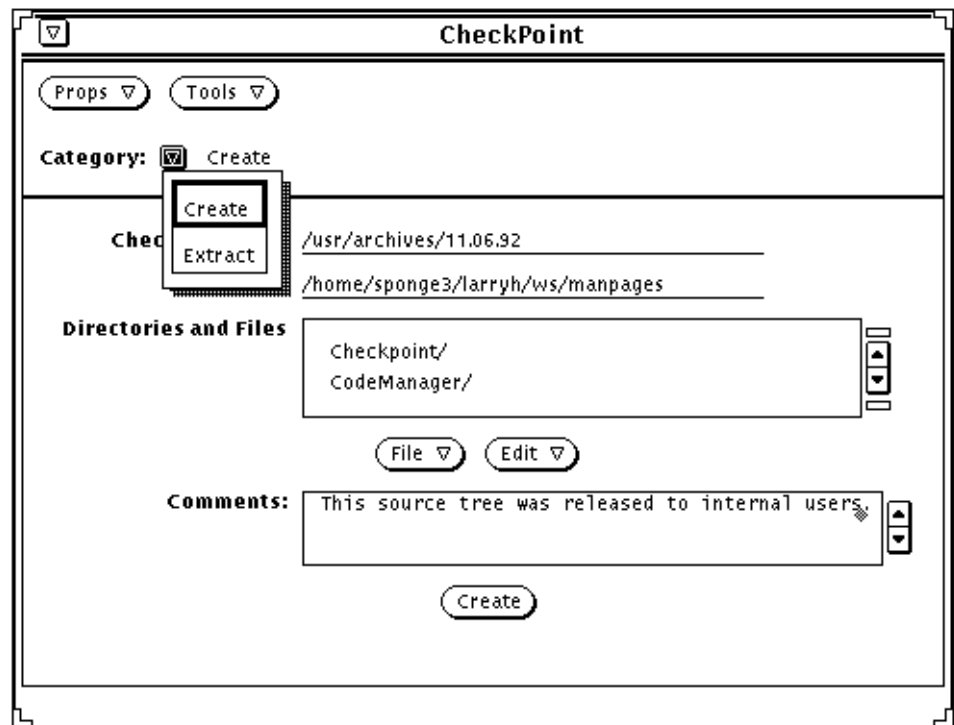


Figure 13-2 Choosing the FreezePoint Create Pane

2. Enter the name of a freeze point file.

- When FreezePoint initially appears, the FreezePoint File text field is automatically set to contain the file `freeze_point.out` appended to the path name of the directory from which freeze_point.
- Delete `freeze_point.out` and type the path name of your freeze point file in the FreezePoint File text field.

Note that path names that are not absolute are assumed to be relative to the directory in which FreezePoint is started.

3. Enter the name of the source workspace.

When you start FreezePoint, 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 FreezePoint 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 freezept 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;¹ double-click SELECT on the  icon to ascend.

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 freezept file for future reference.

6. Select the Create button to create the freezept file.

A counter on the bottom right corner of the base window footer displays the progress of the freezept operation.

1. Alternatively, you can select a directory icon and click on the Load Directory button to hierarchically descend.

Viewing or Modifying a Freezepoint File

Freezepoint 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 freezepoint 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 freezepoints. 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.

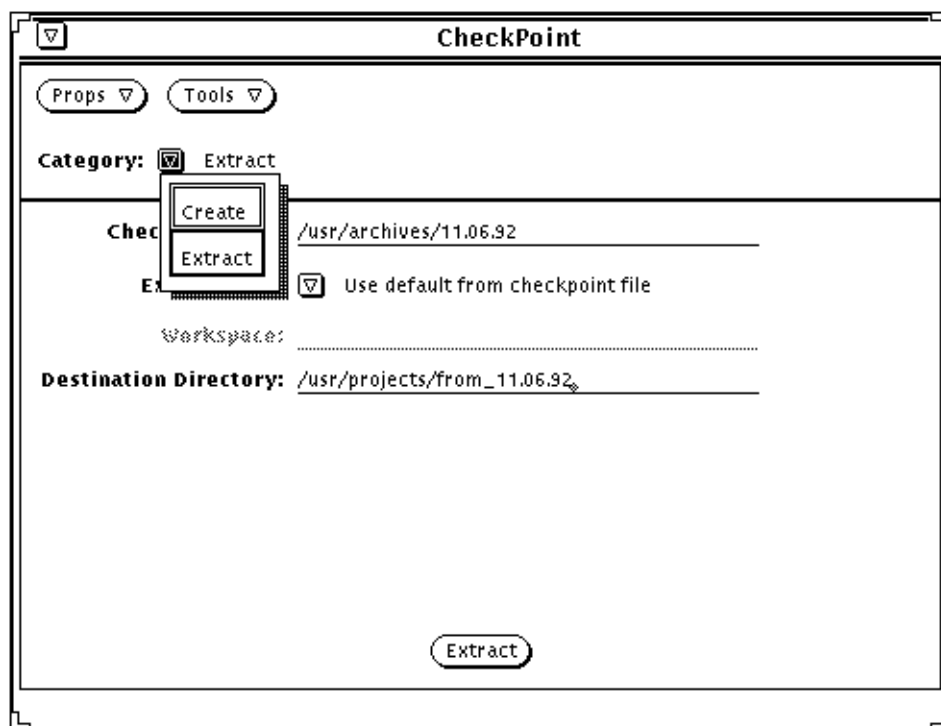


Figure 13-3 Choosing the FreezePoint Extract Pane

2. Type the name of an existing freeze point file.

Type the path name of your freeze point file in the FreezePoint File text field.

Note that path names that are not absolute are assumed to be relative to the directory in which FreezePoint is started.

3. Use the Extract From menu to choose how you will specify the source workspace.

By default, FreezePoint extracts files from the source workspace path name stored in the freeze point file when it was created.¹ By default, FreezePoint uses this path name as the source workspace from which to extract files. If

1. You can edit the freeze point file and change the path name of the source workspace.

you choose the Show Default menu item from the Extract From menu, FreezePoint displays the path name of the source workspace in the Workspace text field.¹

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 FreezePoint 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 `sccs 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 FreezePoint

- 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.

1. If you type Return after entering the path name of the freeze point file, FreezePoint automatically displays the default source workspace in the Workspace text field.

- If during an extraction, FreezePoint 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 FreezePoint.

Details about the Freezeptoint File

A freezeptoint 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 13-4 Three Entries From a Freezeptoint 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 FreezePoint to work properly with files in which SIDs have been renumbered as part of a CodeManager 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 FreezePoint to create a freezeptoint file, it calculates

the SMID for both the current delta and the root delta in the SCCS history file. Using both of these values, FreezePoint can identify a delta in a file even if its SID has been changed.

Why are SMIDs Necessary?

Note – This section briefly discusses how CodeManager merges SCCS history files. For more information, see Chapter 8, “How CodeManager Merges SCCS Files.”

When CodeManager 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 CodeManager 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 1.0 of TeamWare, SMID/SID translation is available only through the FreezePoint 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 ledb2f41"
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 "b1e24be3 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 2436calc" = SID 1.10  
SMID "46de4131 b95b9973 93958a07 b960074c" = SID 1.11
```

Troubleshooting VersionTool and FreezePoint

This chapter describes some of the most common problems in VersionTool and FreezePoint. It indicates where to look for information on how to overcome the problem. It is organized into the following sections:

- “Troubleshooting Checklist” on page 267
- “Reporting Problems” on page 268
- “Error Messages” on page 268

Troubleshooting Checklist

If you are having problems using VersionTool or FreezePoint, 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 *Installing SunSoft Developer Products on Solaris*.
- Is `/opt/bin` in your PATH?
If not, see *Installing SunSoft Developer Products on Solaris* for information on how to add `/opt/bin` to your PATH.
- Is `/usr/lang` in your PATH?
If not, see *Installing SunSoft Developer Products on Solaris* for information on how to add `/usr/lang` to your PATH.

- ❑ **Is the HELPPATH environment variable set?**
VersionTool relies on finding the `vertool.info` file in or near the directory that contains `vertool`. FreezePoint relies on finding the `freezepoint.info` file in or near the directory that contains `freezepoint`. Magnify Help provides on-line help for each control, window, pane, and error message displayed on the screen. See “Error Messages” for a list of the VersionTool and FreezePoint 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 VersionTool or FreezePoint 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. For information on how to display the version number, see Chapter 2, “VersionTool Basics,” for VersionTool and Chapter 3, “FreezePoint,” for FreezePoint.

Error Messages

VersionTool and FreezePoint display messages to provide you with information or tell you about an error.

Part 5 — ParallelMake

Introduction to ParallelMake

page 271

Using ParallelMake

page 273

The ParallelMake tool replaces the `make` command. Using ParallelMake, it is possible to distribute the process of building large programs over a number of processes and, in the case of multiprocessor systems, over multiple CPUs.

ParallelMake reads your makefiles and automatically:

- Determines which targets can be built in parallel¹
- Distributes the build of those targets over a number of processes set by you

If you already use the standard `make` command, the transition to ParallelMake is simple; most makefiles require little, if any, alteration, and the command is virtually identical to standard `make`.

The ParallelMake executable file (also named `make`) is executed in place of the standard `make` utility. The ParallelMake executable is installed in a different directory than standard `make`, and should be placed in your search path so that it is called when you execute the `make` command.

Parallel Builds

ParallelMake allows targets to be built in parallel on any single host. This concurrent processing can greatly reduce the elapsed time required to build a large system or project. ParallelMake supplies the special targets `.PARALLEL`, `.NO_PARALLEL`, and `.WAIT` for controlling concurrency and timing.

1. You can exert control over how the build is parallelized by how you write your makefiles.

New Options

There are two new options:

-M *machines_file*

Read the alternate machine specification file *machines_file* rather than the default file `~/ .make.machines`.

-R

Turn parallel build mode off.

Special-Purpose Targets

The following targets are specified in makefiles to control parallel processing.

Note - Makefiles that you write using these targets remain compatible with the standard version of `make` distributed with Solaris 1.0 and Solaris 2.0. Standard `make` accepts these targets without error (and without action).

`.NO_PARALLEL :`

Use this target to indicate which targets are to be processed serially.

`.PARALLEL :`

Use this target to indicate which targets are to be processed in parallel.

`.WAIT`

When you specify this target in a dependency list, ParallelMake waits until the dependencies that precede it are finished before processing those that follow, even when processing is parallel.

This chapter describes how to use ParallelMake. It assumes that you have a working knowledge of the standard `make` utility. It is a supplement to the standard `make` utility documentation.

A Note About Makefiles

The methods and examples in this chapter illustrate the kinds of problems that can be corrected with ParallelMake.

As procedures become more complicated, so do the makefiles that implement them. The trick is to know which approach will yield a makefile that works in a given situation. The examples in this chapter illustrate some common situations and some methods to simplify them using ParallelMake.

If you use a template approach in a project from the outset, chances are that the custom makefiles that evolve from the templates are more familiar, and therefore easier to understand, integrate, maintain, and reuse. After all, the less time you spend editing the makefiles, the more time you have to develop your program or project.

Building Targets in Parallel

Large software projects typically consist of multiple independent modules that can be built in parallel. ParallelMake supports concurrent processing of targets on a single machine; this concurrency can markedly reduce the time required to build a large project.

The .make.machines File

You can control the level of parallelization of a `make` operation on the local machine. By default, ParallelMake spawns a maximum of four concurrent processes. If you want to increase or decrease the maximum number of concurrent processes, you must create a file named `.make.machines` in your home directory¹, adding an entry for the local machine. An entry consists of both:

- The machine name
- A value that specifies the maximum number of concurrent processes allowed on that machine

Parallel Processes

Since most compilers spend more time in disk I/O functions than they do in compute intensive functions, some performance speedup can be achieved by building targets concurrently on the local machine. The optimal number of multiple builds for a machine depends on the computational power in relation to disk access speed. A single processor machine can get reasonable improvement compiling two targets in parallel; a multiprocessor machine can build more.

The number of concurrent builds is also limited by the available swap space and the disk space available in `/tmp`. As a default, ParallelMake attempts to run four builds on the local machine. This default can be modified for different machines by using the `max` option in the `.make.machines` file. This option takes the form:

```
max = n
```

1. You can specify an alternate file using the `-M` option.

where n is the number of concurrent targets. Here is an example:

```
venus max=7
pluto max=2
jupiter max=4
```

Figure 16-1 Example `.make.machines` file using the `max` specification

always uses the local host.

Turning off Parallelism

The `-R` option to ParallelMake turns off parallel processing, even when the `~/ .make.machines` file exists.

How Parallelism is Achieved

When given a target to build, ParallelMake 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 building in parallel, ParallelMake starts every target that it can. As these targets complete, ParallelMake then starts other targets. Nested invocations of ParallelMake are not run in parallel by default, but this can be changed (see Section , “Restricting Parallelism,” on page 278 for more information).

Collected Output

Since ParallelMake builds multiple targets concurrently, the output of each build will be produced at the same time. In order to avoid intermixing the output of various commands, ParallelMake collects output from each build separately. ParallelMake displays the commands before they are executed. If an executed command generates any output, warnings, or errors, ParallelMake 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 in parallel. Targets in makefiles which modify the same files may fail if those files are modified in parallel by two different targets. Some examples of possible problems are discussed in this section.

Dependency Lists

When building targets in parallel, the dependency lists should 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 in parallel. 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
```

Figure 16-2 Example Makefile with Inadequate Dependency Information

When built serially, the target `aux.o` would be built as a dependent of `prog1` and would be up to date for the build of `prog2`. If built in parallel, the link of `prog2` would begin before `aux.o` had been built, and would therefore be 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 `ParallelMake` 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
```

Figure 16-3 Example Use of `.WAIT`

The user may 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 in parallel 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. This can be accomplished 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
```

Figure 16-4 Use of Dynamic Macros in Temporary File Names

Concurrent Library Update

Another example is the default rule for creating libraries that also modifies a fixed file, that is, the library. The inappropriate `.c.a` rule causes `ParallelMake` to build each object file and then archive that object file. When `ParallelMake` archives two object files in parallel, the concurrent updates will corrupt the archive file.

```
.c.a:  
    $(COMPILE.c) -o $% $<  
    $(AR) $(ARFLAGS) $@ $%  
    $(RM) $%
```

Figure 16-5 Incorrect `.c.a` Rule for Parallel Building

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)
    ranlib $@
    $(RM) $(OBJECTS)
```

Figure 16-6 Example Suffix Rule for Building Libraries in Parallel

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
```

Figure 16-7 Parallel `yacc` Rule Using `+` Construct

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
```

Figure 16-8 Common Use of `xstr` in `.c.o` makefile Rule

ParallelMake 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. The user can use the special target `.NO_PARALLEL:` to tell ParallelMake not to build these targets in parallel. For example, if the objects being built using the `.c.o` rule were defined by the `OBJECTS` macro, the following entry would force ParallelMake to build those targets serially:

```
.NO_PARALLEL: $(OBJECTS)
```

Figure 16-9 Use of `.NO_PARALLEL:`

If most of the objects need to be built serially, it is easier and safer to force all objects to default to serial by including the `.NO_PARALLEL:` target without any dependents. Any targets which can be built in parallel can be listed as dependencies of the `.PARALLEL:` target:

```
.NO_PARALLEL:  
.PARALLEL: $(LIB_OBJECT)
```

Figure 16-10 Use of `.PARALLEL:` and `.NO_PARALLEL:`

Nested Invocations of ParallelMake

When ParallelMake encounters a target that invokes another ParallelMake command, it builds that target serially, rather than in parallel. This prevents problems with two different ParallelMake invocations attempting to build the same targets in the same directory. Such a problem might occur when two different programs are built in parallel, and each must access the same library. The only way for each ParallelMake invocation to be sure that the library is up to date is for each to invoke ParallelMake recursively to build that library. Note that ParallelMake only recognizes a nested invocation when the `$(MAKE)` macro is used in the command line.

If you have nested `make` 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 `make` commands that run in parallel, 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.

Error Messages

The following error messages may be issued by ParallelMake and are not documented in the standard `make` utility documentation.

- Ignoring unknown host *hostname*

The `~/ .make.machines` file includes the name of a host which cannot be found in the hosts database. See the file `/etc/hosts` or the NIS.

- Conditional macro conflict encountered

When building targets in parallel, `make` may encounter a problem with the use of conditional macros. This condition is rather obscure, and the warning only appears with the `-d` option. It indicates that `make` has encountered a target that may have to be rebuilt due to a new conditional macro setting, but the previously built object has not yet been used.

Glossary

Access control

The CodeManager facility by which users can control access to workspaces by CodeManager 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 CodeManager 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 CodeManager 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*.

FreezePoint

The TeamWare utility used to make snapshots of workspaces (or portions of them) at important junctures or “freezepoints.”

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 CodeManager transfer transactions are viewed from the child workspace perspective, and all conflicts are resolved in the child workspace.

Codemgr_wsdata directory

Every CodeManager workspace contains a “metadata” directory in its root directory named `Codemgr_wsdata`. CodeManager stores data about the workspace in `Codemgr_wsdata`. The presence of this directory is the sole factor that defines it as a CodeManager workspace (as opposed to a normal directory). CodeManager 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 65 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 CodeManager 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 CodeManager 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 CodeManager 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.
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.
FileMerge	The TeamWare utility used to merge deltas during Resolve transactions. See Chapter 6, “Resolving Conflicts,” and the section on FileMerge.
FLP	An FLP or <i>File List Program</i> is a program or script that generates a list of files to <code>stdout</code> that CodeManager then processes during Bringover and Putback transactions. See <i>def.dir.flp</i> .
g-file (SCCS)	The working copy of a file retrieved from an SCCS history file by the <code>sccs-get</code> command.
Integration workspace	A workspace to which multiple developers put back (merge) their work.
Lock	To assure consistency, the CodeManager file transfer transactions Bringover and Putback lock workspaces while they are working in them. Locks are recorded in the <code>Codemgr_wsdata/lock</code> file in each workspace; the CodeManager commands consult that file before acting in a workspace. See <i>read-lock</i> , <i>write-lock</i> .
Merge	To produce a single version of a file from two conflicting files (deltas). Usually accomplished with the assistance of the FileMerge program.
Notification	A CodeManager 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 <code>Codemgr_wsdata/children</code> file. Parent workspaces are typically used as integration areas, since development, testing, and conflict resolution occur in child workspaces.

Putback

The transaction used to update a parent workspace with respect to files contained in its child workspace. All CodeManager 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 CodeManager 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 CodeManager 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*.

Root directory

The top-level directory of a CodeManager workspace. This directory’s path name is the name by which the workspace is referred.

SCCS history file

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 g-file. See *g-file*.

SID

SCCS delta ID—The number used to represent a specific SCCS delta.

Undo

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.

Update

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.

VersionTool

The TeamWare program that provides a graphical interface to SCCS. See the section on VersionTool.

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. CodeManager 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 CodeManager 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 CodeManager 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

.make.machines, 274
.NO_PARALLEL: special target, 279
.PARALLEL: special target, 279
.WAIT special target, 276
~/.codemgr_resrc, 57
~/.codemgrtoolrc, 57

A

access control, workspace, 77
access_control file, 66, 77
AnswerBook, xvii
archiving libraries, 277
args file, 66, 100
Auto Bringover option, 120

B

backup directory, 66, 125
Base window
 pop-up menu, 219
branch delta, 216
branches, 216
branching, SCCS, 160
Bringover Create transaction, 69, **104 to 173**

effect of checked out files, 107
file system accessibility, 108
Force Conflicts option, 107
path name specification, 108
Preview option, 106
Quiet option, 106
search path, 109
Skip SCCS gets option, 106
Verbose option, 106
workspace locks, 108, 189
Bringover Transaction, **104 to 116**
Bringover Update transaction, **110 to 116**
 action summary table, 116
 conflict detection during, 137
 effect of checked-out files, 113
 file system accessibility, 114
 Force Conflicts option, 113
 path names, 114
 Preview option, 112
 Quiet option, 112
 Skip SCCS gets option, 112
 Verbose option, 112
 workspace locks, 115
Bringover/Putback transaction
 introduction, 28

C

Category menu (Transactions)

- window), 51
- Check In New menu item, 228
- Check In New window, 229
 - Check In button, 230
 - Directory text field, 229
 - Initial Comments pane, 230
 - List field, 229
 - Reset button, 230
- checked-out files, 113, 120
- children file, 66
- chooser, 102
- Chooser window, 55
- CLI, command-line interface
 - umbrella command, 38
- CodeManager
 - base window, 42
 - Chooser, 102
 - control area, 48
 - customization, 57
 - menus, 48
 - moving an existing project, 145
 - properties, 57
 - starting a project, 145
 - starting execution, 40
 - transaction model, 95
 - Workspace Graph pane, 43
- CODEMGR_WS variable, 92
- Codemgr_wsdata, 26, 65
 - access_control file, 66
 - args file, 66, 100
 - backup directory, 66, 125
 - children file, 66
 - history file, 66
 - locks file, 67
 - nametable file, 67
 - notification file, 67, 83
 - parent file, 67
- CODEMGR_WSPATH variable, 93
- codemgrtool, 40
- Command Output menu item, 222
- command-line interface
 - umbrella command, 38
- Commands button, 228
 - Check In menu item, 228
 - Check In New menu item, 228
 - Check Out menu item, 228
 - checking in a new file, 246
 - checking in files, 244
 - checking out files, 243
 - displaying the differences between deltas, 247
 - Edit menu item, 228
 - editing checked out files, 246
 - Latest Diffs menu item, 228
 - unchecking out a file, 247
 - Uncheckout menu item, 228
- Commands button menu, 226
- Commands button, History window, 226
 - Check In menu item, 227
 - Check Out menu item, 227
- Commands Output button, 241
- Commands Output menu item, 241
- comment
 - Putback transaction, 120
- common ancestor delta, 138
- concurrent file modification, 277
- conflict
 - detection during Bringover, 137
 - merging files in conflict, 155
 - resolving, 136
- copy-modify-merge
 - example, 19
 - model, 18
- creating a workspace, 68
- current difference (FileMerge), 144

D

- def.dir.flp, 99
- default list
 - loading, 101
 - saving, 101
- defaults files, 57
 - ~/ .codemgr_resrc, 57
 - ~/ .codemgrtoolrc, 57
- Delete, 69
 - Codemgr_wsdata Directory only, 69

- Sources and Codemgr_wsdata
Directory, 69
- delta, 217
- delta branches, 223
- Delta Comment pane, 226
- delta ID, 217
- dependency lists, 276
 - explicit ordering, 276
 - implicit ordering, 276
- difference (FileMerge)
 - current, 144
 - defined, 144
 - next, 144
 - previous, 144
 - resolved, 144
- Differences button menu, 224
 - Use FileMerge menu item, 225
 - View diff Output menu item, 225
- double-click action
 - Workspace Graph pane, 47

E

- Edit menu, 48
- environment variables, 92
- examples
 - Bringover/Putback/Resolve cycle, 167 to 182
 - merging SCCS history files, 156
 - reparenting, 74

F

- file
 - collision, 278
 - concurrent modification, 277
- File button, 220, 235
 - Load menu item, 221
 - loading a directory, 236
 - Unload menu item, 221
 - unloading a directory, 236
- file chooser, 102
- File History menu item, 222
- File List pane, 55

- changing contents of, 101
- constructing directory and file lists, 100
- selecting files, 101

File List Program, see FLP

- file lists
 - initial state, 100
 - transactions, 100
- File menu, 48
- File status
 - menu item, 223
 - pop-up window, 223
- FileMerge program, 142 to 144
- files
 - merging, 155
 - relationships between files in parent and child workspaces, 30 to 34
 - specifying for transactions, 98

FLP, 99

- default (`def.dir.flp`), 99

footer messages, 62

Force Conflicts option, 107, 113

FreezePoint

- creating a freeze point file, 257
- extract a source directory, 260

FreezePoint terms, 256

- extract, 256
- freeze point file, 256

G

- graphical user interface (GUI),
 - overview, 39
- grouping files

H

- help facilities
 - AnswerBook, xvii
 - Magnify Help, xvii
 - Notices, xvii
- hierarchy, workspace, 146 to 151
- history file, 217

history file, 66, 87
 History window, 223, 239
 Commands button, 226
 Commands button menu, 226
 Delta Comment pane, 226
 Differences button menu, 224
 pop-up menu, 227
 Version Information pane, 226
 View Contents button, 226
 History window Commands button
 Check In menu item, 227
 Check Out menu item, 227
 hosts, definition, 274

I

icons
 drag and drop, 46

L

Latest Diffs menu item, 228
 library update, concurrent, 277
 limitations on makefiles, 276
 Load
 workspaces into Workspace Graph
 pane, 43
 Load button, 221, 237
 Load menu option, 221
 locking workspaces, 90
 locks
 removing workspace locks, 91
 viewing workspace locks (GUI), 91
 locks file, 67

M

-M option, read machines file, 274
 machines
 file, 274
 file option, 274
 macro
 dynamic, 277
 Magnify Help, xvii, 39
 spot help, xvii
 makefiles, limitations, 276
 manual pages, SunOS, xvii
 max option, 274
 maximum builds, controlling, 274
 menu buttons
 Edit, 48
 File, 48
 Properties, 49
 Tools, 49
 Transactions, 49
 View, 48
 menus
 CodeManager, 48
 merging files
 not in conflict, 154
 merging SCCS history files, 33
 example, 156
 in conflict, 155
 messages
 footer, 62
 metadata directory, 26
 mkfile command, 268
 multiple targets, 278

N

name fields, workspace, 44
 nametable file, 67
 new
 special targets for make, 272
 next difference (FileMerge), 144
 nodes, 216
 Notices, xvii
 notification, 83
 notification file, 67

O

On-Line Help
 answerbook, xvii
 notices, xvii
 output, collected, 275

output, commands, 241

output, transaction, 97

P

parallelism

 restricting, 278

 turning off, 275

Parent, 72

parent file, 67

parent/child introduction, 23 to 26

Preview option, 106, 112, 119

previous difference (FileMerge), 144

project

 moving an existing project, 145

 starting a new project, 145

Properties menu, 49

Properties window, 57, 231

Props button, 231, 247

 changing double click action, 249

 changing file list display, 248

 changing history graph display, 249

 changing history information

 display, 249

 defining an editor, 248

 VersionTool Props menu item, 231

Putback, 123

Putback transaction, **117 to 123**

 access control, 122

 action summary table, 123

 Auto Bringover option, 120

 comment, 120

 effect of checked-out files, 120

 file system accessibility, 121

 path names, 121

 Preview option, 119

 Quiet option, 120

 Skip SCCS gets option, 120

 Verbose option, 120

 workspace locks, 122

Putback/Bringover transactions,

 introduction, 28

Q

Quiet option, 106, 112, 120

R

-R option, turn off parallelism, 275

Rename, 70

reparenting a workspace, 26, **71 to 76**

 example, 74

Resolve transaction, **139 to 144**

 introduction, 34

 merging SCCS history files, 161

 preparing files for conflict

 resolution, 138

 summary, 136

restricting parallelism, 278

restrictions on makefiles, 276

S

s.file, 217

SCCS, xiii, 215

 branches, 216

 concepts, 216

 delta, 217

 delta branches, 223

 delta ID, 217

 history file, 217

 nodes, 216

 terminology, 216

 version, 217

 viewing output, 241

SCCS history files, 21, 27, 97, 117, 153

 branching, 160

 common ancestor delta, 138

 merging, 33, **153**

 resolving, 138

s-dot-file, 217

selecting files, transactions, 101

selecting workspaces, 45

Set Default button, 57

SID, 217

Skip SCCS gets option, 106, 112, 120

Source Code Control System, xiii, 215
spot help, see Magnify Help
starting a project, 145
 default FLP, 146
 SCCS file location, 145
swap space, 268

T

targets
 .NO_PARALLEL:, 279
 .PARALLEL:, 279
 .WAIT, 276
 multiple, 278
Tools menu, 49, 263
transaction model, 95
Transaction Output window, 56, 97
transactions
 file lists, 100
 file specification, 98
Transactions menu, 49
Transactions window, 51

U

Uncheckout menu item, 228
Undo transaction, **124 to 127**
 implementation, 125
 workspace locks, 125
Unload menu option, 221
unlocking workspaces, 90
Use FileMerge menu item, 225

V

variables, environment, 92
 CODEMGR_WS, 92
 CODEMGR_WSPATH, 93
Verbose option, 106, 112, 120
version, 217
Version Information pane, 226
VersionTool, 215
VersionTool Props menu item, 231

View button, 222, 238
 Command Output menu item, 222
 File History menu item, 222
 using the History window, 239
 viewing a history graph, 239
 viewing SCCS file status, 241

View Contents button, 226
View diff Output menu item, 225
View menu, 48

W

windows
 Chooser, 55
 Transaction Output window, 56
 Transactions, 51
workspace
 access control, 77, 108, 115
 command history log, 87
 create, 68
 create using Bringover Create, 69
 delete, 69
 event notification, 83
 hierarchies, 24
 hierarchy configuration, 146, 151
 locking, 90
 metadata directory
 (Codemgr_wsdata), 65
 moving, 70
 name fields, 44
 removing locks, 91
 renaming, 70
 reparenting, **71 to 76**
 selection, 45
 viewing locks from GUI, 91
Workspace Create, 68
Workspace Graph pane, 43
 double-click action, 47
 loading workspaces, 43
 pop-up menu, 46
workspace introduction, 21 to 23